۱

مقدمه

سیستم های کامپیوتری در حال تحول هستند. از سال ۱۹۴۵ ،که آغاز دوران کامپیوترهای امروزی محسوب می شود، تا حدود سال های ۱۹۸۵کامپیوترها هم بزرگ بودند و هم گران . بطوریکه یک مینی کامپیوتر حداقل هزاران دلار قیمت داشت. در نتیجه، اغلب سازمانها فقط تعداد انگشت شماری کامپیوتر در اختیار داشتند و به دلیل عدم امکان اتصال، همین کامپیوترها هم مستقل از یکدیگر کار می کردند.

اما تقریباً از اواسط دهه هشتاد میلادی، دو پیشرفت در فن آوری باعث تغییر شرایط شد . اولین تغییر مربوط به توسعه ریزپردازنده های[۱] قدرتمند بود . در ابتدا، این ریزپردازنده ها ماشین های ۸ بیتی بودند، اما کمی بعدتر پردازنده های ۱۶، ۳۲ و ۶۴ بیتی هم متداول شدند. قدرت محاسبه بسیاری از این ریزپردازنده ها به اندازه کامپوترهای بزرگ[۲] اما با کسری از قیمت آنها بود.

میزان رشدی که طی نیم قرن گذشته در عرصه فن آوری کامپیوترها رخ داده ، واقعاً مبهوت کننده و در مقایسه با صنایع دیگر، کاملاً بی سابقه است . بطوریکه از ماشین هایی با قیمت ۱۰ میلیون دلار که قادر به اجرای فقط یک دستورالعمل در عرض یک ثانیه بودند، امروزه به ماشین هایی رسیده ایم که ۱۰۰۰ دلار قیمت داشته و در عوض قادر به اجرای یک میلیارد دستورالعمل در واحد ثانیه بوده و نسبت کارآیی به قیمت آنها 10^{13} برابر شده است . اگر ماشین ها هم در این فاصله زمانی با همین سرعت رشد می کردند، امروز رولزرویس فقط یک دلار قیمت داشت و در یک میلیارد مایل فقط حدود چهار لیتر بنزین مصرف می کرد.

پیشرفت دوم مربوط به پیدایش شبکه های کامپیوتری پرسرعت بود. **شبکه های محلی**[۳] یا **LANها** باعث می شوند تا صدها ماشین در یک ساختمان طوری به هم متصل شوند که بتوان مقادیر کمی از اطلاعات را در عرض تقریباً چند میلی ثانیه بین آنها انتقال داد. انتقال مقادیر بزرگ تر داده ها در بین ماشین هایی با سرعت ۱۰۰ میلیون تا ۱۰ میلیارد بیت بر ثانیه امکان پذیر است . **شبکه های گسترده**[۴] یا **WANها** امکان اتصال میلیونها ماشین در سرتاسر جهان با سرعت متغیر از ۶۴ کیلوبیت در ثانیه(Kbps) تا گیگابیت در ثانیه را فراهم کرده است .

در اثر تمامی این فن آوری ها ،امروزه در کنار هم قرار دادن سیستم های محاسباتی متشکل از تعداد زیادی کامپیوتر که بوسیله یک شبکه کامپیوتری پرسرعت بهم متصل شده باشند ، امروزه نه تنها امکان پذیر ، بلکه آسان هم شده است. این شبکه های کامپیوتری یا به اصطلاح **سیستم های توزیعی** [۵] با

---

[1] microprocessor
[2] mainframe
[3] local-area networks
[4] wide-area networks
[5] distributed systems

سیستم های متمرکز [۶](یا **سیستم های تک پردازشگری**[۷]) قدیمی که فقط از یک کامپیوتر، دستگاه های جانبی و احتمالاً پایانه های دور[۸] تشکیل می شدند، متفاوت است.

۱-۱-تعریف سیستم های توزیعی

در منابع مختلف تعاریف مختلفی هم برای سیستم های توزیعی ارائه شده است. اما هیچیک نه کامل است و نه با دیگری همخوانی دارد. برای اهداف ما در این کتاب ، ارائه یک تعریف تقریبی هم کفایت می کند :

سیستم توزیعی در واقع مجموعه ای از کامپیوترهای مستقل[۹] *است که برای کاربر خود ، مانند یک سیستم منسجم*[۱۰] *و منفرد*[۱۱]*به نظر می رسد.*

این تعریف نکته های مختلف و مهمی در خود دارد. اول آنکه، یک سیستم توزیعی از مؤلفه های(یعنی، کامپیوترهای) خود مختار تشکیل می شود . نکته دوم اینکه، کاربران(چه افراد و چه برنامه ها) تصور می کنند که با یک سیستم منفرد کار می کنند. به این معنا که مؤلفه های خودمختار[۱۲] بایستی به روشی با هم همکاری[۱۳] کنند. نحوه ایجاد این تشریک مساعی هسته سیستم های توزیعی را تشکیل می دهد. توجه داشته باشید که هیچ فرضی راجع به نوع کامپیوترها صورت نمی گیرد. اما در مجموع، حتی در داخل یک سیستم منفرد هم، انواع کامپیوترها، از کامپیوترهای بزرگ کارآ گرفته تا گره های کوچک در شبکه های حسگر، به چشم می خورد. به همین ترتیب، هیچ فرضی راجع به نحوه اتصال کامپیوترها صورت نمی گیرد. در ادامه فصل راجع به این جنبه های مختلف بحث خواهیم کرد.

شاید مفیدتر باشد که به جای تعریف مفاهیم، روی ویژگی های مهم سیستم های توزیعی تمرکز کنیم. یکی از ویژگی ها این است که تفاوتهای بین کامپیوترهای مختلف و نحوه ارتباط آنها تا حد زیادی از دید کاربران مخفی می باشد . همین امر درمورد سازمان داخلی[۱۴] سیستم های توزیعی صدق می کند. ویژگی مهم دیگر اینکه کاربران و برنامه های کاربردی می توانند به صورتی منسجم و متحد با سیستم توزیعی ارتباط برقرار کنند، مستقل از این که این تعامل در کجا و کی صورت پذیرد.

در مجموع ، توسعه یا به مقیاس درآوردن[۱۵] سیستم های توزیعی باید نسبتاً به راحتی انجام شود. این ویژگی، هرچند پیامد مستقیم استقلال کامپیوترهاست ، در عین حال باعث مخفی سازی کلی نقش واقعی این کامپیوترها در سیستم می شود. حتی در صورت خرابی موقت برخی بخشهای سیستم

---

[6] centralized systems
[7] single-processor systems
[8] Remote terminals
[9] independent computers
[10] coherent
[11] single
[12] autonomous components
[13] collaborate
[14] internal organization
[15] scale

توزیعی، این سیستم ها معمولاً همواره در دسترس هستند . با این وجود،کاربران و برنامه های کاربردی نبایستی متوجه تعمیر یا تعویض بخش های معیوب و یا افزودن یا عدم افزودن بخش های جدید برای خدمت رسانی به کاربران یا برنامه های کاربردی بشوند.

مطابق شکل ۱-۱، سیستم های توزیعی بمنظور پشتیبانی از کامپیوترها و شبکه های ناهمگن[۱۶] و در عین حال ارائه نمایی تک سیستمی، غالباً بوسیله لایه ای از نرم افزار سازماندهی می شوند – یعنی، بصورت منطقی بین یک لایه سطح بالاتر، شامل کاربران و برنامه های کاربردی و یک لایه زیرین، حاوی سیستم های عامل و امکانات ارتباطی قرار داده می شوند. به همین دلیل، این نوع سیستم توزیعی گاهی اوقات **میان افزار**[۱۷] نامیده می شود.

شکل ۱-۱- سیستم توزیعی که بصورت میان افزار سازماندهی شده است . لایه میان افزار به چندین ماشین گسترش یافته و برای تمامی برنامه های کاربردی یک رابط واحد ارائه می کند(داخل شکل: کامپیوتر۱، برنامه کاربردیA، سیستم عامل محلی ۱،کامپیوتر۲،کامپیوتر۳، برنامه کاربردی B، سیستم عامل محلی۲، سیستم عامل محلی۳، کامپیوتر ۴، برنامه کاربردی C، سیستم عامل محلی ۴، لایه سیستم توزیعی(میان افزار)، شبکه).

شکل ۱-۱، چهار کامپیوتر شبکه شده و سه برنامه کاربردی را نمایش می دهد که برنامه کاربردی B آن در بین ماشین های ۲و۳ توزیع شده است . برای هریک از برنامه های کاربردی یک رابط واحد ارائه می شود. سیستم توزیعی امکانی را برای ارتباط مؤلفه های یک برنامه کاربردی منفرد با یکدیگر و همچنین برای ارتباط برنامه های کاربردی مختلف با یکدیگر فراهم می کند. درعین حال، به بهترین و منطقی ترین روش ممکنه، تفاوتهای بین سخت افزار و سیستم های عامل را از برنامه های کاربردی مخفی می کند.

۱-۲- اهداف

امکان ساخت سیستم های توزیعی لزوماً به معنای خوب بودن این ایده نیست. مثلاً هرچند فن آوری نوین امکان قرار دادن چهار فلاپی دیسک درایو روی کامپیوترهای شخصی را فراهم آورده ، ولی به دلیل بی مورد بودن کسی حتی به فکر انجام آنهم نمی افتد . در این بخش، راجع به چهار هدف مهمی بحث خواهیم کرد که ساخت سیستم های توزیعی فقط در سایه برآورده شدن آنها موجه و معقول است : یک سیستم توزیعی باید امکان دسترسی راحت به منابع را فراهم کند ؛ باید به صورتی منطقی و صحیح توزیع منابع در سرتاسر شبکه را مخفی کند ؛ باید باز باشد؛ و بالاخره اینکه باید مقیاس پذیر باشد.

۱-۲-۱- دسترس پذیر کردن منابع

[16] heterogeneous computers and networks
[17] middleware

هدف اصلی سیستم های توزیعی آن است که کاربران (وبرنامه های کاربردی) بتوانند به راحتی به منابع از راه دور دسترسی پیدا کرده و بصورتی کنترل شده و مؤثر، در آنها شریک[۱۸] شوند. این منابع ممکن است از هر نوعی باشند، اما بعنوان برخی نمونه ها می توان به چاپگرها، کامپیوترها، تجهیزات ذخیره سازی، داده ها ، فایلها، صفحات وب و شبکه ها اشاره کرد. دلایل متعددی را می توان برای لزوم اشتراک منابع ، از جمله مسائل اقتصادی ، ذکر کرد. بعنوان مثال، از نظر اقتصادی به صرفه تر است که در یک دفتر کار کوچک چندین کاربر از یک چاپگر واحد استفاده کنند تا اینکه هریک مجبور به خرید و نگهداری از یک چاپگر جداگانه باشد. به همین ترتیب، از نظر اقتصادی به صرفه است که از منابع هزینه بری از قبیل ابرکامپوترها، سیستم های ذخیره کارآ، شکل نگارها[۱۹] و دیگر وسائل جانبی گران قیمت به صورت اشتراکی استفاده کنیم.

بعلاوه، اتصال کاربران و منابع باعث تسهیل همکاری و تبادل اطلاعات می شود که از نمونه های آن می توان به موفقیت اینترنت در سایه پروتکلهای ساده آن جهت تبادل فایلها، کارهای پستی، اسناد، صوت و شکل اشاره کرد. امروزه، اتصال به اینترنت باعث ایجاد سازمانهای مجازی[۲۰] متعددی شده که در آنها ،گروههای انسانی که گاهی اوقات حتی چند صد هزار کیلومتر از هم فاصله دارند ، با استفاده از **گروه افزار** [۲۱]— یعنی، نرم افزاری برای ویرایش اشترکی، کنفرانس های تلفنی و غیره - با هم کار می کنند. به همین ترتیب، ارتباطات اینترنتی با فراهم کردن امکان تجارت الکترونیک، موجب شده اند تا بتوان بدون مراجعه به فروشگاه یا حتی خروج از خانه انواع کالاها را خرید و فروش کرد.

اما همین افزایش امکان ارتباط و شراکت ، نیاز روزافزون به امنیت را هم مطرح می کند. در حال حاضر، سیستم ها از محافظت کمی در برابر استراق سمع[۲۲] و نفوذ [۲۳] در ارتباط برخوردار هستند. کلمات عبور و دیگر اطلاعات حساس غالباً بصورت متن واضح[۲۴] (یعنی فاقد رمز) از طریق شبکه ارسال شده و یا در خدمتگزارهایی ذخیره می شوند که نمی توان اطمینان چندانی به معتبر بودن آنها داشت. در این زمینه هنوز عرصه برای بهبود باز است. بعنوان مثال، امروزه خریداران می توانند صرفاً با ارائه یک شماره کارت اعتباری، حتی در صورتی هم که صاحب واقعی کارت نباشند، کالا خریداری کنند. اما در آینده، سفارش دهی به این ترتیب فقط در صورتی امکان پذیر خواهد بود که با قرار دادن کارت در کارت خوان ، دارنده کارت بصورت فیزیکی هم صاحب کارت باشد .

[18] share
[19] imagesetters
[20] virtual organizations
[21] groupware
[22] eavesdropping
[23] intrusion
[24] cleartext

مشکل امنیتی دیگر مربوط به ردیابی ارتباطات برای ساخت نمودار ترجیحی[۲۵] از کاربر مورد نظر است(Wang وگروه همکاران ۱۹۹۸). این دست ردیابی ها ، خصوصاً اگر بدون اطلاع کاربر انجام شود، آشکارا باعث نقض حریمهای خصوصی[۲۶] افراد خواهد شد. مشکل دیگر این است که افزایش قابلیت ارتباط می تواند باعث ارتباطات ناخواسته[۲۷] و از جمله پست های الکترونیکی بیهوده ای شود که غالباً مزاحم [۲۸]نامیده می شوند. در این موارد، لازم است بکمک فیلترهای اطلاعاتی خاص که براساس محتوای پیام های ورودی اقدام به انتخاب آنها می کنند، از خود مراقبت کنیم.

۲-۲-۱- شفافیت توزیع[۲۹]

یکی از اهداف مهم سیستم های توزیعی مخفی سازی این واقعیت است که فرآیندها و منابع مربوط به آن به صورت فیزیکی در بین ماشین های متعدد توزیع شده اند. سیستم توزیعی که در نظر کاربران و برنامه های کاربردی خود بصورت یک سیستم کامپیوتری منفرد جلوه کند ، اصطلاحاً **شفاف**[۳۰] نامیده می شود. ابتدا بیایید نگاهی به انواع شفافیت ها در سیستم های توزیعی داشته باشیم. سپس به این پرسش کلی پاسخ می دهیم که آیا همیشه نیازمند شفافیت هستیم یا فقط گاهی اوقات به آن نیاز داریم .

انواع شفافیت

گرچه مفهوم شفافیت را می توان درمورد جنبه های مختلف یک سیستم توزیعی بکار برد ، مهمترین آنها در شکل ۲-۱ ارائه شده است.

شفافیت	شرح
دسترسی[۳۱]	مخفی سازی تفاوتها در ارائه داده و نحوه دسترسی به منابع
مکان[۳۲]	مخفی سازی محل استقرار منبع
مهاجرت[۳۳]	مخفی سازی احتمال جابجایی منبع به محل دیگر
مکان یابی مجدد[۳۴]	مخفی سازی جابجایی منبع به محل دیگر در حین استفاده از آن
کپی برداری[۳۵]	مخفی سازی وجود چند کپی از منبع
همروندی[۳۶]	مخفی سازی اشتراک منبع در بین کاربران رقیب

[25] preference profile
[26] privacy
[27] unwanted communication
[28] spam
[29] distribution transparency
[30] transparent
[31] Access
[32] Location
[33] Migration
[34] Relocation
[35] Replication
[36] Concurrency

خرابی[۳۷]	مخفی سازی خرابی و ترمیم خرابی منبع

شکل ۲-۱- اشکال مختلف شفافیت در سیستم های توزیعی(۱۹۹۵، ISO)

شفافیت دسترسی به بحث در مورد مخفی سازی تفاوت های ارائه داده و نحوه دسترسی به منابع بوسیله کاربران می پردازد. در پایین ترین سطح ، هدف مخفی سازی تفاوت بین معماری ماشین هاست، اما مهمتر از آن رسیدن به توافق در زمینه نحوه ارائه داده توسط ماشین ها و سیستم عامل های مختلف است. به عنوان مثال، سیستم های کامپیوتری موجود در یک سیستم توزیعی ممکن است سیستم های عامل مختلفی را اجرا کرده و هریک از قراردادهای نامگذاری فایل[۳۸] مخصوص به خود استفاده کنند .تفاوت بین قراردادهای نامگذاری و همچنین نحوه دستکاری[۳۹] فایل ها، بایستی تماماً از کاربران و برنامه های کاربردی مخفی باشد.

گروه مهم دیگری از انواع شفافیت مربوط به محل منبع است. هدف از این **شفافیت مکان** آن است که کاربران نتوانند محل استقرار فیزیکی منبع در سیستم را شناسایی کنند. نامگذاری نقش مهمی در کسب شفافیت توزیعی دارد. شفافیت مکان بخصوص از طریق نسبت دادن اسامی منطقی[۴۰] به منابع - یعنی اسامی که در آنها نام محل بصورت مخفیانه کدگذاری نشده باشد – امکانپذیر است. بعنوان نمونه، در نامگذاری URL *http://www.prenhall.com/index.html* هیچ نشانه ای در مورد محل خدمتگزار اصلی وب Prentice Hall به چشم نمی خورد. بعلاوه، از URL نمی توان دریافت که آیا *index.html* همیشه در موقعیت فعلی خود بوده یا اخیراً به آن منتقل شده است . سیستم های توزیعی که بتوان منابع آنها را بدون تأثیرگذاری بر نحوه دسترسی به آنها انتقال داد، اصطلاحاً دارای **شفافیت مهاجرت** هستند. حالت دیگر هنگامی است که بتوان منابع را درحین دسترسی به آنها و بدون کوچکترین اطلاعی به کاربر یا برنامه کاربردی مجداً مکان یابی کرد. در این موارد، سیستم ها اصطلاحاً از **شفافیت مکان یابی مجدد** پشتیبانی می کنند. بعنوان نمونه ای از این شفافیت مکان یابی مجدد می توان به امکان ادامه استفاده کاربران متحرک از لپ تاپهای خود در حین جابجایی از یک نقطه به نقطه دیگر و بدون قطع ارتباط(حتی موقت هم) اشاره کرد .

همانطور که بعداً خواهیم دید، کپی برداری نقش مهمی در سیستم های توزیعی ایفا می کند. بعنوان مثال، می توان منابع راکپی کرده و با قراردادن یکی از کپی ها در مجاورت محل مورد دسترسی به آن ، سطح کارآیی یا دسترسی را افزایش ارتقاء داد. **شفافیت کپی برداری** به مخفی سازی وجود چندین کپی از یک منبع می پردازد. برای مخفی سازی کپی ها از کاربران، تمام کپی ها باید نام واحدی داشته باشند. در نتیجه، سیستمی که از شفافیت کپی برداری پشتیبانی می کند لزوماً باید از شفافیت مکان

[37] Failure
[38] file-naming conventions
[39] manipulate
[40] logic names

یابی هم پشتیبانی کند، چون در غیراینصورت ارجاع به کپی ها در محل های مختلف غیرممکن خواهد بود .

گفتیم که یکی از اهداف مهم سیستم های توزیعی ، برقراری امکان اشتراک منابع[۴۱] است. هرچند در بسیاری از موارد، از جمله ارتباطات، اشتراک منابع با همکاری انجام می شود ، نمونه های زیادی هم می توان از اشتراک رقابتی منابع[۴۲] ذکر کرد. بعنوان مثال، ممکن است دو کاربر مستقل فایلهای خود را روی یک خدمتگزار فایل واحد ذخیره کرده و یا به جداول واحدی در یک پایگاه داده ای مشترک[۴۳] دسترسی داشته باشند. در این موارد، هیچیک از کاربران نباید کوچکترین اطلاعی از واقعیت استفاده کاربر دیگر از آن منبع داشته باشند. این پدیده به نام **شفافیت همروندی** خوانده می شود . اما مهم این است که دسترسی همزمان به یک منبع مشترک، منبع را در حالت منسجم نگهدارد. انسجام[۴۴] از طریق مکانیزم های قفل کردن[۴۵] قابل ایجاد بوده و به کمک آن ، کاربران دسترسی انحصاری به منبع مورد نظر خواهند داشت . مکانیزم اصلاح شده دیگر ، استفاده از تراکنش ها[۴۶] است. اما همانطور که در فصلهای بعد خواهیم دید، پیاده سازی تراکنش ها در سیستم های توزیعی بسیار مشکل است.

لسلی لمپرت تعریف دیگری برای سیستم های توزیعی ارائه می دهد:" هنگامی خواهید دانست یک سیستم توزیعی دارید که شکست یک کامپیوتر که نشنیده اید، ادامه کار شما را متوقف کند". این تعریف دقیقاً روی یکی دیگر از مسائل مهم در طراحی سیستم های توزیعی، یعنی مبحث خرابی ها ، انگشت می گذارد. ایجاد **شفافیت خرابی** در سیستم های توزیعی به این معناست که کاربر متوجه خرابی و عملکرد نادرست یک منبع نشده (حتی شاید کوچکترین گزارشی هم از آن دریافت نکند) و سپس سیستم اقدام به ترمیم آن خرابی کند. مخفی سازی خرابی ها یکی از مشکل ترین مسائل در سیستم های توزیعی بوده و همانطور که در فصل ۸ خواهیم گفت، برخی فرضیات بظاهر واقع بینانه آنرا حتی غیرممکن هم می کند. مشکل اصلی در مخفی سازی خرابی ها ناشی از عدم توانایی در تمایز بین منبع مرده[۴۷] از منبع کند[۴۸] است. مثلاً در بعضی موارد، پس از چندین تماس با یک خدمتگزار وب مشغول، جستجوگر به سررسید زمانی می رسد و گزارش می دهد که صفحه وب مورد نظر غیرقابل دسترسی است. در این حالت کاربر نباید نتیجه گیری کند که خدمتگزار واقعاً ازکارافتاده است.

درجه شفافیت

[41] sharing of resources
[42] competitive sharing of resources
[43] shared database
[44] Consistency
[45] locking mechanisms
[46] transaction
[47] dead resource
[48] slow resource

هرچند شفافیت توزیعی معمولاً برای تمامی سیستم های توزیعی مناسب تصور می شود، در برخی موارد مخفی سازی کامل تمامی جنبه های توزیع از کاربران چندان هم قابل قبول نیست. بعنوان نمونه، مثلاً تمایل دارید که روزنامه الکترونیکی مورد علاقه تان همیشه پیش از ساعت هفت صبح به وقت محلی در صندوق پستی تان قرار بگیرد ، در حالیکه ممکن است در آن ساعت در جای دیگری از دنیا و در محدوده زمانی دیگری زندگی کنید. بنابراین ممکن است با دیدن روزنامه "به اصطلاح صبحتان " خیلی تعجب کنید . همچنین نمی توان انتظار داشت که سیستم توزیعی ناحیه گسترده ای که فرآیندی را در سانفرانسیسکو به فرآیندی در آمستردام متصل می کند ، بتواند ظرف کمتر از ۳۵ میلی ثانیه پیامی را از یک فرآیند به فرآیند دیگر ارسال و قانون طبیعت را مخفی کند. در عمل، اینکار با استفاده از یک شبکه کامپیوتری صورت می گیرد و چندصد میلی ثانیه طول می کشد. انتقال سیگنال علاوه بر وابستگی به سرعت نور، با ظرفیت های پردازشی سوییچ های میانی هم مرتبط است.

همچنین نوعی توازن[۴۹] بین درجه بالای شفافیت و کارایی سیستم وجود دارد. بعنوان مثال، بسیاری از برنامه های کاربردی اینترنت ممکن است بارها برای اتصال به یک خدمتگزار تلاش کرده و درصورت عدم موفقیت از اینکار صرفنظر کنند. در نتیجه، تلاش برای مخفی سازی خرابی موقت خدمتگزار پیش از آزمودن خدمتگزار دیگر ممکن است در کل باعث کاهش سرعت سیستم شود. در این موارد، بهتر است کاربر زودتر منصرف شود و یا حداقل از تلاشهای مکرر برای ایجاد اتصال صرفنظر کند .

در نمونه دیگر مثلاً باید انسجام همیشگی کپی های مختلف که در قاره های متفاوت واقع شده اند را تضمین کنیم . به بیان دیگر، درصورت تغییر یکی از کپی ها ، این تغییر بایستی پیش از هر عمل دیگری به تمام کپی ها انتشار پیدا کند. واضح است که تکمیل فقط یک عمل به روزآوری ممکن است چندین ثانیه طول بکشد، و این چیزی است که نمی توان از کاربران مخفی کرد.

و بالاخره اینکه در برخی موارد ، نمی توان بطور قطع ادعا کرد که مخفی سازی شفافیت ایده درستی است . از آنجایی که سیستم های توزیعی در حال گسترش به دستگاه های قابل حملی هستند که اهمیت آگاهی از مکان و بافت[۵۰] در آنها بیش از پیش در حال افزایش است، شاید بهتر باشد که بجای مخفی سازی شفافیت حتی آنرا عملاً هم *در معرض دید قرار دهیم*. اهمیت این بی حفاظی در برابر توزیع در سیستم های توزیعی همه جا حاضر[۵۱] و نهفته[۵۲] در ادامه همین فصل بیشترآشکار خواهد شد. بعنوان یک نمونه ساده ، در نظر بگیرید که کارمندی بخواهد فایلی را از نوت بوک خود چاپ کند. در اینصورت بهتر است که نسخه چاپی را به یک چاپگر نزدیک مشغول ارسال کنیم تا به چاپگر بیکاری در دفتر اصلی شرکت در کشور دیگر.

---

[49] trade-off
[50] Context
[51] ubiquitous
[52] embedded

استدلال های دیگری را هم می توان علیه شفافیت توزیعی مطرح کرد. با این فرض که دستیابی به شفافیت توزیعی کامل تقریباً غیرممکن است، باید از خود بپرسیم که آیا حتی *تظاهر* به برخورداری از این ویژگی عاقلانه هست یا خیر. شاید بسیار بهتر باشد توزیع را چنان آشکار کنیم که کاربر و سازنده برنامه کاربردی هیچگاه حتی به فکر شفافیت هم نیفتند . در نتیجه، کاربران بهتر رفتار (شاید غیرمنتظره) سیستم توزیعی را درک کرده و بنابراین آمادگی بسیار بیشتری برای برخورد با این رفتار پیدا خواهند کرد.

بنابراین، هرچند هدف قرار دادن شفافیت توزیعی در طراحی و پیاده سازی سیستم های توزیعی ممکن است پسندیده باشد، اما باید به همراه آن مسائل دیگری از قبیل کارآیی و جامعیت[۵۳] هم لحاظ شود. هزینه کسب شفافیت کامل ممکن است واقعاً کمرشکن باشد.

۳-۲-۱- بازبودن[۵۴]

یکی دیگر از اهداف مهم سیستم های توزیعی، بازبودن است . **سیستم توزیعی باز** [۵۵]سیستمی است که براساس قوانین استاندارد تشریح کننده معنا و نحو[۵۶] خدمات ، اقدام به ارائه آنها می کند. بعنوان مثال، در شبکه های کامپیوتری، قوانین استانداردی بر قالب، محتوا و معنای پیامهای ارسالی و دریافتی حاکم است. این دست قوانین در قالب پروتکل مدون می شوند. در سیستم های توزیعی، سرویس ها غالباً از طریق **رابط ها** [۵۷]تخصیص یافته و غالباً توسط **زبان تعریف رابط(IDL)** [۵۸]تشریح می شوند. تعاریف رابط نوشته شده در IDL ، تقریباً همیشه فقط درمورد نحو خدمات صحبت می کنند . به بیان دیگر، صرفاً اسامی توابعی را تعیین می کند که به همراه انواع پارامترها، مقادیر بازگشت[۵۹]، استثنائات احتمالی و غیره موجود هستند. بخش مشکل کار تخصیص وظیفه دقیق این سرویس ها، یعنی معنای رابط ها است. در عمل، این دست ویژگی ها همیشه به صورتی غیر متعارف، یعنی بوسیله زبان طبیعی ارائه می شوند.

در صورتی که رابط به نحو مناسبی تعریف شود، اجازه می دهد یک فرآیند دلخواه که این رابط را دارد با فرآیند دیگری که این رابط را ارائه می دهد صحبت نماید. همچنین باعث می شود تا دو طرف مستقل، پیاده سازی های کاملاً متفاوتی را از آن رابط ایجاد کرده و در نتیجه، دو سیستم توزیعی جداگانه، اما با عملکرد یکسان، بوجود می آید . اما این ویژگی ها باید کامل[۶۰] و خنثی[۶۱] باشند. کامل بودن یعنی، تمام آنچه برای پیاده سازی لازم است واقعاً تعریف شده باشد. با این وجود، بسیاری از تعاریف رابط به هیچ

[53] comprehensiveness
[54] Openness
[55] open distributed system
[56] semantics and syntax
[57] interfaces
[58] interface definition language
[59] return values
[60] Complete
[61] Neutral

وجه کامل نبوده و سازنده آنها باید جزئیات مربوط به پیاده سازی را به آنها اضافه کند. به همین اندازه خنثی بودن اهمیت دارد به این معنی که نحوه و شکل پیاده سازی ویژگی ها تجویز نشود. کامل بودن و خنثی بودن باعث کسب قابلیت کار با هم[۶۲] و قابلیت جابجایی[۶۳] می شود (Blair و Stefani ۱۹۹۸).
قابلیت کار با هم بیان می کند که اشکال مختلف پیاده سازی سیستم ها یا مؤلفه های سازندگان مختلف تا چه حد می توانند صرفاً با تکیه بر سرویس های یکدیگر که بوسیله یک استاندارد مشترک تعریف شده است، همزیستی و با هم کار کنند. **قابلیت جابجایی** بیان می دارد که برنامه کاربردی ایجاد شده برای سیستم توزیعی مفروض A تا چه حد، بدون نیاز به تغییر و اصلاح، روی سیستم توزیعی دیگر B که همان رابط های A را پیاده سازی کرده است، قابل اجراست.
یکی دیگر از اهداف مهم سیستم های توزیعی باز آن است که پیکره بندی سیستم توسط مؤلفه های مختلف (شاید از سازندگان مختلف) براحتی امکان پذیر باشد. همچنین، افزودن مؤلفه های جدید یا تعویض مؤلفه های فعلی نباید هیچ تأثیری بر دیگر مؤلفه ها داشته باشد. به بیان دیگر، یک سیستم توزیعی باز باید **قابل گسترش** [۶۴]هم باشد. بعنوان مثال، در یک سیستم قابل گسترش ، افزودن بخشهایی که روی یک سیستم عامل متفاوت اجرا می شوند یا حتی تعویض کل سیستم فایل باید تقریباً به راحتی انجام شود. اما باید اعتراف کرد که کسب انعطاف پذیری در حرف بسیار آسانتر از عمل است.

تفکیک سیاست از مکانیزم

برای کسب انعطاف پذیری[۶۵] در سیستم های توزیعی باز ، سیستم بایستی لزوماً بصورت مجموعه ای از مؤلفه های براحتی قابل انطباق یا قابل تعویض و نسبتاً کوچک سازماندهی شود. به این معنا که، بایستی تعاریفی را هم برای رابط های بالاترین سطح – یعنی رابط هایی که بوسیله کاربران و برنامه های کاربردی قابل رؤیت هستند- و هم برای رابط های بخشهای داخلی سیستم ایجاد و نحوه ارتباط این بخشها را توصیف کرد. این روش تقریباً جدید است. بسیاری از سیستم های قدیمی تر و حتی تقریباً جدید ، براساس شیوه یکپارچه ای سازماندهی شده اند که در آن ،مؤلفه ها فقط به صورت منطقی تفکیک شده ولی به صورت یک برنامه واحد و عظیم پیاده سازی شده اند . این روش، تعویض یا انطباق مؤلفه ها بدون تأثیرگذاری بر کل سیستم را مشکل می نماید. بنابراین ، سیستم های یکپارچه[۶۶] ، به جای باز بودن، تمایل به بسته بودن دارند .

[62] Interoperability
[63] Portability
[64] extensible
[65] flexibility
[66] monolithic approach

لزوم تغییر سیستم های توزیعی غالباً ناشی از مؤلفه ای است که سیاست بهینه ای را برای یک کاربر یا برنامه کاربردی ایجاد نمی کند. بعنوان مثال، ذخیره نهان[۶۷] در شبکه جهانی وب را در نظر بگیرید. جستجوگرها معمولاً به کاربرها امکان می دهند تا با تعیین ابعاد حافظه نهان و لزوم یا عدم لزوم چک کردن دائمی سند ذخیره نهان شده از نظر انسجام ، و یا چک کردن انسجام فقط برای یک مرتبه در طول جلسه ، سیاست ذخیره نهان[۶۸] خود را منطبق کنند. با این وجود ، کاربر نمی تواند تغییری در دیگر پارامترهای ذخیره نهان، از قبیل مدت زمان ماندگاری سند در حافظه نهان یا تعیین سندی که باید با پرشدن حافظه نهان حذف شود ، ایجاد کند . همچنین، نمی توان براساس *محتوای* سند راجع به مخفی سازی آن تصمیم گیری کرد. بعنوان مثال، یک کاربر ممکن است با اطلاع از اینکه جداول زمانی راه آهن به ندرت تغییر می کنند ، این اطلاعات ، و نه اطلاعات مربوط به شرایط فعلی ترافیک در بزرگراهها، را در حافظه نهان قرار دهد .

در اینجا لازم است که بین مکانیزم و سیاست تفاوت قائل شویم. بعنوان مثال، در مورد ذخیره نهان شبکه جهانی وب، جستجوگر بایستی امکاناتی را صرفاً جهت ذخیره سازی اسناد فراهم کرده و در عین حال ، به کاربران اجازه دهد تا راجع به اسناد قابل ذخیره شده و مدت زمان ذخیره تصمیم گیری کند. در عمل، اینکار از طریق ارائه مجموعه کاملی از پارامترهای قابل تنظیم برای کاربر (به صورتی پویا) پیاده می شود. حتی بهتر است که کاربر بتواند سیاست خود را به شکل یک مؤلفه قابل اتصال به جستجوگر پیاده کند. البته، مؤلفه مذکور بایستی یک رابط قابل فهم برای جستجوگر را داشته باشد تا جستجوگر بتواند روال های این رابط را فراخوانی کند.

۴-۲-۱- مقیاس پذیری[۶۹]

امروزه اتصال جهانی از طریق اینترنت، مانند امکان ارسال یک کارت پستال برای هرکسی در هرگوشه ای از جهان، تبدیل به امری عادی شده است . به همین دلیل ، مقیاس پذیری یکی از مهمترین اهداف طراحی برای سازندگان سیستم های توزیعی محسوب می شود.

مقیاس پذیری یک سیستم را می توان حداقل در سه بعد مختلف اندازه گیری نمود (Neuman ۱۹۹۴). اولاً ، یک سیستم می تواند با توجه به اندازه خود مقیاس پذیر باشد، به این معنا که بتوان به راحتی کاربران و منابع دیگری را به سیستم اضافه نمود. ثانیاً ، یک سیستم مقیاس پذیر جغرافیایی سیستمی است که ممکن است کاربران و منابع آن در فاصله های دوری از هم قرار گرفته باشند. ثالثاً ، یک سیستم ممکن است از نظر مدیریت اجرایی مقیاس پذیر باشد، به این معنا که حتی اگر سازمان هایی با مدیریت اجرایی مستقل را هم بهم پیوند دهد، باز به راحتی قابل مدیریت باشد. متأسفانه، اغلب

[67] caching
[68] caching policy
[69] scalability

سیستم هایی که از یک یا چند بعد مقیاس پذیر هستند ، با افزایش مقیاس پذیری سیستم، تا حدودی با افت عملکرد مواجه می شوند .

مشکلات مقیاس پذیری

پیش از ایجاد امکان مقیاس پذیری در یک سیستم باید انواع مختلفی از مشکلات را حل و فصل نمود. ابتدا بیایید نگاهی به مسأله مقیاس پذیری بر اساس اندازه داشته باشیم. پشتیبانی از تعداد بیشتری از کاربران یا منابع غالباً با محدودیتهای سرویس ها، داده ها و الگوریتمهای متمرکز مواجه می شویم (مراجعه کنید به شکل ۳-۱). بعنوان مثال، بسیاری از سرویس ها از این نظر متمرکز هستند که بوسیله فقط یک خدمتگزار روی یکی از ماشین های سیستم توزیعی پیاده می شوند. مشکل این دست طرحها مشخص است: با افزایش تعداد کاربران و برنامه های کاربردی، ممکن است خدمتگزار تبدیل به یک گلوگاه شود. حتی در صورت نامحدود بودن ظرفیت ذخیره سازی و پردازش، برقراری ارتباط با این دست خدمتگزارها مانعی در برابر رشد خواهد بود.

متأسفانه، گاهی اوقات مجبور هستیم که فقط از یک خدمتگزار استفاده کنیم . تصور کنید که برای مدیریت اطلاعات بسیار محرمانه ای از قبیل گزارشات پزشکی، حسابهای بانکی و غیره یک سرویس در اختیار داشته باشیم. در این موارد، یکی از بهترین گزینه ها پیاده سازی آن سرویس با استفاده از یک سرویس جداگانه و در یک اتاق جداگانه بسیار حفاظت شده و ایمن از بخشهای دیگر سیستم توزیعی از طریق مؤلفه های شبکه ای خاص باشد. کپی کردن خدمتگزار در محلهای مختلف از آنجایی که باعث کاهش سطح امنیتی سرویس می شود امکان پذیر نمی باشد.

مفهوم	مثال
سرویس های متمرکز	فقط یک سیستم برای تمام کاربران
داده های متمرکز	فقط یک دفترچه تلفن واحد آن لاین
الگوریتم های متمرکز	مسیریابی براساس اطلاعات کامل شبکه

شکل ۳-۱- نمونه هایی از محدودیت های مقیاس پذیری

داده های متمرکز درست به اندازه سرویس های متمرکز نامناسب هستند. چگونه باید شماره تلفن و آدرس پنجاه میلیون نفر را پی گیری کرد؟ فرض کنید هر رکورد داده ای ۵۰ کاراکتر باشد. بنابراین ، یک بخش ۲/۵ گیگابایتی دیسک فضای ذخیره کافی برای اینکار ایجاد می کند. اما در اینجا هم داشتن فقط یک پایگاه داده ای باعث اشغال شدن تمام خطوط ارتباطی داخلی و خارجی خواهد شد. همچنین تصور کنید اگر سیستم نام دامنه[۷۰] (DNS) بعنوان یک جدول منفرد پیاده شود، چه عملکردی خواهد داشت ؟ DNS اطلاعات مربوط به میلیون ها کامپیوتر در سرتاسر جهان را نگه داشته و یکی از

[70] Domain Name System

سرویس اصلی برای مکان یابی خدمتگزارهای وب محسوب می شود. اما اگر قرار بود که هریک از درخواست های تجزیه[۷۱] URL را به یک و تنها خدمتگزار DNS ارسال کرد ، هیچکس تمایلی به استفاده از وب نمی داشت .

نهایتاً اینکه الگوریتمهای متمرکز هم چندان مناسب نیستند. در یک سیستم توزیعی بزرگ، تعداد زیادی پیام باید روی تعداد زیادی خط مسیردهی شوند. از نقطه نظر تئوری، روش بهینه برای انجام اینکار، جمع آوری اطلاعات کامل درباره بار تمامی ماشین ها و خطوط و سپس اجرای الگوریتمی برای محاسبه مسیرهای بهینه[۷۲] است. سپس می توان جهت ارتقاء مسیردهی، این اطلاعات را در اطراف سیستم پراکنده کرد.

مشکل در آنجاست که جمع آوری و انتقال تمامی اطلاعات ورودی و خروجی هم ایده مناسبی نیست، چون این پیامها بخشی از اضافه بار شبکه را ایجاد می کنند. در واقع، از هر الگوریتمی که براساس جمع آوری اطلاعات از تمامی سایتها عمل می کند و اطلاعات را جهت پردازش به یک ماشین مشخص ارسال کرده و سپس نتایجی را توزیع می کند اجتناب شود . برای این منظور فقط باید از الگوریتم های غیرمتمرکز استفاده شود . این الگوریتم ها معمولاً دارای ویژگی های زیر بوده که وجه تمایز آنها از الگوریتم های متمرکز محسوب می شود:

۱. هیچیک از ماشین ها اطلاعات کاملی راجع به وضعیت سیستم در اختیار ندارد.
۲. ماشین ها فقط براساس اطلاعات محلی تصمیم گیری می کنند.
۳. خرابی یک ماشین باعث خرابی الگوریتم نمی شود.
۴. هیچ فرض ضمنی راجع به وجود نوعی ساعت جهانی[۷۳] وجود ندارد.

از آنچه تا به اینجا گفته شد سه حالت اول به راحتی قابل درک و موجه است . مورد آخر هم گرچه خیلی واضح نیست اما مهم است. از آنجایی که همگام سازی دقیق تمام ساعتها غیرممکن است، بنابراین هر الگوریتمی که با عبارتی مشابه ″ دقیقاً در ۱۲:۰۰:۰۰ تمام ماشین ها باید اندازه صف خروجی خود را ثبت کنند″ آغاز شود، با شکست مواجه خواهد شد. در الگوریتم ها بایستی عدم همگامی دقیق ساعتها مدنظر قرار گیرد. هرقدر که سیستم بزرگ تر باشد، عدم قطعیت هم بیشتر افزایش خواهد یافت. در یک LAN منفرد، شاید بتوان با صرف تلاش کافی همه ساعت ها را تا چند میلی ثانیه همگام کرد ، اما انجام اینکار در سطح کشوری یا بین المللی آرزویی دست نیافتنی است.

مقیاس پذیری جغرافیایی[۷۴] هم مشکلات مخصوص به خود را دارد. یکی از مهمترین دلایل بروز اشکال در مقیاس پذیری سیستم های توزیعی موجود که برای شبکه های محلی طراحی شده اند این است که

[71] resolve
[72] optimal routes
[73] global clock
[74] geographical scalability

براساس **ارتباط همگام** [۷۵] استوار هستند. در این شکل از ارتباط، طرف درخواست کننده سرویس که معمولاً به نام **مشتری**[۷۶] خوانده می شود، تا زمان بازگشت پاسخ مسدود باقی می ماند. این روش معمولاً برای LANهایی مناسب است که در آنها ارتباطات بین دو ماشین بطور معمول حداکثر چند صد میلی ثانیه طول می کشد. اما در سیستم های حوزه گسترده ، ارتباطات میان فرآیندی ممکن است صدها میلی ثانیه یعنی هزار برابر بیشتر طول بکشد. ساخت برنامه های کاربردی تعاملی[۷۷] با استفاده از ارتباط همگام در سیستم های حوزه گسترده مستلزم صرف دقت زیادی است.

مشکل دیگر در کسب مقیاس پذیری جغرافیایی این است که ارتباطات در شبکه های حوزه گسترده ذاتاً نامطمئن و تقریباً همیشه نقطه- به -نقطه[۷۸] است . بالعکس، شبکه های منطقه محلی معمولاً امکانات ارتباطی بسیار مطمئن براساس پخش[۷۹] ایجاد می کنند و همین امر باعث سادگی زیاد در ایجاد سیستم های توزیعی می شود. بعنوان مثال، مسئله یافتن یک سرویس را در نظر بگیرید . در سیستم های محلی، فرآیند می تواند با استفاده از پخش پیام به تمامی ماشین ها از آنها درباره سرویس مورد نیاز خود پرسش نماید. فقط آندسته از ماشین هایی که سرویس را در اختیار داشته باشند پاسخ داده و هریک آدرس شبکه ای خود را در پیام پاسخ قرار می دهد. استفاده از چنین طرح های مکان یابی در سیستم های حوزه گسترده ای از قبیل اینترنت غیرممکن است . در عوض، سرویس های مکان یابی خاص باید طراحی شوند که مقیاس پذیر در سطح جهانی بوده و قادر به سرویس دهی به یک میلیارد کاربر باشند. در فصل ۵ مجدداً راجع به این سرویس ها بحث خواهیم کرد.

مقیاس پذیری جغرافیایی ارتباط زیادی با مشکلات راه حل های مرکزی دارد که مانع از مقیاس پذیری اندازه[۸۰] می شود. اگر یک سیستم با مؤلفه های مرکزی متعدد در اختیار داشته باشیم، مقیاس پذیری جغرافیایی به دلیل مشکلات کارآیی و قابلیت اطمینان ناشی از ارتباط حوزه گسترده دچار محدودیت خواهد بود. بعلاوه، مؤلفه های مرکزی باعث اتلاف منابع شبکه می شوند. تصور کنید که برای تمام کشور فقط از یک خدمتگزار پست الکترونیکی استفاده شود. در اینصورت ، پست الکترونیکی که برای همسایه تان ارسال می کنید باید ابتدا به خدمتگزار پستی مرکزی ارسال شود که ممکن است صدها کیلومتر از شما فاصله داشته باشد. مشخص است که این روش به هیچ وجه مناسب و مفید نیست.

نهایتاً، یکی از سؤالات مشکل و در برخی موارد باز درباره نحوه مقیاس پذیری یک سیستم توزیعی در بین چندین دامنه مدیریتی مستقل است . یکی از مهمترین مشکلات در این رابطه ، تضاد سیاست ها در رابطه با به کارگیری (و پرداخت) و مدیریت و ایمنی منابع است.

---

[75] synchronous communication
[76] client
[77] interactive applications
[78] point-to-point
[79] Broadcast
[80] size scalability

بعنوان مثال، بسیاری از مؤلفه های یک سیستم توزیعی که در داخل یک دامنه مستقر هستند، غالباً مورد اعتماد کاربرانی است که در داخل همان دامنه عمل می کنند . در این موارد، مدیریت اجرایی سیستم ممکن است برنامه های کاربردی آزموده و تأیید شده ای در اختیار داشته و یا برای تضمین عدم دستکاری در این مؤلفه ها اقدامات خاصی اتخاذ کند. بطور خلاصه، کاربران به مدیران اجرایی سیستم خود اعتماد می کنند. با این وجود، این اعتماد طبیعتاً در سرتاسر مرزهای دامنه توسعه نمی یابد.

چنانچه یک سیستم توزیعی به دامنه دیگر گسترش یابد، بایستی دو نوع اقدام امنیتی صورت بپذیرد. در درجه اول، سیستم توزیعی باید از خود در برابر حملات ناخواسته از دامنه جدید محافظت کند. بعنوان مثال، کاربران دامنه جدید ممکن است به سیستم فایل در دامنه اصلی فقط دسترسی خواندن داشته باشند. به همین ترتیب، امکاناتی از قبیل شکل نگار های گران قیمت یا کامپیوترهای پربازده ممکن است برای کاربران خارج از دامنه قابل دستیابی نباشد. ثانیاً، دامنه جدید بایستی از خود در برابر حملات ناخواسته از جانب سیستم توزیعی محافظت کند. بعنوان یک نمونه می توان به دانلود کردن برنامه هایی از قبیل ریزبرنامه ها[۸۱] در جستجوگرهای وب اشاره کرد . اساساً، دامنه جدید نمی داند که باید چه نوع رفتاری را از این کدهای خارجی انتظار داشته باشد و بنابراین ممکن است تصمیم به محدود کردن شدید حقوق دسترسی[۸۲] این کدها بگیرد . همانطور که در فصل ۹ خواهیم دید، نحوه اعمال این محدودیتها خود یکی از مشکلات است .

تکنیکهای مقیاس پذیری

بحث در مورد برخی از مشکلات مقیاس پذیری ما را به مسأله نحوه حل و فصل این مشکلات سوق می دهد. در اغلب موارد، مشکلات مقیاس پذیری در سیستم های توزیعی به صورت مشکلات کارآیی ناشی از محدودیت ظرفیت خدمتگزارها و شبکه ها جلوه می کند . در حال حاضر، اساساً فقط سه تکنیک در زمینه مقیاس پذیری وجود دارد: مخفی سازی تأخیرهای ارتباطات[۸۳]، توزیع و کپی برداری[مراجعه شود به (Neuman ۱۹۹۴).

مخفی سازی تأخیرهای ارتباطی تأثیر مهمی در کسب مقیاس پذیری جغرافیایی دارد. ایده اصلی، اجتناب حداکثری از انتظار برای دریافت پاسخ به درخواستهای سرویس از راه دور(و احتمالاً در فاصله دور) است. بعنوان مثال، وقتی سرویس واقع در ماشین از راه دوری درخواست می شود، می توان بجای انتظار برای دریافت پاسخ از جانب خدمتگزار، کار مفید دیگری در طرف درخواست کننده انجام داد .اساساً، این به معنای ساخت برنامه کاربردی درخواست کننده به صورتی است که فقط از **ارتباط ناهمگام** [۸۴] استفاده کند. با ورود پاسخ، برنامه کاربردی متوقف شده و برای تکمیل درخواست قبلاً صادر

[81] applets
[82] access rights
[83] communication latencies
[84] asynchronous communication

شده، عمل کننده[۸۵] خاصی فراخوانی می شود. از ارتباطات ناهمگام غالباً می توان در سیستم های پردازش دسته ای[۸۶] و برنامه های کاربردی موازی[۸۷] استفاده کرد که در آنها ، وظیفه های کمابیش مستقل را می توان در حالی برای اجرا زمانبندی کرد که کار دیگری منتظر تکمیل ارتباط است . روش دیگر ، آغاز یک نخ کنترلی[۸۸] جدید جهت انجام درخواست است . هرچند این نخ انتظار برای پاسخ را مسدود می کند، نخ های دیگر فرآیند قادر به ادامه کار هستند.

با این وجود، بسیاری از برنامه های کاربردی نمی توانند از ارتباطات ناهمگام بصورتی مفید و کارآمد استفاده کنند. بعنوان مثال، در برنامه های کاربردی تعاملی[۸۹] وقتی کاربر درخواستی را ارسال می کند، معمولاً مجبور است که منتظر پاسخ باقی بماند. در این موارد، راه حل بهتر کاهش کلی ارتباط ،مثلاً ، از طریق انتقال بخشی از محاسبه که معمولاً در خدمتگزار انجام می شود به فرآیند مشتری درخواست کننده سرویس است. یکی از نمونه های اثربخشی این روش ، دسترسی به پایگاه های داده با استفاده از فرم ها است. پرکردن فرم از طریق ارسال پیامی جداگانه برای هریک از فیلدها و انتظار برای دریافت تصدیق از جانب خدمتگزار، مطابق شکل ۴-۱ (الف) ، انجام می شود. بعنوان مثال، خدمتگزار ممکن است پیش از پذیرش ورودی خطاهای نحوی آنرا چک کند. بر اساس شکل ۴-۱ (ب) ، راه حل بسیار بهتر انتقال کد مربوط به پرکردن فرم ، و احتمالاً چک کردن ورودی ها، برای مشتری و بازگرداندن فرم پرشده توسط مشتری است. این روش انتقال کد امروزه بطور گسترده توسط وب به صورت ریزبرنامه های جاوا[۹۰] و جاوا اسکریپت[۹۱] پشتیبانی می شود.

شکل ۴-۱- تفاوت بین بررسی فرمها توسط الف) خدمتگزار و ب) مشتری در حین تکمیل آنها(داخل شکل: الف) مشتری، نام کوچک، نام خانوادگی، پست الکترونیک، خدمتگزار، چک کردن فرم، پردازش فرم، ب) مشتری، نام کوچک، نام خانوادگی، پست الکترونیک، چک کردن فرم ، خدمتگزار، پردازش فرم).

یکی دیگر از تکنیکهای مهم مقیاس پذیری، **توزیع**[۹۲] است. توزیع به معنای انتخاب یک مؤلفه، تجزیه آن به قطعات کوچکتر و سپس پخش این قطعات در بین سیستم هاست. یکی از بهترین نمونه های توزیع ، سیستم نام دامنه (DNS) است. مطابق شکل۵-۱، فضای نام DNS بصورت سلسله مراتبی به صورت سه **دامنه**[۹۳] سازمان داده شده و هر دامنه به **مناطق**[۹۴] نا هم پوشان[۹۵] تقسیم می شود.اسامی در

[85] Handler
[86] batch-processing systems
[87] parallel application
[88] thread of control
[89] interactive applications
[90] Java applets
[91] Javascript
[92] distribution
[93] domain
[94] Zones
[95] nonoverlapping

هریک از مناطق بوسیله یک خدمتگزار نام واحد اداره می شود. بدون وارد شدن به جزئیات می توان تصور کرد که نام هر مسیر در واقع نام میزبانی در اینترنت خواهد بود و بنابراین با آدرس شبکه ای آن میزبان مرتبط است. اساساً، تجزیه نام به معنای بازگشت آدرس شبکه ای میزبان مربوطه است. بعنوان مثال، نام *nl.vu.cs.flits* را در نظر بگیرید . برای تجزیه، ابتدا این نام به خدمتگزار منطقه*Z1* داده شده (مراجعه شود به شکل ۵-۱) و آنهم آدرس خدمتگزار منطقه *Z2* را باز می گرداند که مابقی نام یعنی *vu.cs.flits* به آن داده می شود. خدمتگزار مربوط به*Z2* آدرس خدمتگزار مربوط به منطقه Z3 را بازمی گرداند که آنهم با کار برروی بخش پایانی نام ،آدرس میزبان مربوطه را بازمی گرداند.

شکل ۵-۱- نمونه ای از تقسیم فضای نام DNS به چندین ناحیه(داخل شکل: عمومی، کشورها).

این مثال نمونه بارزی از توزیع *سرویس نامگذاری* [۹۶]،که بوسیله DNS ایجاد می شود، در بین چندین ماشین و بنابراین جلوگیری از اشتغال فقط یک خدمتگزار با تمامی درخواستهای مربوط به تجزیه نام[۹۷] است.

بعنوان مثال دیگر، شبکه جهانی وب را در نظر بگیرید. برای اکثر کاربران، وب یک سیستم اطلاعاتی عظیم براساس سند است که هریک از اسناد آن دارای نام منحصر به فردی به شکل URL است. از نظر مفهومی، حتی ممکن است گاهی تصور شود که فقط یک خدمتگزار وجود دارد. در حالیکه وب از نظر فیزیکی در بین تعداد زیادی از خدمتگزارها توزیع شده که هریک روی تعدادی از اسناد وب کار می کند. نام خدمتگزاری که روی سند کار می کند در URL آن سند کد گذاری می شود. دقیقاً به خاطر همین ویژگی توزیع اسناد است که وب تا به حال قادر به مقیاس پذیری با ابعاد فعلی خود بوده است.

با این فرض که مشکلات مقیاس پذیری غالباً به شکل افت کارآیی ظاهر می شود، معمولاً بهتر است که مؤلفه ها را واقعاً در داخل سیستم توزیعی **کپی برداری** [۹۸]کنیم. این کپی برداری علاوه بر افزایش قابلیت دسترسی ، به تعادل بار در بین مؤلفه ها کمک کرده و در نتیجه باعث افزایش کارآیی هم خواهد شد. بعلاوه، در سیستم های دارای پراکندگی جغرافیایی زیاد ، در اختیار داشتن یک کپی دم دستی باعث مخفی سازی بخش عمده ای از مشکلات تأخیر در ارتباطات می شود که قبلاً مورد بحث قرار گرفت.

هرچند **به کار گیری حافظه نهان**[۹۹] شکل خاصی از کپی برداری است ، تفاوتهای جزئی بین آنها وجود دارد. به کار گیری حافظه نهان هم مانند کپی برداری، باعث ایجاد یک کپی از منبع معمولاً در نزدیکی مشتری می شود که به آن منبع دسترسی دارد. در حالیکه برخلاف کپی برداری تصمیمی است که

[96] naming service
[97] name resolution
[98] replicate
[99] caching

بوسیله مشتری منبع گرفته می شود و نه بوسیله مالک منبع. بعلاوه، حافظه نهان در صورت اعلام نیاز به کار گرفته می شود، در حالیکه کپی برداری طرحی از پیش برنامه ریزی شده است.

اما به کار گیری حافظه نهان و کپی برداری یک نقطه ضعف عمده دارند که ممکن است تأثیر شدیدی بر مقیاس پذیری داشته باشند . ازآنجاییکه بکمک این روش چندین کپی از یک منبع ایجاد می شود ، اصلاح یک کپی باعث تفاوت آن با سایر کپی ها خواهد شد. در نتیجه، به کار گیری حافظه نهان و کپی برداری مشکلاتی را از نظر **انسجام**[100] ایجاد می کنند.

میزان تحمل عدم انسجام[101] تا حد زیادی بستگی به کاربری یک منبع دارد. بعنوان مثال، بسیاری از کاربران وب مشکلی ندارند که جستجوگر آنها سند موجود در حافظه نهان را بازگرداند که اعتبار[102] آن در چند دقیقه اخیر بررسی نشده باشد. با این وجود، در بسیاری از موارد ، از قبیل مزایده ها یا بورس اوراق بهادار الکترونیک، بایستی انسجام قوی تضمین شود. مشکل ناشی از انسجام قوی آن است که هرنوع بروزآوری بایستی فوراً به تمامی کپی های دیگر پخش شود. بعلاوه، چنانچه دو به روزآوری بصورت همزمان انجام شود، غالباً لازم است که هریک از کپی ها به یک ترتیب به روزآوری شوند. مواردی از این دست معمولاً مستلزم وجود نوعی مکانیزم همگام سازی جهانی است. متأسفانه، پیاده سازی این مکانیزم ها بصورتی مقیاس پذیر بسیار مشکل یا حتی غیرممکن است ، چون فتونها و سیگنالهای الکتریکی بایستی از محدوده سرعت ۳۰۰ کیلومتر بر میلی ثانیه (یا همان سرعت نور) تبعیت کنند. در نتیجه، مقیاس پذیری از طریق کپی برداری ممکن است راه حل های ذاتاً غیرقابل مقیاس پذیری را معرفی کند. در فصل ۷ مجدداً به موضوع کپی برداری و انسجام بازخواهیم گشت.

با بررسی این تکنیک های مقیاس پذیری به راحتی می توان ادعا کرد که از نظر فنی، مقیاس پذیری اندازه[103] کمترین مشکل را دارد. در بسیاری از موارد، صرفاً با افزایش ظرفیت ماشین می توان (حداقل بطور موقت و احتمالاً با صرف هزینه های زیاد) مشکل را حل کرد. در مقایسه، مقیاس پذیری جغرافیایی مشکل بزرگتری است چون می خواهیم مادر طبیعت را رام خود کنیم. در عمل ترکیب تکنیک های توزیع، کپی برداری و به کار گیری حافظه نهان با انواع مختلف انسجام، عملکرد مناسبی ارائه می دهد. نهایتاً اینکه مقیاس پذیری مدیریت اجرایی[104] ، تا حدی به دلیل نیاز ما به حل مشکلات غیرتکنیکی(از قبیل سیاستهای سازمانی و همکاری های انسانی)، مشکل ترین نوع مقیاس پذیری ها به نظر می رسد . با این وجود، صرفاً با *نادیده گرفتن* دامنه های مدیریت اجرایی ، شاهد پیشرفتهایی در این زمینه بوده ایم . ایجاد و کاربرد گسترده فناوری نظیر به نظیر[105] در دنیای امروز بیانگر چیزی است که با قرارگیری کنترل در اختیار کاربران نهایی قابل دستیابی می باشد (Aberer and Hauswirth ،

[100] consistency
[101] inconsistency
[102] validity
[103] size scalability
[104] administrative scalability
[105] peer-to-peer technology

2005 ؛ Lau وگروه همکاران ۲۰۰۵ و Oram، ۲۰۰۱). با این وجود لازم به ذکر است که فناوری نظیر به نظیر در بهترین حالت می تواند فقط یک راه حل نسبی برای حل مشکلات مقیاس پذیری در مدیریت اجرایی محسوب شود و به همین دلیل هنوز جای بحث و بررسی دارد.

۵-۲-۱- طرح اشکال

اکنون دیگر باید بدانید که توسعه سیستم های توزیعی می تواند واقعاً کار مشکلی باشد. همانطور که در بسیاری از موارد در سرتاسر این کتاب خواهیم دید، مسائل زیادی را باید به طور هم زمان در نظر گرفت و نتیجه آن افزایش پیچیدگی است. با این وجود، با تبعیت از چند اصل کلی در طراحی ها می توان سیستم های توزیعی را به گونه ای ایجاد کردکه همخوانی زیادی با اهداف ترسیم شده در این فصل داشته باشد. بسیاری از اصول از قوانین اساسی و صحیح مهندسی نرم افزار تبعیت کرده و به همین دلیل از تکرار آنها در اینجا خودداری می کنیم.

با این وجود، وجه تمایز سیستم های توزیعی با نرم افزار متداول، پراکندگی مؤلفه های آنها در سرتاسر شبکه است. لحاظ نکردن این پراکندگی در طراحی ها باعث پیچیدگی بیهوده بسیاری از سیستم ها و نهایتاً منجر به اشتباهاتی می شود که باید بعدها به دنبال راه چاره ای برای رفع آنها باشیم . پیتر دوچ که زمانی در شرکت سان میکروسیستمز مشغول فعالیت بود، این اشتباهات را در قالب تصورات اشتباهی مدون و ارائه کرد که مبتدی ها را در اولین تلاش ها برای ایجاد برنامه های کاربردی توزیعی گرفتار خود می کند:

۱. شبکه قابل اطمینان[۱۰۶] است.
۲. شبکه ایمن[۱۰۷] است.
۳. شبکه همگن[۱۰۸] است.
۴. توپولوژی شبکه[۱۰۹] ثابت است.
۵. تأخیر[۱۱۰] صفر است.
۶. پهنای باند[۱۱۱] نامحدود است.
۷. هزینه انتقال[۱۱۲] صفراست.
۸. فقط یک مدیر اجرایی[۱۱۳] وجود دارد.

[106] reliable
[107] secure
[108] homogeneous
[109] network topology
[110] latency
[111] bandwidth
[112] transport cost
[113] administrator

به رابطه بین این فرضیات با ویژگی های مختص سیستم های توزیعی یعنی قابلیت وثوق، ایمنی، همگن بودن، توپولوژی شبکه، تأخیر، پهنای باند، هزینه های انتقال و نهایتاً دامنه های مدیریتی توجه کنید . در هنگام ایجاد برنامه های کاربردی غیرتوزیعی بسیاری از این مشکلات اصلاً مطرح نمی شوند .
اغلب اصول مورد بحث در این کتاب ارتباط نزدیکی با مفروضات بالا دارد. در تمام موارد، ما برای مشکلاتی راه حل ارائه می کنیم که در اثر نادرست بودن یک یا چند فرض بالا ایجاد شده اند . بعنوان مثال، چیزی به نام شبکه های قابل اطمینان به هیچ وجه وجود خارجی ندارد و بنابراین کسب شفافیت خرابی هم غیرممکن خواهد بود. یک فصل کامل کتاب را به این بحث اختصاص داده ایم که ارتباطات شبکه ای ذاتاً غیرایمن هستند. تاکنون استدلال آورده ایم که در سیستم های توزیعی مسأله ناهمگن بودن باید مدنظر قرار گیرد. در همین راستا ، در مبحث مربوط به کپی برداری و حل مشکلات مقیاس پذیری ، مفصلاً به مشکلات پهنای باند و تأخیر اشاره خواهیم کرد. همچنین در موارد مختلف در ضمن بحث درباره مسائل مدیریتی ، بطور ضمنی تصورات غلط در مورد صفر بودن هزینه انتقال و یکتا بودن دامنه های مدیرتی را مطرح می کنیم.

۳-۱- انواع سیستم های توزیعی

پیش از شروع بحث در مورد اصول سیستم های توزیعی، نگاه دقیقتری به انواع مختلف سیستم های توزیعی خواهیم داشت. در ادامه، تفاوت بین سیستم های محاسبات توزیعی[۱۱۴] ، سیستم های اطلاعات توزیعی[۱۱۵] و سیستم های نهفته توزیعی[۱۱۶] را تشریح خواهیم کرد.

۱-۳-۱- سیستم های محاسبات توزیعی

یکی از مهم ترین گروه های سیستم های توزیعی در محاسبات بسیار کارآ استفاده دارد. در این رابطه دو زیرگروه می توان شناسایی کرد. در **محاسبه خوشه** [۱۱۷]، سخت افزار زیربنایی متشکل از مجموعه ای از ایستگاههای کاری یا رایانه های شخصی(PCها) مشابه است که از طریق یک شبکه ناحیه محلی پرسرعت ارتباط نزدیکی با هم پیدا کرده اند . بعلاوه، تمامی گره ها سیستم عامل واحدی را اجرا می کند.
اما در مورد **محاسبه توری**[۱۱۸]، وضعیت تا حدودی متفاوت است. این زیرگروه متشکل از سیستم های توزیعی است که غالباً بشکل فدراسیونی از سیستم های کامپیوتری سازماندهی می شوند . هریک از

[114] distributed computing systems
[115] distributed information systems
[116] distributed embedded systems
[117] cluster computing
[118] grid computing

این سیستم ها ممکن است در دامنه مدیریت اجرایی متفاوتی قرار گرفته و حتی سخت افزار، نرم افزار و فن آوری شبکه بکاررفته در آنها بسیار با هم متفاوت باشد.

سیستم های محاسبه خوشه ای

سیستم های محاسبه خوشه ای محصول بهبود نسبت قیمت به کارآیی کامپیوترهای شخصی و ایستگاههای کاری محسوب می شود. در مقطعی در گذشته ، از هر دو بعد مالی و فنی امکان ساخت ابرکامپیوترها با استفاده از فن آوری در دسترس همگان[۱۱۹] و صرفاً با قرار دادن مجموعه ای از کامپیوترهای نسبتاً ساده در یک شبکه پرسرعت فراهم شد . تقریباً در تمام موارد، از محاسبه خوشه ای برای برنامه نویسی های موازی[۱۲۰] استفاده می شود که در آنها یک برنامه (با محاسبات فراوان) به صورت موازی روی چندین ماشین اجرا شود.

شکل ۶-۱- نمونه ای از سیستم های محاسبه خوشه ای(داخل شکل: گره اصلی[۱۲۱]،برنامه کاربردی مدیریت، کتابخانه های موازی، سیستم عامل محلی،شبکه دسترسی از راه دور، گره محاسبه، مؤلفه برنامه کاربردی موازی، سیستم عامل محلی،گره محاسبه، مؤلفه برنامه کاربردی موازی، سیستم عامل محلی.... گره محاسبه، مؤلفه برنامه کاربردی موازی، سیستم عامل محلی، شبکه استاندارد،شبکه پرسرعت).

بعنوان یکی از نمونه های معروف کامپوترهای خوشه ای می توان به خوشه های بیولف[۱۲۲] براساس Linux اشاره کرد که پیکر بندی کلی آن در شکل ۶-۱ ارائه شده است. هریک از خوشه ها متشکل از مجموعه ای از گره های محاسبه[۱۲۳] است که بوسیله یک گره اصلی قابل کنترل و دسترسی هستند. این گره اصلی تخصیص گره ها به یک برنامه موازی خاص را هدایت کرده ، دسته ای از کارهای محوله در اختیار دارد و رابطی برای کاربران سیستم ایجاد می کند. به این ترتیب، گره اصلی واقعاً میان افزار لازم برای اجرای برنامه ها و مدیریت خوشه را اجرا می کند، در حالیکه گره های محاسبه غالباً فقط به یک سیستم عامل استاندارد نیاز دارند .

یکی از مهمترین بخشهای این میان افزار از کتابخانه هایی برای اجرای برنامه های موازی تشکیل شده است. همانطور که در فصل ۴ هم خواهیم گفت، بسیاری از این کتابخانه ها عملاً فقط امکانات پیشرفته ارتباطی براساس پیام ایجاد کرده ولی قادر به اداره فرآیند های معیوب، مسائل امنیتی و غیره نخواهند بود.

---

[119] Off-the-self
[120] parallel programming
[121] master node
[122] Beowulf
[123] compute nodes

بعنوان نمونه ای از این سازمان سلسله مراتبی می توان به روش متقارن[۱۲۴] مورد استفاده در سیستم های MOSIX(Amar،۲۰۰۴ وگروه همکاران) اشاره کرد. MOSIX تلاش می کند تا **شکل تک سیستمی**[۱۲۵] از خوشه ارائه دهد؛ به این معنی که یک کامپیوتر خوشه ای با تظاهر به اینکه یک کامپیوتر منفرد است، شفافیت توزیعی بالایی را به فرآیند ارائه می دهد. همانطور که قبلاً هم ذکر کردیم، ایجاد چنین شکلی در همه شرایط غیرممکن است. در MOSIX، شفافیت توزیعی بالا با ایجاد امکان مهاجرت پویا[۱۲۶] و با قبضه[۱۲۷] فرآیندها در بین گره های تشکیل دهنده خوشه محقق می شود. مهاجرت فرآیند به کاربر امکان می دهد تا برنامه کاربردی را روی هریک از گره ها(که گره خانه[۱۲۸] نامیده می شود) آغاز کرده و پس از آن ، مثلاً جهت استفاده مفید و مؤثر از منابع، این گره را به صورتی شفاف به گره های دیگر انتقال دهد. در فصل ۳ مجدداً به موضوع مهاجرت فرآیند بازخواهیم گشت.

سیستم های محاسبه توری

همگن بودن ، یکی از ویژگی های بارز محاسبه خوشه است. در اغلب موارد، کامپیوترهای داخل خوشه تاحد زیادی یکسان هستند، دارای سیستم عامل واحدی بوده و همگی از طریق یک شبکه واحد متصل می شوند. در حالیکه، سیستم های محاسبه توری تا حد زیادی ناهمگن هستند، یعنی هیچ فرضی در مورد سخت افزار، سیستم های عامل، شبکه ها، دامنه های مدیریت اجرایی، سیاستهای امنیتی و غیره وجود ندارد.

یکی از مهمترین مسائل در سیستم های محاسبه توری آن است که منابعی از سازمانهای مختلف در کنار هم قرار داده می شوند تا امکان تشریک مساعی گروه هایی از افراد یا مؤسسات فراهم شود. این نوع تشریک مساعی در قالب **سازمان مجازی** [۱۲۹]محقق می شود. افراد متعلق به یک سازمان مجازی حق دسترسی به منابعی تأمین شده برای همان سازمان را دارند. این منابع نوعاً شامل خدمتگزاران محاسبه (از جمله ابرکامپیوترهایی که احتمالاً به صورت کامپیوترهای خوشه ای پیاده شده اند) ، امکانات ذخیره سازی و پایگاه های داده ای و در برخی موارد ، تعدادی از دستگاه های شبکه ای شده از قبیل تلسکوپ، حسگر و غیره می باشند .

به دلیل ماهیت این سیستم ها، قسمت اعظم نرم افزار مربوط به عملی کردن محاسبات توری، جهت ایجاد دسترسی به منابع مربوط به دامنه های مدیریت اجرایی مختلف و فقط برای کاربران و برنامه های کاربردی متعلق به همان سازمان مجازی تکامل یافته اند . به همین دلیل، عمدتاً روی مسائل معماری

[124] symmetric approach
[125] single-system image
[126] Dynamic
[127] Preemptive
[128] home node
[129] virtual organization

تمرکز می شود . نمونه ای از معماری های پیشنهادی (۲۰۰۱، Foster و گروه همکاران) در شکل ۷-۱ ارائه شده است.

شکل ۷-۱- معماری لایه ای سیستم های محاسبه توری (داخل شکل: برنامه های کاربردی، لایه جمعی،لایه اتصال، لایه منبع، لایه فابریک).

این معماری از چهار لایه تشکیل شده است. پایین ترین لایه ، یعنی *لایه فابریک*[۱۳۰]، رابط هایی را برای منابع محلی در محلی خاص بوجود می آورد. توجه داشته باشید که این رابط ها به گونه ای ساخته می شوند تا امکان اشتراک منابع در داخل یک سازمان مجازی فراهم شود . این رابط ها توابعی را برای پرس و جوی وضعیت و توانایی های منابع و همچنین توابعی برای مدیریت واقعی منابع (از قبیل قفل کردن منابع) ایجاد می کنند.

لایه اتصال [۱۳۱]متشکل از پروتکل های ارتباطی جهت پشتیبانی از تراکنش توری است که از چندین منبع استفاده می کنند . بعنوان مثال، برای انتقال داده ها در بین منابع یا صرفاً دسترسی به منابع از مکان دور نیازمند پروتکل هایی هستیم. بعلاوه، این لایه حاوی پروتکلهای امنیتی برای تأیید هویت کاربران و منابع است. توجه داشته باشید که در بسیاری از موارد هویت کاربران انسانی تأیید نمی شود؛ بلکه در عوض، برنامه هایی که به نمایندگی از جانب آنها عمل می کنند تأیید هویت می شوند . از این نقطه نظر، تفویض حقوق از کاربر به برنامه ها کارکرد مهمی است که باید در لایه اتصال پشتیبانی شود. در مبحث امنیت در شبکه های توزیعی مفصلاً راجع به موضوع تفویض صحبت خواهیم کرد.

لایه منبع [۱۳۲]مسئول مدیریت در شبکه های منفرد است . این لایه با استفاده از توابع فراهم شده بوسیله لایه اتصال، رابط های فراهم شده بوسیله لایه فابریک را مستقیماً فراخوانی می کند. بعنوان مثال، این لایه کارکردهای مربوط به کسب اطلاعات پیکربندی روی یک منبع خاص، یا، در کل، برای انجام عملیاتهای خاصی از قبیل ایجاد فرآیند یا خواندن داده را ارائه می دهد. بنابراین لایه منبع مسئول کنترل دسترسی بوده و از اینرو، روی تأیید هویت صورت گرفته بعنوان بخشی از لایه اتصال تکیه می کند.

لایه بعد در سلسله مراتب بالا، *لایه جمعی* [۱۳۳]است. این لایه مسئول هدایت دسترسی به منابع متعدد بوده و نوعاً متشکل از سرویس هایی جهت کشف منبع، تخصیص و زمانبندی امور روی منابع متعدد، کپی سازی داده و غیره است. برخلاف لایه اتصال و منبع که از مجموعه استاندارد و نسبتاً کوچکی از پروتکلها تشکیل می شود ، لایه اتصال ممکن است متشکل از پروتکلهای بسیار متفاوت برای اهداف بسیار متفاوت بوده و این خود بیانگر طیف وسیع سرویس هایی است که لایه مذکور می تواند به یک سازمان مجازی ارائه دهد.

---

130 fabric layer
131 connectivity layer
132 resource layer
133 collective layer

لایه برنامه کاربردی [۱۳۴]متشکل از برنامه های کاربردی است که در داخل یک سازمان مجازی عمل کرده و از محیط محاسبه توری[۱۳۵] استفاده می کنند.

لایه های جمعی، اتصال و منبع رویهم رفته هسته چیزی به نام لایه میان افزار توری[۱۳۶] را تشکیل می دهند. این لایه ها در مجموع امکان دسترسی و مدیریت منابعی را فراهم می کنند که بالقوه در سرتاسر سایتهای متعدد پراکنده شده اند. یکی از مهمترین مشاهدات از نقطه نظر میان افزاری این است که در محاسبه توری، همیشه ایده سایت(یا یک واحد مدیریت اجرایی) بچشم می خورد . این جهت گیری مشترک با تغییر موضع تدریجی به سمت **معماری سرویس محور** [۱۳۷] - که امکان دسترسی به لایه های مختلف را از طریق مجموعه ای از سرویس های وب فراهم می کند- تشدید شد (Joseph،۲۰۰۴ وگروه همکاران). تاکنون، این معماری منجر به تعریف معماری دیگری به نام **معماری سرویس های توری باز(OGSA)**[۱۳۸] شده است. در این شیوه معماری ، تنوع لایه ها و تعدد مؤلفه ها باعث ایجاد پیچیدگی می شود. به نظر می رسد که سرنوشت تمامی فرآیندهای استانداردسازی[۱۳۹] به پیچیدگی ختم می شود. جزئیات مربوط به OGSA را در کتاب (۲۰۰۵، Foster و گروه همکاران) مطالعه کنید.

۲-۳-۱-سیستم های اطلاعات توزیعی

یکی دیگر از گروههای مهم سیستم های توزیعی در سازمانهایی مشاهده می شود که گرچه با مجموعه ای از برنامه های کاربردی شبکه ای شده سروکار دارند ، اما قابلیت کار با هم برای آنها تجربه ای بسیار سخت است. بسیاری از راه حلهای میان افزاری موجود نتیجه کار با زیرساختی[۱۴۰] است که در آنها ادغام برنامه های کاربردی در یک سیستم اطلاعاتی شرکتی[۱۴۱] آسان تر بوده است.

ادغام[۱۴۲] در سطوح مختلفی قابل تشخیص است. در بسیاری از موارد، یک برنامه کاربردی شبکه ای شده شده فقط متشکل از خدمتگزار اجرا کننده آن برنامه کاربردی (در اغلب موارد مجهز به یک پایگاه داده ای) بود که در اختیار برنامه های از راه دوری بنام **مشتریان** [۱۴۳] قرار می گرفت. این مشتریان قادر به ارسال درخواست اجرای یک عملیات خاص برای خدمتگزار و متعاقب آن دریافت پاسخ بودند. ادغام در پایین ترین لایه به کاربران اجازه می داد تا تعدادی از درخواستها را ، که گاهی اوقات برای

---

134 applications
135 grid computing environment
136 grid middleware layer
137 service-oriented architecture
138 open grid services architecture
139 standardization process
140 infrastructure
141 enterprise-wide
142 integration
143 client

خدمتگزارهای مختلف بودند ، در قالب یک درخواست بزرگ بسته بندی کرده و تحت عنوان **تراکنش توزیعی** [۱۴۴] اجرا کند . ایده اصلی آن بود که یا تمام درخواستها اجرا شود و یا هیچیک ازآنها.
با پیشرفته تر شدن برنامه های کاربردی و تفکیک تدریجی آنها به مؤلفه های مستقل (خصوصاً تمایز بین مؤلفه های پایگاه داده ای از مؤلفه های پردازشی) ، مشخص شد که بهتر است ادغام از طریق ایجاد امکان ارتباط مستقیم برنامه های کاربردی با یکدیگر هم انجام شود. این ویژگی اکنون منجر به صنعت عظیمی به نام **ادغام برنامه کاربردی شرکتی (EAI)** [۱۴۵] شده است. در ادامه بیشتر راجع به این دو شکل از سیستم های توزیعی صحبت خواهیم کرد.

سیستم های پردازش تراکنش[۱۴۶]

برای روشن تر شدن بحث، روی برنامه های کاربردی پایگاه داده ای تمرکز می کنیم. در عمل، عملیات روی پایگاه داده ای معمولاً در قالب **تراکنش ها** انجام می شود. برنامه نویسی با استفاده از تراکنش ها مستلزم آشنایی با عمل های پایه ای[۱۴۷] خاصی است که یا باید بوسیله سیستم توزیعی زیربنایی[۱۴۸] تأمین شود و یا از طریق سیستم زمان اجرای زبان[۱۴۹]. برخی از نمونه های عمل های پایه ای تراکنش در جدول ۸-۱ ارائه شده است. لیست دقیق عمل های پایه ای بستگی به نوع اشیاء مورد استفاده در تراکنش دارد(Gray and Reuter ،۱۹۹۳). مثلاً در یک سیستم پستی، با عمل های پایه ای ارسال، دریافت و پیش بردن پست مواجه هستیم . اما در یک سیستم حسابداری، ممکن است عمل های پایه ای کاملاً متفاوتی به چشم بخورد . اما *READ*و *WRITE*از نمونه های معروف و متداول در اغلب سیستم ها هستند. جمله های عادی[۱۵۰]، فراخوانی های روال و امثال آن هم در تراکنش ها مجاز هستند. همانطور که در ادامه خواهیم گفت، فراخوانی های روال دور(RPCها) هم غالباً به شکل تراکنش بسته بندی شده و چیزی به نام **RPC تراکنشی** [۱۵۱]بوجود می آورد. در فصل ۴ مفصلاً راجع به RPCها صحبت خواهیم کرد.

عمل های پایه	شرح
BEGIN_TRANSACTION	آغاز تراکنش را علامتگذاری کن
END_TRANSACTION	تراکنش را خاتمه بده و تلاش کن متعهد شوی
ABORT_TRANSACTION	تراکنش را نابود کن و مقادیر قدیمی را بازگردان

144 distributed transaction
145 enterprise application integration
146 transaction processing systems
147 Primitives
148 underlying distributed system
149 language runtime system
150 ordinary statements
151 Transactional RPC

READ	داده ها را از یک فایل، جدول یا امثال آن بخوان
WRITE	داده ها را به یک فایل، جدول یا امثال آن بنویس

شکل ۸-۱- نمونه ای از عمل های پایه در تراکنش ها

از TRANSACTION_BEGIN و TRANSACTION_END جهت مشخص کردن محدوده تراکنش ها استفاده شده و عملیات بین آنها بدنه تراکنش را تشکیل می دهند.ویژگی بارز تراکنش آن است که یا تمام این عملیاتها اجرا می شوند و یا هیچیک از آنها. عملیات ممکن است بسته به نوع پیاده سازی، فراخوانی های سیستم ، روالهای کتابخانه ای، یا جملات داخل پرانتز یک زبان باشند.

این ویژگی "یا همه چیز یا هیچ چیز" تراکنش ها یکی از چهار ویژگی عمده تمام تراکنش ها است. به بیان دقیقتر، تراکنش ها دارای ویژگی های زیر هستند:

۱. اتمی (Atomic): از دید دنیای خارج، تراکنش بصورتی تفکیک ناپذیر انجام می شود.
۲. پایدار (Consistent): تراکنش ثابت های سیستم را مخدوش نمی کند.
۳. ایزوله (Isolated): تراکنش ها همزمان با یکدیگر تداخل ندارند.
۴. دائمی (Durable): تغییرات ناشی از تراکنش ها دائمی هستند.

گاهی اوقات این ویژگی ها با چسباندن حروف اول آنها ، در مجموع ACID (**اسید**)خوانده می شوند.

اولین ویژگی مهم تمامی تراکنش ها **اتمی بودن** است. براساس این ویژگی، تمامی تراکنش ها یا بطور کامل ، و البته طی فقط یک اقدام فوری و تفکیک ناپذیر، انجام می شود و یا اصلاً رخ نمی دهد. در حین انجام فرآیند، فرآیندهای دیگر (چه خود آنها در تراکنش ها شرکت داشته باشند چه نه) قادر به مشاهده هیچیک از وضعیتهای میانی[۱۵۲] تراکنش نیستند. .

دومین ویژگی **پایداری** تراکنش است. به این معنی که سیستم دارای ثابت هایی است که باید همواره برقرار باشند. بعنوان مثال، در سیستم های بانکداری، یکی از ثابت های مهم قانون حفاظت از پول است. پس از هر نقل و انتقال داخلی، مقدار وجه موجود در بانک باید معادل مقدار آن پیش از انتقال باشد. اما در صورت از بین رفتن ثبات در حین تراکنش ، حتی به صورت لحظه ای هم، این اشکال از دید دنیای بیرون کاملاً مخفی خواهد ماند.

ویژگی سوم به این معناست که تراکنش ها **ایزوله** یا **قابلیت سری شدن**[۱۵۳] دارند. به این معنا که چنانچه دو یا چند تراکنش به صورت همزمان در حال انجام باشند، نتیجه نهایی برای هریک از آنها و برای فرآیندهای دیگر، چنان است که گویی تمام تراکنش ها با ترتیب مشخص(وابسته به سیستم) انجام شده اند .

[152] intermediate states
[153] serialized

ویژگی چهارم **دائمی بودن** است. به این معنی است که تراکنش ها در صورت تعهد، بدون کوچکترین توجهی به رویدادها و اتفاقات جانبی، به پیش رفته و نتایج دائمی می شوند. پس از انجام تعهد، هیچ خرابی نمی تواند نتایج را ملغی کرده یا باعث مفقود شدن آنها شود.(در مورد ویژگی دائمی بودن در فصل ۸ صحبت خواهیم کرد).

تا به اینجا، تراکنش ها یک پایگاه داده ای منفرد تعریف شده است. مطابق شکل ۹-۱، **تراکنش جاسازی شده**[۱۵۴] از تعدادی زیرتراکنش تشکیل می شود. ممکن است در بالاترین سطح تراکنش با منشعب کردن فرزندانی[۱۵۵] که به صورت موازی باهم ، اما روی ماشین های مختلف اجرا می شوند، باعث افزایش عملکرد یا تسهیل برنامه نویسی شود. هریک از این فرزندان هم ممکن است یک یا چند زیر تراکنش اجرا کرده و یا فرزندان خود را منشعب کنند.

شکل ۹-۱- نمایی از یک تراکنش جاسازی شده(تراکنش جاسازی شده، زیرتراکنش، زیر تراکنش، پایگاه داده ای شرکت هواپیمایی، پایگاه داده ای هتل، دو پایگاه داده ای (مستقل) مختلف).

زیرتراکنش ها اشکالی کوچک اما مهم را مطرح می کند. فرض کنید که یک تراکنش چندین زیرتراکنش را به صورت موازی با هم راه اندازی و یکی از آنها را تعهد می کند و همین امر باعث قابل رؤیت شدن نتایج برای تراکنش والد [۱۵۶]می شود. با پیشروی محاسبات ، پدر فرزند خود را قطع[۱۵۷] کرده و کل سیستم را به وضعیت پیش از آغاز تراکنش بالاترین لایه بازمی گرداند. بنابراین، نتایج زیرتراکنش تعهد شده باید خنثی[۱۵۸] شود. بنابراین مفهوم ماندگاری که در بالا گفته شد، فقط در مورد تراکنش ها بالاترین لایه[۱۵۹] صدق می کند.

ازآنجاییکه تراکنش ها را می توان با عمق دلخواه جایگذاری کرد، برای انجام درست کارها نیازمند مدیریت اجرایی قابل ملاحظه ای هستیم. اما معانی مشخص هستند. با آغاز هر تراکنش ها یا زیرتراکنش، از نظر مفهومی کپی اختصاصی از تمامی داده های موجود در کل سیستم در اختیار آنها قرار داده می شود تا در صورت لزوم این داده ها را دستکاری کند. در صورت قطع شدن، دنیای خصوصی آن ناپدید می شود به طوری که گویا اصلاً وجود نداشته است. در صورت تعهد، این دنیای خصوصی جایگزین دنیای والد می شود. بنابراین ، چنانچه یک زیر تراکنش تعهد شده و سپس زیر تراکنش جدیدی آغاز شود، زیرتراکنش[۱۶۰] اخیر نتایج تولیدی بوسیله زیرتراکنش اول را مشاهده خواهد

[154] nested transaction
[155] children
[156] parent transaction
[157] Abort
[158] undo
[159] top-level transactions
[160] subtransaction

کرد. به همین ترتیب، در صورت قطع شدن تراکنش (سطح بالاتر) در بر گیرنده سایر تراکنش ها، تمامی زیرتراکنش ها آن هم قطع خواهند شد.

تراکنش ها جایگذاری شده ، از آنجایی که روش طبیعی برای توزیع تراکنش در بین ماشین های مختلف بوجود می آورند، در سیستم های توزیعی دارای اهمیت هستند. این تراکنش ها از تقسیم بندی *منطقی* [۱۶۱]کار تراکنش اصلی تبعیت می کنند. مثلاً، تراکنش مربوط به طرح ریزی سفری را که باید برای آن سه پرواز مختلف رزرو شوند، می توان منطقاً به سه زیرتراکنش تقسیم کرد. هریک از این زیرتراکنش ها بصورتی جداگانه و مستقل از دو تای دیگر قابل مدیریت است .

در اوائل ابداع سیستم های میان افزاری شرکتی[۱۶۲]، مؤلفه ای که کار تراکنش ها توزیعی(یا جایگذاری شده) را انجام می داد، در واقع هسته ادغام کننده برنامه های کاربردی در سطح خدمتگزار یا پایگاه داده ای را تشکیل می داد. این مؤلفه **ناظر پردازش تراکنش** [۱۶۳]یا بطور خلاصه **ناظر TP** خوانده می شد. وظیفه این مؤلفه، ایجاد امکان دسترسی برنامه های کاربردی به پایگاههای داده ای/ خدمتگزاران متعدد از طریق ارائه یک مدل برنامه نویسی تراکنشی به آن ، مشابه شکل ۱۰-۱ ، بود .

شکل ۱۰-۱- نقش ناظر هایTP در سیستم های توزیعی(داخل شکل: برنامه کاربردی مشتری، تراکنش، درخواستها، پاسخ، ناظرTP ، پاسخ، پاسخ، درخواست، پاسخ، خدمتگزار).

ادغام برنامه کاربردی شرکتی

همانطور که قبلاً گفته شد، با افزایش تعداد برنامه های کاربردی جداشده از پایگاههای داده ای زیربنایی خود ، لزوم وجود امکاناتی برای تلفیق برنامه های کاربردی بصورتی مستقل از پایگاههای داده ای آنها بیشتر آشکار شد. بخصوص اینکه، مؤلفه های برنامه کاربردی بایستی قادر به ارتباط مستقیم با یکدیگر باشند، و نه این که فقط از طریق رفتار درخواست/ پاسخ که در سیستم های پردازش تراکنش مورد پشتیبانی است، با یکدیگر ارتباط داشته باشند.

همین نیاز به وجود ارتباط در میان برنامه های کاربردی باعث ایجاد مدل های ارتباطی مختلف شد که مفصلاً در همین کتاب مورد بحث قرار خواهند گرفت . مطابق شکل ۱۱-۱، ایده اصلی آن بود که برنامه های کاربردی موجود بتوانند مستقیماً با یکدیگر تبادل داده کنند.

شکل ۱۱-۱- میان افزار به صورت تسهیل کننده ارتباط[۱۶۴] در ادغام برنامه کاربردی شرکتی (داخل شکل: برنامه کاربردی مشتری، میان افزار ارتباطی، برنامه کاربردی طرف خدمتگزار).

---

[161] logical division
[162] enterprise middleware systems
[163] transaction processing monitor
[164] communication facilator

انواع مختلفی از نرم افزارهای ارتباطی وجود دارند. در **فراخوانی های روال دور(RPC)** [۱۶۵]، مؤلفه های برنامه کاربردی می توانند با فراخوانی روال محلی، درخواستی را برای مؤلفه های دیگر برنامه کاربردی ارسال کند . در ادامه ، درخواست به صورت یک پیام بسته بندی و برای فراخوان شده[۱۶۶] ارسال می شود. نتیجه هم بازگردانده شده و بعنوان نتیجه فراخوانی روال به برنامه کاربردی عودت داده می شود.

همگام با افزایش مقبولیت فن آوری شیءگرا، تکنیکهایی هم برای فراخوانی های اشیاء از راه دور ایجاد شدند که امروزه تحت عنوان **فراخوانی های متد دور**[۱۶۷] **(RMI)** شناخته می شود. RMI اساساً همان RPC، اما با این تفاوت جزئی است که بجای برنامه های کاربردی، روی اشیاء عمل می کند.

نقطه ضعف RMI و RPC آن است که فراخواننده و فراخوان شده هردو باید در زمان ارتباط برقرار و در حال اجرا باشند. بعلاوه، باید دقیقاً از نحوه ارجاع به یکدیگر اطلاع داشته باشند. این وابستگی عمیق غالباً یک نقطه ضعف جدی تلقی شده و منجر به تولید چیزی به نام **میان افزار پیام گرا**[۱۶۸] یا بطور ساده **MOM** شده است. در این حالت، برنامه های کاربردی صرفاً پیامهایی را به نقاط تماس منطقی[۱۶۹] ارسال می کنند که غالباً بوسیله یک موضوع تعریف شده اند . به همین ترتیب، برنامه های کاربردی می توانند علاقه خود را به نوع خاصی از پیام اعلان کرده و پس از آن، میان افزار ارتباطی مراقب ارائه همان پیامها به آن برنامه های کاربردی خواهد شد. سیستم های موسوم به **انتشار/اشتراک**[۱۷۰] یکی از مهمترین و روبه گسترش ترین گروههای سیستم های توزیعی هستند. به همین دلیل بطور مفصل در فصل ۱۳ مورد بحث قرار گرفته اند.

۳-۳-۱- سیستم های فراگیر توزیعی

اغلب سیستم های توزیعی که تا به اینجا مورد بحث قرار گرفت ، از ویژگی مشترکی بنام ثبات[۱۷۱] برخوردار بودند ؛به این معنا که گره ها ثابت بوده و اتصال کمابیش باکیفیت و دائمی با شبکه برقرار می کنند. این ثبات تا حدودی از طریق روشهای مختلف مورد بحث در این کتاب -که هدف اغلب آنها کسب شفافیت توزیعی است- محقق می شود . بعنوان مثال، با آگاهی از وجود انبوهی از تکنیکهای مربوط به مخفی سازی و بازسازی خرابی ها می توان گفت این تصور وجود دارد که خرابی ها اموری موردی هستند و به ندرت روی می دهند. به همین ترتیب، چون قادر به مخفی سازی مسائل مربوط به محل واقعی گره ها در شبکه بوده ایم، و به طور مؤثر این توهم را برای کاربران وبرنامه های کاربردی ایجاد نماییم که گره ها درجای خود میخکوب هستند.

---

165 remote method calls
166 callee
167 remote method invocations
168 message-oriented middleware
169 logical contact points
170 publish/subscribe
171 stablity

با این وجود، با ابداع و عرضه دستگاه های محاسبه نهفته و متحرک امور تا حدی پیچیده تر شده است. اکنون دیگر با سیستم های توزیعی مواجه هستیم که بی ثباتی، رفتار پیش فرض[۱۷۲] آنها تلقی می شود. اغلب دستگاه های این **سیستم های فراگیر توزیعی**[۱۷۳] کوچک بوده، با باتری کار می کنند ، متحرک بوده و فقط یک اتصال بی سیم دارند، اگرچه این ویژگی ها در همه دستگاه ها مشاهده نمی شوند. بعلاوه، این ویژگی ها را نباید لزوماً محدود کننده تلقی کرد، همچنان که در امکان پذیری تلفن های هوشمند مدرن دیده می شود(۲۰۰۵، Roussos وگروه همکاران).

سیستم های فراگیر توزیعی، همانطور که از نام آنها هم برمی آید، بخشی از محیط اطراف ما می باشد(و از همینرو، در اغلب موارد ذاتاً توزیعی هستند). یکی از مهمترین ویژگی این سیستم ها ، فقدان عمومی کنترل مدیریت اجرایی انسانی در آنهاست . در بهترین حالت، این دستگاه ها بوسیله مالکانشان قابل پیکربندی هستند، ولی گذشته از این باید بصورتی اتوماتیک محیط خود را کشف کرده و به بهترین نحو ممکنه، "در آن جایگذاری شوند" . این جایگذاری بوسیله Grimm وگروه همکاران (۲۰۰۴) با تعیین سه شرط زیر برای برنامه های کاربردی فراگیر فرموله شده است:

۱. شمول تغییرات بافتی[۱۷۴] .
۲. تشویق ترکیب موردی[۱۷۵] .
۳. شناسایی اشتراک بعنوان پیش فرض.

شمول تغییرات بافتی به این معناست که یک دستگاه بایستی دائماً از این حقیقت آگاه باشد که محیط آن هر لحظه دستخوش تغییر است . یکی از ساده ترین تغییرات آن است که مثلاً چون کاربر در بین ایستگاههای پایه در حال جابجایی است، شبکه دیگر در دسترسی نباشد. در این حالت، برنامه کاربردی بایستی از طریقی، مثلاً با ایجاد اتصال اتوماتیک با یک شبکه دیگر یا به هر صورت مناسب دیگر از خود واکنش نشان دهد.

ویژگی تشویق ترکیب موردی اشاره به این واقعیت دارد که در سیستم های فراگیر، بسیاری از دستگاه ها به روشهای بسیار متفاوت توسط کاربران متفاوت مورد استفاده قرار می گیرند . در نتیجه، پیکربندی مجموعه برنامه های کاربردی که روی یک دستگاه اجرا می شوند، چه بوسیله کاربر و چه از طریق موقعیت های میانی خودکار[۱۷۶](اما تحت کنترل)، باید ساده باشد.

یکی از جنبه های بسیار مهم سیستم های فراگیر آن است که دستگاه ها غالباً با هدف دسترسی (احتمالاً ارائه) اطلاعات به سیستم ملحق می شوند. این امر لزوم ایجاد ابزاری جهت تسهیل خواندن،

---

[172] default behavior
[173] distributed pervasive systems
[174] contextual changes
[175] Ad hoc composition
[176] automated interpositions

ذخیره سازی، مدیریت و تشریک اطلاعات را مطرح می کند. به دلیل تناوب و تغییر اتصال دستگاه ها، فضایی که اطلاعات قابل دسترسی در آن وجود دارد، به احتمال قوی دائماً تغییر خواهد کرد.

Mascolo وگروه همکاران (۲۰۰۴) و همچنین Niemelaو Latvakoski (۲۰۰۴) به نتایج مشابهی دست یافتند: به دلیل شرایط متحرک بودن، دستگاه ها بایستی از انطباق آسان و وابسته به برنامه کاربردی به محیط محلی خود پشتیبانی کنند. آنها باید بصورتی مؤثر قادر به کشف سرویس ها و واکنش مناسب باشند. در مجموع، می توان نتیجه گرفت که شفافیت توزیعی در سیستم های فراگیر دیده نمی شود. در واقع، توزیع داده ها، فرآیندها و کنترل، *ذاتی* [۱۷۷] این سیستم ها بوده و به همین دلیل، شاید بهتر باشد که به جای تلاش برای مخفی سازی، این ویژگی را حتی در معرض دید قرار دهیم . حال بیایید نگاهی به برخی ازنمونه های واقعی سیستم های فراگیر داشته باشیم.

سیستم های خانگی

بعنوان یکی از انواع سیستم های فراگیر که کاربرد روزافزونی هم پیدا کرده و در عین حال کمترین محدودیت را دارد، می توان به سیستم های مرتبط با شبکه های خانگی اشاره کرد . این سیستم ها گرچه معمولاً متشکل از یک یا چند کامپیوتر شخصی هستند، اما برخی لوازم الکترونیکی مصرفی از قبیل دستگاه های تلویزیون، تجهیزات صوتی و شکلی، دستگاه های بازی ، تلفن ها (هوشمند)، PDAها[۱۷۸] و دیگر لوازم پوشیدنی شخصی را در یک سیستم واحد تلفیق می کنند. علاوه بر آن می توان انتظار داشت که در آینده انواع دستگاه ها از قبیل لوازم آشپزخانه، دوربین های نظارتی ، ساعت ها، کنترل کننده های روشنایی و غیره هم در یک سیستم توزیعی واحد ادغام شوند.

از دیدگاه سیستمی، پیش از محقق شدن سیستم های خانگی فراگیر چالش های زیادی باید در نظر گرفته شود. یکی از مهمترین مسائل این است که چنین سیستم هایی باید کاملاً خود-پیکربند[۱۷۹] و خود-مدیریت[۱۸۰] باشند. در صورتی که مؤلفه های این سیستم های خانگی توزیعی مستعد خطا باشند (که در بسیاری از دستگاه های جدید هم چنین است)، نمی توان انتظار داشت که کاربران نهایی علاقمند و قادر به نگهداری و استفاده از آنها باشند. استانداردهای **وصل کن و کار کن**[۱۸۱] **جهانی** (**UPnP**[۱۸۲]) تا حد زیادی این امر را محقق کرده اند؛ به طوری که دستگاه ها به صورت اتوماتیک آدرس IP بدست می آورند، یکدیگر را کشف می کنند و بسیاری توانایی های دیگر(مجمع UPnP، ۲۰۰۳). اما هنوز هم مشکلاتی وجود دارد. بعنوان مثال، هنوز نمی دانیم چگونه می توان نرم افزار و سخت افزار این دستگاه ها را به راحتی و بدون دخالت انسانی به روزآوری کرد، یا این به روزآوری ها چه موقع انجام شوند تا سازگاری با دیگر دستگاه ها از بین نرود.

---

177 inherent
178 Personal Digital Assistant
179 self-configuring
180 self- managing
181 Plug and Play
182 Universal Plug and Play Standards

نکته مهم دیگر مدیریت چیزی به نام "فضای شخصی[۱۸۳]" است. با این فرض که سیستم های خانگی از تعداد زیادی دستگاه مشترک و همچنین شخصی تشکیل شده اند ، و اینکه داده ها در سیستم های خانگی هم دچار محدودیت های اشتراک هستند، تلاش زیادی برای تحقق این فضاهای شخصی صورت گرفته است . بعنوان مثال، بخشی از فضای شخصی آلیس شامل دستور کار[۱۸۴] او، عکس های خانوادگی، دفترچه خاطرات، موسیقی و تصاویر ویدئویی خریداری شده و غیره می باشد. دارایی های شخصی باید به صورتی ذخیره شوند که آلیس به محض نیاز به آنها دسترسی پیدا کند. بعلاوه، بخش هایی از این فضای شخصی (بطورموقت) باید بوسیله دیگران قابل دسترسی باشد، مثلاً برای گذاشتن قرارهای کاری. خوشبختانه مسائل ساده تر خواهند شد. مدت ها است که این تصور وجود دارد که فضاهای شخصی مربوط به سیستم های خانگی ذاتاً در سرتاسر دستگاه های مختلف توزیع شده اند. مشخص است که چنین پراکندگی می تواند منجر به مشکلات عدیده ای در روند همگام سازی شود. اما با افزایش سریع ظرفیت هارد دیسک ها در عین کاهش ابعاد آنها، می توان مشکلات را کاهش داد. پیکربندی یک واحد ذخیره سازی چند ترابایتی برای یک کامپیوتر شخصی مشکل واقعی نیست . همچنین در حال حاضر هارد دیسک های قابل جابجایی با ظرفیت صدها گیگابایت در داخل مدیاپلیر[۱۸۵]های قابل حمل نسبتاً کوچک جایگذاری می شوند. با افزایش مداوم ظرفیت ها، شاید در آینده شاهد استفاده از نوعی معماری در سیستم های فراگیر خانگی باشیم که در آن یک ماشین منفرد (که در جایی در زیرزمین در نزدیکی سیستم گرمایش مرکزی مخفی شده) بعنوان رئیس عمل کرده و تمامی دستگاه های ثابت دیگر صرفاً نقش رابط دسترسی را برای انسانها بازی کنند. در اینصورت ممکن است دستگاه های شخصی برای اطلاعات لازم روزمره تحت فشار شدید قرار گیرند، ولی منبع ذخیره آنها هرگز به پایان نمی رسد .

با این وجود، در اختیار داشتن منبع ذخیره کافی حلال مشکل مدیریت فضاهای شخصی محسوب نمی شود. بلکه توان ذخیره سازی مقادیر عظیم داده ها مسئله را به سمت ذخیره سازی داده های *مربوط*[۱۸۶]و پیدا کردن آنها برای استفاده های آتی سوق می دهد. هر روز بیش از پیش شاهد خواهیم بود که سیستم های فراگیری از قبیل شبکه های خانگی، بکمک برنامه هایی به نام **توصیه کننده ها** [۱۸۷] (برنامه هایی که با برنامه های کاربران دیگر مشورت می کنند تا سلیقه های مشابه شناسایی شوند) راجع به محتوایی که باید در فضای شخصی افراد قرار گیرد ، تصمیم گیری کنند . جالب است بدانید که مقدار اطلاعاتی که برنامه های توصیه کننده برای انجام وظیفه خود نیاز دارند، در اغلب موارد به اندازه کافی کوچک است به طوری که می توانند روی PDAها نیز اجرا شوند (Miller ،۲۰۰۴ وگروه همکاران).

---

183 personal space
184 agenda
185 Media player
186 relevant data
187 recommenders

سیستم های الکترونیکی مراقبت از سلامت

یکی دیگر از مهمترین و رو به افزایش ترین انواع سیستم های فراگیر ، سیستم های الکترونیکی مراقبت از سلامت (فردی) است. به دلیل افزایش هزینه های درمانی، دستگاه های جدیدی برای نظارت بر سلامت افراد و برقراری تماس اتوماتیک با پزشکان در صورت نیاز ابداع شدند. یکی از مهمترین اهداف بسیاری از این سیستم ها جلوگیری از بستری شدن افراد در بیمارستان است.

سیستم های مراقبت از سلامت فردی غالباً مجهز به حسگرهای مختلفی هستند که به صورت یک شبکه(ترجیحاً بی سیم) سطح بدن[188](BAN) سازماندهی شده اند. مهم این است که این شبکه ها در بدترین حالت حداقل سختی را برای شخص داشته باشند. بنابراین شبکه باید بتواند علیرغم اتصال هیچ رشته (یعنی سیمی) به دستگاه های غیرمتحرک، در حین حرکت فرد باز هم عمل کند .

براساس شکل ۱۲-۱، در این رابطه می توان دو سازمان مشخص شناسایی کرد. در سازمان اول، یک هاب مرکزی [189] بعنوان بخشی از BAN ، داده های لازم را جمع آوری می کند. سپس این داده ها در فواصل زمانی به یک دستگاه ذخیره بزرگتر آفلود[190] می شوند. مزیت طرح این است که هاب قادر به مدیریتBAN هم می باشد . در حالت دوم، BAN پیوسته و از طریق یک اتصال بی سیم، به یک شبکه خارجی مرتبط شده و داده های نظارتی را به آن ارسال می کند. برای مدیریت BAN، باید از تکنیک های مجزایی استفاده شود. البته، ممکن است اتصالات دیگری هم با پزشک یا افراد دیگر ایجاد شود.

شکل ۱۲-۱- نظارت بر یک فرد در یک سیستم الکترونیکی فراگیر مراقبت از سلامت با استفاده از الف) هاب مرکزی و ب) اتصال بی سیم پیوسته (داخل شکل: حسگر یکبری شدن،حسگر ECG، PDA، حسگرهای حرکتی، شبکه سطح بدنی، ب) فرستنده، منبع ذخیره خارجی، GPRS/UMTS، شبکه سطح بدن).

از نقطه نظر سیستم های توزیعی، پرسشهایی از این قبیل مطرح می شود:

۱. داده های نظارت شده در کجا و چگونه باید ذخیره شوند؟
۲. چگونه می توان از مفقود شدن داده های حیاتی جلوگیری کرد؟
۳. برای تولید و پخش هشدارها نیازمند چه زیرساختی هستیم؟
۴. پزشکان چگونه می توانند بازخورد آن لاین[191] ایجاد کنند؟
۵. چگونه می توان سیستم نظارتی بسیار مقاوم را محقق نمود؟
۶. مسائل امنیتی چه هستند و چگونه می توان سیاستهای مناسبی را به کار گرفت؟

[188] body-area networks
[189] central hub
[190] offload
[191] online feedback

برخلاف سیستم های خانگی، نمی توان انتظار داشت که معماری سیستم های فراگیر مراقبت از سلامت به سمت سیستم های تک خدمتگزاری تمایل پیدا کرده و در نتیجه دستگاه های نظارتی با حداقل کارکرد عمل کنند. بلکه ، به دلایل بهره وری، دستگاه ها و شبکه های سطح بدن باید از **پردازش داده های دورن شبکه ای** [۱۹۲]پشتیبانی کنند؛ به این معنا که ، داده های نظارتی بایستی، مثلاً ، پیش از ذخیره دائم یا ارسال به پزشک، متراکم شوند. برخلاف سیستم های اطلاعات توزیعی، هنوز هیچ پاسخ مشخصی برای این پرسشها ارائه نشده است.

شبکه های حسگر

مثال آخر مربوط به سیستم های فراگیر، شبکه های حسگر است. در بسیاری از موارد، این شبکه ها بخشی از فن آوری توسعه فراگیری را تشکیل داده و ، همانطور که خواهیم دید، بسیاری از راه حلهای مربوط به شبکه های حسگر به برنامه های کاربردی فراگیر باز می گردد. قابل توجه بودن شبکه های حسگر از نقطه نظر سیستم های توزیعی آن است که تقریباً در تمام موارد، جهت پردازش اطلاعات مورد استفاده قرار می گیرند . از این نظر، وظیفه آنها چیزی بیش از صرفاً تهیه سرویس های ارتباطی است که کار شبکه های کامپیوتری مرسوم است. در (Akyildiz ۲۰۰۲ و گروه همکاران) شرح کلی راجع به شبکه های حسگر از نقطه نظر شبکه ای و در (Zhao and Guibas ۲۰۰۴) از نقطه نظر سیستمی ارائه شده است . موضوع مرتبط دیگر ، **شبکه های مش**[۱۹۳]هستند که از مجموعه ای از گره ها(ی ثابت) تشکیل شده اند که از طریق پیوندهای بی سیم با هم ارتباط برقرار می کنند. این شبکه ها می توانند اساس بسیاری از سیستم های توزیعی مقیاس متوسط[۱۹۴] را تشکیل دهند که شرح کلی از آنها در (Akyildiz ۲۰۰۵ و گروه همکاران) آمده است .

شبکه های حسگر نوعاً از ده ها تا ده ها هزار گره نسبتاً کوچکی تشکیل شده اند که هریک مجهز به یک دستگاه حس کننده می باشد. اغلب شبکه های حسگر از ارتباط بی سیم[۱۹۵] استفاده کرده و انرژی گره ها غالباً از طریق باتری تأمین می شود. محدودیت منابع ، امکانات ارتباطی محدود و مصرف توان محدود باعث می شود تا در طراحی این شبکه ها موضوع بهره وری جایگاه ویژه ای داشته باشد.

در نظر گرفتن شبکه های حسگر به عنوان پایگاه های داده ای توزیعی، ارتباط آنها را با سیستم های توزیعی مشخص می کند. تصور این که بسیاری از شبکه های حسگر برای برنامه های کاربردی نظارتی و اندازه گیری پیاده سازی شده اند (Bonnet ،۲۰۰۲ وگروه همکاران)، سادگی و قابل فهم بودن این دیدگاه را نشان می دهد. در این موارد، اپراتور می تواند صرفاً با صدور پرس وجوهایی از قبیل:″ بار ترافیکی به سمت شمال در بزرگراه ۱ چقدر است؟″ اطلاعات را از (بخشی از) شبکه استخراج کند . این نوع پرس و جوها مشابه پرس و جوها در پایگاه های داده ای متعارف است. در این حالت، شاید لازم

[192] in-network data processing
[193] mesh networks
[194] medium-scale distributed systems
[195] wireless communication

باشد که پاسخ با تشریک مساعی تعداد زیادی از حسگرهای واقع در اطراف بزرگراه ۱ ارائه شده و بقیه حسگرها به حال خود رها شوند.

مطابق شکل ۱۳-۱، برای سازماندهی شبکه های حسگر به صورت پایگاهای داده ای توزیعی دو راه وجود دارد . در روش اول ، حسگرها تشریک مساعی نمی کنند، بلکه صرفاً داده های خود را به یک پایگاه داده ای متمرکز واقع در سایت اپراتور ارسال می کنند. در روش دوم، پرس و جوها به حسگرهای مربوطه ارسال شده و هر حسگر پاسخی را محاسبه می کند ، البته اپراتور هم باید پاسخهای عودت داده شده را بصورتی منطقی متمرکز کند.

هیچیک از ایندو راه حل خیلی جالب نیستند. راه حل اول مستلزم آن است که حسگرها تمامی داده های اندازه گیری شده خود را از طریق شبکه ارسال کنند که البته اینکار باعث اتلاف انرژی و منابع شبکه خواهد شد. راه حل دوم هم می تواند باعث اتلاف شود، اگر توانایی متراکم سازی حسگرها و در نتیجه کاهش محسوس تعداد داده های بازگشتی به اپراتور ندیده گرفته شود. در واقع مشابه سیستم های فراگیر مراقبت از سلامت، در این شبکه ها هم نیازمند امکاناتی برای **پردازش داده های داخل شبکه ای** هستیم .

شکل ۱۳-۱- سازماندهی یک پایگاه داده ای شبکه حسگر در حین ذخیره سازی و پردازش داده الف) فقط در سایت اپراتور یا ب) فقط در حسگرها(داخل شکل: الف) سایت اپراتور، داده های حسگر مستقیماً به اپراتور ارسال می شود، شبکه حسگر، ب) سایت اپراتور، هریک از حسگرها قادر به پردازش و ذخیره داده ها هستند، شبکه حسگر، پرس وجو، حسگرها فقط پاسخها را ارسال می کنند).

پردازش داخل شبکه ای به اشکال مختلفی قابل انجام است. یک راه حل مشخص آن است که از طریق درختی که جامع تمام گره ها باشد، برای تمام گره های حسگرها یک پرس و جو ارسال کرده و سپس، نتایج را در حین پخش بازگشتی به ریشه -که محل آغاز آنها محسوب می شود- متراکم[۱۹۶] کنیم . تراکم در جایی رخ خواهد داد که دو یا چند شاخه از درخت با هم تلاقی می کنند. این طرح با وجود سادگی ، سه پرسش عمده را مطرح می کند:

۱. چگونه می توان در یک شبکه حسگر (به صورتی پویا) یک درخت مفید و مؤثر ایجاد کرد؟
۲. متراکم سازی نتایج چگونه انجام می شود و آیا قابل کنترل است؟
۳. در صورت شکست پیوندهای شبکه چه اتفاقی می افتد؟

این پرسشها تا حدودی در TinyDB – که رابطی توصیفی[۱۹۷] (پایگاه داده ای) را برای شبکه های حسگر بی سیم پیاده می کند- پاسخ داده شده اند. بطور خلاصه، TinyDB می تواند از هر نوع

[196] aggregate
[197] declarative

الگوریتم مسیردهی براساس درخت استفاده کند. یکی از گره های میانی نتایج را از بچه های خود جمع آوری کرده و به همراه یافته های خود متراکم کرده و به سمت ریشه ارسال می کند . برای افزایش اثربخشی ، پرس وجوها با تعریف یک دوره زمانی ، امکان زمانبندی دقیق عملیات و در نتیجه، استفاده بهینه از منابع و انرژی شبکه را فراهم می کنند . جزئیات بیشتر را در (Madden ،۲۰۰۵ وگروه همکاران) مطالعه کنید.

اما اگر بتوان پرس و جوها را از نقاط مختلف در شبکه آغاز کرد، استفاده از درختهای تک ریشه ای از قبیلTinyDB شاید چندان هم مؤثر نباشد. در روش دیگر، شبکه های حسگر به گره های ویژه ای مجهز می شوند که نتایج و همچنین پرس وجوهای مربوط به آن نتایج به آنها ارسال می شوند . بعنوان یک نمونه ساده، پرس وجوها و نتایج مربوط به خواندن درجه حرارت در نقطه ای متفاوت نسبت به نقطه اندازه گیری رطوبت جمع آوری می شوند. این روش مستقیماً مشابه ایده سیستم های انتشار / اشتراک است که در فصل ۱۳ مورد بحث قرار خواهند گرفت.

۴-۱- خلاصه

سیستم های توزیعی ازکامپیوترهای خودمختاری تشکیل شده اند که ضمن همکاری با هم، نمایی از یک سیستم منسجم و منفرد، ارائه می دهند. یکی از مهم ترین مزایای این دست سیستم ها آن است که تلفیق برنامه های کاربردی مختلف را ، که روی کامپیوترهای مختلف هم در حال اجرا هستند، در یک سیستم واحد تسهیل می کنند. مزیت دیگر اینکه سیستم های توزیعی در صورت طراحی مناسب، به خوبی با ابعاد شبکه زیربنایی مقیاس برداری می شوند . اما هزینه ای که باید در قبال این مزایا بپردازیم ، افزایش پیچیدگی نرم افزار، افت کارآیی و کاهش سطح امنیتی است. با وجود تمام این اشکالات، هنوز هم علاقه زیادی به ساخت و نصب سیستم های توزیعی در سرتاسر جهان وجود دارد.

هدف غالب سیستم های توزیعی، مخفی سازی بسیاری از پیچیدگی های مربوط به توزیع فرآیندها، داده و کنترل است. اما کسب این شفافیت توزیعی نه تنها باعث افت عملکرد می شود، در موقعیتهای عملی هم هرگز به طور کامل محقق نمی شود . باید در طراحی سیستم های توزیعی، مسأله ایجاد توازن در کسب اشکال مختلف شفافیت توزیعی لحاظ شده و همین امر درک آنها را پیچیده می کند.

پیچیدگی بیشتر ناشی از این واقعیت است که بسیاری از سازندگان در ابتدای کار فرضیات اساساً نادرستی راجع به شبکه زیربنایی در نظر دارند . بعد ها که این فرضیات با شکست مواجه می شود، ممکن سرپوش گذاشتن بر رفتار ناخواسته ناشی از آنها مشکل ساز شود. بعنوان مثال، این فرض که تأخیرهای شبکه ناچیز هستند را در نظر بگیرید. بعداً، حین انتقال سیستم موجود به یک شبکه حوزه گسترده، مخفی سازی تأخیرها ممکن است تأثیر شدیدی بر طرح اولیه سیستم داشته باشد. از نمونه فرض های نابجای دیگر می توان به فرض قابل اطمینان بودن ، ثبات ، ایمنی و همگن بودن شبکه اشاره کرد.

انواع مختلف سیستم های توزیعی را می توان در سه گروه سیستم های پشتیبان محاسبات،پردازش اطلاعات و فراگیری دسته بندی کرد. سیستم های محاسبه توزیعی نوعاً برای برنامه های کاربردی با کارآیی بالا ایجاد شده اند که از حوزه محاسبه موازی سرچشمه می گیرند. گروه دیگری از سیستم های توزیعی را می توان در محیط های دفاتر کار سنتی مشاهده کرد که پایگاههای داده ای در آنها نقش مهمی را ایفا می کنند. معمولاً در این محیط ها از سیستم های پردازش تراکنش[۱۹۸] استفاده می شود. در آخرین گروه سیستم های توزیعی نوظهور، مؤلفه ها کوچک بوده و سیستم به صورت موردی ساخته شده است ، اما مدیریت آنها دیگر برعهده مدیر اجرایی سیستم نمی باشد . نمونه بارز این گروه، محیط های محاسبه همه جا حاضر[۱۹۹] شناخته می شود.

مسائل فصل

۱- یکی دیگر از تعاریفی که برای سیستم های توزیعی ارائه می شود به این صورت است: ″مجموعه ای از کامپیوترهای مستقل که مانند یک *سیستم واحد* بنظر می رسد. یعنی حتی این که چندین کامپیوتر هستند از دید کاربران مخفی است″. نمونه ای از مواردی را ذکر کنید که این دیدگاه برای آن بسیار کاربردی است.

۲- راجع به نقش میان افزارها در سیستم های توزیعی توضیح دهید.

۳- بسیاری از سیستم های شبکه ای شده به صورت یک دفترکار عقبی[۲۰۰] و یک دفتر کار جلویی[۲۰۱] سازمان دهی شده اند. سازمانها چگونه با نمای منسجم سیستم های توزیعی هماهنگ می شوند؟

۴- راجع به معنی شفافیت(توزیعی) توضیح داده و نمونه هایی از انواع مختلف شفافیت ذکر کنید.

۵- توضیح دهید که چرا در برخی موارد مخفی سازی وقوع و ترمیم خرابی در سیستم های توزیعی بسیار سخت است؟

۶- چرا کسب حداکثر درجه شفافیت ممکنه همواره ایده خوبی نیست؟

۷- سیستم توزیعی باز را تشریح کرده و مزایای باز بودن را توضیح دهید؟

۸- دقیقاً معنای سیستم های مقیاس پذیر توضیح دهید.

۹- مقیاس پذیری به روشهای مختلف قابل دستیابی است. این روش ها چیستند؟

۱۰- سازمان های مجازی را توضیح داده و اشاره ای به نحوه پیاده سازی این دست سازمانها داشته باشید.

---

[198] transaction processing
[199] ubiquitous
[200] Back office
[201] Front office

۱۱- گفتیم که در صورت سقط شدن تراکنش ، دنیا طوری به وضعیت سابق خود باز می گردد که گویا اصلاً تراکنشی انجام نشده بود . اما دروغ گفتیم. نمونه ای از مواردی ذکر کنید که بازگرداندن دنیا به صورت قبل غیر ممکن باشد.

۱۲- اجرای تراکنشهای جایگذاری شده مستلزم وجود شکلی از هماهنگی است. راجع به وظیفه واقعی یک هماهنگ کننده[۲۰۲] توضیح دهید.

۱۳- گفتیم که شفافیت توزیعی ممکن است در سیستم های فراگیر موجود نباشد. اما این گفته در مورد همه انواع شفافیتها صدق نمی کند . یک مثال ذکر کنید.

۱۴- در متن درس نمونه هایی از سیستم های توزیعی فراگیر از قبیل سیستم های خانگی، سیستم های الکترونیک مراقبت از سلامت و شبکه های حسگر ذکر کردیم . نمونه های دیگری را از این مجموعه نام ببرید.

۱۵- (**تکلیف آزمایشگاهی**) طرحی برای یک سیستم خانگی متشکل از یک خدمتگزار رسانه مجزا با امکان اتصال یک مشتری بی سیم به آن ترسیم کنید . این مشتری به تجهیزات صوتی/ تصویری (آنالوگ) متصل بوده و جریانات رسانه ای دیجیتال را به خروجی آنالوگ تبدیل می کند. خدمتگزار روی ماشین جداگانه ای اجرا می شود که احتمالاً به اینترنت متصل است ، اما هیچ صفحه کلید و/ یا صفحه نمایشی به آن متصل نیست.

[202] coordinator

۲
معماری

سیستم های توزیعی غالباً قطعات پیچیده نرم افزاری هستند که براساس تعریف مؤلفه[۲۰۳] های آنها در سرتاسر ماشینهای مختلف پراکنده شده اند. جهت فائق آمدن بر پیچیدگی این سیستم ها، باید آنها را به نحو مناسبی سازماندهی نمود. روشهای متعدد و مختلفی برای بررسی نحوه سازماندهی سیستم های توزیعی وجود دارد. اما یک روش مشخص، تمایز قائل شدن میان سازمان منطقی مجموعه مؤلفه های نرم افزاری و از طرف دیگر، درک فیزیکی واقعی است.

چگونگی سازمان سیستم های توزیعی تا حد زیادی در رابطه با مؤلفه های نرم افزاری تشکیل دهنده سیستم است. این **معماری های نرم افزار** بیانگر نحوه سازماندهی مؤلفه های مختلف نرم افزاری و تقابل آنهاست. در این فصل، ابتدا اشاره ای خواهیم داشت به برخی از پرکاربردترین روش هایی که غالباً جهت سازماندهی سیستم های کامپیوتری (توزیعی) مورد استفاده قرار می گیرد.

تحقق یک سیستم توزیعی مستلزم آن است که مؤلفه های نرم افزار ایجاد و روی ماشین های واقعی قرار داده شوند. برای انجام اینکار روشهای مختلفی وجود دارد. نحوه استقرار نهایی معماری نرم افزار غالباً **معماری سیستم** هم خوانده می شود. در این فصل ابتدا نگاهی خواهیم داشت به معماری های متمرکز مرسوم که درآنها، پیاده سازی اغلب مؤلفه های (و بنابراین کارکردهای) نرم افزاری بر روی یک سرور واحد قرار دارد و مشتری های دور قادر هستند تا با روشهای ساده ارتباطی به آن سرور دسترسی پیدا کنند. سپس، معماری های غیرمتمرکز را بررسی خواهیم کرد که در آنها ، چندین ماشین نقشهای کمابیش یکسانی ایفا کرده و همچنین نگاهی خواهیم داشت به سازمانهای تلفیقی.

همانطور که در فصل ۱ توضیح داده شد، یکی از اهداف مهم سیستم های توزیعی ، تفکیک برنامه های کاربردی از بسترهای زیربنایی از طریق ایجاد یک لایه میان افزاری است. استفاده از چنین لایه ای یکی از مهمترین تصمیمات معماری محسوب شده و هدف اصلی آن، ایجاد شفافیت توزیعی است. بااین وجود، دستیابی به این شفافیت مستلزم ایجاد برخی موازنه[۲۰۴] ها است. همین امر منجر به تنوع روشهای تطبیق پذیر[۲۰۵] در میان افزارها شده است. از آنجاییکه این روشها بر سازمان میان افزار تأثیر می گذارند، راجع به پرکاربردترین آنها در همین فصل صحبت خواهیم کرد.

روش دیگر برای ایجاد انطباق پذیری در سیستم های توزیعی ، ایجاد امکان نظارت بر رفتار برای خود سیستم و اتخاذ اقدامات مناسب در صورت لزوم است. این دیدگاه موجب تولید انواعی از سیستمها به نام **سیستم های خودکار**[۲۰۶] شده است. این دست سیستم های توزیعی معمولاً به شکل حلقه کنترل

[203] Component
[204] Trade-off
[205] Adaptive
[206] Automatic

بازخوردی سازماندهی شده و یکی از عوامل مهم معماری درزمان طراحی سیستم می باشند. در این فصل، بخش مستقلی را به سیستم های توزیعی خودکار اختصاص داده ایم.

۱-۲- شیوه های معماری[۲۰۷]

بحث خود درمورد معماری ها را ابتدا با بررسی سازمان منطقی سیستم های توزیعی به مؤلفه های نرم افزاری یا همان **معماری نرم افزار** آغاز می کنیم (۲۰۰۳، Bass و گروه همکاران). تحقیق در مورد معماری های نرم افزارها به میزان قابل ملاحظه ای پیشرفت کرده و امروزه دیگر اغلب باور دارند که طراحی یا استفاده ازنوع خاصی از معماری در توسعه موفقیت آمیز سیستم های بزرگ اهمیت کلیدی دارد.

در بحث حاضر مفهوم **شیوه معماری** اهمیت به سزا دارد. چنین شیوه ای شامل مؤلفه ها، نحوه اتصال مؤلفه ها به یکدیگر، تبادل داده ها در بین مؤلفه ها و در نهایت، نحوه پیکربندی این عوامل برای تشکیل یک سیستم واحد است. منظور از **مؤلفه** واحد پیمانه ای[۲۰۸] با واسط[۲۰۹] های کاملاً تعریف شده لازم و موجود است که در داخل محیط خود قابل تعویض هستند (OMG،۲۰۰۴b). همانطور که ذیلاً هم خواهیم گفت، نکته مهم در مورد یک مؤلفه برای سیستم های توزیعی آن است که مشروط به لحاظ کردن واسط آن، مؤلفه قابل تعویض است. یکی از مفاهیمی که درک آن تاحدودی مشکل است، **اتصال گر[۲۱۰] ها** هستند که غالباً به صورت مکانیزمی برای میانجی گری بین ارتباطات، هماهنگ سازی، یا همکاری در بین مؤلفه ها تعریف می شوند (۲۰۰۰، Mehta و گروه همکاران ؛۱۹۹۷، Shaw and Clements). به عنوان مثال، می توان با استفاده از امکانات مربوط به فراخوانی روال ها (از راه دور)، تبادل پیام، یا جریان های داده ای یک اتصال گر ساخت.

بااستفاده از مؤلفه ها و اتصال گرها ها می توان پیکربندی های مختلفی ایجاد نمود که نهایتاً در قالب شیوه های معماری دسته بندی می شوند. شیوه های مختلفی شناسایی شده اند که از مهمترین آنها در سیستم های توزیعی می توان به موارد زیر اشاره کرد:

۱. معماری های لایه ای[۲۱۱]
۲. معماری های براساس شیء[۲۱۲]
۳. معماری های داده محور[۲۱۳]
۴. معماری های بر اساس رویداد[۲۱۴]

---

[207] Architectural Styles
[208] Modular
[209] Interface
[210] Connector
[211] Layered
[212] Object-based
[213] Data-centered
[214] Event-based

ایده اساسی در شیوه لایه ای بسیار ساده است: مؤلفه ها **به صورت لایه ای** سازماندهی می شوند . همانطور که در شکل الف) ۱-۲ مشاهده می کنید، مؤلفه واقع در لایه L_i می تواند مؤلفه های واقع در لایه زیرین آن L_{i-1} (و نه برعکس) را فراخوانی نماید. این مدل بطور گسترده در جوامع شبکه ای به کار برده شده و در فصل ۴ شرح مختصری ازآنرا ارائه خواهیم کرد. یکی از مشاهدات کلیدی این است که کنترل غالباً از یک لایه به لایه دیگر جریان می یابد. به بیان دیگر، درخواستها بصورت سلسله مراتبی از بالا به پایین و نتایج از پایین به بالا جریان پیدا می کنند.

در حالیکه در **معماری های براساس شیء** از ساختار بسیار آزادانه تر شکل ب) ۱-۲ تبعیت می شود. بطور خلاصه، هر شیء متناظر با چیزی است که ما آنرا مؤلفه می نامیم و این مؤلفه ها از طریق یک مکانیزم فراخوانی روال (از راه دور) به هم متصل می شوند. تعجبی ندارد اگر بگوییم که این معماری نرم افزاری با معماری سیستم مشتری- سرور که قبلاً گفته شد، هماهنگی دارد. معماری های لایه ای و براساس شیء هنوز هم مهمترین شیوه های سیستم های بزرگ نرم افزاری محسوب می شوند (۲۰۰۳، Bass وگروه همکاران).

شکل ۱-۲- شیوه معماری الف) لایه ای و ب) براساس شیء(داخل شکل: لایه N ، لایهN-1 ، جریان درخواست، جریان پاسخ، لایه۲، لایه۱، شیء، فراخوانی روش).

معماری های داده محور[۲۱۵] براساس ایده برقراری تماس بین فرآیندها از طریق یک منبع مشترک(فعال یا غیرفعال) شکل گرفته اند . می توان استدلال کرد که اهمیت این معماری ها در سیستم های توزیعی به اندازه معماری های لایه ای و شیء-محور است. بعنوان مثال، مجموعه بزرگی از برنامه های کاربردی تحت شبکه براساس سیستم فایل توزیعی اشتراکی بوجود آمده اند که در آنها ، تقریباً تمام ارتباطات از طریق فایلها انجام می شود. به همین ترتیب، سیستم های توزیعی براساس وب، که به تفصیل در فصل ۱۲ مورد بحث قرار خواهند گرفت، تا حد زیادی داده محور هستند. یعنی در آنها فرآیندها با استفاده از سرویسهای داده ای مشترک براساس وب ارتباط برقرار می کنند.

در حالت **معماری براساس رویداد**[۲۱۶]، فرآیندها اساساً از طریق انتشار رویدادها ارتباط برقرار کرده و مطابق شکل الف) ۲-۲، بصورت اختیاری داده هم حمل می کنند. در سیستم های توزیعی، پخش رویداد غالباً با آنچه به نام **سیستم های انتشار/ اشتراک**[۲۱۷] خوانده می شود(۲۰۰۳، Eugster وگروه همکاران)، مرتبط است. ایده اساسی این است که فرآیندها، در صورتی اقدام به پخش رویدادها خواهند کرد که میان افزار اطمینان دهد که فقط فرآیندهای قبلاً مشترک شده در آن رویداد آنرا دریافت می کنند. مزیت عمده سیستم های براساس رویداد آن است که فرآیندها دارای اتصال سست[۲۱۸]

[215] Data-Centric
[216] Event-based
[217] Publish/Subscribe
[218] Loosely coupled

هستند و در مجموع، لزومی ندارد که آشکارا به یکدیگر ارجاع دهند. این حالت به نام بازشدن اتصال در فضا یا به بیان دیگر ، **بازشدن اتصال مرجعی**[۲۱۹] هم خوانده می شود.

شکل ۲-۲- شیوه معماری الف) براساس رویداد و ب) فضای داده ای اشتراکی (داخل شکل: مؤلفه، تحویل رویداد، گذرگاه رویداد، انتشار، مؤلفه، ب) مؤلفه، تحویل داده، انتشار، فضای داده ای (دائم) اشتراکی).

معماری های براساس رویداد را می توان با معماری های داده محور تلفیق کرد و چیزی به نام **فضاهای داده ای اشتراکی**[۲۲۰] ایجاد نمود . بطور خلاصه ، معنای فضاهای داده ای اشتراکی آن است که فرآیندها اکنون از نظر زمانی هم به یکدیگر اتصال ندارند و دیگر لازم نیست که هردو در حین انجام ارتباط فعال باشند. بعلاوه، بسیاری از فضاهای داده ای اشتراکی از یک رابط شبه SQL برای منبع اشتراکی استفاده می شود تا بتوان- مشابه حالت فایلها- با استفاده از توصیف ونه یک مرجع آشکار، به داده ها دسترسی پیدا کرد. به دلیل اهمیت موضوع کل فصل ۱۳ را به این شیوه ویژه معماری اختصاص داده ایم.

وجه اهمیت این معماری های نرم افزاری در سیستم های توزیعی آن است که هدف همه آنها دستیابی به شفافیت توزیعی(در حدی معقول و منطقی) است. با این وجود، همانطور که قبلاً هم بحث کردیم، شفافیت توزیعی مستلزم ایجاد موازنه[۲۲۱] هایی میان کارآیی، تحمل خرابی، سهولت برنامه نویسی و غیره است. از آنجایی که هیچ راه حل منحصربه فردی وجود ندارد که در شرایط تمامی برنامه های کاربردی توزیعی ممکنه صدق کند، محققان دیگر به دنبال ایجاد سیستم توزیعی واحدی نیستند که در ۹۰٪ تمامی حالات ممکنه قابل کاربرد باشد .

۲-۲- معماری های سیستم

حال که تاحدوی با برخی از شیوه های متعارف معماری آشنا شدیم ، اجازه دهید نگاهی داشته باشیم به آمار واقعی سیستم های توزیعی که براساس محل قرارگیری مؤلفه های نرم افزاری سازمان داده شده اند. تصمیم گیری درمورد مؤلفه های نرم افزاری ، تعامل آنها و محل قرارگیری آنها منجر به حالتی از معماری نرم افزاری می شود که به نام **معماری سیستم** هم معروف است (۲۰۰۳، Bass وگروه همکاران). در ادامه راجع به سازمانهای متمرکز و غیرمتمرکز و انواع مختلف ترکیب آنها صحبت خواهیم کرد.

۱-۲-۲- معماری های متمرکز

علیرغم نبود اتفاق نظر در مورد بسیاری از مسائل سیستم های توزیعی، بسیاری از محققان و صاحبنظران فن برسر این مسأله توافق کرده اند که این دیدگاه که *مشتریان* از *سرورها* تقاضای سرویس

[219] Referentially decoupled
[220] Shared data spaces
[221] Trade-off

می کنند، به ما در درک و اداره پیچیدگی سیستم های توزیعی کمک کرده و دیدگاه مفیدی محسوب می شود.

در مدل اصلی مشتری- سرور، فرآیندها در سیستم توزیعی به دو گروه (احتمالاً دارای هم پوشانی) تقسیم می شوند: **سرور،** فرآیند اجراکننده یک سرویس خاص از قبیل سرویس سیستم فایل یا سرویس پایگاه داده ای است. در حالیکه **مشتری،** فرآیند درخواست کننده یک سرویس خاص از سرور از طریق ارسال درخواست برای سرور و سپس، انتظار برای دریافت پاسخ از جانب آن است. تقابل مشتری- سرور که **رفتار درخواست- پاسخ** هم نامیده می شود، در شکل ۳-۲ نمایش داده شده است.

شکل ۳-۲- تعامل عمومی بین مشتری و سرور(داخل شکل: مشتری، انتظار برای نتیجه، درخواست، پاسخ، سرور، ارائه سرویس، زمان).

ارتباط بین مشتری و سرور را می توان در صورت مطمئن بودن شبکه زیربنایی- که درمورد بسیاری از شبکه های محلی صدق می کند- با استفاده از یک پروتکل بدون اتصال ساده پیاده نمود. در این موارد ، هنگامیکه مشتری سرویس خاصی را تقاضا می کند، پیامی را برای سرور بسته بندی می کند. داخل پیام سرویس مورد نظر مشخص شده و داده های ورودی نیز قرار داده شده اند. سپس پیام مذکور برای سرور ارسال می شود. سرور که به نوبه خود همواره درحالت انتظار برای درخواست جدید قرار دارد ، پیام را پردازش ، نتایج را به شکل پیام پاسخ بسته بندی و در نهایت برای مشتری ارسال می نماید.

مزیت آشکار استفاده از پروتکل بدون اتصال ،کارآمدی است. مادامی که پیام ها مفقود یا معیوب نشوند، پروتکل درخواست/ پاسخ فوق به خوبی عمل خواهد کرد. متأسفانه، مقاوم سازی پروتکلها در برابر خرابی های موردی سیستم ارسال چندان هم کار ساده ای نیست. تنها راه حل ممکنه شاید این باشدکه به مشتری اجازه دهیم تا در صورت عدم دریافت پیام پاسخ، درخواست خود را مجدداً ارسال نماید. اما مسأله این است که مشتری نمی تواند تشخیص دهد کند که آیا مشکل در اثر مفقود شدن پیام درخواست اصلی ایجاد شده و یا ارسال پاسخ با شکست مواجه شده است. در صورت مفقود شدن پاسخ، آنگاه ارسال مجدد درخواست ممکن است منجر به انجام دوباره عملیات شود.در عملیات مفروض " مبلغ ۱۰۰۰۰دلار از حساب بانکی من انتقال بده" ، مشخص است که روش بهتر، دادن گزارش خطاست. اما در عملیات "چه مقدار پول در حسابم باقی مانده" ارسال مجدد درخواست کاملاً بدون اشکال است. عملیاتی را که بتوان آنرا بدون هیچ مشکلی چندین مرتبه تکرار کرد، اصطلاحاً دارای **احتمال نشانه گذاری مجدد**[۲۲۲] خوانده می شود. ازآنجاییکه برخی درخواست ها دارا این ویژگی و برخی دیگر فاقد آن هستند، پس هیچ راه حل واحدی برای توصیه درباره موضوع مفقود شدن پیامها وجود ندارد. در فصل ۸ به طور مفصل راجع به نحوه برخورد با خرابی در ارسال پیام بحث خواهیم کرد.

[222] Idempotent

در روش دیگر، بسیاری از سیستم های مشتری- سرور ازیک پروتکل اتصال گرای مطمئن استفاده می کنند. هرچند این راه حل به دلیل کارآیی نسبتاً پایین در شبکه های محلی مناسب نیست، در سیستم های حوزه گسترده[۲۲۳] که ارتباطات ذاتاً غیرمطمئنی دارند، عملکرد خیلی خوبی دارد. بعنوان نمونه، تقریباً تمامی پروتکل های برنامه کاربردی اینترنت براساس اتصالات قابل وثوق TCP/IP استوار هستند. در این حالت، هرزمانی که مشتری درخواست سرویس نماید، پیش از ارسال درخواست، ابتدا اتصالی را با سرور برقرار می نماید. سرور هم برای ارسال پیام پاسخ غالباً از همین اتصال استفاده نموده و سپس اتصال ازبین می رود. مشکل این است که ایجاد و ازبین بردن اتصال، خصوصاً در صوررت کوچک بودن پیامهای درخواست و پاسخ، تاحدودی هزینه براست.

لایه بندی برنامه کاربردی

سالهاست که مدل مشتری- سرور بحث ها و مناظره های زیادی براه انداخته است. یکی از مهمترین مسائل در همه این سال ها ایجاد تمایزی آشکار بین مشتری و سرور بوده است. اما تعجبی ندارد اگر بگوییم که در اغلب موارد هیچ تفاوت آشکاری بین ایندو وجود ندارد. بعنوان مثال، سرور یک پایگاه داده ای توزیعی ممکن است دائماً بعنوان مشتری عمل کند، به این ترتیب که درخواست ها را به سرورهای فایل مختلفی ارسال می کندکه مسئول پیاده سازی جداول پایگاه داده ای هستند. در چنین حالتی، خود سرور پایگاه داده ای اساساً کاری غیر از پردازش پرس وجوها انجام نمی دهد.

بااین وجود، از آنجایی که هدف بسیاری از برنامه های کاربردی مشتری- سرور پشتیبانی از دسترسی کاربر به پایگاههای داده ای است، بسیاری از کارشناسان براساس شیوه معماری لایه ای که اخیراً مورد بحث قرار گرفت، تفاوت میان سه سطح زیر توصیه کرده اند:

:

۱. لایه رابط/کاربر
۲. لایه پردازش
۳. لایه داده

سطح کاربر- رابط حاوی تمامی ضروریات برای رابطه مستقیم با کاربر از قبیل مدیریت نمایش است. لایه پردازش حاوی برنامه های کاربردی است. سطح داده مدیریت داده های واقعی را که روی آنها عمل می شود بر عهده دارد.

مشتریان معمولاً سطح رابط کاربر را پیاده سازی می کنند. این لایه حاوی برنامه هایی است که برای کاربر نهایی امکان تعامل با برنامه های کاربردی را فراهم می آورد. تفاوت عمده ای بین برنامه های پیشرفته رابط کاربر وجود دارد.

[223] Wide area

ساده ترین برنامه رابط کاربر چیزی بیش از یک صفحه نمایش براساس کاراکتر نیست. از چنین رابط هایی نوعاً در محیط های کامپیوتری بزرگ[۲۲۴] استفاده می شود. در مواردی که کامپیوتر بزرگ تمامی تعامل ها از جمله با صفحه کلید و با صفحه نمایش را تحت کنترل دارد، دیگر نمی توان از محیط مشتری/سرور سخنی به میان آورد. بااین وجود، در بسیاری از موارد ترمینال کاربر برخی از پردازش های محلی جزئی از قبیل انعکاس حروف تایپ شده یا پشتیبانی از رابط های فرم ها را انجام می دهد که باید پیش از ارسال ورودی به کامپیوتر اصلی بطور کامل ویرایش شوند.

امروزه، حتی در محیط کامپیوترهای بزرگ هم شاهد واسط های کاربر پیشرفته تری هستیم. در اغلب مواردً، ماشین مشتری حداقل یک صفحه نمایش گرافیکی ارائه می دهد که در آن از منوهای بالا آوری[۲۲۵] یا پایین آوری[۲۲۶] استفاده شده و بسیاری از کنترل های صفحه به جای صفحه کلید توسط موس انجام می شود. از نمونه های چنین واسط هایی می توان به واسط های ویندوز X اشاره کرد که در بسیاری از محیط های UNIX استفاده می شود، همچنین می توان به واسط های قدیمی تر که در کامپیوتر های شخصی MS-DOS و Apple Macintosh کاربرد داشت اشاره کرد.

واسط های کاربر امروزی با ایجاد امکان اشتراک برنامه های کاربردی در یک پنجره گرافیکی واحد و استفاده از این پنجره برای تبادل داده ها توسط کاربر میزان کارکرد را تا حد بسیار زیادی افزایش داده اند. بعنوان مثال، برای حذف یک فایل ، معمولاً می توان آیکن نماینده آنرا به آیکن سطل آشغال انتقال داد. به همین ترتیب، بسیاری از پردازشگرهای کلمه[۲۲۷] به کاربر امکان می دهد تا با استفاده از موس، متن داخل یک سند مفروض را به موقعیت دیگری انتقال دهند. در فصل ۳ مجدداً در مورد واسط های کاربر بحث خواهیم کرد.

بسیاری از برنامه های کاربردی مشتری- سرور را می توان بکمک سه بخش تقریباً متفاوت ایجاد نمود: بخشی که کار تعامل با کاربر را در اختیار دارد، بخشی که روی یک پایگاه داده ای یا سیستم فایل عمل می کند، و بخش میانی که غالباً شامل کارکرد اصلی برنامه کاربردی است. این بخش میانی منطقاً در سطح پردازش واقع است. برخلاف واسط کاربر و پایگاه داده ای، سطح های پردازشی جوانب مشترک زیادی ندارد. بنابراین ، برای درک بهتر این لایه چندین نمونه را ذکر خواهیم کرد.

بعنوان اولین نمونه، یک موتور جستجوی اینترنت را درنظر بگیرید. اگر بنرهای متحرک، تصاویر و دیگر زینت های پنجره را در نظر نگیریم، رابط کاربر موتور جستجو بسیار ساده است: کاربر، زنجیره ای از کلمات کلیدی را تایپ کرده و سپس ، لیستی از عناوین صفحات وب در اختیار او گذاشته می شود. پس زمینه[۲۲۸] از پایگاه داده ای عظیمی از صفحات وب تشکیل می شود که از پیش واکشی شده و فهرست

[224] Mainframe
[225] Pop-up
[226] Pull-down
[227] Word processors
[228] Back-end

بندی شده اند. هسته موتور جستجو را برنامه ای تشکیل می دهد که زنجیره کلمات کلیدی کاربر را به یک یا چند پرس وجو در پایگاه داده ای تبدیل می کند. سپس، نتایج را به صورت لیست رتبه بندی کرده و لیست را به مجموعه ای از صفحات HTML ترجمه می کند. در مدل مشتری- سرور، این بخش بازیابی اطلاعات نوعاً در لایه پردازشی قرار داده می شود، همان طور که در شکل ۴-۲ دیده می شود.

شکل ۴-۲- سازمان ساده شده یک موتور جستجوی اینترنت به صورت سه لایه(داخل شکل: رابط کاربر، سطح رابط کاربر، بیان کلمات کلیدی، صفحه HTML حاوی لیست، تولیدکننده HTML، تولیدکننده پرس و جو، پرس و جوهای پایگاه داده، لیست رتبه بندی شده حاوی عناوین صفحات، الگوریتم رتبه بندی، سطح پردازشی، عناوین صفحات وب حاوی فوق اطلاعات،پایگاه داده حاوی صفحات وب، لایه داده).

بعنوان مثال دوم، یک سیستم تصمیم یار برای تالار بورس را در نظربگیرید. این سیستم را هم می توان مشابه موتور جستجو به یک نرم افزار پیش-انتهایی[۲۲۹] اجراکننده رابط کاربر و یک نرم افزار پس-انتهایی[۲۳۰] جهت دسترسی به پایگاه داده اطلاعات مالی و برنامه های آنالیزی بین ایندو تقسیم کرد. جهت تجزیه و تحلیل اطلاعات مالی ممکن است از روش ها و تکنیکهای پیشرفته آماری و هوش مصنوعی استفاده شود. در برخی موارد، ممکن است حتی لازم باشد که هسته سیستم تصمیم یار مالی روی کامپیوترهای با کارآیی بالا اجرا شود تا بدین ترتیب، گذردهی و پاسخ دهی که کاربران آن انتظار دارند، حاصل شود.

بعنوان آخرین نمونه، یک بسته رومیزی معمول متشکل از یک پردازشگر کلمه[۲۳۱]، یک برنامه کاربردی صفحه گسترده[۲۳۲] ، امکانات ارتباطی و غیره را در نظر بگیرید. این مجموعه های "آفیس" غالباً با رابط کاربر مشترک که از اسناد ترکیبی پشتیبانی نموده مجتمع می شوند و بر روی فایلهایی از دایرکتوری کاربر عمل می کنند. (در محیط آفیس، چنین دایرکتوری غالباً روی یک سرور فایل دور قرار داده میشود.) در این مثال، سطح پردازشی متشکل از مجموعه نسبتاً بزرگی از برنامه هاست که هریک از امکانات پردازشی نسبتاً ساده ای برخوردار هستند.

سطح داده در یک مدل مشتری – سرور شامل برنامه هایی است که از داده های واقعی که برنامه های کاربردی روی آنها عمل می کنند، نگهداری می کنند. یکی از ویژگی های مهم این لایه آن است که داده ها غالباً **پایدار**[۲۳۳] هستند، یعنی حتی در صورتی هم که هیچ برنامه کاربردی در حال اجرا نباشد، داده جهت استفاده های بعدی در جایی ذخیره خواهد شد. این لایه در ساده ترین شکل خود، متشکل

[229] Front-end
[230] Back-end
[231] Word processor
[232] Spread sheet
[233] Persistent

از یک سیستم فایلی است ، اما معمولتر آن است که از یک پایگاه داده ای تمام عیار استفاده شود. در مدل مشتری- سرور، لایه داده نوعاً در طرف سرور پیاده سازی می شود.

لایه داده علاوه بر ذخیره داده ها، غالباً مسئول حفظ انسجام[۲۳۴] داده ای در تمام برنامه های کاربردی مختلف هم می باشد. در مورد پایگاههای داده ای، حفظ انسجام به این معناست که ابرداده هایی از قبیل شرح جداول، محدودیتهای ورودی و ابرداده های ویژه برنامه کاربردی هم در این لایه ذخیره خواهند شد. بعنوان مثال در امور بانکداری،می خواهیم به محض رسیدن بدهی کارت اعتباری یکی از مشتریان به مقداری مشخصی ، برای او اطلاعیه ای صادر کنیم . این نوع اطلاعات را می توان بوسیله تریگر[۲۳۵] پایگاه داده ای حفظ نمود. این تریگر یک راه انداز[۲۳۶] را در زمان مناسب فعال خواهد ساخت.

در اغلب محیط های کسب و کار گرا، لایه داده بصورت پایگاه داده ای رابطه ای سازمان داده می شود. در اینجا استقلال داده اهمیت کلیدی پیدا خواهد کرد. داده ها ، مستقل از برنامه های کاربردی و به صورتی سازمان داده می شوند که نه تغییرات آن سازمان تأثیری بر برنامه های کاربردی داشته و نه برنامه های کاربردی تأثیری بر سازمان داده ای باقی بگذارند. از آنجایی که لایه پردازش و لایه داده مستقل از یکدیگر تلقی می شوند، بنابراین استفاده از پایگاههای داده ای رابطه ای در مدل مشتری- سرور به ما در تفکیک ایندو لایه بسیار کمک خواهد کرد.

با این وجود، پایگاههای داده ای رابطه ای همواره بهترین گزینه نیستند. یکی از ویژگیهای بارز بسیاری از برنامه های کاربردی این است که روی انواع داده های پیچیده عمل کرده و بنابراین مدلسازی آنها براساس شیء بسیار آسانتر از مدلسازی براساس روابط خواهد بود . از انواع این داده ها می توان به چندضلعی ها و دوایر ساده تا نمایش طرحهای هواپیما اشاره کرد که در سیتمهای طراحی بکمک کامپیوتر (CAD) استفاده می شود.

در مواردی که بیان عملیات روی داده از طریق دستکاری شیء راحت تر است، توصیه می شود که لایه داده با استفاده از پایگاه داده ای شیء گرا یا شیء- رابطه پیاده سازی شود. لازم به ذکر است که پایگاههای شیء- رابطه رواج و کاربرد گسترده ای پیدا کرده است، چون این پایگاههای داده ای علاوه بر ارائه مزایای مدل شیء گرا، براساس مدل داده رابطه ای بسیار پراکنده هم استوار هستند.

معماری های چندطبقه ای[۲۳۷]

ایده تقسیم بندی به سه سطح منطقی که تا به اینجا مورد بحث قرار گرفت، امکاناتی را برای توزیع فیزیکی برنامه های کاربردی مشتری- سرور در میان چند ماشین مختلف مطرح می کند. در ساده ترین سازمان ، ممکن است فقط با دو نوع ماشین سروکار باشیم:

۱. ماشین مشتری که فقط شامل برنامه های اجراکننده (بخشی از) لایه رابط/ کاربر است

[234] Consistency
[235] Trigger
[236] Handler
[237] Multitiered

۲. و ماشین سرور که شامل مابقی برنامه ها ، یعنی برنامه های اجراکننده لایه داده و پردازش است.

در این سازمان ، همه امور توسط سرور هدایت شده و مشتری اساساً چیزی بیش از یک ترمینال غیرهوشمند و احتمالاً دارای یک رابط گرافیکی زیبا نخواهد بود. احتمالات دیگری هم وجود دارند که برخی از معمول ترین آنها رادر ادامه همین بخش مورد بحث قرار خواهیم داد.

یک رویکرد برای سازماندهی مشتری و سرور، توزیع برنامه های لایه های برنامه کاربردی مشروح در بخش قبل در میان ماشینهای مختلف مطابق شکل ۵-۲ است[همچنین مراجعه شود به (۱۹۹۷) Umar؛(۱۹۹۹) , Jingو گروه همکاران] . بعنوان اولین گام، بین فقط دو نوع ماشین تفاوت قائل می شویم: یکی ماشینهای سرور و دیگری ماشینهای مشتری که باعث ایجاد چیزی به نام **معماری دوطبقه ای**[۲۳۸]**(فیزیکی)** میشود:

شکل ۵-۲- انواع سازمانهای سرور- مشتری الف) تا ه)(داخل شکل:الف)رابط کاربر، برنامه کاربردی، پایگاه داده، ب)رابط کاربر، برنامه کاربردی، پایگاه داده، ج) ماشین مشتری، رابط کاربر، برنامه کاربردی، پایگاه داده، د) رابط کاربر، برنامه کاربردی، پایگاه داده، ه) رابط کاربر، برنامه کاربردی، پایگاه داده، پایگاه داده).

یکی از روشهای سازماندهی آن است که مطابق شکل الف) ۵-۲، فقط بخش وابسته به ترمینال رابط مشتری روی ماشین مشتری قرار داشته باشد و کنترل برنامه های کاربردی برای ارائه داده های خود از دور انجام شود . یک روش دیگر، قرار دادن تمامی نرم افزار رابط کاربر در طرف مشتری مطابق شکل ب) ۵-۲ است. در چنین مواردی، اساساً برنامه کاربردی به نرم افزار پیش-انتهایی گرافیکی تقسیم می کنیم که از طریق یک پروتکل ویژه برنامه کاربردی، با مابقی برنامه کاربردی (واقع در سرور) ارتباط برقرار می کند.در این مدل، نرم افزار پیش-انتهایی (نرم افزار مشتری) هیچ پردازشی بجز آنچه برای ارائه رابط کاربر لازم است انجام نمی دهد.

در ادامه همین مسیر منطقی می توانیم بخشی از برنامه کاربردی را هم مطابق شکل ج) ۵-۲ به نرم افزار پیش-انتهایی انتقال دهیم. یکی از نمونه های کاربرد این روش هنگامی است که در برنامه کاربردی از فرمی استفاده شود که بایستی پیش از پردازش بطور کامل پرشود. پس نرم افزار پیش-انتهایی می تواند صحت و انسجام فرم را بررسی کرده و در صورت لزوم ، با کاربر تعامل نماید. مثال دیگر از سازمان شکل ج) ۵-۲ ،پردازشگر کلمه ای است که در آن کارکردهای اصلی ویرایشی در طرف مشتری اجرا می شود. این کارکردها روی داده های داخل حافظه نهان محلی یا در حافظه داده پیاده سازی می شوند، در حالی که ابزارهای پشتیبانی پیشرفته ای از قبیل چک کردن املاء و گرامر در طرف سرور اجرا می شوند.

[238] Two-tiered

در بسیاری از محیط های مشتری- سرور، سازمان های نشان داده شده در شکل د) ۵-۲ و شکل ه) ۵-۲ کاربرد زیاد دارند. از این سازمان ها در مواردی استفاده می شوند که ماشین مشتری ،کامپیوتر شخصی یا ایستگاه کاری است که از طریق شبکه به یک سیستم فایل توزیعی یا پایگاه داده ای متصل می باشد .در اینصورت بخش اعظم برنامه کاربردی روی ماشین مشتری اجرا می شود ،اما تمامی عملیات روی فایلها و ورودی های پایگاه داده ای به سرور محول می شود. بعنوان مثال، بسیاری از برنامه های کاربردی بانکداری روی ماشین کاربر انتهایی اجرا می شود که در آن کاربر ، نقل و انتقالات و اموری از این دست را آماده می کند. پس از پایان کار، برنامه کاربردی با پایگاه داده واقع در سرور بانک ارتباط برقرار کرده و نقل و انتقالات را جهت پردازش بیشتر به آن بالاگذاری[۲۳۹] می کند. شکل ۵-۲ حالتی را نشان می دهد که در آن دیسک محلی مشتری حاوی بخشی از داده هاست. در اینصورت مشتری می تواند مثلاً در حین جستجو در وب، بتدریج حافظه نهان بزرگی از آخرین صفحات وب مشاهده شده را بصورت محلی روی دیسک ایجاد نماید.

اشاره می کنیم که طی چندسال اخیر، تمایل شدیدی نسبت به دوری از پیکربندی های شکل د) ۵-۲و ه) ۵-۲ برای مواردی بوجود آمده که نرم افزار مشتری در ماشین های کاربر انتهایی قرار داشته باشد. در این موارد، عمده کار ذخیره سازی و پردازش داده در طرف مشتری انجام می شود. دلیل امر ساده است: هرچند ماشین های مشتری کار زیادی انجام می دهند، ولی مدیریت آنها مشکل زا است. افزایش کارکردی ماشین مشتری باعث افزایش احتمال خطای نرم افزار طرف مشتری و همچنین وابستگی آن به بستر زیربنایی مشتری (یعنی منابع و سیستم عامل) می شود. از نقطه نظر مدیریت سیستم ، داشتن اصطلاحاً **مشتریان فربه**[۲۴۰] بهترین گزینه نیست. بلکه بهتر است از **مشتریان لاغر**[۲۴۱] سازمان های شکل های الف) تا ج) ۵-۲ استفاده شود. البته هزینه این کار احتمالاً استفاده از واسط های کاربر قدیمی تر و کاهش کارآیی از دیدگاه مشتری خواهد بود .

توجه داشته باشید که این جهت گیری به این معنا نیست که دیگر نیازی به سیستم های توزیعی نخواهیم داشت. بلکه برعکس،راه حل های طرف سرور روز به روز توزیع یافته تر می شوند، زیرا بجای یک سرور واحد از چندین سرور استفاده می شود که روی ماشین های مختلف اجرا می شوند. بخصوص، در هنگام تمایز بین فقط ماشینهای سرور و مشتری که تا به اینجا انجام شد ، این نکته را از قلم انداختیم که ممکن است در بعضی موارد لازم باشد تا مطابق شکل ۶-۲، سرور بعنوان مشتری ایفای نقش نموده و **معماری سه طبقه**[۲۴۲] **(فیزیکی)** را بوجود آورد.

---

[239] Upload
[240] Fat clients
[241] Thin clients
[242] Three-tiered

شکل ۶-۲- نمونه ای از مواردیکه سرور بعنوان مشتری عمل می کند(داخل شکل: رابط کاربر(ارائه)، انتظار برای بازگشت، عملیات درخواست، بازگشت نتیجه، انتظار برای داده، سروربرنامه کاربردی، درخواست داده، بازگشت داده، سرور پایگاه داده، زمان).

در این معماری، برنامه هایی که بخشی از سطح پردازش را تشکیل می دهند ممکن است روی یک سرور جداگانه قرار داشته باشند، اما تاحدودی در ماشینهای مشتری و سرور هم توزیع شده باشند. یکی از نمونه های استفاده از معماری سه طبقه در پردازش تراکنش است. همانطور که در فصل ۱ هم گفتیم، فرآیندی جداگانه به نام ناظر پردازش تمامی تراکنش ها بین سرورهای داده احتمالاً مختلف را هماهنگ خواهد کرد.

نمونه دیگر اما بسیار متفاوت که غالباً در معماری های سه طبقه شاهد هستیم ، سازمان وب سایت ها است. در این حالت، یک سرور وب بعنوان نقطه ورودی به سایت عمل کرده و جهت پردازش واقعی درخواستها ،آنها را به سرور برنامه کاربردی عبور می دهد. این سرور برنامه کاربردی هم متقابلاً با سرور پایگاه داده ای تعامل دارد. بعنوان مثال، یک سرور برنامه کاربردی ممکن است مسئول اجرای کد جهت نظارت بر لیست برخی کالاهای موجود در یک مغازه لوازم الکترونیک باشد. بدین منظور، بایستی با پایگاه داده حاوی داده های خام موجودی تعامل نماید. در فصل ۱۲ مجدداً به موضوع سازمان وب سایت ها باز خواهیم گشت.

۲-۲-۲- معماری های غیرمتمرکز

معماری های سرور- مشتری چند طبقه ای یکی از پیامدهای مستقیم تقسیم برنامه های کاربردی به رابط/کاربر، مؤلفه های پردازشی و یک لایه داده ای هستند. طبقات مختلف مستقیماً با سازمان منطقی برنامه های کاربردی مربوط هستند. در بسیاری از محیط های کسب و کاری، پردازش توزیعی بمعنای سازماندهی یک برنامه کاربردی سرور- مشتری بصورت معماری چندطبقه است. ما از این نوع توزیع بنام **توزیع عمودی**[۲۴۳] یاد می کنیم . ویژگی بارز توزیع عمودی آن است که با قرارگیری مؤلفه های *منطقاً* متفاوت روی ماشینهای مختلف بوجود می آید. این اصطلاح با مفهوم تکه کردن عمودی[۲۴۴] ارتباط دارد که در پایگاه های داده رابطه ای توزیعی بکار برده شده و به این معناست که جداول به صورت ستونی تفکیک و سپس، در میان ماشینهای مختلف توزیع می شوند(Oszu and Valduriez، ۱۹۹۹).

مجدداً باید اشاره کرد که از نقطه نظر مدیریت سیستم ، توزیع عمودی می تواند باعث تقسیم شدن منطقی و فیزیکی کارکردها در بین ماشینهای متعددی شود که هریک برای گروه خاصی از کارکردها سفارشی شده است . با این وجود، توزیع عمودی فقط یکی از روشهای سازماندهی برنامه های کاربردی مشتری- سرور است. در معماری های مدرن ، غالباً تکیه برروی توزیع مشتری ها و سرورها قرار می

[243] Vertical distribution
[244] Vertical fragmentation

گیرد که از آن بانام **توزیع افقی**[۲۴۵] یاد می شود. در این نوع توزیع، یک مشتری یا سرور بایستی به بخشهای منطقاً برابر تقسیم شود؛ هر بخش روی سهم و مجموعه کامل داده های خود عمل می کند، و به این ترتیب بار متعادل خواهد شد. در این بخش، نگاهی خواهیم داشت به گروهی از معماری های نوین سیستم به نام **سیستم های نظیر به نظیر**[۲۴۶] که از توزیع افقی پشتیبانی می کنند.

از دیدگاه سطح بالا، فرآیندهای تشکیل دهنده سیستم های نظیربه نظیر همگی برابر هستند. به این معنا که کارکردهایی که باید اجرا شوند، بوسیله هریک از فرآیندهای تشکیل دهنده سیستم های توزیعی ارائه می شوند. در نتیجه، قسمت اعظم تبادل بین فرآیندها متقارن بوده و هر فرآیند بطور همزمان هم بعنوان مشتری و هم سرور عمل خواهد کرد (و بنابراین اصطلاحاً به نام **پیشخدمت**[۲۴۷] هم نامیده می شود).

براساس این رفتار متقارن ، مسأله مهم در معماری های نظیر به نظیر نحوه سازماندهی فرآیندها در یک **شبکه فوقانی**[۲۴۸] است.و اما شبکه فوقانی شبکه ای است که در آن گره ها بوسیله فرآیندها شکل گرفته و پیوندها بیانگر کانال های ارتباطی ممکن هستند (که معمولاً به توسط اتصالات TCP انجام می شوند). بطور کلی، هیچ فرآیندی نمی تواند مستقیماً با فرآیند اختیاری دیگر ارتباط برقرار کند، بلکه پیامها باید از طریق کانال های ارتباطی موجود ارسال شوند. دو نوع شبکه فوقانی وجود دارد: شبکه های دارای ساختار و شبکه های فاقد ساختار. این دو نوع بطور مفصل در مرجع (۲۰۰۵) Lau و گروه همکاران همراه با ذکر مثالهای متعدد مربوطه مورد بررسی قرار گرفته اند. کتاب (۲۰۰۵)Aberer وگروه همکاران مرجع خوبی برای انواع معماری هاست و امکان مقایسه رسمی تر بین انواع مختلف سیستم های نظیر به نظیر را فراهم می آورد. بررسی بر اساس دیدگاه توزیع محتوی را در(۲۰۰۴) Androutsellis-Theotokis and Spinellis مطالعه کنید.

معماری های نظیربه نظیر دارای ساختار

در معماری های نظیر به نظیر دارای ساختار، شبکه فوقانی با استفاده از یک روال مشخص ایجاد می شوند. پرکاربردترین روال تا به امروز سازماندهی فرآیندها با استفاده از **جدول توزیعی درهم**[۲۴۹](**DHT**) بوده است. درسیستم های براساس جدول توزیعی درهم، به هریک از آیتم های داده ای کلید تصادفی از یک فضای بزرگ شناسه ای از قبیل یک شناسه ۱۲۰ یا ۱۶۰ بیتی تخصیص داده می شود. بعلاوه ، به گره های داخل سیستم هم یک عدد تصادفی از همان فضای شناسه داده می شود. بنابراین مسئله اصلی سیستم های براساس DHT ، اجرای طرحی مشخص و مؤثر است که بتواند بکمک یک مقیاس سنجش فاصله، شناسه گره منحصربه فردی را به کلید یک آیتم داده ای مفروض،

[245] Horizontal distribution
[246] Peer-to-peer systems
[247] Servent
[248] Overlay network
[249] Distributed hash table

نگاشت نماید (Balakrishnan،۲۰۰۳). مهم ترین مسأله این است که در هنگام جستجوی یک آیتم داده ای ، آدرس شبکه ای گره مسئول آن آیتم داده ای بازگردانده خواهد شد. در عمل، اینکار توسط *مسیریابی* درخواست آن آیتم داده ای به گره مربوطه انجام خواهد شد.

بعنوان مثال، در سیستم Chord(۲۰۰۳، Stocia و گروه همکاران) گره ها منطقاً بصورت یک حلقه سازماندهی شده و آیتم داده ای با کلید k به گره ای با کوچکترین شناسه $id \leq k$ نگاشت می شود. این گره را اصطلاحاً *جانشین*[۲۵۰] کلیدk خوانده و مطابق شکل ۷-۲، بصورت $succ(k)$ نمایش می دهیم. بنابراین برای جستجوی واقعی یک آیتم داده ای مفروض، لازم است که برنامه کاربردی درحال اجرا برروی گره اختیاری ،تابع LOOKUP(k) را فراخوانی نماید. به این ترتیب، آدرس شبکه ای $succ(k)$ بازگردانده خواهد شد. در اینجا، برنامه کاربردی می تواند ضمن تماس با گره، کپی از آیتم داده ای مورد نظر را به دست آورد.

شکل ۷-۲- نگاشت آیتم داده ای روی گره هایChord (داخل شکل: گره واقعی، کلیدهای داده مربوطه).

درحال حاضر برای جستجوی کلید به سراغ الگوریتم های مربوطه نمی رویم و بحث درمورد آنها را به فصل ۵ وجزئیات سیستم های نامگذاری مختلف موکول می کنیم. در عوض، نگاهی خواهیم داشت به نحوه سازماندهی گره ها در یک شبکه فوقانی و یا به بیان دیگر، **مدیریت عضویت**[۲۵۱]. در ادامه بحث باید بدانیم که جستجوی یک کلید مفروض از سازمان منطقی گره ها در حلقه شکل ۷-۲ تبعیت نمی کند. بلکه هر گره میانبرهایی به دیگر گره ها به نحوی ایجاد می کند که بتواند این دست جستجوها را طی $O(log(N))$ مرحله انجام دهد- N درواقع تعداد گره های مشارکت کننده در فوقانی است.

حال مجدداً مثال Chord را درنظر می گیریم. هنگامی که گرهی بخواهد به سیستم ملحق شود،کار را با تولید شناسه تصادفی id آغاز خواهد کرد. توجه داشته باشید که مشروط بر بزرگ بودن کافی فضای شناسه، چنانچه تولیدکننده عدد تصادفی از کیفیت خوبی برخوردار باشد، احتمال تولید شناسه ای که درهمان زمان به یک گره واقعی تخصیص داده شده باشد، نزدیک به صفر خواهد بود. سپس ، گره می تواند به راحتی جستجو را روی id انجام دهدکه اینکار باعث بازگشت آدرس شبکه ای $succ(id)$ خواهد شد. در اینجا، گره الحاقی می تواند به راحتی با $succ(id)$ و گره قبل از آن تماس برقرار و خود را در حلقه درج کند. البته، این طرح مستلزم آن است که هرگره اطلاعات را روی گره قبل از خود هم ذخیره کند. درج کردن همچنین باعث خواهد شد تا هر آیتم داده ای که کلید آن با گره (id) مربوط است ، از $succ(id)$ انتقال داده می شود.

خروج از حلقه هم بسیار ساده است : گره (id) جانشین و گره قبل از خود را از ترک حلقه آگاه کرده و آیتمهای داده ای خود را به $succ(id)$ انتقال می دهد.

[250] Successor
[251] Membership Management

در دیگر سیستم های براساس DHT هم از روشهایی مشابه استفاده می شود. بعنوان مثال، **شبکه آدرس دهی محتوا[۲۵۲](CAN)** را در نظر بگیرید که در مرجع (۲۰۰۱) Ratnasamy و گروه همکاران شرح داده شده است. در CAN از فضای d بعدی مختصات دکارتی استفاده می شود که بطور کامل در بین تمامی گره های مشارکت کننده در آن سیستم تقسیم بندی شده است . برای ایجاد شکل ذهنی از اینمورد، اجازه دهید فقط حالت دو بعدی را درنظر بگیریم که نمونه ای از آن در شکل ۸-۲ ارائه شده است.

شکل ۸-۲- الف) نگاشت آیتم های داده ای روی گره ها در CAN.ب) تقسیم منطقه در هنگام الحاق یک گره.

شکلالف) ۸-۲، نحوه تقسیم فضای دوبعدی [۰،۱]×[۰،۱] دربین شش گره را نمایش می دهد. هر گره دارای یک منطقه مربوطه بوده و به هریک از آیتمهای داده ای موجود در CAN نقطه منحصر به فردی در این فضا داده می شود. پس از این دیگر مشخص است که کدام گره مسئول آن داده می باشد (آیتم های داده ای واقع روی مرز چند منطقه در نظر نمی گیریم. در این حالت از قانون تخصیص مشخص استفاده می شود).

اگر گره P بخواهد به سیستم CAN ملحق شود، نقطه اختیاری را از فضای مختصاتی انتخاب کرده و سپس گره Q را جستجو خواهد کرد که این نطقه در آن واقع می شود . این جستجو از طریق مسیریابی براساس موقعیت انجام خواهد شد که در پایان همین فصل مورد بحث قرار خواهد گرفت. سپس گره Q منطقه خود را به دونیم تقسیم کرده و همانطور که در شکل ب) ۸-۲ مشاهده می کنید، یک نیمه به گره P تخصیص می یابد. گره ها ،گزارشات همسایگان خود یعنی گره های مسئول منطقه مجاور را نگه می دارند. در هنگام تقسیم یک منطقه، گره الحاقی P می تواند صرفاً با سؤال از گرهQ براحتی همسایگان جدید را شناسایی کند. مشابه Chord، در اینجا هم آیتمهای داده ای که گره P اکنون مسئول آنهاست، از گره Q انتقال داده می شوند.

مسأله خروج در CAN تا حدودی مشکلتر است. فرض کنید که در شکل ۸-۲ ، گره دارای مختصات (۰،۷،۰،۶) قصد خروج داشته باشد. منطقه آن به یکی از همسایگانش مثلاً گره واقع در (۰،۹،۰،۹) تخصیص خواهد یافت ،اما مشخص است که صرف ادغام گره و ساخت یک مستطیل کافی نخواهد بود. در این حالت، گره واقع در(۰،۹،۰،۹) فقط مراقب آن منطقه بوده و همسایگان قدیمی را از این خبر جدید مطلع خواهد کرد. اینکار باعث کاهش تقارن در فضای مختصات خواهد شد. به همین دلیل یک فرآیند پس زمینه ای متناوباً اقدام به تقسیم بندی مجدد کل فضا خواهد نمود.

معماری های نظیر به نظیر بدون ساختار

[252] Content addressable network

در سیستم های نظیر به نظیر بدون ساختار برای ساخت شبکه فوقانی تا حد زیادی برروی الگوریتم های تصادفی تکیه می شود. ایده اصلی این است که هرگره لیستی از همسایه ها را در اختیار داشته باشد؛ اما این لیست باید بصورت کمابیش تصادفی ایجاد شود. بعلاوه فرض می شود که اقلام داده ای به صورت تصادفی روی گره ها قرار می گیرند. در نتیجه، وقتی گرهی اقدام به تعیین مکان یک قلم داده ای خاص می نماید، تنها روش ممکن و مؤثر این است که شبکه را با سیل[۲۵۳] پرس و جو جستجو مواجه سازد (Risson and Moors ،۲۰۰۶). در فصل ۵ مجدداً به موضوع جستجو در شبکه های فوقانی بدون ساختار باز خواهیم گشت و اکنون روی موضوع مدیریت عضویت تمرکز خواهیم کرد.

یکی از اهداف بسیاری ازسیستم های نظیر به نظیر بدون ساختار، ساخت شبکه فوقانی نظیر **گراف تصادفی**[۲۵۴] است. مدل اصلی این است که هر گره حاوی لیستی از c همسایگان بوده و در حالت ایده آل، هر همسایه نماینده یک گره *زنده* تصادفاً انتخاب شده از مجموعه فعلی گره ها باشد. لیست همسایگان بنام **نمای بخشی**[۲۵۵] خوانده می شود. راه های مختلفی برای ساخت نمای بخشی وجود دارد .در مرجع (۲۰۰۵،۲۰۰۴) Jelasity وگروه همکاران چارچوبی برای در بر گیری الگوریتم های مختلف برای ساخت شبکه فوقانی ارائه داده اند تا امکان ارزیابی و مقایسه هریک فراهم آید. در این چارچوب، فرض می شود که گره ها بصورتی منظم مدخل ها را ازنمای بخشی خود تبادل می نمایند. هریک از مدخل ها گره دیگری را در شبکه شناسایی نموده و برای بیان قدمت آن گره در منبع، از عددی بعنوان سن استفاده می شود . مطابق شکل ۹-۲ ، در این رابطه از دو نخ[۲۵۶] استفاده می شود.

شکل ۹-۲- الف) مراحل انجام شده بوسیله نخ فعال.ب) مراحل انجام شده بوسیله نخ غیرفعال

نخ فعال برای اولین بار اقدام به برقراری ارتباط با گره دیگر می کند. این گره از نمای بخشی فعلی آن انتخاب می شود. با این فرض که ورودی ها باید به نظیر انتخابی *هل*[۲۵۷] *داده شوند*، کار را با ایجاد یک بافر حاوی c/2+1 مدخل از جمله ورودی شناسنده خود ادامه خواهد داد. مابقی مدخل ها از نمای بخشی فعلی گرفته می شوند.

چنانچه گره در حالت *کشیدن*[۲۵۸] باشد، منتظر پاسخی از جانب نظیر انتخابی باقی خواهد ماند. در این فاصله زمانی، ما هم بوسیله نخ غیرفعال شکل ب) ۹-۲ ، یک بافر ساخته ایم. فعالیت های این نخ بسیار شبیه نخ فعال نظیر خود است.

[253] Flood
[254] Random Graph
[255] Partial view
[256] Thread
[257] Push
[258] Pull

نکته مهم ،ساخت نمای بخشی جدید است. این نما برای هر دو نظیر تماس گرفته شده و آغازین دقیقاً حاوی c مدخل خواهد بود و بخشی از آن از بافر دریافتی بدست خواهد آمد. بطور خلاصه، دو روش برای ساخت نمای بخشی وجود دارد . در حالت اول، دو گره ممکن است اقدام به دورریختن مدخل هایی کنند که قبلاً برای هم ارسال کرده بودند . اینکار باعث خواهد شد تا بخشی از نماهای اصلی آنها با هم *مبادله شود*. روش دوم دورریختن حداکثری مدخل های *قدیمی* است. در مجموع باید گفت که ایندو روش در واقع مکمل هم هستند[برای مطالعه جزئیات مراجعه شود به(۲۰۰۵) Jelasity وگروه همکاران]. می توان نتیجه گرفت که بسیاری از پروتکل های مدیریت عضویت برای شبکه های فوقانی بدون ساختار از این چارچوب تبعیت می کنند. چندین مشاهده جالب وجود دارد.

اولاً ، فرض کنیم که وقتی گرهی تصمیم به الحاق می گیرد، با گره اختیاری دیگر که احتمالاً ازروی لیستی از نقاط دسترسی معروف انتخاب خواهد شد، تماس برقرار می نماید . این نقطه دسترسی صرفاً عضو دائمی از شبکه فوقانی است و همچنین بسیار دسترس پذیر می باشد. در این حالت می توان نتیجه گرفت که پروتکل هایی که فقط از *روش هل دادن* یا فقط از *روش کشیدن* استفاده می کنند، ممکن است براحتی شبکه های فوقانی غیرمتصل ایجاد کنند . به بیان دیگر، گروه های گره ها ایزوله شده و دیگر توان دسترسی به همه گره های موجود در شبکه را نخواهند داشت. ناگفته پیداست که این ویژگی مطلوب نبوده و به همین دلیل معقولانه تر این است که گره ها بتوانند واقعاً مدخل ها را با هم *تبادل* نمایند.

ثانیاً، بشرطی که گره ها بصورتی منظم اقدام به تبادل نماهای بخشی نمایند، عملیات خروج از شبکه بسیار آسان خواهد شد. در این حالت، هر گره می تواند به راحتی و بدون اطلاع دادن به گره های دیگر خارج شود. در واقع ، هنگامیکه گره مفروض P یکی از همسایگان ظاهری خود مثلاً Q را انتخاب می کند و می فهمد که Q دیگر پاسخ نخواهد داد، مدخل را از نمای بخشی خود حذف و نظیر دیگری را انتخاب خواهد کرد. بنابراین گره در هنگام ساخت نمای بخشی جدید، از سیاست دورریزی حداکثر ممکنه مدخل ها تبعیت خواهد کرد و گره های خارج شده به سرعت فراموش می شوند. به بیان دیگر، مدخل های ارجاع شده به گره های خروجی بصورت اتوماتیک و سریعاً از نماهای بخشی حذف خواهند شد.

بااین وجود، تبعیت از این راهکار هزینه هایی هم در پی دارد. به ازای گره P ، مجموعه ای از گره ها را فرض کنید که در نمای بخشی خود مدخلی دارد که به P ارجاع می کند. این ویژگی از نظر فنی **درجه مدخلیت**[۲۵۹] گره نامیده می شود. با افزایش زاویه درونی گره P ، احتمال برقراری تماس توسط گره های دیگر با گره P هم افزایش خواهد یافت . به بیان دیگر، این خطر وجود دارد که P تبدیل به گره مورد علاقه شده و بارکاری زیاد وضعیت نامتعادلی را در گره ایجاد کند. دورریختن مدخل های قدیمی

[259] Indegree

به طور سیستماتیک باعث ترغیب گره ها به گره های دارای درجه مدخلیت بیشتر می شود. انواع دیگری هم از مقایسه ها وجود دارند که خوانندگان مشتاق را به مطالعه آنها در مرجع (۲۰۰۵) Jelasityو گروه همکاران دعوت می کنیم.

مدیریت توپولوژی شبکه های فوقانی

هرچند ممکن است به نظر برسد که سیستم های نظیر به نظیر دارای ساختار و بدون ساختار کاملاً از یکدیگر مستقل هستند، در عمل ضرورتاً اینطور نیست[مراجعه شود به (۲۰۰۵) Castroو گروه همکاران]. یکی از مهمترین مشاهدات این است که با تبادل و انتخاب آگاهانه ورودی ها از نماهای بخشی، امکان ساخت و نگهداری توپولوژی های مشخص شبکه های فوقانی فراهم می آید. مدیریت این توپولوژی با اتخاذ روش دولایه ای مطابق شکل ۱۰-۲ امکان پذیر می شود.

شکل ۱۰-۲- روش دولایه ای برای ساخت و نگهداری از توپولوژی های مشخص فوقانی با استفاده از روشهای سیستم های نظیر به نظیر بدون ساختار(داخل شکل: فوقانی دارای ساختار، پروتکل مربط به فوقانی مشخص، پیوند به دیگر گره های دارای توپولوژی مشخص، نظیر تصادفی، فوقانی تصادفی، پروتکل مربوط به نمای تصادفی، پیوند به دیگر گره های تصادفاً انتخاب شده).

پایین ترین لایه، سیستم نظیر به نظیر بدون ساختار را تشکیل می دهد که گره های موجود در آن به منظور کسب یک گراف تصادفی دقیق، متناوباً مدخل های نماهای بخشی خود را مبادله می کنند. در این مورد دقت به این مطلب اشاره دارد که نمای بخشی باید با مدخل های رجوع کننده به گره های *زنده* تصادفاً انتخاب شده پر شود.

پایین ترین لایه نمای بخشی خود را به لایه بالاتر عبور داده که در آنجا انتخاب دیگری روی مدخل ها انجام می شود. اینکار منجر به تهیه لیست ثانویه ای از همسایگان متناظر با توپولوژی مورد نظر خواهد شد.(۲۰۰۵) Jelasity and Babaoglu پیشنهاد داده اند که با استفاده از مفهومی به نام *تابع رتبه بندی*[۲۶۰]، گره ها برحسب ضابطه ای مرتبط با آن گره مفروض مرتب شوند . وظیفه یک تابع ساده رتبه بندی آن است که با افزایش فاصله از گره مفروض P مجموعه ای از گره ها را مرتب کند. در این حالت، چنانچه پایین ترین لایه همچنان به عبوردادن گره های تصادفاً انتخاب شده خود ادامه دهد،گره P تدریجاً لیستی از نزدیکترین همسایگان خود ایجاد خواهد کرد.

برای درک بهتر یک ساختار مشبک منطقی با ابعاد $N \times N$ را تصور کنید که در آن یک گره روی هریک از نقاط شبکه قرار گرفته باشد. هر گره باید حاوی لیستی ازتعداد c نزدیکترین همسایگان باشد. فاصله بین گره واقع در $(a_1 ، a_2)$ و $(b_1 ، b_2)$، به صورت $d_1 + d_2$ تعریف شده و درآن $|a_i - b_i|$

[260] Ranking function

$d = min(N- |a_i - b_i|, |a_i|)$. چنانچه پایین ترین لایه متناوباً و مطابق شکل ۹-۲ اقدام به اجرای پروتکل نماید، توپولوژی حاصله به یک توروس[۲۶۱] شبیه شکل ۱۱-۲ نزدیک خواهد شد.

شکل ۱۱-۲- تولید یک شبکه فوقانی مشخص با استفاده از یک سیستم نظیربه نظیر دولایه ای بدون ساختار[اقتباس از (۲۰۰۵) Jelasity and Babaoglu].

البته از توابع رتبه بندی کاملاً متفاوتی می توان استفاده کرد که در آن میان ، توابع مربوط به کسب **مجاورت معنایی**[۲۶۲] به اقلام داده ای که در گره نظیر ذخیره شده اند، از بقیه جالبتر است . این مجاورت، امکان ایجاد **شبکه های فوقانی معنایی** را فراهم می کند که امکان ساخت الگوریتم های جستجوی بسیار مؤثر در سیستم های نظیربه نظیر بدون ساختار را می دهد. در فصل ۵ در حین بحث درمورد نامگذاری براساس ویژگی مجدداً به این سیستم ها بازخواهیم گشت .

ابرنظیرها[۲۶۳]

قابل ذکر است که در سیستم های نظیر به نظیر بدون ساختار، با رشد شبکه، مکان یابی آیتمهای داده ای مرتبط هم مشکل تر خواهد شد. دلیل این مشکل در مقیاس پذیری آشکار است: چون هیچ روش کاملاً مشخص برای مسیریابی درخواست جستجو به یک قلم داده ای مفروض وجود ندارد و اساساً تنها روشی که یک گره مفروض می تواند بدان متمسک شود، ارسال سیل درخواست است. روشهای مختلفی برای مانع سازی در برابر این سیل وجود دارد که در فصل ۵ با آنها اشاره خواهیم کرد. اما روش جایگزین برای بسیاری از سیستم های نظیر به نظیر، استفاده از گره های ویژه ای است که ایندکسی از اقلام داده ای را در خود نگه دارد.

نمونه های دیگری از مواردی که بهتر است از ماهیت متقارن سیستم های نظیر به نظیر صرفنظر کرد وجود دارد. مثلاً تشریک مساعی گره هایی را در نظر بگیرید که منابعی را به یکدیگر ارائه می کنند. بعنوان مثال، در یک **شبکه تحویل محتوا**[۲۶۴] **(CDN)** با تشریک مساعی، ممکن است گره ها فضاهای ذخیره سازی برای کپی های صفحات وب ارائه کرده تا به این ترتیب، مشتریان وب بتوانند به صفحات دسترسی پیدا کرده و در نتیجه امکان دسترسی سریع برای آنها فراهم شود. در این حالت، ممکن است لازم باشد که گره مفروض P در بخش خاصی از شبکه به دنبال منابع مورد نظر خود بگردد. در این حالت، استفاده از واسطه[۲۶۵] ای برای جمع آوری کاربری منبع برای تعدادی از گره های واقع در مجاورت یکدیگر، امکان انتخاب سریع گره دارای منابع کافی را فراهم می آورد.

گره هایی از قبیل آنها که ایندکس را نگه می دارند یا بعنوان واسطه عمل می کنند، غالباً بنام **ابرنظیر** خوانده میشوند. همانطور که از نام آنها هم برمی آید، ابرنظیرها هم غالباً در شبکه های نظیر به نظیر

---

[261] Torus
[262] Semantic proximity
[263] Superpeers
[264] Content delivery network
[265] Broker

سازماندهی شده و منجر به تولید سازمان سلسله مراتبی مشابه چیزی می شوند که در مرجع (۲۰۰۳) Yang and Garcia- Molina تشریح شده است. یکی از انواع ساده این سازمانها در شکل ۱۲-۲ نمایش داده شده است. در این سازمان، هر یک از نظیرهای عادی بعنوان مشتری به یک ابرنظیر متصل شده است . تمامی ارتباطات از / به نظیر عادی از طریق ابرنظیر مربوط به همان نظیر انجام خواهد شد.

شکل۱۲-۲- سازمان سلسله مراتبی گره ها در یک شبکه ابرنظیر(داخل شکل: نظیر عادی، ابرنظیر، شبکه ابرنظیر).

در بسیاری از موارد، رابطه مشتری – ابرنظیر ثابت است؛ یعنی هروقت که یک نظیر عادی به شبکه ملحق می شود، به یکی از ابرنظیرها متصل شده و تا زمان خروج از شبکه این اتصال را حفظ خواهد کرد. قاعدتاً انتظار می رود که ابرنظیرها فرآیندهایی باعمر طولانی و دسترس پذیری بالا باشند. برای جبران رفتار بعضاً ناپایدار یک ابرنظیر، می توان از طرحهای پشتیبان از قبیل جفت کردن هر ابرنظیر با ابرنظیردیگر و الزام مشتریان برای اتصال به هردو استفاده نمود.

داشتن ارتباط ثابت با یک ابرنظیر ممکن است همواره بهترین راه حل نباشد. بعنوان مثال، در مورد شبکه های اشتراک فایل ، شاید گزینه بهتر برای مشتری اتصال به ابرنظیری است که حاوی ایندکس فایل هایی باشد که مشتری غالباً به آنها تمایل دارد. در این حالت، احتمال اطلاع ابرنظیر از محل فایل خاص مورد جستجوی مشتری بالا خواهد بود. Garbacki (۲۰۰۵)و گروه همکاران طرح نسبتاً ساده ای ارائه کرده اند که می توان درآن ،در صورتیکه مشتری ها ابرنظیرهای بهتری را برای مشارکت پیدا کنند، رابطه مشتری- ابرنظیر را تغییر داد. مشخصاً ابرنظیری که نتیجه عملیات جستجو را بازمی گرداند، نسبت به دیگر ابرنظیرها ارجحیت دارد.

همانطور که تا به اینجا دیدیم، شبکه های نظیر به نظیر ابزار انعطاف پذیری را برای پیوستن به شبکه یا خروج از شبکه برای گره ها ارائه می دهند. بااین وجود، شبکه های ابرنظیر مشکل تازه ای را مطرح می کنند که عبارتست از نحوه انتخاب گره هایی که شایسته تبدیل شدن به ابرنظیر هستند. این مسأله ارتباط نزدیکی با **مسأله انتخاب هماهنگ کننده** دارد که در فصل ۶ در حین بحث در مورد انتخاب ابرنظیرها در یک شبکه نظیر به نظیر راجع به آن صحبت خواهیم کرد.

۳-۲-۲- معماری های هیبریدی (پیوندی)

تا به اینجا روی معماری های مشتری- سرور و تعدادی از معماری های نظیر به نظیر تمرکز کرده ایم. در بسیاری از سیستم های توزیعی، ویژگی های معماری با هم ترکیب می شوند،همچنان که در شبکه های ابرنظیری مطرح کردیم. در این بخش نگاهی خواهیم داشت به برخی از انواع خاص سیستم های توزیعی که درآنها راه حلهای مشتری- سرور با معماری های غیرمتمرکز ترکیب شده است.

سیستم های سرورلبه ای[۲۶۶]

[266] Edge-server systems

یکی از مهمترین انواع سیستم های توزیعی که براساس معماری هیبریدی سازماندهی شده، **سیستم های سرورلبه ای** است. این سیستمها روی اینترنت در جاهایی پیاده شده اند که سرورها "در لبه شبکه" قرار می گیرند. این لبه براساس مرز بین شبکه های شرکتی و اینترنت واقعی تشکیل می شود که مثلاً بوسیله **فراهم کنندگان سرویس اینترنت**[۲۶۷] **(ISP)** ارائه می شود. به همین ترتیب، هنگامیکه کاربران نهایی خانگی از طریق ISP خود به اینترنت متصل می شوند، می توان تصور کرد که ISP در لبه اینترنت واقع شده و در نهایت سازمانی مشابه شکل ۱۳-۲ بوجود خواهد آمد.

شکل ۱۳-۲- نمای اینترنت بصورت مجموعه ای متشکل از سرورهای لبه ای (داخل شکل: مشتری، فراهم کننده محتوا، سرور لبه ای، هسته اینترنت، شبکه شرکتی).

کاربران نهایی ، یا بطور کلی مشتریان، بوسیله سرور لبه ای به اینترنت متصل می شوند. وظیفه اصلی سرور لبه ای خدمات رسانی به محتوا ، احتمالاً پس از اعمال کارکردهای فیلترینگ و تبدیل کد است. جالب تر اینکه از مجموعه ای از سرورهای لبه ای می توان برای بهینه سازی محتوا و توزیع برنامه کاربردی استفاده نمود. در مدل اصلی این گونه است که برای یک سازمان خاص یک سرور لبه ای می تواند به عنوان سرور مبدأعمل کند که تمام محتوا از آن نشأت می گیرد. این سرور هم می تواند از دیگر سرورهای لبه ای جهت کپی برداری از صفحات وب و نظایر آن استفاده کند(۲۰۰۲، Rabinovich and Spatscheck؛و ۲۰۰۴،Nayate و گروه همکاران؛۲۰۰۴، Leff وگروه همکاران). در فصل ۱۲ در مبحث مربوط به راه حلهای براساس وب مجدداً به این موضوع برخواهیم گشت.

سیستم های توزیعی با تشریک مساعی[۲۶۸]

ساختارهای هیبریدی علی الخصوص در سیستم های توزیعی با تشریک مساعی پیاده سازی شده اند. مسئله اصلی بسیاری از این سیستم ها اولین راه اندازی است که بدین منظور غالباً یک طرح مشتری-سرور متعارف اجرا شده است. پس از الحاق یک گره مفروض به سیستم، این گره می تواند جهت تشریک مساعی از طرحی کاملاً غیرمتمرکز استفاده کند.

برای درک موضوع ابتدا نگاهی خواهیم داشت به سیستم اشتراک فایل BitTorrent(۲۰۰۳، Cohen). BitTorrent یک سیستم دانلود فایل نظیربه نظیر است که عملکرد اصلی آن در شکل ۱۴-۲ ارائه شده است. ایده اصلی این است که وقتی کاربر نهایی به دنبال فایل خاصی می گردد، تکه هایی از فایل را آنقدر از کاربران دیگر دانلود کند که این تکه ها بتوانند با کنار هم قرار گرفتن، یک فایل کامل را ایجاد کنند. یکی از مهمترین اهداف طراحی این سیستم ها تضمین تشریک مساعی بوده است. دراغلب سیستم های اشتراک فایل، بخش اعظمی از شرکت کنندگان صرفاً فایلها را دانلود می کنند، و در بقیه موارد نقش آنها چیزی نزدیک به صفر است (۲۰۰۵، Yang وگروه همکاران؛و ۲۰۰۳،Saroiuو گروه همکاران؛و ۲۰۰۰،Adar and Huberman). بنابراین ، یک فایل را فقط هنگامی می توان دانلود

[267] Internet service provider
[268] Collaborative distributed systems

کرد که مشتری در حال دانلود، در حال تهیه محتوا برای فرد دیگری باشد. کمی بعدتر مجدداً به این رفتار" تلافی جویانه" بازخواهیم گشت.

شکل ۱۴-۲- اصل کلی نحوه عملکرد BitTorrent [اقتباس با کسب اجازه از (۲۰۰۴) Pouwelse وگروه همکاران](داخل شکل: تعداد kگره از Nگره،گره مشتری، جستجوی(F)، صفحه وب BitTorrent،سرور وب،ارجاع به سرور فایل ، فایل torrent. برای F،سرور فایل، ارجاع به ردیاب، لیست گره های ذخیره کننده F،ردیاب، گره۱،گره۲، گرهN).

جهت دانلود فایل ، کاربر باید به دایرکتوری جهانی دسترسی داشته باشد که فقط یکی از چندین وب سایت معدود شناخته شده و معروف است. چنین دایرکتوری حاوی ارجاعاتی به چیزی بنام فایلهای *torrent.* است. یک فایل *torrent.* حاوی اطلاعات لازم برای دانلود یک فایل خاص است. مشخصاً به چیزی به نام **ردیاب**[۲۶۹] اشاره دارد که در واقع سروری است برای نگهداری گزارش دقیقی از گره های *فعالی* که حاوی(تکه های) فایل درخواست شده باشند. گره فعال گرهی است که در حال دانلود کردن فایل دیگری باشد. ناگفته پیداست که ردیابهای مختلفی ممکن است وجود داشته باشند، اما غالباً فقط یک ردیاب برای هر فایل(یا مجموعه فایل ها) موجود است.

پس از آنکه گره ها از محل مناسب برای دانلود تکه ها اطلاع پیدا کردند، گره درحال دانلود فعال خواهد شد. در این صورت گره مجبور می شود به دیگر گره ها کمک کند، مثلاً با ارائه تکه هایی از فایلی که او در حال دانلود و دیگران فاقد آن هستند. این اجبار ناشی از یک قانون بسیار ساده است: چنانچه گره P مطلع شود که گره Q درحال دانلود کردن بیش از آن چیزی است که آپلود می کند، P ممکن است تصمیم بگیرد که از سرعت خود در ارسال داده ها به Q بکاهد. مادامی که P چیزی برای دانلود کردن از Q داشته باشد ، این طرح به خوبی عمل خواهد کرد . به همین دلیل، غالباً مرجع هایی به بسیاری از گره های دیگر در اختیار گره ها قرار می گیرد تا در وضعیت بهتری برای معامله قرار گیرند.

ناگفته پیداست که BitTorrent صورت ترکیبی از راه حل های متمرکز و غیرمتمرکز است. بنابراین نقطه ضعف این دست سیستمها در واقع همین ردیابها هستند.

بعنوان نمونه دیگر، شبکه توزیع محتوا با تشریک مساعی Globule(۲۰۰۶، Pierre and Van Steen) را درنظر بگیرید. معماری Globule تا حد زیادی مشابه معماری سرور لبه ای است که قبلاً بحث شد. دراین حالت، بجای سرور لبه ای، کاربران نهایی (و همچنین سازمانها) بطور داوطلبانه سرورهای وب تقویت شده ای را ایجاد می کنند که قادر به تشریک مساعی در کپی کردن صفحات وب هستند. در ساده ترین شکل، چنین سروری دارای مؤلفه های زیر است:

۱. مؤلفه ای که می تواند درخواستهای مشتری را به طرف سرورهای دیگر هدایت کند.
۲. مؤلفه ای برای آنالیزکردن الگوهای دسترسی.

[269] Tracker

۳. مؤلفه ای برای مدیریت کپی برداری از صفحات وب.

سرور ایجاد شده بوسیله کاربر مفروض آلیس سرور وبی است که معمولاً ترافیک مربوط به وب سایت آلیس را هدایت کرده و **سرور مبدأ** آن سایت نامیده می شود. این سرور با سرورهای دیگر، مثلاً سرور ایجادشده بوسیله باب تشریک مساعی کرده و میزبان صفحات سایت باب می شود. از این نقطه نظر، Globule یک سیستم توزیعی غیرمتمرکز محسوب می شود. درخواستهای مربوط به وب سایت آلیس درابتدا به سرور او ارسال می شوند و ممکن است از آنجا به یکی دیگر از سرورها هدایت شود. از هدایت مجدد توزیعی هم پشتیبانی می شود.

بااین وجود،Globule یک مؤلفه متمرکز دیگری هم به شکل **واسطه**[۲۷۰] دارد. این واسطه مسئول ثبت سرورها و شناساندن آنها به دیگر سرورها می باشد. نحوه ارتباط سرورها با واسطه کاملاً مشابه چیزی است که از سیستم های مشتری- سرور انتظار می رود. جهت ایجاد قابلیت دسترسی، واسطه را می توان کپی نمود ؛ اما همانطور که در ادامه همین کتاب خواهیم دید، این نوع کپی برداری به میزان گسترده ای جهت تضمین قابلیت اطمینان محاسبات سرور- مشتری مورد استفاده قرار می گیرد.

۳-۲- مقایسه معماری ها با میان افزار

با بررسی مسائل معماری که تا به اینجا مورد بحث قرار گرفت، این پرسش مطرح می شود که میان افزار در این بین چه جایگاهی دارد. همانطور که در فصل ۱ و شکل۱-۱ هم مشاهده کردیم، میان افزار لایه ای را بین برنامه های کاربردی و سکو های توزیع شده ایجاد می کند. یکی از مهم ترین اهداف اینکار، ایجاد درجه ای از شفافیت توزیعی است که بتوان بکمک آن توزیع داده، توزیع پردازش و توزیع کنترل را از برنامه های کاربردی مخفی کرد.

اما غالباً در عمل مشاهده می شود که سیستم های میان افزاری از شیوه معماری خاصی تبعیت می کنند. بعنوان مثال، بسیاری از راه حلهای میان افزاری از قبیل CORBA(OMG،۲۰۰۴) از شیوه معماری براساس شیء استفاده می کنند. برخی دیگر، ازجمله TIB/Rendezvous(TIBCO،۲۰۰۵) برای ایجاد میان افزار از شیوه معماری براساس رویداد بهره می گیرند. در فصل های بعد، به برخی از نمونه های شیوه های معماری اشاره خواهیم کرد.

مزیت میان افزارهایی که براساس شیوه معماری خاصی مدل شده اند ،این است که بعضاً باعث تسهیل طراحی برنامه های کاربردی می شوند . اما یکی از نقاط ضعف آشکار آن این است که میان افزار مذکور ممکن است برای برنامه کاربردی بهینه نباشد. بعنوان مثال، CORBA در ابتدا فقط اشیائی را ارائه می داد که توسط مشتری های از راه دور قابل فراخوانی بود . بعد ها احساس شد که استفاده صرف از این نوع تعامل بسیار محدود کننده خواهد بود و به همین دلیل ، دیگر الگوهای تعامل از قبیل پیا م دهی

[270] Broker

هم به آن افزوده شد. ناگفته پیداست که افزودن ویژگی های جدید می تواند باعث افزایش راه حل های میان افزاری شود.

بعلاوه، هرچند هدف از میان افزار ایجاد شفافیت توزیعی است، معمولاً تصور می شود که بهتر است برخی راه حل ها با شرایط برنامه های کاربردی سازگاری پیدا کنند. یک راه حل آن است که نسخه های متعددی از یک سیستم میان افزاری تهیه کرده و هر نسخه را برحسب گروه خاصی از برنامه های کاربردی سفارشی کنیم. روشی معمولاً مفیدتر این است که سیستم های میان افزاری را به نحوی ایجاد کنیم که برحسب برنامه کاربردی مورد نظر به راحتی قابل پیکربندی، سازگاری و تغییر باشند. در نتیجه، امروزه سیستم هایی در حال ایجاد هستند که در آنها تفکیک بیشتر و مشخص تری بین سیاست ها و مکانیزم ها منظور شده باشد. از همین رو، مکانیزم های متعددی جهت بهبود رفتار میان افزارها بوجود آمده اند (Sadjadi and McKinley ،۲۰۰۳). اجازه دهید در ادامه نگاهی داشته باشیم به برخی از متداول ترین روش ها در این زمینه.

۱-۳-۲- راه گیرها[۲۷۱]

از نظر مفهومی، **راه گیر** در واقع ساختار نرم افزاری است که جریان عادی کنترل راشکسته و به کد(ویژه برنامه کاربردی) دیگر اجازه اجرا می دهد. عمومی کردن راه گیرها مستلزم تلاش اجرایی زیادی است که توسط Schmidt (۲۰۰۰) وگروه همکاران) مورد بررسی قرار گرفته است . بطور قطع نمی توان گفت که در چنین مواردی ارجحیت با عمومیت است یا قابلیت اجرای محدود و ساده بودن. همچنین، در بسیاری از موارد داشتن امکانات راه گیری محدود باعث بهبود مدیریت نرم افزار و سیستم توزیعی خواهد شد.

برای استحکام مطلب، موضوع راه گیری را به صورتی در نظر بگیرید که در بسیاری از سیتمهای توزیعی براساس شیء مورد پشتیبانی قرار می گیرد. ایده اصلی ساده است: شیء A قادر به فراخوانی متدی متعلق به شیء B است، اما B روی ماشینی غیراز A قرار دارد. همانطور که در بخش های بعدی همین کتاب خواهیم گفت، این فراخوانی شیء از راه دور طی روش سه مرحله ای زیر انجام خواهد شد:

۱. رابط محلی[۲۷۲] دقیقاً مانند آنچه توسط شیءB ارائه می شود به شیء A ارائه می شود. A هم متد موجود در آن رابط را فراخوانی می کند.
۲. فراخوانی توسط A به فراخوانی شیء عمومی تبدیل می شود که ازطریق رابط فراخوانی شیء عمومی ارائه شده توسط میان افزار موجود در ماشین A، امکان پذیر می شود.
۳. نهایتاً، فراخوانی شیء عمومی تبدیل به پیامی می شود که توسط رابط شبکه ای سطح انتقال ارائه شده بوسیله سیستم عامل محلی A ارسال می شود.

شمای کلی از این مراحل در شکل ۱۵-۲ نمایش داده شده است.

[271] Interceptors
[272] Local interface

شکل ۱۵-۲- استفاده از راه گیرها جهت انجام فراخوانی های شیء از راه دور(داخل شکل: فراخوانی مسدود شده، راه گیر سطح درخواست، راه گیر سطح پیام، برنامه کاربردی مشتری، برنامه کاربردی مجازی، فراخوانی مسدود نشده، میان افزار شیء، سیستم عامل محلی، به سمت شیء B).

پس از مرحله اول، فراخوانی B.do_something(value) تبدیل به فراخوانی عمومی از قبیل invoke(B,&do_something,value) با ارجاع به متد B و پارامترهای همراه فراخوانی می شود . حال تصور کنید که شیء B کپی برداری شود. در اینصورت ، همه کپی ها باید فراخوانی شوند. رعایت این نکته مورد روشنی است که راه گیرها می توانند کمک کنند. کافی است **راه گیر سطح درخواست** اقدام به فراخوانی invoke(B,&do_something,value) برای هر یک از کپی ها می نماید. ظرافت کار در اینجاست که نه نیاز است که شیء A از کپی شدن B اطلاع داشته باشد و نه اینکه میان افزار شیء مؤلفه های ویژه ای برای کار با این فراخوانی کپی برداری شده در اختیار داشته باشد . فقط راه گیر سطح درخواست که ممکن است به میان افزار *اضافه شود*، باید از کپی شدن B مطلع باشد.

بعنوان آخرین نکته باید گفت که فراخوانی یک شیء از راه دور بایستی از طریق شبکه ارسال شود. در عمل، این ویژگی لزوم فراخوانی رابط پیام دهی را مطرح می کند که بوسیله سیستم عامل محلی ارائه می شود. در این سطح ، **راه گیر سطح پیام** ممکن است در انتقال فراخوانی به شیء هدف کمک کند. بعنوان مثال، تصور کنید که پارامتر value، واقعاً متناظر با آرایه بزرگی از داده باشد. در این حالت، عاقلانه تر این است که داده را به قطعات کوچکتری تقسیم نمود تا بعداً در مقصد مجدداً سرهم بندی شوند. این دست تجزیه کردنها ممکن است باعث ارتقاء میزان کارآیی یا قابلیت اطمینان شود. در این مورد هم لازم نیست که میان افزار از این تجزیه اطلاع پیدا کند؛ راه گیر سطح پایین تر به صورتی شفاف با همکاری سیستم عامل محلی اقدام به هدایت ارتباطات خواهد کرد.

۲-۳-۲- رویکردهای عمومی به نرم افزار تطبیقی[۲۷۳]

در واقع می توان گفت که راه گیرها ابزاری جهت تطبیق میان افزار ارائه می کنند. لزوم تطبیق ریشه در این واقعیت دارد که محیط اجرای برنامه های کاربردی توزیعی دائماً دستخوش تغییر می شود. از جمله این تغییرات می توان به جابجایی، تفاوت زیاد کیفیت خدمات شبکه ها، خرابی سخت افزار، تخلیه باتری و بسیاری دیگر اشاره کرد. وظیفه واکنش در برابر این تغییرات بجای برنامه های کاربردی، برعهده میان افزار قرار داده شده است.

این دست تأثیرات شدید بسیاری از طراحان نرم افزار را به فکر ساخت *نرم افزارتطبیقی* انداخته است. اما موفقیت نرم افزار تطبیقی هیچگاه به اندازه ای نبوده که از آن انتظار می رفت. از آنجایی که بسیاری از محققان و سازندگان نرم افزار آنرا یکی از مهم ترین جنبه های سیستم های توزیعی امروزی می دانند،

[273] Adaptive software

در ادامه نگاه مختصری به آن خواهیم داشت . در مرجع (۲۰۰۴) McKinley و گروه همکاران سه تکنیک اساسی در مورد تطبیق نرم افزار مطرح شده است.

۱. تفکیک موارد
۲. انعکاس محاسباتی[۲۷۴]
۳. طراحی براساس مؤلفه

تفکیک موارد با روش متعارف پیمانه ای ساختن سیستم ها مرتبط می باشد، یعنی تفکیک بخشهایی که کارکردها را پیاده سازی می کنند از سایر بخش های مسئول انجام امور دیگر (که به نام فوق کارکرد[۲۷۵] خوانده می شود) از قبیل قابلیت اطمینان، کارآیی، امنیت وغیره. می توان استدلال کرد که ایجاد میان افزار برای برنامه های کاربردی توزیعی تا حدود زیادی با کار با موضوع فوق کارکرد آن هم بصورتی مستقل از برنامه های کاربردی مرتبط است . مشکل اصلی این است که نمی توان از طریق پیمانه ای ساختن، این فوق کارکردها را از هم تفکیک نمود. بعنوان مثال، صرف قرار دادن امنیت در یک پیمانه جداگانه کار نخواهد کرد. به همین ترتیب، نمی توان تصور کرد که تحمل خرابی را در یک جعبه جداگانه قرار داد و بعنوان یک سرویس مستقل ارائه نمود. تفکیک و سپس به هم دوختن این موارد مرتبط در یک سیستم (توزیعی) مفروض از مهمترین نکاتی است که در **ایجاد نرم افزار جنبه گرا**[۲۷۶] (Filman،۲۰۰۵وگروه همکاران) مدنظر قرار می گیرد. با این وجود، جنبه گرایی هنوز بطور موفقیت آمیزی در توسعه سیستم های توزیعی پردامنه اعمال نشده است و بنابراین تا تحقق این امر هنوز راه زیادی در پیش داریم.

انعکاس محاسباتی اشاره دارد به توانایی یک برنامه مفروض در نظارت برخود و ،در صورت لزوم، انطباق رفتار خود(Kon،۲۰۰۲ و گروه همکاران). موضوع انعکاس در برخی از زبان های برنامه نویسی از جمله Java لحاظ شده و امکان گسترده ای را برای اصلاحات زمان اجرا در اختیار قرار می دهد. بعلاوه، برخی از سیستم های میان افزاری امکان اتخاذ روشهای انعکاسی را بوجود می آورند. با این وجود، درست مانند حالت جهت یابی جنبه، میان افزار انعکاسی باید بتواند خود را بعنوان یک ابزار قوی در مدیریت پیچیدگی سیستم های توزیعی پردامنه اثبات کند. همانطور که در مرجع (۲۰۰۴) Blair و گروه همکاران هم مطرح شده، اعمال انعکاس در دامنه گسترده برنامه های کاربردی هنوز بطور کامل محقق نشده است .

نهایتاً، طراحی براساس مؤلفه، از تطبیق به طریق ترکیب پشتیبانی می کند. هر سیستم مفروض را می توان یا به صورت ایستا در زمان طراحی پیکربندی کرد و یا به صورت پویا در زمان اجرا. مورد پویا نیازمند پشتیبانی برای پیوند دیرهنگام[۲۷۷] می باشد. این روش به صورت موفقیت آمیزی هم در محیط

[274] Computational reflection
[275] Extra functionality
[276] Aspect-oriented
[277] Late binding

های زبان برنامه نویسی اعمال شده و هم در سیستم های عاملی که بتوان در آنها پیمانه ها را به دلخواه بارکرد و تخلیه نمود. در حال حاضر، تحقیقات مفیدی در زمینه ایجاد امکان انتخاب اتوماتیک بهترین پیاده سازی مؤلفه در حین زمان اجرا در دست انجام است(Yellin،۲۰۰۳). اما در اینجا هم فرآیند مربوط به سیستم های توزیعی پیچیده است .این مشکل خصوصاً هنگامی رخ می نماید که بدانیم تعویض یک مؤلفه چه تأثیری بر دیگر مؤلفه ها خواهد داشت. در بسیاری از موارد، مؤلفه ها استقلال کمتری از آنچه تصور می شود دارند.

۳-۳-۲- بحث

معماری نرم افزارهای سیستم های توزیعی ، خصوصاً آنهایی که بعنوان میان افزار تلقی می شوند،حجیم و پیچیده است. در اکثرموارد، این حجم و پیچیدگی ناشی از لزوم عمومی بودن و آنهم ناشی از لزوم تأمین شفافیت توزیعی است . در عین حال، برنامه های کاربردی نیازمندی های فوق کارکردی ویژه ای دارند که با هدف دستیابی کامل به این شفافیت در تضاد است. این تضاد نیازمندی ها در عمومیت بخشیدن و ویژه گرایی منجر به ارائه راه حل های میان افزاری بسیار انعطاف پذیر شده است. بهای چنین دستاوردی پیچیده تر شدن است. به عنوان مثال، Zhang and Jacobsen(2004) گزارش داده اند که فقط طی چهار سال اول ، ابعاد یکی از محصولات نرم افزاری ۵۰٪ رشد داشته است.این در حالی است که تعدادکل فایلهای آن محصول طی همین دوره سه برابر شده اند. مشخص است که تشویقی برای ادامه این مسیر وجود ندارد.

باتوجه به اینکه تقریباً تمامی سیستم های بزرگ نرم افزاری امروزه بایستی در محیط شبکه ای شده اجرا شوند، باید از خود بپرسیم که آیا پیچیدگی سیستم های توزیعی صرفاً یک ویژگی ذاتی ناشی از تلاش برای ایجاد شفافیت توزیعی است. البته، مسائلی از قبیل باز بودن[۲۷۸] هم به این اندازه حائز اهمیت است ، اما لزوم انعطاف پذیری تاکنون هرگز به اندازه برای میان افزارها مطرح نبوده است.

Coyler (۲۰۰۳) و گروه همکاران استدلال می کنند که باید تأکید بیشتری روی سادگی، یعنی اتخاذ روشی ساده تر برای ساخت میان افزار بوسیله مؤلفه ها، و استقلال برنامه کاربردی صورت پذیرد. این که کدام یک از روشهای فوق الذکر راه حل مورد نظر است محل بحث است. بخصوص که هیچ یک از روش های پیشنهادی تا کنون نه مورد پذیرش گسترده واقع شده اند و نه بطور موفقیت آمیزی در سیستم های پردامنه مورد استفاده قرار گرفته اند.

فرضیه زیربنایی آن است که ما نیاز به *نرم افزارتطبیقی* داریم؛ به این معنا که نرم افزار بایستی همگام با تغییر محیط قادر به تغییر باشد. با این وجود، باید از خود پرسید که آیا تطبیق با تغییرات محیطی دلیل

[278] Openness

خوبی برای پذیرش تغییر نرم افزار خواهد بود. به نظر می رسد که سخت افزار معیوب، حملات امنیتی، تخلیه انرژی و مواردی از این دست همگی تغییرات محیطی هستند که می توانند(و بهتر است) بوسیله نرم افزار پیش بینی شوند.

مستدل ترین ، و قطعاً معتبرترین، بحث در حمایت از نرم افزار تطبیقی این است که امکان توقف فعالیت بسیاری از سیستم های توزیعی وجود ندارد. همین محدودیت لزوم ارائه راه حلهایی را برای تعویض و ارتقاء مؤلفه ها در زمان اجرا مطرح می کند. اما هنوز مشخص نیست که کدامیک از راه حلهای پیشنهادی فوق بهترین شیوه برای حل این دست مشکلات مربوط به نگهداری می باشد.

آنچه باقی می ماند این است که سیستم های توزیعی باید بتوانند، مثلاً با تغییر سیاستهای مربوط به تخصیص منابع ، در برابر تغییرات محیط از خود واکنش نشان دهند. تمامی مؤلفه های نرم افزاری لازم جهت ایجاد چنین تطبیقی هم اکنون موجود می باشد. در واقع این الگوریتم های واقع در مؤلفه ها هستند که نحوه رفتار برای تغییر مقادیر را دیکته می کنند. مهم این است که اجازه دهیم این رفتار واکنشی بدون دخالت انسانی انجام شود. مشاهده شده که این رویکرد در صورتی عملکرد بهتری خواهد داشت که مثلاً در حین تصمیم گیری درمورد محل قرار گرفتن مؤلفه ها ، راجع به سازمان فیزیکی آنها بحث می کنیم. راجع به این نوع مسائل معماری سیستم در ادامه بحث خواهیم کرد.

۴-۲- خود مدیریتی در سیستم های توزیعی

سیستم های توزیعی- و بخصوص میان افزار مربوط به آنها- بایستی راه حلهای عمومی برای مقاومت در برابر ویژگی های نامطلوب ذاتی شبکه سازی ارائه دهند به طوری که بتوانند ازتعداد زیادی برنامه های کاربردی پشتیبانی نمایند. از طرف دیگر، اغلب برنامه های کاربردی هم عملاً در پی کسب شفافیت توزیعی کامل نیستند و این امر باعث پشتیبانی از راه حل های ویژه برنامه کاربردی می شود. قبلاً استدلال کرده ایم که به همین دلیل، سیستم های توزیعی باید تطبیقی باشند ؛ اما این ضرورت هنگامی است که تطبیق رفتار اجرایی آنها مورد نظر است و نه مؤلفه های نرم افزاری تشکیل دهنده آنها.

هنگامی که ضرورت تطبیق اتوماتیک مطرح می شود ، نوعی کنش متقابل قوی بین معماری های سیستم و معماری های نرم افزار مشاهده می شود. از طرف دیگر، باید مؤلفه های سیستم توزیعی را به نحوی سازماندهی کرد که امکان نظارت و تنظیم فراهم آید. در عین حال، باید در مورد محل مناسب برای اجرای فرآیندهای مجری این انطباق هم تصمیم گیری شود.

در این بخش، نگاه دقیقتری خواهیم داشت به سازماندهی سیستم های توزیعی بصورت سیستم های سطح بالای کنترل بازخورد[۲۷۹] تا امکان تطبیق اتوماتیک با تغییرات فراهم شود. این پدیده به نام **محاسبه اتوماتیک** (Kephart،۲۰۰۳)یا **سیستم های خود-ستاره**[۲۸۰] (۲۰۰۵، Babaoglu وگروه همکاران) هم خوانده می شود. نامگذاری اخیر بیانگر تنوع روشهای کسب تطبیق اتوماتیک است، ازقبیل خود مدیریتی، خود ترمیمی، خود پیکربندی، خود بهینه سازی و غیره. در ادامه بحث از عنوان سیستم های مدیریتی برای اشاره به گونه های مختلف این سیستمها استفاده خواهیم کرد.

۱-۴-۲-مدل کنترل بازخورد

دیدگاههای مختلفی در مورد سیستم های خود مدیریتی وجود دارد، اما نقطه اشتراک (آشکار یا ضمنی) در اغلب آنها این است که تطبیق بوسیله یک یا چند **حلقه کنترل بازخورد** انجام می شود. برهمین اساس، سیستم هایی که بوسیله این نوع حلقه ها سازماندهی می شوند، به نام **سیستم های کنترل بازخورد** نامیده می شوند. سال هاست که کنترل بازخورد در حوزه های مختلف مهندسی مورد استفاده قرار گرفته و بنیانهای ریاضی آنهم تدریجاً در سیستم های محاسباتی نفوذ پیدا کرده اند (۲۰۰۵، Diao وگروه همکاران؛۲۰۰۴، Helleerstein وگروه همکاران). در سیستم های خود مدیریتی، مسائل معماری در نگاه اول جالبترین موضوع بنظر می رسند. ایده زیربنایی این سازمان که در شکل ۱۶-۲ مشاهده می کنید ، بسیار ساده است.

شکل ۱۶-۲- سازمان منطقی یک سیستم کنترل بازخورد(داخل شکل: پارامترهای غیرقابل کنترل(اختلال،نویز)،پیکربندی اولیه،اصلاحات، هسته سیستم توزیعی،خروجی مشاهده شده،اقدامات تنظیمی،ورودی مرجع، برآورد مقیاس، راه اندازهای تنظیم، آنالیز، خروجی اندازه گیری شده).

هسته سیستم کنترل بازخورد از مؤلفه هایی تشکیل می شود که نیازمند مدیریت هستند. فرض می شود که این مؤلفه ها از طریق پارامترهای ورودی قابل کنترل رانده می شوند، در حالیکه ممکن است رفتار آنها تحت تأثیر هرنوع ورودی *غیرقابل کنترل* - که به نام ورودی اختلال[۲۸۱] یا نویز هم خوانده می شود- قراربگیرد . هرچند اختلال غالباً ناشی از محیط اجرای سیستم توزیعی است، ممکن است در برخی موارد تقابل پیش بینی نشده مابین مؤلفه ها هم باعث بروز رفتار غیرمنتظره ای شود.

حلقه کنترل بازخورد اساساً از سه المان تشکیل شده است : اولاً، خود سیستم باید تحت نظارت قرار گیرد، به همین دلیل لازم است که جنبه های مختلف سیستم سنجش شوند . در بسیاری از موارد، صحبت از سنجش رفتار آسانتر از عمل به آن است. بعنوان مثال، تأخیرهای مسیر رفت و برگشت در اینترنت ممکن است بی رویه تغییر کرده و وابسته به این باشد که دقیقاً چه چیزی اندازه گیری شده

[279] Feedback control
[280] Self-star systems
[281] Disturbance

است. در چنین مواردی ، برآورد دقیق تأخیر بسیار پیچیده تر می شود. این مسأله خصوصاً هنگامی پیچیده تر می شود که گره مفروض A بخواهد بدون ایجاد مزاحمت برای دو گره کاملاً متفاوت دیگر مثلاً B وC، تأخیر بین آنها را برآورد کند. به دلایلی از همین قبیل، حلقه های کنترل بازخورد غالباً شامل یک **مؤلفه برآورد مقیاس**[۲۸۲] منطقی هستند.

بخش دیگر حلقه کنترل بازخورد به آنالیز اندازه گیری ها و مقایسه آنها با مقادیر مرجع می پردازد. این **مؤلفه آنالیز بازخورد** در واقع قلب حلقه کنترل بوده و حاوی الگوریتمهایی برای تصمیم گیری در مورد تطبیق های احتمالی است.

گروه آخر مؤلفه ها شامل مکانیزم های مختلفی برای تأثیرگذاری مستقیم بر رفتار سیستم است که از آن جمله می توان به جاگذاری کپی ها، تغییر اولویت های جداول زمان بندی، جابجایی سرویس های انتقال داده ها جهت ایجاد دسترسی، هدایت مجدد درخواستها به سمت سرورهای مختلف و غیره اشاره کرد. مؤلفه آنالیز باید از این مکانیزم ها و تأثیر(انتظاررفته) از آنها بر رفتار سیستم مطلع باشد. به همین دلیل اقدام به راه اندازی یک یا چند مکانیزم خواهدکرد تا بعداً بتواند تأثیر را مشاهده کند.

یکی از مشاهدات جالب این است که حلقه کنترل بازخورد در مدیریت دستی سیستمها هم قابل کاربرد است . تفاوت عمده این است که بجای مؤلفه آنالیز از مدیران انسانی استفاده می شود. با این وجود، جهت مدیریت مناسب سیستم های توزیعی، این مدیران بایستی تجهیزات نظارتی و همچنین مکانیزم های کنترل رفتار سیستم را در اختیار داشته باشند. مشخص است که آنالیز صحیح داده های اندازه گیری شده و راه اندازی اقدامات صحیح و بجا ، توسعه سیستم های خود مدیریتی را مشکل می سازد.

لازم به ذکر است که شکل ۱۶-۲ بیانگر سازمان *منطقی* یک سیستم خود مدیریتی بوده و به همین دلیل، مشابه مورد بحث ما درباره معماری های نرم افزار است. اما سازمان *فیزیکی* ممکن است بسیار متفاوت تر باشد. بعنوان مثال، ممکن است مؤلفه آنالیزی بطورکامل در سرتاسر سیستم توزیع شده باشد. همچنین، اندازه گیری کارآیی معمولاً در هر ماشینی که جزء سیستم توزیعی محسوب می شود انجام می گیرد. در ادامه نگاهی خواهیم داشت به چند نمونه ملموس در مورد نحوه نظارت، آنالیز، و اصلاح سیستم های توزیعی بصورتی خودکار. با مطالعه این مثال ها به تفاوت بین سازمان فیزیکی و منطقی نیز توجه خواهیم کرد.

۲-۴-۲- مثال: نظارت بر سیستمها به کمک Astrolabe

بعنوان اولین نمونه، نگاهی خواهیم داشت به Astrolabe (۲۰۰۳، Van Renesse و گروه همکاران) که در واقع سیستمی برای پشتیبانی از نظارت عمومی بر سیستم های توزیعی بسیار بزرگ است . در مورد سیستم های خود مدیریتی،Astrolabe بایستی بعنوان ابزار عمومی برای نظارت بر رفتار

[282] Metric estimation componenet

سیستمها مدنظر قرار گیرد. از خروجی آن می توان جهت تغذیه مؤلفه آنالیز برای تصمیم گیری درمورد اقدامات اصلاحی استفاده نمود.

Astrolabe مجموعه بزرگی از میزبانها را بصورت سلسله مراتبی از ناحیه ها سازماندهی می کند. ناحیه های واقع در پایین ترین سطح فقط شامل یک میزبان هستند، که بعداً به صورت گروهی در ناحیه هایی با اندازه افزایش یافته گروه بندی می شوند. ناحیه واقع در بالاترین سطح تمامی میزبانها را تحت پوشش قرار می دهد. هر یک از میزبانان، یک فرآیند Astrolabe بنام agent را اجرا می کند که مسئولیت جمع آوری اطلاعات روی ناحیه هایی که شامل این میزبان هستند را دارد. همچنین agent با دیگر agent ها هم ارتباط برقرار می کند تا بدین ترتیب ، بتواند اطلاعات منطقه را در سرتاسر سیستم انتشار دهد.

هر میزبان شامل مجموعه ای از *ویژگی ها* برای جمع آوری اطلاعات محلی است. بعنوان مثال، یک میزبان مفروض گزارش فایل های مشخص شده ای را که ذخیره می کند، کاربرد منابع سیستم، و از این قبیل را نگهداری کند. فقط ویژگی های مستقیماً بیان شده بوسیله میزبانها، یعنی در پایین ترین سطح سلسله مراتب، قابل نوشتن هستند. بعلاوه، هر ناحیه می تواند شامل مجموعه ای از ویژگی ها باشد، اما مقادیر این ویژگی ها برحسب مقادیر ناحیه های سطح پایین تر محاسبه می شوند.

مطابق شکل ۱۷-۲ ، مثال ساده ای را درنظر بگیرید که در آن سه میزبان A و B وC در یک ناحیه گروه بندی شده اند. هریک از ماشینها ،گزارش آدرس IPخود، بار CPU، حافظه آزاد موجود و تعداد فرآیندهای فعال را نگه می دارد. هریک از این ویژگی ها را می توان با استفاده از اطلاعات محلی مربوط به هریک از میزبانها مستقیماً نوشت. در سطح ناحیه، فقط اطلاعات متراکمی از قبیل میانگین بار CPU یا میانگین تعداد فرآیندهای فعال را می توان جمع آوری نمود .

شکل ۱۷-۲- جمع آوری داده و تراکم اطلاعات در Astrolabe(داخل شکل: میانگین بار، میانگین اعضا، میانگین فرآیندها،ماشین A ، ماشین B ، ماشین C ،آدرس IP ،بار،اعضاء، فرآیندها).

شکل ۱۷-۲ نحوه مشاهده اطلاعات گردآوری شده بوسیله هر یک از ماشینها بصورت گزارش در پایگاه داده ای و همچنین الحاق آنها به یکدیگر جهت تشکیل یک رابطه(جدول) را نمایش می دهد . این شکل در واقع بیانگر روش مشاهده تمامی داده های جمع آوری شده بوسیله Astrolabe است. با این وجود، اطلاعات به ازای هر ناحیه فقط براساس گزارشات اصلی که در اختیار میزبانان قرار دارد، قابل محاسبه است.

اطلاعات متراکم بوسیله توابع تراکم قابل برنامه ریزی بدست می آید که بسیار مشابه توابع موجود در زبان پایگاه داده ای رابطه ای SQL هستند . بعنوان مثال، با این فرض که اطلاعات میزبان شکل ۱۷-۲ در جدول محلی به نام *hostinfo* نگهداری می شود، با انجام یک جستجوی SQL ساده به صورت زیر می توان میانگین تعداد فرآیندهای مربوط به ناحیه حاوی ماشینهای A،B،C را محاسبه نمود:

SELECT AVG(procs) AS avg_ procs FROM hostinfo

با چند بهبود جزئی درSQL ، به راحتی می توان پرس و جوهای اطلاعاتی بیشتری را فرمول بندی کرد.

پرس و جوهایی از این دست دائماً توسط هر یک از agent های درحال اجرا روی میزبان ها ارزیابی می شوند. مشخص است که این ارزیابی فقط در صورتی امکان پذیر خواهد بود که اطلاعات ناحیه به تمامی گره های تشکیل دهنده Astrolabe پخش شده باشد .تا به این جا agent در حال اجرا برروی یک میزبان مسئول محاسبه بخش هایی از جداول ناحیه های مربوط به خود می باشد. در حالی که گزارشاتی که میزبان مسئولیت محاسباتی در قبال آن ندارد، بعضاً از طریق یک روال تبادلی ساده اما مؤثر بنام **شایعه پراکنی**[۲۸۳] به آن ارسال می شود. پروتکل های شایعه پراکنی در فصل ۴ بحث می شوند. یک agent نتایج محاسبه شده را به agent های دیگر انتقال می دهد.

نتیجه تبادل اطلاعات آن است که نهایتاً ، تمامی agent هایی که برای کمک به کسب برخی اطلاعات متراکم مورد نیاز هستند، (مشروط براینکه هیچ تغییری در آن زمان انجام نشود) نتایج واحدی را مشاهده خواهند کرد.

۳-۴-۲- مثال: تمایز بین راهکارهای کپی برداری در Globule

حال بیایید نگاهی داشته باشیم به یک شبکه توزیع محتوا بر اساس تشریک مساعی به نام Globule (Pierre and Van Steen ،۲۰۰۶). Globule ، بر قرارگیری سرورهای کاربر نهایی در اینترنت و مشارکت این سرورها در بهینه سازی عملکرد از طریق کپی برداری از صفحات وب تکیه می کند. بدین منظور، هریک از سرورهای مبدأ (یعنی سرور مسئول انجام به هنگام سازی های یک وب سایت خاص)، گزارش الگوهای دسترسی را براساس دسترسی به هر صفحه نگهداری می کند. الگوهای دسترسی بصورت عملیات نوشتن و خواندن برای یک صفحه مفروض بیان شده و هر عمل بوسیله سرور مبدأ خاص آن صفحه مهرزمانی خورده و ثبت می شود.

در ساده ترین شکل Globule فرض می کند که اینترنت به صورت یک سیستم سرورلبه ای (قبلاً توضیح داده شده است) قابل رؤیت است. مطابق شکل ۱۸-۲ ،فرض می شود که درخواستها همیشه از طریق یک سرور لبه ای مناسب قابل عبوردادن هستند . این مدل ساده به سرور مبدأ امکان می دهد تا در صورت وجود کپی روی یک سرور لبه ای خاص، از تمامی اتفاقات مطلع شود . هرچند کاهش فاصله کپی از مشتریان باعث کاهش تأخیر درک شده بوسیله مشتری می شود، در عین حال ممکن است جهت حفظ انسجام نسخه کپی با نسخه اصلی ، موجب افزایش ترافیک بین سرور اصلی و آن سرور لبه ای شود.

[283] Gossiping

شکل ۱۸-۲- مدل سرورلبه ای مفروض در Globule(مشتری، سرور مبدأ، سرور کپی، شبکه شرکتی، اینترنت مقیم، مشتری).

سرور مبدأ پس از دریافت درخواست در مورد یک صفحه خاص، آدرس IP را از محل مبدأ درخواست ثبت کرده و با استفاده از سرویس اینترنتی *WHOIS* ، اقدام به جستجوی شبکه شرکتی یا ISP مربوط به آن درخواست می نماید (۱۹۹۵، Deutschو گروه همکاران). سپس سرور مبدأ به دنبال نزدیکترین سرور کپی موجود می گردد که بتواند بعنوان سرور لبه ای آن مشتری عمل کند. در آخر هم تأخیر تا آن سرور را به همراه پهنای باند حداکثری محاسبه خواهد کرد. در ساده ترین شکل Globule فرض می شود که تأخیر بین سرور کپی و ماشین کاربر در حال درخواست قابل اغماض بوده، و همچنین پهنای باند بین این دو بسیار زیاد است.

پس از جمع آوری درخواستهای کافی برای یک صفحه ، سرور مبدأ اقدام به انجام آنالیز ساده "چه می شد اگر" میکند. این آنالیز در قالب ارزیابی سیاست های مختلف کپی برداری می کند. یک سیاست تعیین می کند که صفحه کجا باشد، و نحوه نگهداری انسجام آن صفحه چگونه باشد. هریک از سیاست های کپی برداری هزینه ای درپی دارد که بصورت تابع خطی ساده زیر قابل بیان است:

$$cost = (w_1 \times m_1) + (w_2 \times m_2) + \ldots + (w_n \times m_n)$$

که در آن m_k مقیاس عملکرد و w_k بیانگر میزان اهمیت آن مقیاس است. از نمونه مقیاس های عملکردی می توان به تأخیر جمع شده بین مشتری و سرور کپی در هنگام بازگشت دادن کپی های صفحات وب، مجموع پهنای باند مصرف شده بین سرور مبدأ و سرور کپی جهت حفظ انسجام کپی، و تعداد کپی های (مجاز) قدیمی که به مشتری بازگشت داده شده اند، اشاره کرد(۲۰۰۲، Pierre وگروه همکاران).

بعنوان مثال، فرض کنید که ms d_c تأخیر بین زمان صدور درخواست توسط مشتری C و بازگشت صفحه از بهترین سرور کپی باشد. توجه داشته باشید که بهترین سرور کپی براساس سیاست کپی برداری تعیین می شود. فرض کنیم که m_1 بیانگر تأخیر جمع شده طی یک دوره زمانی خاص ، یعنی $m_1 = \sum d_c$ باشد . اگر سرور مبدأ بخواهد تأخیر درک شده بوسیله مشتری را بهینه نماید، باید مقدار عددی نسبتاً بزرگی را برای w_1 انتخاب کند. در نتیجه، فقط آندسته از سیاستهایی که واقعاً m_1 را حداقلی می کنند، هزینه نسبتاً ناچیزی را ببار خواهند آورد .

درGlobule ، سرور مبدأ به کمک شبیه سازی براساس مسیر[۲۸۴] ، مرتباً اقدام به ارزیابی چند ده سیاست کپی برداری برای هریک از صفحات جداگانه وب خواهد کرد. از همین شبیه سازی ها بهترین سیاست انتخاب و سپس اجرا می شود. ممکن است اینطور بنظر برسد که کپی های جدید در سرورهای لبه ای مختلفی نصب شده و یا این که از روش متفاوتی برای منسجم نگه داشتن کپی ها استفاده می

[284] Trace-driven simulation

شود. جمع آوری مسیر[۲۸۵]ها، ارزیابی سیاستهای کپی برداری، و اعمال سیاست انتخاب شده همگی بصورت اتوماتیک انجام خواهد شد .

چند نکته ظریف دیگر هم وجود دارند که باید مورد بحث قرار گیرد. در درجه اول، هنوز مشخص نیست که چه تعداد درخواست باید پیش از آغاز ارزیابی سیاست جاری جمع آوری شود. برای توضیح، فرض کنید که در زمان T_i ، سرور مبدأ سیاست p را برای دروه زمانی بعدی یعنی تا T_{i+1} انتخاب می کند. این انتخاب براساس مجموعه درخواستهایی صورت می پذیرد که قبلاً بین زمان T_i , T_{i-1} صادر شده بودند . البته، بعداً سرور ممکن است با بازگشت به زمان گذشته T_{i+1}، به این نتیجه برسد که بهتر بود براساس درخواستهای واقعی که در فاصله زمانی T_{i+1} , T_i صادر شده بودند، سیاست p^* را انتخاب می کرد.

می توان نتیجه گرفت که درصد پیش بینی های خطا بستگی به طول سری درخواستهایی دارد (که اصطلاحاً طول مسیر نامیده می شود) که جهت پیش بینی و انتخاب سیاست بعدی مورد استفاده قرار می گیرد . این وابستگی در شکل ۱۹-۲ ارائه شده است. همانطور که از شکل هم برمی آید، در صورت ناکافی بودن طول مسیر، خطای پیش بینی بهترین سیاست افزایش خواهد یافت. بعنوان دلیل باید گفت که برای انجام یک ارزیابی مناسب، درخواست های کافی باید اجرا شوند. با این وجود، درصورت استفاده از تعداد درخواستهای بسیار زیاد، خطا هم افزایش خواهد یافت. دلیل این است که افزایش بیش از حد طول مسیر باعث بازیابی تعداد زیادی *تغییرات* در الگوهای دسترسی شده و این امر هم متقابلاً باعث مشکل شدن انتخاب بهترین سیاست وحتی غیرممکن شدن آن خواهد شد. برای درک این پدیده معروف فرض کنید که بخواهیم با بررسی اوضاع جوی صدسال اخیر، هوای فردا را پیش بینی کنیم . مسلماً اینکار بی فایده خواهد بود و بررسی سابقه چند روز اخیر نتیجه بهتری ایجاد خواهد کرد.

یافتن طول بهینه مسیر بصورت اتوماتیک هم امکان پذیر است. ارائه راه حل مناسب برای این مشکل را به عنوان تمرین ارائه شده است.

۴-۴-۲- مثال: مدیریت تعمیر اتوماتیک مؤلفه در Jade

وقتی از مجموعه خوشه ای[۲۸۶] از کامپیوترهای مجهز به سرورهای پیشرفته نگهداری می کنید، حل مشکلات مدیریتی تبدیل به نوعی ضرورت می شود. یک روش برای سرورهای ساخته شده براساس رویکرد مؤلفه-پایه[۲۸۷] ، کشف خرابی مؤلفه ها و تعویض اتوماتیک آنهاست. در سیستم Jade از این رویکرد پیروی می کند (۲۰۰۵، Bouchenak و گروه همکاران). در این بخش مختصراً راجع به آن بحث خواهیم کرد.

[285] Trace
[286] Cluster
[287] Component-based

Jade براساس مدل مؤلفه Fractal ساخته شده است و شامل یک پیاده سازی Java از یک چارچوب برای افزودن و حذف مؤلفه ها در زمان اجرا است (۲۰۰۴، Bruneton وگروه همکاران). هر مؤلفه داخل Fractal ممکن است دو رابط[۲۸۸] داشته باشد: یک **رابط سرور**که جهت فراخوانی متد هایی که توسط آن مؤلفه پیاده سازی شده اند، و یک **رابط مشتری** که توسط مؤلفه برای فراخوانی مؤلفه های دیگر استفاده می شود. مؤلفه ها توسط واسط های **پیوندی**[۲۸۹] بیکدیگر متصل می شوند. بعنوان مثال ، رابط مشتری مؤلفه C_1 را می توان به رابط سرور مؤلفه C_2 پیوند داد. عمل اولیه یوند به این معناست که فراخوانی رابط مشتری مستقیماً منجر به فراخوانی رابط پیوند خورده به سرور می شود. در حالت پیوند ترکیبی فراخوانی ممکن است از طریق یک یا چند مؤلفه دیگر انجام شود، مثلاً به این دلیل که رابط مشتری و سرور همخوانی نداشته و بنابراین نیاز به نوعی تبدیل وجود دارد. دلیل دیگر می تواند این باشد که مؤلفه های متصل شده در ماشینهای مختلف قرارگرفته اند .

در Jade از ایده **دامنه مدیریت تعمیر** استفاده می شود. چنین دامنه ای متشکل از تعدادی گره است که هریک نماینده یک سرور و مؤلفه هایی است که توسط آن سرور اجرا می شود. یک مدیر گره جداگانه هم وجود دارد که مسئول افزودن گره ها و حذف آنها از دامنه است. ممکن است برای تضمین دسترسی بالا ، مدیر گره کپی برداری شود.

هر یک از گره ها مجهز به کشف کننده خرابی هستند که بر سلامت گره یا یکی از مؤلفه های آن نظارت کرده و هرنوع خرابی را به مدیر گره گزارش می دهند. نوعاً، این کشف کننده ها تغییرات استثنایی در حالت مؤلفه ، کاربرد منابع، و خرابی واقعی مؤلفه (این حالت ممکن است به این معنی باشد که یک ماشین ازکارافتاده است) را مورد بررسی قرار می دهند.

پس از شناسایی خرابی روال تعمیرآغاز می شود. این روال بوسیله یک سیاست تعمیری گدانده می شود که بخشی از آن بوسیله مدیر گره اجرا می شود. سیاست ها به صورت صریح بیان شده و برحسب خرابی شناسایی شده اجرا می شوند. بعنوان مثال، فرض کنید که خرابی در یک گره شناسایی شده و طبق سیاست تعمیرمراحل زیر باید انجام شود:

۱. توقف هرنوع پیوند بین مؤلفه ها روی یک گره غیرمعیوب و مؤلفه واقع برروی گرهی که به تازگی معیوب شده است.
۲. درخواست از مدیر گره جهت آغازبه کار و افزودن گره جدید به دامنه.
۳. پیکربندی گره جدید با دقیقاً همان مؤلفه های گره از کار افتاده.
۴. برقراری مجدد تمامی پیوندهایی که قبلاً متوقف شده بودند.

[288] Interface
[289] Binding

در این مثال، سیاست تعمیر ساده بوده و فقط در صورتی قابل اجرا خواهد بود که داده های کلیدی گم نشده باشند(مؤلفه های ازکارافتاده اصطلاحاً **بی حالت**[۲۹۰] نامیده می شوند).

روش مورد استفاده درJade نمونه ای است از خود مدیریتی، یعنی اجرای اتوماتیک سیاست تعمیر به محض شناسایی خرابی به منظور بازگرداندن کل سیستم به وضعیت پیش از ازکارافتادگی. در این سیستم که بر اساس مؤلفه است این نوع تعمیر اتوماتیک نیازمند نوعی پشتیبانی است تا قطعات بتوانند در زمان اجرا اضافه و حذف شوند. در مجموع باید گفت که تبدیل برنامه های کاربردی قدیمی به سیستم های خود مدیریتی غیرممکن است.

۵-۲- خلاصه

سیستم های توزیعی را به روشهای مختلف می توان سازماندهی نمود. ما می توانیم بین معماری نرم افزار و معماری سیستم تمایز قائل شویم. در معماری سیستم محل قرارگرفتن مؤلفه های تشکیل دهنده سیستم توزیعی در کل ماشینهای مختلف درنظر گرفته می شود. در معماری نرم افزار بیشتر به بررسی سازمان منطقی نرم افزار می پردازد: یعنی نحوه تعامل مؤلفه های مختلف، شیوه های ساختاربندی آنها، چگونگی مستقل سازی آنها و غیره می پردازد.

یکی از مهم ترین ایده های مورد بحث در معماری ها بحث شیوه های معماری است. شیوه[۲۹۱] در واقع بیانگر اصول اساسی سازماندهی تعامل بین مؤلفه های نرم افزاری است که سیستم توزیعی را ساخته اند. از جمله مهمترین شیوه ها می توان به لایه بندی، شیء گرایی، رویداد گرایی و فضای-داده گرایی اشاره کرد.

سازمان های مختلفی برای سیستم های توزیعی وجود دارد. یک کلاس مهم آن است که ماشین ها به مشتری ها و سرورها تقسیم بندی می شوند. در این حالت، مشتری درخواستی را برای سرور ارسال می کند ،سپس سرور نتیجه را تولید وآنرا به مشتری بازمی گرداند. معماری مشتری- سرور بیانگر روش مرسوم پیمانه ای سازی نرم افزار است که طی آن ، یک پیمانه مفروض اقدام به فراخوانی کارکرد های موجود در پیمانه دیگر می کند. با قراردادن مؤلفه های مختلف در ماشینهای مختلف، به نوعی توزیع فیزیکی و طبیعی از کارکردها در مجموعه ای از ماشینها دست خواهیم یافت.

معماری های مشتری- سرور غالباً بسیار متمرکز هستند. در معماری های غیرمتمرکز، غالباً شاهد هستیم که فرآیندهای تشکیل دهنده سیستم توزیعی -که به نام سیستم های نظیربه نظیر هم خوانده می شوند- به یک اندازه نقش دارند. در سیستم های نظیربه نظیر ، فرآیندها به شکل یک شبکه فوقانی سازماندهی می شوند ؛ شبکه فوقانی درواقع یک شبکه منطقی است که هریک از فرآیندهای آن، لیست محلی از فرایندهای نظیر دیگر را دراختیار دارد و می تواند با آنها ارتباط داشته باشد. شبکه فوقانی می تواند دارای ساختار باشد، به گونه ای که از طرح های مشخصی جهت مسیردهی پیام ها در بین

[290] Stateless
[291] Style

فرآیندها استفاده نماید. در شبکه های فاقد ساختار ، لیست فرایندهای نظیر کمابیش تصادفی است ؛ به این معنا که جهت مکان یابی داده ها یا دیگر فرآیندها نیازمند پیاده سازی الگوریتم های جستجو هستیم.

به عنوان جایگزین، سیستم های توزیعی خودمدیریتی پیاده سازی شده اند. در این سیستمها تا حدودی ایده هایی از معماری های نرم افزار و سیستم با هم ترکیب می شود. سیستم های توزیعی خودمدیریتی را می توان بصورت حلقه های کنترل بازخوردی سازمان دهی کرد. این قبیل حلقه ها، دارای یک مؤلفه نظارتی جهت اندازه گیری رفتار سیستم توزیعی ، یک مؤلفه آنالیزی جهت بررسی لزوم یا عدم لزوم تنظیمات و همچنین ، مجموعه ای از ابزارهای مختلف جهت تغییر رفتار می باشند. حلقه های کنترل بازخوردی را می توان در محل های مختلف در سیستم های توزیعی تلفیق نمود. تا رسیدن به درکی مشترک از نحوه ایجاد این حلقه ها و همچنین اجرای آنها به پژوهش بسیاری نیاز داریم.

مسائل فصل

۱- چنانچه مشتری و سرور خیلی دور از هم قرار داده شوند، تأخیر شبکه تأثیر تعیین کننده ای بر کارآیی خواهد داشت. چه راه حلی برای این مشکل پیشنهاد می کنید؟

۲- معماری مشتری- سرور سه طبقه چیست.

۳- چه تفاوتی بین توزیع عمودی و توزیع افقی وجود دارد؟

۴- زنجیره ای از فرآیندها به صورت $P_1, P_2, \ldots, P_n$ را در نظر بگیرید که معماری چندطبقه ای مشتری- سرور را پیاده سازی می کنند. فرآیند P_i مشتری فرآیند P_{i+1} بوده و P_i فقط پس از دریافت پاسخ از P_{i+1} ،جواب را به P_{i-1} عودت خواهد داد. با بررسی عملکرد درخواست- پاسخ در فرآیند P_1 مهمترین مشکلاتی راکه در این سازمان می بینید ذکر کنید.

۵- در یک شبکه فوقانی دارای ساختار، پیام ها براساس توپولوژی شبکه فوقانی مسیریابی می شوند. یکی از مهمترین معایب این روش را ذکر کنید.

۶- شبکهCAN شکل ۸-۲ را درنظر بگیرید. برای مسیریابی پیام از گره با مختصات (۰،۳،۰،۲۵) به گره دیگر با مختصات (۰،۶،۰،۹) چگونه عمل می کنید؟

۷- با این فرض که گره واقع در CAN از مختصات همسایگان بلافصل خود اطلاع دارد، یکی از منطقی ترین سیاستهای مسیریابی می تواند ارسال پیام به نزدیکترین گره به سوی مقصد باشد. این سیاست چقدر خوب است.

۸- شبکه فوقانی بدون ساختاری را در نظر بگیرید که در آن هر گره بطور تصادفی تعداد c همسایه را انتخاب می کند. چنانچه P و Q هردو همسایه R باشند، احتمال این که همسایه یکدیگر باشند چقدر است؟

۹- مجدداً یک شبکه فوقانی بدون ساختار را در نظر بگیرید که در آن هر گره بطور تصادفی تعداد c همسایه را انتخاب می کند. برای جستجوی یک فایل ، این گره سیل درخواست را به سوی همسایگان جاری کرده و از آنها می خواهد که درخواست را مجدداً به صورت سیل ارسال کنند. به چند گره خواهد رسید ؟

۱۰- هرگرهی را نمی توان در سیستم نظیربه نظیر تبدیل به ابرنظیر نمود. یک گره برای این منظور چه ویژگی هایی باید داشته باشد؟

۱۱- یک سیستم BitTorrent را در نظر بگیرید که در آن هر گره دارای پیوند بیرونی با ظرفیت پهنای باند B_{out} داشته باشد و هر پیوند ورودی دارای ظرفیت پهنای باند B_{in} باشد. تعدادی از این گره ها(به نام بذر) داوطلبانه فایل هایی را برای دانلود کردن به دیگران ارائه می دهند. در صورتیکه فرض کنیم یک مشتری BitTorrent می تواند در هر زمان با حداکثر یک بذر تماس برقرار نماید، حداکثر ظرفیت دانلود مشتری BitTorrent چقدر خواهد بود؟

12- دلیل متقاعد کننده(فنی) مطرح کنید که چرا سیاست تلافی جویانه(tit-for-tat) مورد استفاده در BitTorrent ، در اشتراک فایل در اینترنت بهینه نمی باشد.

۱۳- دو مثال از کاربرد راه گیر ها در میان افزار تطبیقی ارائه شد. چند مثال دیگر از همین مورد ذکر کنید.

۱۴- راه گیرها تا چه اندازه وابسته به میان افزاری هستند که در آن مورد استفاده قرار می گیرند؟

۱۵- در خودروهای مدرن با انواع ابزارهای الکترونیک مواجه هستیم. نمونه هایی از سیستم های کنترل بازخوردی را در این خودروها ذکر کنید.

۱۶- نمونه ای از یک سیستم خود مدیریتی ذکر کنید که در آن مؤلفه آنالیزکننده بطورکامل توزیع یا حتی مخفی شده است.

۱۷- راه حلی برای تعیین اتوماتیک بهترین طول مسیر (Trace) برای پیش بینی سیاستهای کپی برداری درGlobule ذکر کنید.

۱۸- (**تکلیف آزمایشگاهی**) بکمک نرم افزار موجود، یک سیستم برپایه BitTorrent جهت توزیع فایل ها در مشتریان متعدد از یک سرور واحد و قدرتمند طراحی و پیاده سازی کنید. برای سهولت کار می توانید از یک سرور وب استاندارد که می تواند به عنوان ردیاب (Tracker) عمل کند استفاده کنید.

3
فرآیندها[۲۹۲]

در این فصل، نگاه دقیقتری به نقش کلیدی فرآیندهای مختلف در سیستم های توزیعی خواهیم داشت . مفهوم فرآیند ریشه در حوزه سیستم عامل داشته و غالباً بصورت برنامه در حال اجرا تعریف می شود. از نقطه نظر سیستم عامل، مدیریت و زمان بندی فرآیندها را می توان از مهم ترین مسائل مورد بحث به حساب آورد. در حالیکه در سیستم های توزیعی، مسائل دیگر از اهمیت برابر یا بیشتری برخوردار هستند.

بعنوان مثال، جهت سازماندهی مؤثر سیستم های مشتری- سرور، غالباً بهتر است که از تکنیک های چندنخی استفاده کنیم . همانطور که در بخش اول همین فصل هم خواهیم گفت، یکی از مهم ترین نقش های نخ[۲۹۳] ها در سیستم های توزیعی آن است که باعث می شوند تا مشتریان و سرورها به صورتی ایجاد شوند که ارتباطات و پردازش محلی با هم هم پوشانی کرده و همین امر باعث افزایش سطح کارآیی آنها خواهد شد.

در سالهای اخیر، مفهوم مجازی سازی[۲۹۴] کاربرد گسترده ای پیدا کرده است. مجازی سازی به برنامه کاربردی، وشاید به محیط کامل آن و از جمله خود سیستم عامل ، امکان می دهد تا بصورت همزمان با سایر برنامه های کاربردی اجرا شود. اما اینکار بصورتی کاملاً مستقل از سخت افزار و سکو[۲۹۵]های زیرین انجام شده و همین امر منجر به درجه بالایی از جابجاپذیری[۲۹۶] در آنها خواهد شد. بعلاوه، مجازی سازی به تفکیک و مجزاسازی خرابی های ناشی از خطاها و مشکلات امنیتی کمک می کند. مجازی سازی یکی از مفاهیم مهم در سیستم های توزیعی بوده و به همین دلیل بخش مجزایی را به آن اختصاص داده ایم.

همانطور که در فصل ۲ هم گفتیم، سازمان های مشتری- سرور در سیستم های توزیعی از اهمیت بالایی برخوردار هستند. در این فصل، نگاه دقیق تری خواهیم داشت به برخی سازمان بندی های نمونه مشتری و سرور. همچنین راجع به مسائل عمومی طراحی سرورها صحبت خواهیم کرد.

یکی از مهمترین مسائل، خصوصاً در سیستم های توزیعی حوزه گسترده[۲۹۷]، جابجایی فرآیندها در بین ماشین های مختلف است. مهاجرت فرآیند[۲۹۸] یا به بیان مشخص تر، مهاجرت کدها[۲۹۹]، می تواند هم در

---

[292] processes
[293] thread
[294] Virtualization
[295] Platform
[296] portability
[297] wide-area
[298] process migration
[299] code migration

کسب قابلیت مقیاس پذیری[300] مفید واقع شود و هم در پیکربندی پویای مشتری ها و سرورها، مفهوم دقیق مهاجرت کد و اثرات جانبی آن هم در این فصل مورد بررسی قرار گرفته اند.

۱-۳- نخ ها

هرچند فرآیندها یکی از ستونهای تشکیل هنده سیستم های توزیعی محسوب می شوند، در عمل مشاهده شده که دانه بندی[301] فرآیندهای ایجاد شده بوسیله سیستم های عاملی که مبنای سیستم های توزیعی را تشکیل می دهند ، هنوزکافی نیست. پس به این نتیجه می رسیم که کسب دانه بندی ظریف تر از طریق افزایش تعداد نخ ها در هر فرآیند ساخت برنامه های کاربردی توزیعی را آسانتر و باعث افزایش کارآیی می شود. در این بخش، نگاه دقیق تری به نقش نخ ها در سیستم های توزیعی داشته و دلیل اهمیت آنها را توضیح خواهیم داد. اطلاعات بیشتر در مورد نخ ها و نحوه استفاده از آنها برای ایجاد برنامه های کاربردی را در Lewis and Berg(1998) و Stevens(1999) مطالعه کنید.

۱-۱-۳- مقدمه ای بر نخ ها

برای درک نقش و اهمیت نخ ها در سیستم های توزیعی ، باید بدانیم که فرآیند چیست و فرآیندها و بندها چگونه با هم ارتباط پیدا می کنند. سیستم عامل برای اجرای برنامه تعدادی پردازنده مجازی ایجاد کرده و هریک را مسئول اجرای برنامه جداگانه ای می کند. برای ردگیری این پردازنده های مجازی سیستم عامل یک **جدول فرآیند**[302] در اختیار دارد؛ این جدول حاوی ورودی هایی جهت ذخیره مقادیر ثبات های پردازنده، نگاشت های حافظه، فایل های باز، اطلاعات حسابرسی، الویت ها و غیره می باشد. **فرآیند** هم غالباً به صورت برنامه در حال اجرا - یعنی برنامه ای که هم اکنون روی یکی از همین پردازنده های مجازی سیستم عامل در حال اجراست - تعریف می شود. یکی از مسائل مهم این است که سیستم عامل بسیار مراقب است تا مبادا فرآیندهای مستقل سهواً یا عمداً کوچکترین تأثیری بر صحت رفتار یکدیگر بگذارند. به بیان دیگر، این واقعیت که چندین برنامه می توانند به صورت همروند در یک پردازنده واحد یا دیگر منابع سخت افزاری شریک شوند ، شفاف سازی شده است. سیستم عامل برای موفقیت در تفکیک فرآیندها ، معمولاً نیازمند پشتیبانی سخت افزار است.

برای دستیابی به این شفافیت همروندی باید هزینه نسبتاً گزافی پرداخت کرد. بعنوان مثال، هرزمان که فرآیندی ایجاد می شود، سیستم عامل باید یک فضای آدرس کالاً مستقل به ازای آن ایجاد کند. در اینجا تخصیص می تواند به معنای مقدار دهی اولیه حافظه باشد، مثلاً صفر قرار دادن یک قطعه داده ای، کپی کردن برنامه مربوطه به قطعه متنی، و ایجاد پشته برای داده های موقت. به همین ترتیب، سوئیچ کردن پردازنده بین دو فرآیند هم ممکن است تاحدودی هزینه بر باشد. جدای از ذخیره متن

[300] scalability
[301] Granularity
[302] process table

پردازنده (شامل مقادیر ثبات ها، برنامه شمار، اشاره گر پشته، وغیره) سیستم عامل مجبور به تغییر ثبات های واحد مدیریت حافظه[۳۰۳] (MMU) و نامعتبر سازی حافظه های پنهان ترجمه آدرس مانند بافر دم دستی ترجمه[۳۰۴](TLB) نیز می باشد. بعلاوه ، چنانچه تعداد فرآیندهای تحت پشتیبانی سیستم عامل بیش از آن چیزی باشد که می تواند بصورت همزمان در حافظه اصلی نگه دارد، ممکن است مجبور باشد تا پیش از سوئیچ بین فرآیندها، فرآیندهایی را بین حافظه اصلی و دیسک مبادله کند.

مشابه فرآیند، نخ هم قطعه کد مخصوص به خود را، به صورتی مستقل از دیگر نخ ها، اجرا می کند. اما برخلاف فرآیندها، چنانچه کسب درجه بالایی از شفافیت همروندی منجر به افت کارآیی شود، هیچ تلاشی برای دستبابی به آن انجام نمی شود . بنابراین، یک سیستم نخ معمولاً فقط حداقل اطلاعات لازم برای ایجاد امکان اشتراک پردازنده در بین چندین نخ را در خود نگه می دارد. علی الخصوص اینکه، **متن نخ**[۳۰۵] غالباً فقط شامل متن پردازنده به همراه مختصر اطلاعات دیگری جهت مدیریت نخ است. بعنوان مثال، یک سیستم نخ ممکن است فقط پی گیری کند که در حال حاضر روی یک متغیر انحصار متقابل[۳۰۶] مسدود شده است تا به این ترتیب مانع از انتخاب آن برای اجرا شود. آن دسته از اطلاعاتی که ضرورت چندانی در مدیریت چندنخی نداشته باشند، غالباً نادیده گرفته می شوند. به همین دلیل، محافظت از داده ها در برابر دسترسی نامناسب بوسیله نخ ها در هر فرآیند بطور کامل برعهده پدید آورندگان برنامه های کاربردی قرار دارد.

این روش دو اثر جانبی مهم دارد. در درجه اول، کارآیی یک برنامه کاربردی چندنخی چندان ضعیفتر از عملکرد رقیب تک نخی خود نخواهد بود . در واقع، در بسیاری از موارد چند نخی شدن منجر به افزایش کارآیی هم می شود. ثانیاً، از آنجایی که نخ ها به صورت اتوماتیک در برابر یکدیگر به آن صورتی که فرآیندها محافظت می شوند محافظت نشده اند، ایجاد برنامه های کاربردی چند نخی نیاز به تلاش فکری بیشتری دارد. در اینجا هم ، طراحی مناسب و ساده سازی امور بسیار مفید خواهد بود. متأسفانه ، از اوضاع فعلی می توان نتیجه گرفت که این اصل هنوز به خوبی درک و مورد استفاده قرار نگرفته است .

کاربرد نخ در سیستم های غیرتوزیعی

پیش از بحث در مورد نقش نخ ها در سیستم های توزیعی ، اجازه دهید ابتدا در مورد کاربرد آنها در سیستم های متعارف غیر توزیعی صحبت کنیم. فرآیندهای چند نخی مزایای متعددی دارند و همین ویژگی باعث افزایش تمایل نسبت به کاربرد آنها شده است.

مهم ترین مزیت ناشی از این واقعیت است که در یک فرآیند تک نخی ، با اجرای فراخوان سیستمی مسدود شونده، کل فرآیند مسدود خواهد شد. برای درک بهتر موضوع، یک برنامه کاربردی مانند یک

[303] Memory Management Unit
[304] Translation Lookaside Buffer
[305] thread context
[306] mutex

برنامه صفحه گسترده را تصور کنید که کاربر بخواهد بطور دائم و به صورت تعاملی مقادیر را تغییر دهد. یکی از ویژگی های مهم برنامه های صفحه گسترده این است که وابستگی های کارکردی بین سلول های مختلف را حتی از صفحه گسترده های دیگر هم نگه می دارد. بنابراین، با تغییر یک سلول ، تمامی سلولهای وابسته هم بصورت اتوماتیک به هنگام خواهند شد. وقتی کاربر مقدار عددی داخل یک سلول را تغییر می دهد، این اصلاح می تواند باعث ایجاد چندین سری محاسبات شود. در صورت وجود فقط یک نخ کنترل، کار محاسبه نمی تواند همزمان با انتظار برنامه برای ورودی انجام شود. به همین ترتیب، نمی توان در حین محاسبه وابستگی ها، اقدام به ایجاد ورودی نمود. ساده ترین راه حل این است که حداقل از دو نخ کنترلی استفاده کنیم، یکی برای هدایت جریان تعاملی با کاربر و دیگری برای به هنگام سازی صفحه گسترده. در این حین می توان ، همزمان با فعال و در حال کار بودن دو نخ، از نخ سومی هم برای پشتیبانی از صفحه گسترده روی دیسک استفاده نمود .

یکی دیگر از مزایای چند نخی این است که امکان استفاده از موازی گرایی[۳۰۷] را به هنگام اجرای برنامه روی یک سیستم چند پردازنده ای فراهم می آورد . در این حالت، هر نخ به یک پردازنده متفاوت نسبت داده شده و داده های اشتراکی هم در حافظه اصلی اشتراکی ذخیره می شوند. این موازی گرایی در صورتیکه با طراحی مناسب همراه شود، می تواند شفاف باشد؛ یعنی فرآیند می تواند به همان راحتی اما با سرعتی کمتر، روی یک سیستم تک پردازنده اجرا شود. به دلیل افزایش میزان دسترسی به ایستگاههای کاری چندپردازنده ی نسبتاً ارزان قیمت ، چند نخی برای موازی گرایی هم اهمیتی بیش از پیش یافته است .از این دست سیستم های کامپیوتری نوعاً جهت سرورها در برنامه های کاربردی مشتری- سرور استفاده می شود.

از مزایای دیگر چند نخی می توان به کاربرد مفید آنها در متن برنامه های کاربردی بزرگ اشاره کرد. این نوع برنامه های کاربردی غالباً بصورت مجموعه ای از برنامه های همکاری کننده ایجاد می شوند که هریک بایستی توسط یک فرآیند جداگانه اجرا شوند. این روش خصوصاً در محیط UNIX کاربرد دارد. همکاری بین برنامه ها بوسیله مکانیزم های ارتباط میان فرآیندی[۳۰۸] (IPC) اجرا می شود. در سیستم های UNIX، این مکانیزم ها غالباً شامل لوله ها[۳۰۹] (نامگذاری شده)، صف های پیام ها، و قطعه های حافظه اشتراکی می باشد[همچنین مراجعه شود به (۲۰۰۵) Stevens and Rago]. مهمترین نقطه ضعف تمامی مکانیزم های IPC آن است که ارتباط غالباً مستلزم سوئیچ کردن متن به مقدار زیاد است که در قالب سه نقطه مختلف در شکل ۱-۳ نمایش داده شده است.

[307] parallelism
[308] Interprocess Communication
[309] Pipes

شکل۱-۳- سوئیچ کردن متن در نتیجه IPC(داخل شکل: فرآیند A، فرآیندB،S1: سوئیچ کردن از فضای کاربر به فضای هسته اصلی، S2: سوئیچ کردن بافت از فرآیندA به فرآیندB، S3: سوئیچ کردن از فضای هسته اصلی به فضای کاربر).

از آن جایی که IPC نیازمند دخالت هسته اصلی است، غالباً لازم است که فرآیند ابتدا از حالت کاربر به حالت هسته اصلی سوئیچ کند که در شکل ۱-۳ با *S1* نمایش داده شده است. این کار هم مستلزم تغییر نگاشت حافظه[۳۱۰] در واحد مدیریت حافظه (MMU) و همچنین تخلیه TLB است. در داخل هسته اصلی، سوئیچ متن فرآیند انجام می شود(*S2* در شکل فوق) و سپس طرف دیگر با تغییر از حالت هسته اصلی به حالت کاربر فعال می شود (*S3* شکل ۱-۳). تغییر اخیر هم مستلزم تغییر نگاشت MMU و تخلیه TLB است.

به جای استفاده از فرآیندها، می توان برنامه کاربردی ایجاد نمود که بخش های مختلف آن بوسیله نخ های مجزا اجرا شوند. ارتباط بین این بخش ها تماماً با استفاده از داده های مشترک صورت می پذیرد. سوئیچ بین نخ ها ممکن است بعضاً بطور کامل در فضای کاربر انجام شود. با این وجود در بسیاری از پیاده سازی ها، هسته اصلی از نخ ها آگاه بوده و آنها را زمان بندی می کند. این امر ممکن است منجر به ارتقاء چشمگیر کارآیی شود.

دلیل آخر برای استفاده از نخ ها کاملاً مربوط به مهندسی نرم افزار است : ساخت بسیاری از برنامه های کاربردی صرفاً بصورت مجموعه ای از نخ های همکاری کننده بسیار آسانتر است. بعنوان نمونه برنامه های کاربردی را در نظر بگیرید که باید امور (کمابیش مستقل) متعددی را انجام دهند. مثلاً در مورد پردازشگر های کلمه[۳۱۱] ، می توان از نخ های جداگانه برای کار برروی ورودی کاربر، چک کردن گرامر و املاء ، صفحه بندی سند، تهیه فهرست موضوعی و غیره استفاده کرد.

پیاده سازی نخ

نخ ها غالباً به شکل یک بسته ارائه می شوند. این بسته شامل عملیاتی جهت ایجاد و از بین بردن نخ ها و همچنین عملیات روی متغیرهای همگام سازی[۳۱۲] از قبیل متغیر انحصار متقابل (mutex) و متغیرهای شرایطی است. اساساً دو روش برای پیاده سازی بسته نخ وجود دارد. روش اول ساخت کتابخانه نخ است که کاملاً در حالت کاربر اجرا شود. در روش دوم، هسته اصلی آگاه از وجود نخ ها و زمان بندی آنها است.

کتابخانه نخ سطح کاربر چند مزیت دارد. اول، ایجاد و از بین بردن نخ ها ارزان است. از آنجا که تمام مدیریت نخ در فضای آدرس کاربر نگهداری می شود، هزینه ایجاد نخ اساساً همان هزینه تخصیص حافظه برای تهیه یک پشته برای نخ است. به همین ترتیب، از بین بردن نخ عمدتاً شامل آزاد سازی

[310] Memory map
[311] Word Processor
[312] Synchronization

حافظه تخصیص داده شده برای پشته نخ است که دیگر استفاده ندارد. هردو عملیات فوق کم هزینه هستند.

دومین مزیت نخ های سطح کاربر آن است که سوئیچ کردن متن نخ فقط بوسیله چند دستورالمعل قابل انجام است . اساساً، فقط باید مقادیر ثبات های پردازنده را ذخیره کرد و سپس مقادیر قبلاً ذخیره شده نخی که به آن سوئیچ می شود مجدداً بارگذاری کرد. هیچ لزومی به تغییر نگاشت های حافظه، تخلیه TLB، انجام حسابرسی پردازنده و از این قبیل وجود ندارد. تغییر متن نخ در صورت لزوم همگام شدن دو نخ ، مثلاً زمان ورود به بخشی از داده های مشترک مطرح می شود.

با این وجود، نقطه ضعف عمده نخ های سطح کاربر آن است که فراخوان سیستمی مسدود شونده فوراً باعث مسدود شدن کل فرآیندی خواهد شد که این نخ و تمامی نخ های دیگر موجود در فرآیند به آن تعلق دارند . همان طور که قبلاً هم توضیح دادیم، اهمیت ویژه نخ ها در تبدیل برنامه های کاربردی بزرگ به صورت قطعات منطقی قابل اجرای هم زمان است. در این حالت، مسدود شدن روی ورودی-خروجی قاعدتاً نباید مانع از اجرای قطعات دیگر در این زمان باشد. در چنین برنامه های کاربردی، استفاده از نخ های سطح کاربر مفید نخواهد بود.

اغلب این مشکلات را می توان با پیاده سازی نخ ها در هسته اصلی سیستم عامل حل و فصل نمود. متأسفانه، هزینه ای که باید بپردازیم بالاست: تمامی عملیات نخ (از قبیل ایجاد، حذف، همگام سازی و غیره) بایستی بوسیله هسته اصلی اجرا شده و این امر مستلزم فراخوانی سیستمی است. در اینصورت، تغییر متن نخ ممکن است به اندازه تغییر متن فرآیند هزینه بر باشد. در نتیجه، اغلب مزایای عملکردی استفاده از نخ ها بجای فرآیندها از بین خواهد رفت.

یک راه حل استفاده از صورت ترکیبی نخ های سطح کاربر و سطح هسته اصلی است که غالباً به نام **فرآیندهای سبک وزن**[۳۱۳] **(LWP)** خوانده می شود. LWP در متن یک فرآیند (سنگین وزن) اجرا شده و ممکن است در هر فرآیند چندین LWP وجود داشته باشد. هرسیستم علاوه بر دارا بودن LWPها ، یک بسته نخ سطح کاربر را هم ارائه می دهد- این بسته به برنامه های کاربردی، عملیات معمول برای ایجاد و از بین بردن نخ ها ارائه می دهد. بعلاوه ، این بسته امکاناتی از قبیل متغیر انحصار متقابل (mutex) و متغیرهای شرطی را برای همگام سازی نخ ها ارائه می دهد. مسأله مهم این است که بسته نخ تماماً در فضای کاربر اجرا می شود. به بیان دیگر، تمامی عملیات روی نخ ها بدون دخالت هسته اصلی انجام می شود.

شکل ۲-۳- ترکیب فرآیندهای سبک وزن سطح هسته اصلی و نخ های سطح کاربر(فضای کاربر، وضعیت نخ، نخ، فضای هسته اصلی، LWPدر حال اجرای یک نخ ، فرآیند سبک وزن).

[313] Lightweight Processes

مطابق شکل ۲-۳ ، یک بسته نخ ممکن است در بین چندین فرآیند LWP مشترک باشد . به این معنا که هر LWP می تواند نخ (سطح کاربر) ویژه خود را اجرا کند. برنامه های کاربردی چند نخی ازطریق ایجاد نخ ها و سپس، تخصیص هریک از نخ ها به یک LWP ساخته می شود. تخصیص نخ به LWP عمدتاً ضمنی و از برنامه نویس مخفی می باشد.

در ترکیب نخ ها(ی سطح کاربر) و LWPها به این ترتیب عمل می شود: بسته نخ برای زمان بندی نخ بعدی فقط یک روال در اختیار دارد. در هنگام ایجاد LWP(که با فراخوان سیستمی انجام می شود)، پشته خاصLWP در اختیار آن قرار گرفته و به او دستور داده می شود که برای جستجوی نخ مورد نظر برای اجرا، روال زمان بندی را اجرا کند. در صورت تعدد LWPها، هریک از آنها زمان بند را اجرا می کند. بنابراین جدول نخ ها، که برای نگهداری گزارش مجموعه نخ های فعلی مورد استفاده قرار می گیرد، در بین LWPها به اشتراک گذاشته می شود. محافظت از این جدول جهت تضمین دسترسی به صورت انحصاری بوسیله متغیر های انحصار متقابلی انجام می شود که تماماً در فضای کاربر پیاده سازی می شوند. به بیان دیگر، همگام سازی در بین LWPها مستلزم هیچ نوع پشتیبانی از جانب هسته اصلی نمی باشد.

پس از آنکه LWP نخ قابل اجرایی را پیدا کرد، متن را به آن نخ سوئیچ می کند. ممکن است در این فاصله زمانی، LWPهای دیگری هم در جستجوی نخ های قابل اجرای دیگر باشند. چنانچه لازم باشد نخی روی یک متغیر انحصار متقابل یا روی یک متغیر شرطی مسدود شود، مدیریت اجرایی لازمه را انجام داده و نهایتاً روال زمان بندی را فراخوانی می کند. با پیدا شدن نخ قابل اجرای دیگر، تغییر متن به آن انجام خواهد شد. زیبایی همه اینها در این است که نیازی به مطلع ساختنLWP اجراکننده نخ وجود ندارد ؛ یعنی تغییر متن تماماً در فضای کاربر انجام شده و برای LWP بصورت یک کد برنامه عادی به نظر می رسد.

حال ببینیم با فراخوانی سیستمی مسدود شونده توسط نخ چه اتفاقاتی رخ می دهد. در این جا حالت اجرا از حالت کاربر به حالت هسته اصلی تغییر می یابد، اما بازهم در متن LWP فعلی ادامه می یابد. در نقطه ای که LWP فعلی دیگر قادر به ادامه کار نباشد، ممکن است سیستم عامل تصمیم به تغییر متن به LWP دیگری بگیرد. این امر ضمناً به معنای بازگشت حالت اجرایی به حالت کاربر است. LWP انتخاب شده درست از جایی که قبلاً قطع شده بود، ادامه خواهد یافت.

استفاده از صورت ترکیبی LWPها و بسته نخ سطح کاربر مزایای متعددی دارد. اولاً، ایجاد، از بین بردن و همگام سازی نخ ها نسبتاً کم هزینه بوده و به هیچ وجه نیازمند دخالت هسته اصلی نمی باشد. ثانیاً، مشروط بر این که فرآیندی LWPهای کافی در اختیار داشته باشد، فراخوانی سیستمی مسدود شونده باعث تعلیق کل فرآیند نخواهد شد. ثالثاً، نیازی به اطلاع برنامه کاربردی از LWPها وجود ندارد ، بلکه این برنامه فقط با نخ های سطح کاربر سروکار دارد. رابعاً، می توان با اجرای LWPهای مختلف بر روی پردازنده های مختلف، به راحتی از آنها در محیط های چندپردازنده ای استفاده نمود. این

چندپردازشی می تواند کاملاً از برنامه کاربردی مخفی باشد. تنها نقطه ضعف ترکیب فرآیندهای سبک وزن با نخ های سطح کاربر این است که همچنان نیازمند ایجاد و از بین بردن LWPها هستیم که مانند نخ های سطح هسته اصلی هزینه زیادی دارد. با این وجود، ایجاد و از بین بردن LWPها فقط به ندرت انجام شده و غالباً بطورکامل تحت کنترل سیستم عامل قرار دارد.

بعنوان روش دیگر، که مشابه رویکرد فرآیندهای سبک وزن می باشد، می توان به استفاده از **فعال سازی زمان بند**[۳۱۴] اشاره کرد (۱۹۹۱، Andersonوگروه همکاران). عمده ترین تفاوت بین فعال سازی زمان بند و LWP آن است که وقتی یک نخ در اثر فراخوانی سیستم مسدود می شود، هسته اصلی روال زمان بند را جهت انتخاب نخ قابل اجرای بعدی فراخوانی می کند. همین روال در صورت خارج شدن از انسداد تکرار خواهد شد. مزیت روش مذکور آن است که مدیریت LWP بوسیله هسته اصلی انجام نمی شود. با این وجود، استفاده از روش فراخوانی به سمت بالا[۳۱۵] خیلی جالب نیست، زیرا باعث برهم خوردن ساختار سیستم های لایه بندی می شود که در این سیستم ها فقط فراخوانی به لایه سطح پایین تر بعدی مجاز است.

۲-۱-۳- نخ ها و سیستم های توزیعی

یکی از مهمترین ویژگی های نخ ها آن است که می توانند بدون مسدود کردن کل فرآیندی که نخ در آن درحال اجراست، راهکار مناسبی جهت فراخوانی های سیستمی مسدود شونده مهیا کنند. این ویژگی باعث جذابیت ویژه کاربرد نخ ها در سیستم های توزیعی می شود، چون به این ترتیب بیان ارتباط به صورتی همزمان با حفظ اتصالات منطقی متعدد بسیار آسانتر خواهد شد. در ادامه سعی خواهیم کرد تا با بررسی دقیقتر مشتری ها و سرورهای چندنخی این موضوع را تشریح کنیم.

مشتری های چندنخی

ممکن است لازم باشد تا برای دستیابی به شفافیت توزیعی بیشتر، سیستم های توزیعی که در شبکه های حوزه گسترده عمل می کنند ، زمان های انتشار پیام های میان فرآیندی طولانی را مخفی کنند. تأخیر سفر گردشی[۳۱۶] در شبکه حوزه گسترده ممکن است به اندازه صدها میلی ثانیه و حتی صدها ثانیه باشد.

روش معمول در مخفی سازی تأخیرهای ارتباطی، آغاز ارتباط و ادامه بی درنگ کار دیگر است. یکی از کاربردهای این حالت در جستجوگرهای وب است. در بسیاری از موارد، سند وب متشکل از یک فایل HTML است که شامل یک متن عادی به همراه مجموعه ای از تصاویر، آیکن ها و غیره می باشد . برای واکشی هریک از مؤلفه های سند وب، جستجوگر باید با ایجاد یک اتصال TCP/IP داده های ورودی را بخواند و آنرا به مؤلفه صفحه نمایش عبور دهد. ایجاد اتصال و خواندن داده های ورودی ذاتاً

[314] Scheduler Activation
[315] Upcall
[316] Round-trip delay

عملیات مسدود شونده هستند.در هنگام بحث درمورد ارتباط در مسافت طولانی هم با این اشکال مواجه هستیم که ممکن است زمان تکمیل هر یک از عملیات نسبتاً طولانی باشد.

جستجوگرهای وب غالباً کار را با واکشی صفحه HTML آغاز و سپس آنرا نمایش می دهند. جهت مخفی سازی حداکثری تأخیرهای ارتباطی، برخی جستجوگرها کار نمایش داده ها را همزمان با ورود آنها آغاز می کنند. وقتی متن به همراه امکانات مرور کردن و دیگر امکانات در اختیار کاربر قرار گرفت ، جستجوگر کار را با واکشی دیگر فایل های سازنده صفحه از قبیل تصاویر ادامه خواهد داد. تصاویر بصورت همزمان با ورود نمایش داده می شوند. بنابراین دیگر لازم نیست که کاربر منتظر بماند تا تمامی مؤلفه های کل صفحه واکشی شده و سپس صفحه در اختیار او قرار بگیرد.

در عمل شاهد بوده ایم که جستجوگرهای وب چند کار را به صورت همزمان انجام می دهند. بنابراین، ایجاد جستجوگر بعنوان یک مشتری چندنخی شده تاحدود زیادی باعث تسهیل مسائل خواهد شد. به محض واکشی فایل HTML اصلی، می توان جهت واکشی سایر بخش ها، نخ های دیگری را هم فعال نمود. هر یک از نخ ها اتصال جداگانه ای را با سرور برقرار کرده و داده ها را به داخل آن می آورد. ایجاد اتصال و خواندن داده ها از سرور را می توان بکمک فراخوانی های سیستمی استاندارد (مسدود شونده) و با این فرض که فراخوانی مسدود شده باعث تعلیق کل فرآیند نخواهد شد، برنامه ریزی نمود. همان طور که در مرجع Stevens(۱۹۹۸) هم تشریح شده، تمامی نخ ها ازکد واحد و، مهمتر از همه این که، ساده ای برخوردار هستند. در همین حال، کاربر فقط متوجه تأخیر در نمایش تصاویر و نظایر آن شده و می تواند اقدام به جستجو در کل سند نماید.

مزیت مهم دیگری در استفاده از جستجوگرهای چند نخی وب آن است که می توان چندین اتصال را به صورت همزمان باز نمود. در مثال قبل، چندین اتصال به یک سرور واحد انجام شده بود. چنانچه این سرور دارای بار زیاد و یا بسیار کند باشد، روش چند نخی هیچ بهبود واقعی در کارآیی نسبت به دستیابی ترتیبی به فایل ها نخواهد داشت.

با این وجود، در بسیاری از موارد، سرورهای وب در بین چندین ماشین تکرار شده و سرورهای هریک از آنها مجموعه دقیقاً واحدی از اسناد وب را مهیا می کند . سرورهای تکرار شده در سایت واحدی قرار داده شده و با نام واحدی هم خوانده می شوند. وقتی درخواستی در رابطه با یک صفحه وب وارد می شود، این درخواست غالباً براساس راهکار نوبت گردشی[۳۱۷] یا روش متعادل کننده بار دیگری (۱۹۹۴، Katz و گروه همکاران) به یکی از سرورها فرستاده می شود. در صورت استفاده از مشتری چند نخی ممکن است اتصالاتی به کپی های مختلف ایجاد شود. در نتیجه داده ها بصورت موازی انتقال پیدا کرده و همین امر به نحو چشمگیری باعث کاهش کلی در زمان نمایش سند وب در قیاس با سرور تکرار نشده خواهد شد. استفاده از این شیوه فقط در صورتی امکانپذیر خواهد بود که مشتری بتواند با

[317] round-robin

جریانات واقعاً موازی داده های ورودی کار کند. برای این منظور نخ ها بهترین و مناسب ترین گزینه خواهند بود.

سرورهای چندنخی

هر چند مشتری های چند نخی مزایای قابل ملاحظه ای دارند، اما همانطور که مشاهده کردیم، کاربرد عمده چندنخی شدن در سیستم های توزیعی در طرف سرور می باشد. در عمل مشاهده شده که چندنخی شدن علاوه بر ساده سازی چشمگیر کد سرور، باعث سهولت فراوان ایجاد سرورهایی می شود که از ویژگی موازی گرایی برای افزایش کارآیی خود، حتی در سیستم های تک پردازنده ای، استفاده می کنند. بااین وجود، حال که کامپیوترهای چندپردازنده ای در نقش ایستگاههای کاری چند منظوره کاربرد گسترده ای پیدا کرده اند، چندنخی شدن جهت موازی گرایی بیش از پیش کاربرد و پیدا کرده است.

برای درک مزایای استفاده از نخ ها در نوشتن کد سرور، سازمان سرور فایلی را در نظر بگیرید که برحسب مورد باید برای انتظار بر روی دیسک مسدود شود. سرور فایل معمولاً منتظر ورود درخواست برای عملیات فایل باقی می ماند. پس از ورود درخواست آن را اجرا کرده و در پایان پاسخ را باز می گرداند. یک سازمان ممکن و بسیار متداول در شکل ۳-۳ نمایش داده شده است. در اینجا یک نخ به نام **پخش کننده**[۳۱۸] اقدام به خواندن درخواست های ورودی برای عملیات فایل می کند. درخواست ها بوسیله مشتریان به نقطه انتهایی[۳۱۹] شناخته شده ای برای سرور مذکور ارسال می شوند. پس از بررسی درخواست، سرور یک **نخ کارگر**[۳۲۰] بی کار را انتخاب و درخواست را به آن تقدیم می کند.

شکل۳-۳-یک سرور چند نخی که به شکل مدل پخش-کننده/ کارگر سازمان داده شده است(نخ پخش کننده، درخواست داده شده به نخ کارگر، سرور، نخ کارگر، درخواست ورودی از شبکه، سیستم عامل).

کارگر با انجام خواندن مسدود شونده روی سیستم فایل *محلی* ادامه می دهد، و اینکار ممکن است باعث تعلیق نخ تا زمان واکشی داده ها از دیسک شود. در صورت معلق بودن نخ، نخ دیگری برای اجرا انتخاب خواهد شد. بعنوان مثال، ممکن است پخش کننده برای انجام کار بیشتر انتخاب شود. البته ممکن است نخ کارگر دیگری که هم اکنون آماده اجرا باشد انتخاب شود.

حال ببینیم سرور فایل در غیاب نخ ها چگونه نوشته می شود. یک راه حل این است که سرور بعنوان تک نخ عمل کند. حلقه اصلی سرور فایل درخواست را گرفته، آنرا بررسی می کند و پیش از گرفتن درخواست بعدی اقدام به تکمیل آن می کند. سرور در حین انتظار برای دیسک، بیکار بوده و هیچ درخواست دیگری را پردازش نمی کند. در نتیجه، نمی توان روی درخواستهای مشتریان دیگر کارکرد.

[318] dispatcher
[319] end point
[320] worker thread

بعلاوه، چنانچه سرور فایل روی ماشین اختصاص یافته در حال اجرا باشد - که معمولاً هم همینطور است – در فاصله زمانی که سرور فایل منتظر دیسک است، پردازنده کاملاً بیکار خواهد بود. درنتیجه ، تعداد درخواست های پردازش شده در ثانیه افت خواهد کرد. بنابراین نخ ها کارآیی بالایی دارند، ولی هرنخ به صورت ترتیبی و به طور معمول برنامه ریزی خواهد شد.

بنابراین تا به اینجا با دو نوع طراحی مواجه شده ایم : سرور فایل چندنخی و سرور فایل تک نخی. اما حالتی را فرض کنید که چند نخی موجود نباشد، اما برای طراحان سیستم حالت تک نخی به دلیل افت کارآیی قابل قبول نیست. بنابراین روش سوم، راه اندازی سرور بعنوان یک ماشین حالت متناهی[۳۲۱] بزرگ است . با ورود درخواست جدید، تنها نخ موجود آنرا بررسی خواهد کرد . در صورتی که بتواند از طریق حافظه پنهان دیسک به درخواست رسیدگی کند که هیچ، و الاّ پیامی برای دیسک ارسال می کند.

به جای مسدود شدن، سرور وضعیت درخواست فعلی را در جدولی ثبت کرده و به سراغ پیام بعدی میرود. پیام بعدی هم ممکن است یا درخواست برای کار جدید باشد و یا پاسخی از جانب دیسک درمورد عملیات قبلی. اگر درخواست کار جدید باشد، کار آغاز خواهد شد. اگر پاسخ از دیسک باشد، اطلاعات مربوطه از جدول واکشی شده، پاسخ پردازش شده و سپس برای مشتری ارسال می گردد. در این طرح، سرور مجبور خواهد بود که از فراخوانی های غیر مسدود شونده **send** و **receive** استفاده کند.

در این طرح، مدل "فرآیند ترتیبی" موجود در دو مورد اول دیگر بچشم نمی خورد. وضعیت محاسبه بایستی برای تمامی پیامهای ارسالی و دریافتی به صورت آشکار در جدول ذخیره و بازیابی شود. در واقع، ما نخ ها و پشته های آنها را با دشواری شبیه سازی کرده ایم. این فرآیند بصورت ماشین حالت متناهی عمل می کند، یعنی یک رویداد راگرفته و بسته به محتویات آن، در برابر آن واکنش نشان می دهد.

مدل	ویژگی ها
چند نخ	موازی گرایی، فراخوانی های سیستمی مسدود شونده
فرآیند تک نخی	غیر موازی ، فراخوانی های سیستمی مسدود شونده
ماشین حالت محدود	موازی گرایی، فراخوانی های سیستمی غیرمسدود شونده

شکل ۴-۳- سه روش برای ساخت سرور

نخ ها این امکان را می دهند ایده فرآیند های ترتیبی که به وسیله فراخوان های سیستمی مسدود شونده انجام می شوند (مثلاً RPC برای ارتباط با دیسک) حفظ شود و همچنان موازی سازی به دست

[321] Finite-State Machine

آید. فراخوان های سیستمی مسدود شونده برنامه نویسی را تسهیل کرده و موازی گرایی هم باعث ارتقاء کارآیی خواهد شد. درسرور تک نخی سهولت و سادگی فراخوانی های سیستمی مسدود شونده حفظ شده، اما کارآیی کاهش پیدا می کند. در روش ماشین حالت متناهی، موازی گرایی باعث ارتقاء کارآیی شده ولی به کار گیری فراخوان های غیر مسدود شونده برنامه نویسی را با برخی پیچیدگی ها مواجه می سازد. این مدل ها بطور خلاصه در شکل ۴-۳ ارائه شده اند.

۲-۳- مجازی سازی[۳۲۲]

نخ ها و فرآیندها را می توان بعنوان روشی برای افزایش کمیّت کار در زمان یکسان تلقی نمود. در واقع، بکمک آنها می توانیم (قطعاتی از) برنامه هایی را بسازیم که ظاهراً بصورت همزمان اجرا می شوند. البته ، در یک کامپیوتر تک پردازنده ای، اجرای همزمان فقط تصور می شود. از آنجا که فقط یک پردازنده وجود دارد، در هر زمان فقط یک دستورالعمل از یک فرآیند یا یک نخ قابل اجراست. با سوئیچ کردن سریع بین نخ ها و فرآیندها تصور موازی گرایی ایجاد می شود.

این تفکیک بین داشتن یک پردازنده و تظاهر به وجود بیش از یکی را می توان به منابع دیگر هم بسط داده و در نتیجه به چیزی به نام **مجازی سازی منبع**[۳۲۳] دست یافت. هرچند مفهوم مجازی سازی از چند دهه پیش مورد استفاده قرار گرفته است ، اما با افزایش کاربرد و پیچیدگی سیستم های کاربردی (توزیعی) علاقمندی جدیدی به آن مشاهده می شود، زیرا نرم افزار برنامه های کاربردی در اکثریت قریب به اتفاق موارد بیشتر از سخت افزار و نرم افزار سیستم های زیربنایی خود عمر می کنند. در این بخش، تا حدودی به نقش مجازی سازی و نحوه فعلیت بخشیدن به آن خواهیم پرداخت.

۱-۲-۳- نقش مجازی سازی در سیستم های توزیعی

در عمل، تمامی سیستم های کامپیوتری (توزیعی)، مطابق شکل الف) ۵-۳، واسط برنامه ریزی را برای نرم افزار سطح بالاتر ارائه می دهند. واسط ها انواع مختلفی دارند که از آن میان می توان به مجموعه دستورالعمل پایه که بوسیله پردازنده ارائه می شود تا مجموعه گسترده ای از واسط های برنامه نویسی کاربردی در بسیاری ازسیستم های میان افزار امروزی اشاره کرد. بطور خلاصه و مطابق شکل ب) ۵-۳، مجازی سازی به توسعه یا تعویض واسط فعلی با هدف تقلید رفتار سیستم دیگر می پردازد. در ادامه راجع به جزئیات فنی مجازی سازی بحث خواهیم کرد، اما اجازه دهید پیش ازآن نگاهی داشته باشیم به دلیل اهمیت مجازی سازی در سیستم های توزیعی.

[322] virtualization
[323] resource virtualization

یکی از مهمترین دلایل برای معرفی مجازی سازی در سالهای دهه هفتاد ایجاد امکان اجرا شدن نرم افزار باقیمانده از گذشته[۳۲۴] بر روی سخت افزار کامپیوتر های بزرگ[۳۲۵] گران قیمت بود. این نرم افزار علاوه بر برنامه های کاربردی مختلف، شامل سیستم عامل های ایجاد شده برای آنها نیز بود. این شیوه پشتیبانی از نرم افزار باقیمانده از گذشته بطور موفقیت آمیزی در کامپیوتر های بزرگ IBM 370 (و جانشینان آنها) اجرا شده و در نتیجه ماشین مجازی بوجود آمد که سیستم عامل های مختلف به آن انتقال داده می شدند .

هم زمان با کاهش قیمت سخت افزار، کامپیوترها هم قدرتمندتر شده، تنوع سیستم های عامل کاهش یافت و مجازی سازی دیگر مانند قبل مورد توجه نبود. با این وجود، به چند دلیل که در ادامه خواهیم گفت ، از اواخر سال های دهه نود شرایط مجدداً دستخوش تغییراتی شد.

شکل ۵-۳- الف) سازمان عمومی بین برنامه، واسط و سیستم.ب) سازمان عمومی مجازی سازی سیستم A در بالای سیستم B (داخل شکل: الف) برنامه، واسط A، سخت افزار/ نرم افزار سیستم A، ب) برنامه، واسطA، پیاده سازی تقلید A روی B، واسطB، سخت افزار/ نرم افزار سیستم B).

اول، هرچند سخت افزار و نرم افزار سیستم های سطح پایین با سرعت زیادی تغییر می کنند، نرم افزار در سطوح بالاتر انتزاع (بعنوان مثال، میان افزار و برنامه های کاربردی)، ثبات خیلی بالاتری دارند. به بیان دیگر، نمی توان نرم افزار باقیمانده از گذشته را بر روی سکوهای گذشته نگهداری کرد. در اینجا، مجازی سازی می تواند بصورت انتقال واسط های باقیمانده از گذشته به سکوهای جدید و سپس، استفاده فوری از آنها برای گروه های بزرگی از برنامه های موجود مفید واقع شود.

یک واقعیت مهم دیگر این است که شبکه سازی کاملاً فراگیر شده است. نمی توان تصور کرد که کامپیوترهای امروزی به هیچ شبکه ای متصل نباشند. در عمل، این امر مستلزم آن است که مدیران اجرایی سیستم ها مجموعه ای بزرگ و ناهمگن[۳۲۶] از کامپیوترهای سرور در اختیار داشته باشند که هریک برنامه های کاربردی بسیار مختلفی را اجرا کرده، و مشتری ها به راحتی بتوانند به این برنامه های کاربردی دسترسی پیدا کنند. همچنین این برنامه های کاربردی باید به راحتی به منابع مختلف دسترسی داشته باشند. مجازی سازی می تواند بسیار مفید باشد، چون می تواند تنوع سکوها و ماشینها را کاهش دهد، به این ترتیب که هر یک از برنامه های کاربردی روی ماشین مجازی ویژه خود، که احتمالاً شامل کتابخانه های مربوطه و سیستم عامل می باشد، که به نوبه خود روی یک سکوی مشترک قرار دارند، اجرا می شود.

[324] legacy
[325] mainframe computers
[326] Heterogeneous

نوع آخر مجازی سازی جابجا پذیری[۳۲۷] و انعطاف پذیری بالایی را ایجاد می کند. بعنوان مثال، Awadallah and Rosenblum(۲۰۰۲) بحث می کنند که برای تحقق بخشیدن به ایده شبکه های تحویل محتوا[۳۲۸] که براحتی قادر به پشتیبانی از کپی برداری محتوای پویا باشند، اگر از سرورهای لبه ای[۳۲۹] که از مجازی سازی پشتیبانی کرده و در نتیجه برای کل سایت (شامل محیط آن) امکان کپی شدن فعالانه را فراهم آورند استفاده شود، شاهد تسهیل چشمگیر مدیریت خواهیم بود. همانطور که بعداً هم خواهیم گفت، مباحث جابجا پذیری مجازی سازی را تبدیل به یکی از مکانیزم های مهم در سیستم های توزیعی نموده است.

۲-۲-۳- معماری های ماشین های مجازی

برای تحقق مجازی سازی روش های بسیار مختلفی وجود دارد. Smith وNair (2005) مروری بر این روش ها ارائه داده اند. برای درک تفاوت ها در مجازی سازی، لازم است بدانیم که سیستم های کامپیوتری غالباً چهار نوع واسط مختلف در چهار سطح مختلف ارائه می دهند:

۱. یک واسط بین سخت افزار و نرم افزار، شامل **دستورالعمل های ماشین** که ازطریق هر برنامه ای قابل اجرا است.
۲. یک واسط بین سخت افزار و نرم افزار ، شامل دستورالعمل های ماشین که فقط توسط برنامه های ممتاز[۳۳۰] از قبیل سیستم عامل قابل اجرا است.
۳. یک واسط شامل **فراخوان های سیستمی** که توسط سیستم عامل ارائه می شود.
۴. یک واسط شامل فراخوان های کتابخانه ای که غالباً باعث ایجاد چیزی به نام **واسط برنامه نویسی کاربردی**[۳۳۱]**(API)** می شود. در بسیاری از موارد، فراخوان های سیستمی بوسیله یک API مخفی می شوند.

این انواع مختلف در شکل ۶-۳ نمایش داده شده است . بطور خلاصه، هدف از مجازی سازی تقلید رفتار این واسط ها است.

شکل ۶-۳- واسط های مختلف ارائه شده توسط سیستم های کامپیوتری(برنامه کاربردی، توابع کتابخانه، کتابخانه، فراخوان های سیستمی، سیستم عامل، دستور العملهای ممتاز، دستورالعملهای عمومی، سخت افزار).

مجازی سازی ممکن است به دو روش مختلف انجام شود. اولاً، می توانیم یک سیستم زمان اجرا[۳۳۲] بسازیم که اساساً مجموعه دستورالعمل انتزاعی را برای اجرای برنامه های کاربردی ارائه دهد. این

[327] Portability
[328] Content Delivery Networks
[329] Edge Servers
[330] Priviledged
[331] Application Programming Interface
[332] Runtime System

دستورالعمل ها می توانند ترجمه شوند (مانند محیط زمان اجرای Java)، همچنین می توانند تقلید شوند مشابه کاری که در اجرای برنامه های کاربردی ویندوز روی سکوهای UNIX انجام می شود. توجه داشته باشید که در این حالت، مقلد[۳۳۳] مجبور خواهد بود تا رفتار فراخوان های سیستمی را نیز تقلید کند که در عمل چندان هم ساده نیست. این نوع مجازی سازی منجر به چیزی می شود که Smithو Nair (۲۰۰۵) از آن با عنوان **ماشین مجازی فرآیند**[۳۳۴] یاد کرده اند و تأکیدی است بر این واقعیت که مجازی سازی اساساً فقط برای یک فرآیند انجام می شود.

یک روش دیگر برای مجازی سازی، ایجاد سیستمی است که اساساً بعنوان لایه ای پیاده سازی شده است که سخت افزار اصلی را می پوشاند و مجموعه دستورالعمل کاملی از همان نوع (یا سخت افزار دیگر) را بعنوان واسط ارائه می کند. نکته مهم آن است که این واسط را می توان *بصورت همزمان* به برنامه های مختلف ارائه کرد. در اینصورت ، امکان اجرای مستقل و همزمان چندین سیستم عامل مختلف روی یک سکو واحد فراهم خواهد آمد. این لایه غالباً به نام **ناظر ماشین مجازی**[۳۳۵]**(VMM)** خوانده می شود. از نمونه های این روش می توان به VMware (۲۰۰۱، Sugermanو گروه همکاران) و Xen (۲۰۰۳،Barham و گروه همکاران) اشاره کرد. این دو رویکرد مختلف در شکل ۷-۳ نمایش داده شده است.

شکل ۷-۳- الف) یک ماشین مجازی فرآیند همراه با چند نمونه از ترکیبات (برنامه کاربردی، سیستم زمان اجرا). ب) ناظر ماشین مجازی همراه با چند نمونه های از ترکیبات (برنامه های کاربردی، سیستم عامل)(داخل شکل: الف)برنامه کاربردی، سیستم زمان اجرا، سیستم عامل، سخت افزار، ب) برنامه های کاربردی، سیستم عامل، ناظر ماشین مجازی، سخت افزار).

همانطور که توسط (۲۰۰۵) Rosenblumو Garfinkel هم ارائه شده، اهمیت VMMها در بافت قابلیت وثوق و امنیت سیستم ها(ی توزیعی) به صورت روزافزونی افزایش خواهد یافت. همچنین، از آنجایی که VMMها باعث تفکیک کل برنامه کاربردی از محیط آن می شوند، خرابی ناشی از خطا یا حملات امنیتی دیگر بر کل ماشین تأثیرگذار نخواهد بود. بعلاوه، همانطور که قبلاً هم گفتیم، VMMها با فراهم آوردن امکان تفکیک بیشتر بین سخت افزار و نرم افزار و امکان انتقال کامل محیط از یک ماشین به ماشین دیگر، باعث افزایش چشمگیر جابجا پذیری می شوند.

[333] Emulator
[334] Process Virtual Machine
[335] virtual machine monitor

۳-۳- مشتری ها

در فصلهای قبل راجع به مدل مشتری- سرور ، نقش های مشتری ها و سرورها و روشهای تعامل آنها صحبت کردیم. حال بیایید نگاه دقیقتری به ساختار مشتری و سرور داشته باشیم . این بخش را با بحث در مورد مشتری ها آغاز کرده و سرورها را به بخش بعد موکول می کنیم.

۱-۳-۳- واسط های کاربر شبکه ای شده[۳۳۶]

یکی از مهمترین وظایف ماشین های مشتری، ایجاد ابزار لازم برای تعامل کاربران با سرورهای دور است. برای پشتیبانی از این تعامل دو راه وجود دارد: اولاً ، به ازای هر سرویس از راه دور، ماشین مشتری یک نظیر جداگانه داشته باشد تا بتواند روی شبکه با سرویس مورد نظر تماس برقرار نماید. بعنوان نمونه می توان به دستور کاری[۳۳۷] اشاره کرد که روی PDA کاربر اجرا شده و بایستی با یک دستور کار دور، و احتمالاً اشتراکی دیگر همگام شود. در این حالت و مطابق شکل الف) ۸-۳، یک پروتکل در سطح برنامه کاربردی[۳۳۸] کار همگام سازی را انجام خواهد داد.

شکل ۸-۳- الف) یک برنامه کاربردی شبکه ای با پروتکل خود.ب) یک راه حل کلی برای ایجاد دسترسی به برنامه های کاربردی دور(داخل شکل: الف) ماشین مشتری، برنامه کاربردی، میان افزار، سیستم عامل محلی، پروتکل ویژه برنامه کاربردی، ماشین سرور، برنامه کاربردی، میان افزار، سیستم عامل محلی، شبکه، ب) ماشین مشتری، برنامه کاربردی، میان افزار، سیستم عامل محلی، پروتکل مستقل از برنامه کاربردی، ماشین سرور، برنامه کاربردی، میان افزار، سیستم عامل محلی، شبکه).

راه حل دوم، ایجاد دسترسی مستقیم به سرویسه ای دور صرفاً با استفاده از یک واسط کاربر کارآمد است. در عمل این بدان معناست که ماشین مشتری صرفاً بعنوان پایانه ای بدون نیاز به ذخیره سازی محلی بکار برده شده و در نتیجه، راه حل مستقل از برنامه کاربردی مطابق شکل ب) ۸-۳ بوجود خواهد آمد. در واسط های کاربر شبکه ای همه چیز در طرف سرور پردازش و ذخیره می شود. همراه با افزایش امکان اتصال به اینترنت و پیچیده تر شدن وسایل دستی، رویکرد **روش مشتری لاغر**[۳۳۹] بیش از پیش مورد توجه قرار گرفته است. همانطور که در فصل قبل هم گفتیم، دلیل دیگر برای کاربرد فراوان راه حل های مشتری لاغر آن است که باعث تسهیل مدیریت سیستم می شوند. در ادامه نگاهی خواهیم داشت به نحوه پشتیبانی از واسط های کاربر شبکه ای.

---

[336] networked user interfaces
[337] agenda
[338] application-level protocol
[339] thin-client approach

مثال: سیستم X Window

احتمالاً یکی از قدیمی ترین و درعین حال پرکاربردترین واسط های کاربر شبکه ای شده ، **سیستم X Window** می باشد. سیستم X Window، که غالباً فقط به نام X خوانده می شود، برای کنترل پایانه های نگاشت بیتی[۳۴۰] بکار می رود که شامل یک صفحه نمایش، صفحه کلید و ابزار اشاره مانند ماوس است. از یک نظر، X را می توان بخشی از سیستم عامل تلقی نمود که پایانه را کنترل می کند. قلب سیستم از چیزی به نام **هسته اصلیX**[۳۴۱] تشکیل شده که شامل تمام درایورهای وسائل ترمینال بوده که معمولاً به شدت به سخت افزار وابسته می باشند.

هسته اصلیX واسط نسبتاً سطح پایینی را برای کنترل صفحه نمایش و همچنین گرفتن رویدادها از صفحه کلید و موس ارائه می دهد. مطابق شکل ۹-۳ ، این واسط بصورت کتابخانه ای به نام *Xlib* در اختیار برنامه های کاربردی قرار می گیرد.

شکل ۹-۳-سازمان اصلی سیستمX Window(داخل شکل: سرور برنامه کاربردی، مدیر پنجره، Xlib، سیستم عامل محلی، سرور برنامه کاربردی، برنامه کاربردی، واسط Xlib، Xlib، سیستم عامل محلی، پروتکل X، پایانه کاربر، هسته اصلیX ، درایور های دستگاه ، پایانه (شامل صفحه نمایش، صفحه کلید، ماوس و غیره).

یکی از نکات جالب توجه در مورد X آن است که هسته اصلیX و برنامه های کاربردی X نباید لزوماً روی یک ماشین واحد مقیم باشند. خصوصاً اینکه ، X پروتکل ارتباطی سطح برنامه کاربردی به نام **پروتکل X** [۳۴۲] ارائه می دهد که بوسیله یک مورد از *Xlib* ،قادر به تبادل داده ها و رویدادها با هسته اصلیX خواهد بود. بعنوان مثال، *Xlib* قادر است تا برای ایجاد یا حذف پنجره ، تنظیم رنگها و تعریف نوع شاخص صفحه نمایش و بسیاری از درخواست های دیگر، اقدام به ارسال درخواست به هسته اصلی X نماید. هسته اصلیX هم متقابلاً با بازگرداندن بسته های رویداد به *Xlib* ، در برابر وقایع محلی از قبیل ورودی ماوس و صفحه کلید واکنش نشان خواهد داد.

امکان ارتباط همزمان چندین برنامه کاربردی با هسته اصلیX فراهم شده است. یکی از برنامه های کاربردی بنام **مدیر پنجره**[۳۴۳] حق ویژه ای در این رابطه دارد. این برنامه کاربردی قادر است تا" ظاهر و حس" صفحه نمایش را به همان صورتی که به نظر کاربر می رسد، مشخص کند. بعنوان مثال، مدیر پنجره می تواند تعیین کند که هریک از پنجره ها با استفاده از دکمه های اضافی چگونه تزئین شوند، پنجره ها چگونه روی صفحه نمایش قرار گیرند وغیره. دیگر برنامه های کاربردی هم باید از این قوانین تبعیت کنند.

---

340 bit-mapped terminals
341 X kernel
342 X protocol
343 window manager

بهتر است درمورد تناسب عملی سیستم X Window در سیستم مشتری- سرور هم نکته ای را بیان کنیم. از آنچه تا به اینجا گفته شد می توان نتیجه گرفت که هسته اصلیX درخواست هایی را برای دستکاری[۳۴۴] صفحه نمایش دریافت می کند. این درخواست ها از برنامه های کاربردی(احتمالاً دور) دریافت می شوند. در این حالت، هسته اصلیX بعنوان سرور عمل خواهد کرد، در حالی که برنامه های کاربردی در نقش مشتری ظاهر می شوند. از این واژه گزینی در X استفاده می شود، و اگرچه صحیح است ولی ممکن است باعث ابهام شود.

مشتری لاغر در کاربردهای شبکه ای

برنامه های کاربردی بکمک فرامین خاص صفحه نمایش که توسط X ارائه می شود، صفحه نمایش را دستکاری می کنند. این فرامین غالباً از طریق شبکه ارسال و سپس به وسیله هسته اصلیX اجرا می شوند. برنامه های کاربردی نوشته شده برای X باید منطق برنامه کاربردی را از فرامین واسط کاربر تفکیک کنند. متأسفانه غالباً این حالت اتفاق نمی افتد. در نتیجه، همانطور که (۲۰۰۲) Lai and Nieh هم گزارش داده اند، قسمت عمده منطق برنامه کاربردی و تعامل کاربر ارتباط محکمی با هم دارند؛ به این معنی که برنامه کاربردی درخواست های زیادی را برای هسته اصلی X ارسال کرده و در عوض انتظار دارد تا پیش از اقدام به مرحله بعد، پاسخی از جانب آن دریافت کند. این نوع رفتار همگام، در صورت اعمال روی یک شبکه حوزه گسترده با تأخیرهای طولانی ، ممکن است تأثیرات زیادی بر عملکرد داشته باشد.

راه حل های مختلفی برای این مشکل وجود دارد . یک روش، طراحی مجدد پیاده سازی پروتکل X است، همان گونه که در NX (Pinzari،۲۰۰۳) انجام شده است. یک بخش مهم این کار، کاهش پهنای باند از طریق متراکم سازی پیامهای X است. در ابتدا فرض می شود که پیام ها از یک بخش ثابت، که بعنوان شناسه تلقی می شود، و یک بخش متغیر تشکیل شده اند. در بسیاری از موارد، چندین پیام یک شناسه واحد دارند که در اینصورت، غالباً حاوی داده های مشابهی می باشند . از این ویژگی می توان فقط برای ارسال تفاوتها بین پیامهای دارای شناسه واحد استفاده کرد.

هر دو طرف فرستنده و گیرنده دارای حافظه پنهان محلی هستند که می توان با استفاده از شناسه پیام، اقدام به جستجو در درایه[۳۴۵] های آن نمود. پس از ارسال پیام، جستجو ابتدا در حافظه پنهان محلی انجام می شود. اگر در آنجا یافت شود، به این معناست که قبلاً پیامی با همان شناسه، اما با داده های احتمالاً متفاوت، ارسال شده بود. در این حالت، برای ارسال صرفاً تفاوتهای بین ایندو از کدگذاری تفاضلی[۳۴۶] استفاده می شود. در طرف دریافت کننده ، پیام در حافظه پنهان محلی هم جستجو می شود و سپس ، رمزگشایی از طریق تفاوتها صورت می پذیرد. در صورت یافت نشدن در حافظه مخفی، از

[344] Manipulate
[345] Entry
[346] differential encoding

روشهای متراکم سازی استانداردی استفاده می شود که غالباً باعث بهبود ضریب چهاری پهنای باند می شود. در مجموع، گزارش شده که این تکنیک منجر به کاهش پهنای باند تا ۱۰۰۰ برابر شده و در نتیجه، X از طریق پیوندهای با پهنای باند کم تا kbps ۹۶۰۰ هم قابل اجرا خواهد بود.

یکی از اثرات جانبی مهم ذخیره پیام ها در حافظه پنهان آن است که فرستنده و گیرنده اطلاعات مشترکی درمورد وضعیت فعلی صفحه نمایش خواهند داشت. بعنوان مثال، یک برنامه کاربردی می تواند صرفاً با درخواست جستجوها در حافظه پنهان محلی ، اطلاعات هندسی درباره اشیاء مختلف درخواست نماید. با در اختیار داشتن صرفاً همین اطلاعات اشتراکی، تعداد پیامهای لازم جهت همگام نگه داشتن برنامه کاربردی و صفحه نمایش کاهش خواهد یافت.

علیرغم تمام این تغییرات، هنوز هم X نیازمند داشتن یک سرور صفحه نمایش در حال اجراست. این امر ممکن است زیاد به نظر برسد، خصوصاً در مواردی که صفحه نمایش یک تلفن همراه ساده باشد. یک راه حل برای حفظ سادگی نرم افزار در صفحه نمایش آن است که تمام پردازش ها در طرف برنامه کاربردی انجام شود. در عمل این بدان معناست که کل صفحه نمایش تا سطح پیکسل در طرف برنامه کاربردی کنترل می شود. سپس تغییرات نگاشت بیتی از طریق شبکه به صفحه نمایش ارسال و از آنجا فوراً به حافظه میانگیر فریم محلی[۳۴۷] انتقال داده می شوند.

این رویکرد مستلزم استفاده از تکنیکهای فشرده سازی پیشرفته است تا بتواند مانع شود که در دسترس بودن پهنای باند به مشکل تبدیل شود. به عنوان مثال نمایش یک جریان ویدوئی با سرعت ۳۰ فریم در هر ثانیه روی صفحه نمایشی با ابعاد ۲۴۰× ۳۲۰ را درنظر بگیرید. صفحه نمایشی با این ابعاد در بسیاری از PDAها به چشم می خورد. چنانچه هریک از پیکسلها با ۲۴ بیت کدگذاری شود، آنگاه بدون فشرده سازی نیازمند پهنای باندی به اندازه تقریبی 53 Mbps خواهیم بود. ناگفته پیداست که در این حالت فشرده سازی ضروری است و در حال حاضر از تکنیکهای زیادی برای اینکار استفاده می شود. اما توجه داشته باشید که فشرده سازی مستلزم بازسازی در طرف مشتری است و این ممکن است در صورت عدم حمایت سخت افزاری، هزینه های محاسباتی بالایی داشته باشد. پشتیبانی سخت افزاری می تواند مهیا شود، اما باعث افزایش قیمت آن دستگاه خواهد شد.

نقطه ضعف ارسال داده های خام پیکسل در قیاس با پروتکلهای سطح بالاتری از قبیل X آن است که معنی شناسی[۳۴۸] های برنامه کاربردی در سطح ارسال داده های خام از دست می روند و امکان استفاده از آنها وجود ندارد. Baratto (۲۰۰۵) و گروه همکاران راه حل متفاوتی را پیشنهاد کرده اند. در این روش که THINC نام دارد، تعداد محدودی فرمان نمایش سطح بالا ایجاد کرده اند که در سطح درایورهای دستگاه ویدئو عمل می کنند. این فرامین، وابسته به دستگاه هستند و قدرتمندتر از عملیات داده های خام خواهند بود، و در عین حال در قیاس با آنچه پروتکلی از قبیل X ارائه می دهد، ضعیف

[347] local frame buffer
[348] Semantics

تر هستند. نتیجه اینکه سرورهای نمایش می توانند بسیار ساده تر شوند که برای میزان به کارگیری پردازنده خوب است. در عین حال، می توان از بهینه سازی های وابسته به برنامه کاربردی جهت کاهش پهنای باند و همگام سازی استفاده نمود.

در THINC، درخواستهای صفحه نمایش از برنامه کاربردی گرفته شده و به فرامین سطح پایین تر ترجمه می شوند. بکمک گرفتن درخواست های برنامه های کاربردی، THINC قادر است تا از معنی شناسی های برنامه کاربردی جهت تصمیم گیری در مورد بهترین صورت ترکیبی فرامین سطح پایین تر استفاده کند. فرامین ترجمه شده فوراً به صفحه نمایش ارسال نشده ، بلکه در عوض صف بندی می شوند. بسته بندی[۳۴۹] چندین فرمان مختلف باعث متراکم سازی فرامین صفحه نمایش به شکل یک فرمان واحد و در نتیجه کاهش تعداد پیام ها خواهد شد. بعنوان مثال، اگر پیام جدیدی در مورد نقاشی کردن در منطقه خاصی از صفحه نمایش عملاً برچیزی که فرمان (هنوز صفبندی شده) قبلی ایجاد کرده رونویسی شود ، دیگر نیازی به ارسال فرمان قبلی به صفحه نمایش وجود نخواهد داشت . نهایتاً اینکه، به جای اینکه صفحه نمایش درخواست به هنگام سازی داشته باشد، THINC همواره به هنگام سازی ها را به همان ترتیب آماده شدن ارسال خواهد کرد (هل می دهد). بعلاوه ، چون دیگر لزومی برای ارسال درخواست به هنگام سازی توسط صفحه نمایش وجود ندارد، روش هل دادن باعث کاهش تأخیر خواهد شد.

در نتیجه، روش مورد استفاده در THINC باعث افزایش عملکردکلی می شود، و هماهنگی بسیار زیادی با NX دارد. جزئیات بیشتر در مورد مقایسه عملکردها را در (۲۰۰۵) Baratto و گروه همکاران مطالعه کنید.

اسناد ترکیبی[۳۵۰]

واسط های کاربر امروزی بسیار بیشتر از آن چه به توسط سیستم هایی از قبیل X یا برنامه های کاربردی ساده آن انجام می شود کار می کنند. به خصوص، بسیاری از واسط های کاربر به برنامه های کاربردی امکان می دهند تا در یک پنجره گرافیکی واحد شریک شده و از طریق عملیات کاربر، از این پنجره جهت تبادل داده ها استفاده کنند. از دیگر کارهای قابل انجام توسط کاربر می توان به ترتیب به عملیات **کشیدن و رها کردن**[۳۵۱] و **ویرایش در محل**[۳۵۲] اشاره کرد.

بعنوان یکی از نمونه های کاربرد کشیدن و رها کردن می توان به انتقال آیکن نماینده فایل A به آیکن نماینده سطل آشغال و حذف آن اشاره کرد. در این حالت، واسط کاربر باید کاری بیش از صرفاً مرتب سازی آیکنهای روی صفحه نمایش انجام دهد؛ یعنی بایستی به محض انتقال آیکنA به بالای آیکن

[349] batching
[350] compound documents
[351] drag-and-drop
[352] in-place editing

برنامه کاربردی سطل آشغال ، نام فایل A را به برنامه کاربردی مربوط به سطل آشغال انتقال دهد. به راحتی می توانید مثال های دیگری در این رابطه ذکر کنید.

ویرایش در محل را می توان به بهترین نحو در اسنادی که شامل متن و تصاویر گرافیکی هستند مشاهده کرد. فرض کنید که سند در داخل یک پردازشگر کلمه استاندارد نمایش داده شده باشد. به محض اینکه کاربر موس را در بالای شکل قرار می دهد، واسط کاربر اطلاعات آن را به برنامه نقاشی می دهد تا کاربر بتواند شکل را اصلاح نماید. بعنوان مثال، ممکن است کاربر شکل را چرخانده باشد که اینکار بر محل قرارگیری شکل در داخل سند تأثیر خواهد گذاشت . بنابراین واسط کاربر ارتفاع و عرض شکل جدید را محاسبه کرده و این اطلاعات را به پردازشکر کلمه می دهد. به این ترتیب، پردازشگر قادر به به هنگام سازی اتوماتیک آرایش صفحه سند خواهد شد.

ایده اصلی در واسط های کاربر همان ایده **سند ترکیبی** است که بصورت مجموعه ای از اسناد احتمالاً بسیار متفاوت (از قبیل متن، شکل، صفحه گسترده و غیره) تعریف می شود که بصورتی روان در سطح کاربر- واسط ادغام شده اند. واسط کاربری که می تواند با سندهای ترکیبی کار کند این واقعیت را که برنامه های کاربردی مختلف روی بخشهای مختلف سند کار می کنند را مشخص می کند. از نظر کاربر ، تمامی بخشها به صورتی روان با هم تجمیع شده اند. چنانچه تغییر یک بخش بر بخشهای دیگر هم تأثیر بگذارد، واسط کاربر می تواند کار مناسب را انجام دهد، مثل مطلع ساختن برنامه های کاربردی.

مشابه آنچه در مورد سیستم X Window گفته شد، لزومی ندارد برنامه های کاربردی مربوط به یک سند ترکیبی روی ماشین مشتری اجرا شوند. با این وجود، باید بدانیم که واسط های کاربری که از اسناد ترکیبی پشتیبانی می کنند، ممکن است کار پردازش بیشتری نسبت به دیگرانی که فاقد این مزیت هستند انجام دهند.

۲-۳-۳- نرم افزار طرف مشتری برای شفافیت توزیعی[۳۵۳]

نرم افزار مشتری از چیزی بیش از صرفاً واسط های کاربر تشکیل شده است. در بسیاری از موارد، بخشهایی از سطح پردازش و سطح داده برنامه کاربردی مشتری- سرور نیز در طرف مشتری اجرا می شوند. ماشین های پاسخ گوی اتوماتیک[۳۵۴] (ATMها)، ماشین های فروش نقدی[۳۵۵]، دستگاه های خواننده بارکد، دستگاه های روی تلویزیون[۳۵۶] و غیره است. در این موارد دیده می شود واسط کاربر بخش نسبتاً کوچکی از نرم افزار مشتری است، برخلاف تجهیزات ارتباطی و پردازش محلی.

نرم افزار مشتری ، علاوه بر واسط کاربر و نرم افزارهای مربوط به برنامه کاربردی، متشکل از مؤلفه هایی برای کسب شفافیت توزیعی است. ایده آل آن است که مشتری از ارتباط خود با فرآیندهای راه دور اطلاع نداشته باشد. درحالی که به دلایل کارآیی و درستی، توزیع غالباً برای سرورها شفافیت کمتری

[353] distribution transparency
[354] Automati Teller Machine
[355] Cash Register
[356] TV Set-Top Box

دارد. بعنوان مثال، در فصل ۶ خواهیم گفت که در مواردی لازم است سرورهای کپی شده با هم ارتباط برقرار کنند تا عملیات با ترتیب خاص در تمامی نسخه های کپی اجرا شوند.
شفافیت دسترسی معمولاً با تولید بخش باقیمانده مشتری[۳۵۷] از تعریف واسط[۳۵۸] برای آنچه سرور باید ارائه دهد، ایجاد می شود. این باقیمانده همان واسط موجود برای سرور را ارائه می دهد، اما تفاوت های احتمالی در معماری های ماشین و همچنین ارتباطات واقعی را مخفی می کند.
روشهای مختلفی برای اجرای شفافیت مکانی، مهاجرتی، و تغییر مکانی وجود دارد. همانطور که در فصل بعد هم خواهیم دید، استفاده از یک سیستم نامگذاری مناسب ضرورت دارد . در بسیاری از موارد، همکاری با نرم افزار طرف مشتری هم اهمیت پیدا می کند. بعنوان مثال، در حالی که مشتری به سرور بسته شده است، می توان تغییر محل سرور را مستقیماً به مشتری اطلاع داد. در این حالت، میان افزار مشتری می تواند محل استقرار جغرافیایی فعلی سرور را از کاربر مخفی نموده و در صورت لزوم، به صورتی شفاف مجدداً به سرور بسته شود. در بدترین حالت ، برنامه کاربردی مشتری ممکن است متوجه افت کارآیی موقت شود.
به صورتی مشابه، بسیاری از سیستم های توزیعی به کمک راه حل های طرف مشتری اقدام به اجرای شفافیت نسخه برداری می نمایند. بعنوان مثال، یک سیستم توزیعی مجهز به سرورهای نسخه برداری شده را در نظر بگیرید. مطابق شکل ۱۰ - ۳، این نسخه برداری می تواند از طریق ارسال درخواست برای هریک از نسخه ها به انجام برسد. نرم افزار طرف مشتری می تواند بصورتی شفاف تمامی پاسخها را جمع آوری کرده و یک پاسخ واحد را به برنامه کاربردی طرف مشتری انتقال دهد.

شکل ۱۰-۳- نسخه برداری شفاف از سرور با استفاده از راه حل طرف مشتری (داخل شکل: ماشین مشتری، برنامه کاربردی مشتری، طرف مشتری درخواست چندگانه را اداره می کند. سرور۱، برنامه کاربردی سرور، درخواست نسخه برداری شده، سرور۲، برنامه کاربردی سرور، سرور۳، برنامه کاربردی سرور).

در آخر هم نوبت به شفافیت خرابی[۳۵۹] می رسد. مخفی سازی خرابی های ارتباطی با یک سرور نوعاً توسط میان افزار مشتری انجام می شود. بعنوان مثال، میان افزار مشتری را می توان به صورتی پیکره بندی کرد که به صورت مکرر برای اتصال به یک سرور تلاش کرده و یا این که پس از چندین تلاش ناموفق، سرور دیگری را آزمایش کند. حتی در برخی موارد، میان افزار مشتری داده هایی را که طی جلسه قبل در حافظه پنهان قرار داده بود ، بازمی گرداند – گاهی اوقات جستجوگرهای وب که قادر به اتصال به سرور نمی شوند ، به این صورت عمل می کنند.

[357] Client Stub
[358] Interface Definition
[359] Failure

شفافیت همروندی[۳۶۰] را می توان بکمک برخی سرورهای میانی، مثلاً مانیتورهای تراکنش، با نیاز کمتر به پشتیبانی از جانب نرم افزار مشتری اجرا نمود . شفافیت ماندگاری[۳۶۱] هم در اغلب موارد بطور کامل در طرف سرور انجام می شود.

۴-۳- سرورها

حال بیایید نگاه دقیقتری به سازمان سرورها داشته باشیم . در صفحات بعد، ابتدا روی چند مسأله کلی در طراحی سرورها تمرکز کرده و سپس به سراغ مبحث خوشه های سرور[۳۶۲] می رویم.

۱-۴-۳- مسائل کلی مربوط به طراحی ها

سرور یک فرآیند اجراکننده یک سرویس خاص برای مجموعه ای از مشتری هاست. در واقع هر سرور از سازمان واحدی تبعیت می کند : منتظر ورورد درخواست از جانب مشتری می ماند، به درخواست ورودی رسیدگی کرده و سپس مجدداً منتظر ورود درخواست بعدی می شود.

روشهای مختلفی برای سازماندهی سرورها وجود دارد. در مورد **سرور تکراری**[۳۶۳] ، خود سرور به درخواست رسیدگی کرده و، در صورت لزوم، پاسخی را به مشتری درخواست کننده عودت می دهد. در حالیکه **سرور همروند**[۳۶۴] روی درخواست عمل نمی کند، بلکه آنرا به یک نخ جداگانه یا فرآیند دیگر رد کرده و سپس به سرعت منتظر ورورد درخواست بعدی می شود. سرور چندنخی نمونه ای از یک سرور همروند می باشد. یک پیاده سازی جایگزین برای سرور همروند می تواند ایجاد فرآیند جدید به ازای هر درخواست جدید اشاره کرد. از این روش در بسیاری از سیستم های UNIX استفاده می شود. نخ یا فرآیندی که روی در خواست کار می کند مسئول بازگرداندن درخواست به مشتری درخواست کننده هم می باشد.

مسأله دیگر محل برقراری تماس توسط مشتریان با سرور است. در تمام موارد، مشتریان درخواستها را برای یک **نقطه پایانی**[۳۶۵] ،یا به عبارت دیگر یک **پایانه**[۳۶۶]، روی ماشین اجرا کننده سرور ارسال می کنند. هر سرور به نقطه پایانی مشخصی گوش می دهد. مشتریان چگونه نقطه پایانی یک سرویس را شناسایی می کنند؟ یک روش برای اینکار، تخصیص دادن نقاط پایانی جهانی برای سرویسهای معروف است. بعنوان مثال، سرورهایی که روی درخواستهای FTP اینترنت کار می کنند، همواره به پایانه ۲۱ TCP و سرور HTTP شبکه جهانی وب همیشه به پایانه ۸۰ TCP گوش می دهد . این نقاط پایانی

---

360 Concurrency
361 Persistency
362 Server Clusters
363 iterative server
364 concurrent server
365 end point
366 port

به وسیله بخش IANA[۳۶۷] تخصیص یافته و اسناد آن در مرجع (۱۹۹۴) Reynolds and Postel ارائه شده است. با تخصیص نقاط پایانی ، مشتری فقط باید آدرس شبکه ای ماشینی را پیدا کند که سرور در آن درحال اجراست. همانطور که در فصل بعد هم خواهیم گفت، برای این منظور می توان از سرویس های نامگذاری استفاده نمود.

بسیاری از سرویس ها نیازی به نقاط پایانی از پیش تخضیض یافته ندارند. بعنوان مثال، یک سرور زمان روز[۳۶۸] ممکن است از نقطه پایانی استفاده کند که به ضورت پویا و به توسط عامل محلی به آن نسبت داده شده است. در این حالت، ابتدا مشتری باید نقطه پایانی را شناسایی کند. یک راه حل برای اینکار، اجرا کردن فرآیند محافظ[۳۶۹] روی هریک از ماشینهای اجراکننده سرورهاست. این فرآیند محافظ نقطه پایانی فعلی تمامی سرویس های پیاده سازی شده در این ماشین را پی گیری کرده و در اختیار دارد. خود فرآیند محافظ هم به یک نقطه پایانی معروف گوش می دهد. مطابق شکل الف) ۱۱-۳، مشتری ابتدا با فرآیند محافظ تماس برقرار نموده ، نقطه پایانی را درخواست و سپس با سرور مورد نظر تماس برقرار می کند.

شکل ۱۱-۳- الف) پیوند مشتری به سرور با استفاده از فرآیند محافظ.ب) پیوند مشتری به سرور با استفاده از ابرسرور (داخل شکل: الف) ماشین مشتری، مشتری، ۲- درخواست سرویس، ۱- درخواست نقطه پایانی، ماشین سرور، سرور، فرآیند محافظ، ثبت نقطه پایانی، جدول نقطه پایانی، ب) ماشین مشتری، مشتری، ۲- ادامه سرویس،۱-درخواست سرویس، ماشین سرور، سرور واقعی، ابرسرور، ایجاد سرور برای سرویس درخواست شده).

معمولاً به هر سرور یک نقطه پایانی نسبت داده می شود. با این وجود، اجرای واقعی هریک از سرویسها بوسیله یک سرور جداگانه ممکن است نوعی اتلاف منابع محسوب شود. بعنوان مثال ، در یک سیستم UNIX ، معمولاً با تعداد زیادی سرور مواجه هستیم که بطور همزمان اجرا شده و اغلب آنها تا زمان رسیدن درخواست جدید از جانب مشتری در حال انتظار و غیرفعال باقی می مانند. به جای پی گیری چندین فرآیند غیرفعال ، مطابق شکل ب) ۱۱-۳، در اغلب موارد بهتر است که یک **ابرسرور**[۳۷۰] به تمامی نقاط پایانی مربوط به سرورهای مشخصی گوش دهد. از این روش بعنوان مثال در فرآیند محافظ *inetd* در UNIX استفاده شده است. *inetd* به تعدادی از پایانه های معروف سرویس های اینترنت گوش می دهد. با ورورد درخواست جدید، فرآیند محافظ فرآیند جدیدی را برای رسیدگی به درخواست ورودی ایجاد می کند. این فرآیند پس از تکمیل سرویس خارج خواهد شد.

[367] Internet Assigned Numbers Authority
[368] time-of-day
[369] daemon
[370] superserver

مسأله دیگری که باید در طراحی سرورها لحاظ شود، امکان و چگونگی وقفه[۳۷۱] سرورهاست . بعنوان مثال، کاربری را در نظر بگیرید که قصد دارد فایل بسیار بزرگی را به یک سرور FTP آپلود کند . اما ناگهان متوجه می شود که فایل اشتباه بوده و تصمیم می گیرد سرور را متوقف کند تا تبادل داده های بعدی انجام نشود. برای اینکار روشهای مختلفی وجود دارد. یکی از روش هایی که کارکرد بسیار خوبی در دنیای اینترنت امروز پیدا کرده (و در بعضی موارد تنها گزینه هم محسوب می شود) این است که کاربر بطور ناگهانی از برنامه کاربردی مشتری خارج شده (اینکار باعث شکست اتوماتیک اتصال به سرور خواهد شد)، بی درنگ آنرا مجدداً راه اندازی نماید، و تظاهر کند که هیچ اتفاقی نیفتاده. سرور هم با این تصور که مشتری احتمالاً خراب شده است، نهایتاً اتصال قدیمی را می شکند.

یک روش بسیار بهتر برای ایجاد وقفه در ارتباطات آن است که سرور و مشتری بگونه ای ایجاد شوند که امکان ارسال داده های **خارج از باند**[۳۷۲] - یعنی داده هایی که باید پیش از ورود داده های دیگر از همان مشتری، پردازش شوند- فراهم آید . یک راه حل این است که سرور به نقطه پایانی کنترل جداگانه ای گوش دهد که داده های خارج از باند توسط مشتری به آن ارسال می شود و درعین حال (با اولویت کمتر) به نقطه پایانی دیگری گوش دهد که داده های عادی از طریق آن عبور داده می شوند. راه حل دیگر ، ارسال داده های خارج از باند از طریق همان اتصالی است که مشتری بوسیله آن درخواست اصلی را ارسال نموده است. بعنوان مثال، در TCP، می توان داده های فوریتی[۳۷۳] را ارسال نمود. سرور پس از دریافت داده های فوریتی ، متوقف شده (بعنوان مثال در سیستم های UNIXاینکار توسط یک سیگنال انجام می شود) و سپس داده ها را بررسی نموده و به نحو مقتضی روی آنها عمل کند.

آخرین و مهمترین نکته ای که باید در طراحی ها مدنظر قرار گیرد آن است که آیا سرور بدون حالت[۳۷۴] است یا خیر. **سرور بدون حالت** ، اطلاعات مربوط به وضعیت مشتریان خود را نگه نداشته و می توانند بدون اطلاع مشتریان تغییر وضعیت دهد (Birman ،۲۰۰۵). بعنوان مثال، سرورهای وب بدون حالت بوده و صرفاً به درخواستهای HTTP ورودی پاسخ می دهند که ممکن است یا مربوط به آپلود کردن فایل به سرور باشند و یا (در اغلب موارد) جهت واکشی فایل. پس از پردازش درخواست، سرور وب کاملاً مشتری را فراموش می کند. همچنین می توان مجموعه فایلهایی را که یک سرور(احتمالاً با همکاری با یک سرور فایل) مدیریت می کند ، بدون نیاز به مطلع ساختن مشتریان تغییر داد.

توجه داشته باشید که در بسیاری از طرح های بدون حالت، سرور عملاً اطلاعاتی از مشتریان خود را نگه می دارد ، اما واقعیت مهم این است که مفقود شدن این اطلاعات باعث وقفه در سرویس ارائه شده از طرف سرور نخواهد شد. بعنوان مثال، یک سرور وب غالباً تمامی درخواستهای مشتریان را ثبت می کند.

---

371 interrupt
372 out-of-band
373 urgent
374 Stateless

این اطلاعات مثلاً در هنگام تصمیم گیری در مورد لزوم یا عدم لزوم کپی برداری برخی اسناد و محل کپی شدن آنها ممکن است مورد استفاده قرار گیرد. ناگفته پیداست که در صورت مفقود شدن ثبتی ها، احتمالاً تنها مشکل آن است که کارآیی کمتر از حد بهینه خواهد شد.

حالت خاصی از طراحی بدون حالت هنگامی است که سرور دارای **حالت نرم**[۳۷۵] باشد. در این حالت، سرور متعهد می شود که به نمایندگی از جانب مشتری و فقط برای مدت زمانی محدود، وضعیت مشتری را حفظ کند. پس از منقضی شدن این زمان، سرور به رفتار پیش فرض خود بازگشته و تمامی اطلاعاتی را که از مشتری مورد نظر در اختیار دارد، دور خواهد ریخت. بعنوان یک نمونه از این نوع حالت، می توان سروری را ذکر کرد که تهعد نموده تا مشتری مفروض را برای مدت زمانی کوتاهی از به هنگام سازی ها آگاه نگه دارد . پس از گذشت این زمان، مشتری باید درمورد به هنگام سازی ها از سرور پرسش نماید. رویکردهای هویت نرم گرچه از طراحی پروتکل در شبکه های کامپیوتری اقتباس شده اند ، اما می توانند به همان گونه در طراحی سرور هم کاربرد داشته باشند (Clark،۱۹۹۸؛۲۰۰۴، Lui وگروه همکاران).

برعکس، **سرور دارای حالت**[۳۷۶] غالباً اطلاعات ماندگار از مشتریان خود را نگه می دارد. به این معنی که اطلاعات باید صریحاً به توسط سرور حذف شوند. بعنوان یک نمونه می توان سرور فایلی را ذکر کرد که به مشتری اجازه می دهد تا حتی برای انجام عملیات به هنگام سازی، کپی محلی از فایل را در اختیار داشته باشد. چنین سروری جدولی شامل درایه های *(مشتری، فایل)* را نگه خواهد داشت. این جدول به سرور امکان می دهد تا گزارش لحظه ای از این که کدام مشتری اجازه به هنگام سازی کدام فایل را دارد در اختیار داشته باشد، و احتمالاً همان مشتری آخرین نسخه از آن فایل را هم در اختیار دارد.

این روش می تواند باعث افزایش کارآیی عملیات خواندن و نوشتن از دیدگاه مشتری شود. افزایش کارآیی در قیاس با سرورهای بدون حالت یکی از مزایای مهم طرح های دارای حالت تلقی می شود. در عین حال، مثال گفته شده بیانگر نقطه ضعف عمده سرورهای دارای حالت هم می باشد. در صورت شکست سرور بایستی جدول درایه های *(مشتری، فایل)* خود را بازسازی کند، والاّ نمی تواند تضمین کند که آخرین به هنگام سازی ها را روی فایل پردازش کرده باشد. در کل، سرور دارای حالت باید تمامی وضعیت خود را بازسازی کرده و به حالت پیش از خرابی بازگردد. همانطور که در فصل ۸ هم خواهیم گفت، ایجاد امکان بازسازی پیچیدگی های زیادی دارد. در طرح بدون حالت، برای بازسازی سرور از کارافتاده نیازمند انجام هیچ اقدام خاصی نیستیم . بلکه فقط مجدداً راه اندازی شده و سرور منتظر ورود درخواستهای مشتری باقی خواهد ماند.

---

[375] soft state
[376] Stateful

Ling (۲۰۰۴) وگروه همکاران استدلال می کنند که باید بین **حالت جلسه**[۳۷۷] (موقت) و حالت دائم تمایز قائل شویم. مثال گفته شده نمونه ای از حالت جلسه است و در رابطه با مجموعه عملیات صورت گرفته توسط یک کاربر است که باید برای مدت زمانی معین، اما نه نامشخص ، حفظ شود. در نتیجه، حالت جلسه غالباً در معماری های مشتری- سرور سه طبقه ای[۳۷۸] استفاده می شود که در آنها، سرور برنامه کاربردی بایستی پیش از پاسخ به مشتری درخواست کننده، از طریق مجموعه ای از پرس و جوها به سرور پایگاه داده مناسبی دسترسی پیدا کند. نکته این است که در صورت از دست رفتن حالت جلسه، مشروط براینکه مشتری بتواند درخواست اصلی را مجدداً صادر نماید، اشکالی بوجود نخواهد آمد. این مشاهده امکان ذخیره سازی ساده تر ولی با قابلیت اطمینان کمتر حالت را مطرح می کند.

آنچه در وضعیت دائم باقی می ماند غالباً اطلاعاتی است که در پایگاه های داده نگهداری می شوند، از قبیل اطلاعات خریداران، کلیدهای مربوط به نرم افزار خریداری شده و غیره. با این وجود، در اغلب سیستم های توزیعی، حفظ حالت جلسه بیانگر یک طرح دارای حالت است که درصورت وقوع خرابی ها حتماً نیازمند انجام اقدامات خاص و همچنین ارائه فرضیات آشکار در مورد ماندگاری حالت ذخیره شده در سرور می باشد. در مبحث مربوط به تحمل خرابی راجع به این مسائل بحث خواهیم کرد.

در هنگام طراحی سرور، انتخاب هریک از گزینه های طراحی بدون حالت و دارای حالت نبایستی بر خدمات ارائه شده توسط سرور تأثیر بگذارد. بعنوان مثال، در صورتی که باید فایل ها پیش از خوانده شدن یا نوشته شدن باز شوند، پس یک سرور بدون حالت باید به گونه ای این رفتار را تقلید کند. یک راه حل معمول که با جزئیات بیشتر در فصل ۱۱ بحث خواهد شد این است که ابتدا سرور با بازکردن فایل ارجاع داده شده در برابر درخواست نوشتن یا خواندن واکنش داده و سپس واقعاً عملیات خواندن و نوشتن را انجام دهد و فایل را فوراً دوباره ببندد.

در بقیه موارد، ممکن است سرور بخواهد گزارش رفتار مشتریان خود را نگهداری کند تا بتواند به نحو مؤثرتری به درخواست های آنان پاسخ دهد. بعنوان مثال، سرورهای وب برخی اوقات می توانند مشتری را مستقیماً به صفحات مورد علاقه او هدایت کنند. این روش فقط در صورتی قابل اجرا خواهد بود که سرور سابقه ای ازآن مشتری در اختیار داشته باشد. زمانی که سرور قادر به حفظ وضعیت نباشد، یک راه حل معمول این است که مشتری اطلاعات اضافی درمورد دسترسی های قبلی خود ارسال کند. در مورد وب، این اطلاعات غالباً به صورتی شفاف توسط جستجوگر مشتری در چیزی به نام **کوکی**[۳۷۹] ذخیره خواهد شد؛ کوکی در واقع قطعه کوچکی از داده های حاوی اطلاعات ویژه مشتری است که سرور به آنها علاقه مند می باشد. این کوکی ها هرگز توسط جستجوگر اجرا نشده و صرفاً ذخیره می شوند.

---

[377] session
[378] Three-tiered
[379] cookie

وقتی مشتری برای اولین بار به سرور دسترسی پیدا می کند، سرور به همراه صفحات وب درخواستی یک کوکی را هم به جستجوگر باز می گرداند و او هم کوکی را برای استفاده های بعدی در جایی ذخیره می کند . ازآن به بعد، هرزمانیکه مشتری به سرور دسترسی پیدا می کند، کوکی مربوط به آن سرور به همراه درخواست برای او ارسال می شود. این روش در مجموع کارآیی مناسبی دارد و این واقعیت که کلوچه ها برای محافظت توسط جستجوگر عودت داده می شوند بطور کامل از کاربران مخفی نگه داشته می شود. البته این کار در تناقض با حفظ حریم خصوصی است.

۲-۴-۳- خوشه های سرور[۳۸۰]

در فصل ۱ مختصراً راجع به محاسبات خوشه ای بعنوان یکی از چندین نوع نمای سیستم های توزیعی بحث کردیم. در اینجا نگاه دقیقتری خواهیم داشت به سازمان خوشه های سرور و همچنین مهمترین مشکلات مسائل طراحی این خوشه ها.

سازمان عمومی

به بیان ساده، خوشه سرور در واقع مجموعه ای از ماشینهایی است که از طریق یک شبکه به هم متصل شده و هر ماشین یک یا چند سرور را راه اندازی می کند. خوشه های سروری که تا به اینجا مورد بررسی قرار گرفت از جمله خوشه هایی هستند که ماشینهای آنها از طریق یک شبکه محلی به هم متصل شده و غالباً پهنای باند زیاد ، ولی تأخیر ناچیزی را ایجاد می کنند.

در اغلب موارد و مطابق شکل ۱۲-۳، خوشه سرور از نظر منطقی بصورت سه طبقه سازماندهی می شود. طبقه اول متشکل از سوئیچ(منطقی) است که درخواستهای مشتری از طریق آن مسیریابی می شوند. این سوئیچها ممکن است بسیار متفاوت باشند. بعنوان مثال، همانطور که در ادامه هم خواهیم گفت، سوییچهای لایه انتقال درخواستهای اتصال TCP ورودی را پذیرفته و آنها را به یکی از سرورهای داخل خوشه عبور می دهند. یک نمونه کاملاً متفاوت دیگر سرور وبی است که گرچه درخواستهای HTTP ورودی را می پذیرد ، بخشی از درخواست ها را جهت پردازش بیشتر به سرورهای برنامه کاربردی ارسال می کند تا بعداً بتواند نتایج را جمع آوری و پاسخی را به HTTP بازگرداند.

شکل ۱۲-۳- سازمان کلی یک خوشه سرور سه طبقه ای(داخل شکل: طبقه اول، سوییچ منطقی(احتمالاً متعدد)، درخواست های مشتری، درخواست اعزام شده، طبقه دوم، سرورهای برنامه کاربردی/ محاسبه، طبقه سوم، سیستم پایگاه داده ای/ فایل توزیع شده).

مشابه تمامی معماری مشتری – سرور چندطبقه ای ، بسیاری از خوشه های سرور حاوی سرورهای اختصاص یافته جهت پردازش برنامه کاربردی هستند. در محاسبه خوشه ای، این سرورها نوعاً روی سخت افزار خاص با کارآیی بالا برای تحویل توان محاسباتی بالا اجرا می شوند. اما در خوشه های سرور

[380] server clusters

سازمانی[۳۸۱] ، از آنجایی که مشکل کار دسترسی به منبع است نه کسب توان محاسباتی ، ممکن است لازم باشد که برنامه های کاربردی فقط روی ماشینهای نسبتا پایین اجرا شوند.

طبقه سوم متشکل از سرورهای پردازش داده و مخصوصاً، سرورهای فایل و پایگاه داده ای است. این سرورها هم بسته به کاربرد خوشه سرور، ممکن است روی ماشینهای خاص منظوره اجرا شده و برای دسترسی پرسرعت به دیسک پیکربندی شوند و از حافظه های مخفی بزرگ برای داده های طرف سروراستفاده کنند.

البته تمام خوشه های سرور از این دسته بندی کاملاً مشخص پیروی نمی کنند. در بسیاری از موارد، هر ماشین مجهز به منبع ذخیره محلی خود بوده و غالباً ، برنامه کاربردی و پردازش داده را در یک سرور تجمیع کرده و در نتیجه معماری دوطبقه ای ایجاد خواهد شد. بعنوان مثال، در مبحث جریان رسانه[۳۸۲] با استفاده از خوشه سرور، معمولاً رسم براین است که از معماری سیستم دوطبقه ای استفاده شده و هر ماشین، بعنوان یک سرور رسانه اختصاصی شده عمل می کند (۲۰۰۴، Steinmetz and Nahrstedt).

در صورتیکه یک خوشه سرور چندین سرویس را ارائه کند، ممکن است ماشینهای مختلف سرورهای برنامه کاربردی مختلفی را اجرا کنند. در نتیجه، سوییچ قادر خواهد بود تا بین سرویسها تمایز قائل شود، و الاّ قادر به ارسال درخواستها به ماشینهای مربوطه نخواهد بود. در نتیجه، بسیاری از ماشین های طبقه دوم فقط یک برنامه کاربردی را اجرا می کنند. این محدودیت علاوه بر دلالت بر وابستگی به سخت افزار و نرم افزار موجود ، ناشی از این واقعیت هم می باشد که برنامه های کاربردی مختلف غالباً بوسیله مدیران اجرایی مختلف مدیریت می شوند. این مدیران هم تمایلی به دخالت در کار ماشین های یکدیگر ندارند.

در نتیجه مشاهده می شود که برخی ماشینها موقتاً بیکار هستند ، در حالی که درخواست های بسیار به برخی دیگر باعث اضافه بار آنها می شود. در اینصورت بهتر است که سرویسها را موقتاً به ماشینهای بیکار مهاجرت دهیم. راه حل پیشنهادی در (۲۰۰۴) Awadallah and Rosenblum ، استفاده از ماشینهای مجازی جهت مهاجرت نسبتاً آسان کد به ماشینهای واقعی است. در بخشهای بعدی همین فصل به مسأله مهاجرت کد باز خواهیم گشت.

حال بیایید طبقه اول را که حاوی سوییچ است، به دقت بررسی کنیم. یکی از اهداف مهم در طراحی خوشه های سرور، مخفی سازی واقعیت وجود چندین سرور است. به بیان دیگر، برنامه های کاربردی که روی ماشینهای مختلف اجرا می شوند، قاعدتاً نیازی به اطلاعات راجع به سازمان داخلی خوشه ندارند. این شفافیت دسترسی بوسیله یک نقطه دسترسی ارائه می شود که آنهم متقابلاً از طریق نوعی سوییچ سخت افزار مانند یک ماشین اختصاصی پیاده سازی می شود.

---

[381] enterprise

[382] Streaming Media

سوئیچ در واقع نقطه ورودی برای خوشه سرور بوده و تنها یک آدرس شبکه ای دارد. جهت ایجاد قابلیت های مقیاس پذیری و دسترسی پذیری، خوشه سرور ممکن است چندین نقطه دسترسی داشته باشد که هریک بوسیله یک ماشین اختصاصی مشخص می شود. ما در اینجا فقط مورد یک نقطه دسترسی را در نظر می گیریم.

یکی از روش های استاندارد برای دسترسی به خوشه سرور ، ایجاد اتصال TCP است تا پس از آن درخواست های سطح برنامه کاربردی از طریق آن و بعنوان بخشی از جلسه ارسال شوند. پایان جلسه با شکستن اتصال اعلان می شود. در مورد **سوییچ های لایه انتقال**، سوییچ درخواستهای اتصالات TCP ورودی را پذیرفته و این اتصال را به یکی از سرورها تقدیم[۳۸۳] می کند(۱۹۹۸، Hunt وگروه همکاران؛ ۱۹۹۸، Pai وگروه همکاران). اصل کلی چیزی که معمولاً به نام **تقدیم کردن TCP** خوانده می شود، در شکل ۱۳-۳ ارائه شده است.

شکل ۱۳-۳- اصل کلی تقدیم TCP(داخل شکل: یک اتصال منطقاً جداگانه TCP،مشتری، درخواست، پاسخ، سرور، سوئیچ، (تقدیم) درخواست، سرور).

پس از آنکه سوییچ درخواست اتصالTCP را دریافت می کند، بهترین سرور را برای تقدیم آن درخواست انتخاب و بسته درخواست را به آن سرور ارسال می کند. سرور هم متقابلاً تصدیقی را به مشتری درخواست کننده می دهد، و همچنین آدرس IP سوئیچ را در حوزه منبع سرآیند بسته IP حامل قطعه TCP درج می نماید. توجه داشته باشید که این ترفند برای ادامه اجرای پروتکل TCP توسط مشتری ضروری است؛ چون مشتری در انتظار پاسخ از جانب سوییچ است و نه از جانب سرور دیگری که تا به حال حتی نامی از آن برده نشده. مشخص است که پیاده سازی تقدیم TCP نیازمند انجام برخی اصلاحات در سطح سیستم عامل است.

حتماً تا به اینجا متوجه شده اید که سوییچ می تواند نقش مهمی در توزیع بار در بین سرورهای مختلف داشته باشد. با تصمیم گیری در مورد محل ارسال درخواست، سوییچ در مورد سرور مسئول انجام پردازش بیشتر روی درخواست هم تصمیم گیری می کند. ساده ترین سیاست برای تعادل بار آن است که سوییچ از روش نوبت گردشی بهره ببرد: یعنی هربار سرور بعدی را از لیست خود انتخاب و درخواست را برای آن ارسال کند.

از ضوابط پیشرفته تری هم می توان برای انتخاب سرور استفاده نمود. بعنوان مثال، فرض کنید که یک خوشه سرور سرویسهای متعددی را ارائه می دهد. چنانچه سوییچ بتواند در صورت ورود درخواست باز هم بین این سرویسها تمایز قائل شود، بنابراین قادر خواهد بود که تصمیمات آگاهانه ای در مورد محل ارسال درخواست اتخاذ کند. انتخاب سرور می تواند همچنان در لایه انتقال انجام شود، البته مشروط به اینکه سرویس ها بوسیله شماره درگاه از هم تفکیک شده باشند. گام بعدی آن است که اجازه دهیم

[383] Hand-off

سوئیچ ماموریت درخواست ورودی را مشخص کند. از این روش فقط در صورتی می توان استفاده نمود که بداند ماموریت چگونه به نظر می رسد. بعنوان مثال، درمورد سرورهای وب، سوییچ نهایتاً در انتظار یک درخواست HTTP است تا براساس آن درمورد واحد پردازش کننده آن تصمیم بگیرد. در فصل ۱۲ در مبحث مربوط به سیستم های براساس وب مجدداً به موضوع **توزیع درخواست محتواآگاه**[۳۸۴] بازخواهیم گشت.

سرورهای توزیعی[۳۸۵]

اغلب خوشه های سروری که تا به اینجا مورد بحث قرار گرفت، بیشتر به صورت آماری پیکر بندی شده بودند. در این خوشه ها، معمولاً یک ماشین مدیریت اجرایی جداگانه وجود دارد که گزارش سرورهای موجود را نگهداری کرده و به نحو مقتضی، این اطلاعات را به ماشین های دیگری از قبیل سوئیچ می دهد.

همانطور که قبلاً هم گفتیم ، اغلب خوشه های سرور فقط یک نقطه دسترسی ارائه می کنند. در صورت خرابی این نقطه، خوشه هم غیرقابل دسترسی خواهد شد. برای حذف این مشکل احتمالی، می توان چندین نقطه دسترسی ایجاد نموده و آدرسهای آنها را در اختیار عموم قرار داد. بعنوان مثال، **سیستم نام دامنه**[۳۸۶]**(DNS)** می تواند آدرسهای متعددی را بازگرداند که همگی متعلق به یک نام میزبان واحد باشند. در این روش هم مشتریان باید در صورت خراب شدن یکی از آدرسها، مجدداً تلاش های متعددی بنماید. بعلاوه، راه حلی برای مشکل نیاز به نقاط دسترسی ثابت ارائه نمی دهد.

داشتن ثبات، مانند داشتن نقطه دسترسی باعمر طولانی ، یکی از ویژگی های مطلوب از نقطه نظر مشتری و سرور است. از طرف دیگر، بهتر است در پیکر بندی خوشه سرور (شامل سوییچ) از انعطاف پذیری بالایی برخوردار باشیم . این مشاهده منجر به طراحی **سرور توزیعی** شده که عملاً مجموعه ای از ماشین ها است که تعداد آنها به صورت پویا تغییر می کند و احتمالاً دارای نقاط دسترسی متغیری است ؛ اما برای ناظران بیرونی فقط یک ماشین منفرد و قدرتمند می باشد. طراحی این دست سرورهای توزیعی در کتاب Szymaniak (۲۰۰۵) و گروه همکاران ارائه شده است. ذیلاً مختصری در مورد آن خواهیم گفت.

ایده اصلی در سرورهای توزیعی آن است که مشتریان از مزایای وجودی یک سرور مقاوم، با کارآیی بالا و پایدار برخودار شوند . این ویژگی ها را غالباً به وسیله کامپیوترهای بزرگ به دست می آورند که برخی، میانگین زمانی بین دو خرابی آنها را حتی تا بیش از ۴۰ سال هم ذکر کرده اند. اما، با دسته بندی شفاف ماشین های ساده تر در یک خوشه و تکیه نکردن بر موجود بودن فقط یک ماشین ، ممکن است بتوان به درجه پایداری بهتری در مقایسه با هریک از مؤلفه ها دست یافت . بعنوان مثال، چنین

---

384 content-aware
385 distributed servers
386 domain name system

خوشه ای را می توان به صورت پویا با استفاده از ماشین های کاربران پیکربندی نمود، همچنان که در مورد سیستم های توزیعی همکاری کننده[۳۸۷] عمل می شود.

حال ببینیم چگونه می توان در این دست سیستم ها یک نقطه دسترسی ثابت[۳۸۸] ایجاد کرد . ایده اصلی استفاده از سرویس های شبکه سازی موجود، خصوصاً پشتیبانی جابجایی برای نسخه ۶ IP[۳۸۹] (MIPv6) است. در MIPv6 ، فرض می شود که یک گره متحرک دارای یک **شبکه خانگی**[۳۹۰] است که معمولاً در آن سکونت داشته و به ازای آن، آدرس ثابتی به نام **آدرس خانه**[۳۹۱] (**HoA**) دارد. به این شبکه خانگی، مسیریاب خاصی به نام **نماینده خانه**[۳۹۲] متصل است که مراقب جریان ترافیک به سمت گره متحرک، در صورت دور بودن از آن است. وقتی یک گره متحرک به یک شبکه خارجی متصل می شود، **آدرس مراقبتی**[۳۹۳] (**CoA**) موقتی دریافت می کند که از آن طریق به گره دسترسی می شود. این آدرس مراقبتی به نماینده خانه گره گزارش شده و آنهم مراقب است که از آن پس تمام ترافیک به گره متحرک پیش برده شود. توجه داشته باشید که برنامه های کاربردی که با گره متحرک ارتباط برقرار می کنند، فقط آدرس شبکه خانگی گره را درک کرده و هرگز آدرس مراقبتی را مشاهده نخواهند کرد.

از این اصل کلی می توان جهت دادن آدرسی ثابت به سرورهای توزیعی استفاده نمود. در این حالت، ابتدا یک **آدرس تماس**[۳۹۴] منحصربه فرد و جداگانه به خوشه سرور نسبت داده می شود. این آدرس تماس، آدرس عمری سرور است که در تمام ارتباطات آن با دنیای خارج مورد استفاده قرار خواهد گرفت. در هر زمان، یکی از گره های داخل سرور توزیعی بعنوان نقطه دسترسی این آدرس تماس عمل خواهد کرد؛ اما این نقش قابل واگذاری به گره های دیگر هم می باشد . در واقع داستان به این صورت است که، نقطه دسترسی مفروض آدرس خود را بعنوان آدرس مراقبتی در نماینده خانه سرور توزیعی ثبت می کند. دراین مقطع ، تمام ترافیک به نقطه دسترسی هدایت خواهد شد و نقطه هم از آن پس، مراقب توزیع درخواستها در بین گره هایی است که در همان زمان در خوشه هستند. در صورت خرابی نقطه دسترسی، یک مکانیزم کنترل خرابی ساده وارد صحنه شده و به این ترتیب ، یک نقطه دسترسی دیگر آدرس مراقبتی جدید را گزارش خواهد داد.

در همین پیکر بندی ساده ممکن است نماینده خانه و نقطه دسترسی تبدیل به نقطه شکست یکتا شوند، چون تمام ترافیک از طریق ایندو ماشین جاری خواهد شد. می توان بکمک یکی از ویژگی های

---

[387] collaborative
[388] stable access point
[389] Mobile IP version 6
[390] home network
[391] home address
[392] home agent
[393] care-of address
[394] contact address

MIPv6 به نام *بهینه سازی مسیر* [۳۹۵]از بروز این مشکل جلوگیری کرد. بهینه سازی مسیر به اینصورت عمل خواهد کرد: هنگامیکه یک گره متحرک با آدرس خانگی *HA* آدرس مراقبتی فعلی خود ، مثلاً *CA* ، را گزارش می دهد، نمایندگی خانه می تواند *CA* را برای یک مشتری ارسال نماید. مشتری هم جفت *(HA،CA)* را به صورت محلی ذخیره خواهد کرد . از این لحظه به بعد، ارتباطات مستقیماً برای *CA* پیش برده خواهد شد. هرچند برنامه کاربردی طرف مشتری هنوز هم می تواند از آدرس خانگی استفاده کند، نرم افزار پشتیبان زیربنایی MIPv6 این آدرس را به *CA* ترجمه کرده و بجای آن مورد استفاده قرار خواهد داد.

شکل ۱۴-۳- بهینه سازی مسیر در یک سرور توزیعی(داخل شکل: مشتری ۱، باور می کند که سرور آدرس HA را در اختیار دارد، باور می کند که سرور به X متصل است، باور می کند که X در محل CA1 مستقراست، مشتری۲، باور می کند که سرور آدرس HA را در اختیار دارد، باور می کند که سرور به X متصل است، باور می کند که X در محل CA2 مستقراست، سرور توزیعی X، سرور ۱، می داند که مشتری ۱ باور می کند که X است، نقطه دسترسی با آدرس CA1، سرور۲، نقطه دسترسی با آدرس CA2، می داند که مشتری ۲ باور می کند که X است، اینترنت).

مطابق شکل ۱۴-۳ ، از بهینه سازی مسیر می توان برای قبولاندن این مطلب به مشتریان استفاده کرد که با یک سرور واحد تماس دارند، در حالی که هر مشتری در ارتباط با گره مجزایی از سرور توزیعی است. هنگامیکه یکی از نقاط دسترسی داخل سرور توزیعی درخواستی را از مشتری C_1 به مثلاً گره S_1 (با آدرس CA_1) پیش می برد، آنقدراطلاعات به S_1 عبور می دهد که برای آغاز روال بهینه سازی مسیر کفایت کند. بر اساس این اطلاعات، مشتری نهایتاً باور می کند که آدرس مراقبتی، CA_1 است. به این ترتیب، C_1 قادر به ذخیره سازی جفت *(HA،CA1)* می شود. در جریان این روال، نقطه دسترسی (و همچنین نماینده خانه) بعنوان تونل هدایت عمده ترافیک بین C_1 و S_1 عمل می کند. این کار از این که نماینده خانه باور کند تغییر آدرس مراقبتی داشته است جلوگیری می کند و همچنان به ارتباط خود با نقطه دسترسی ادامه دهد.

البته، در حین انجام روال بهینه سازی مسیر، ممکن است درخواست هایی از جانب مشتریان دیگر وارد شود. این درخواست ها در نقطه دسترسی در وضعیت معلق باقی خواهند ماند تا این که بتوانند پیش برده شوند. پس ممکن است درخواست یکی دیگر از مشتریان به نام C_2 به سوی گره عضو S_2 (با آدرس CA_2) پیش برده شده و S_2 اجازه دهد مشتری C_2 جفت *(HA و CA۲)* را ذخیره کند . در نتیجه، مشتریان مختلف مستقیماً با اعضای مختلف سرور توزیعی ارتباط برقرار می کنند، در حالی که

[395] route optimization

همچنان هر برنامه کاربردی مشتری این تصور را دارد که سرور آدرس HA را دارد. نماینده خانگی همچنان به ارتباط خود با نقطه دسترسی در حال گفتگو با آدرس تماس ادامه خواهد داد.

۳-۴-۳- مدیریت خوشه های سرور

یک سرور خوشه ای باید برای دنیای خارج مانند یک کامپیوتر منفرد بنظر برسد- که در عمل غالباً هم همینطور است. با این وجود، وقتی کار به مدیریت خوشه ها می رسد، شرایط به شدت تغییر می کند. همانطور که در ادامه خواهیم گفت، تلاشهای زیادی برای تسهیل مدیریت خوشه های سرور انجام گرفته است.

روشهای متعارف

تا به حال ، متعارف ترین روش در مدیریت خوشه های سرور، بسط کارکردهای سنتی مدیریت تک کامپیوترها به خوشه بوده است. در ساده ترین شکل ، مدیر اجرایی در گره یکی از مشتریان دور ثبت نام[۳۹۶] کرده و فرامین مدیریت محلی مربوط به نظارت، نصب و تغییر مؤلفه ها را اجرا می کند.

در حالت کمی پیشرفته تر ، ثبت نام در یک گره مفروض مخفی می شود و به جای آن یک واسط در ماشین مدیر اجرایی ارائه می شود که امکان جمع آوری اطلاعات از یک یا چند سرور ، ارتقاء مؤلفه ها ، حذف یا اضافه گره ها و غیره را فراهم می آورد. مزیت عمده این روش ، تسهیل عملیات جمعی است که بر روی گروهی از سرورها انجام می شود. این نوع مدیریت خوشه های سرور کاربرد عملی فراوانی پیدا کرده و ازنمونه های آن می توان به نرم افزار مدیریتی از قبیل مدیریت سیستم های خوشه ای[۳۹۷] IBM اشاره کرد(Hochstetler and Beringer، ۲۰۰۴).

با این وجود، چنانچه تعداد خوشه ها به بیش از چند ده گره برسد، این نوع مدیریت دیگر کارساز نخواهد بود. بسیاری از مراکز داده ای مدیریت هزاران سروری را برعهده دارند که گرچه به شکل چندین خوشه سازماندهی شده اند، اما همگی با همکاری عمل می کنند. انجام اینکار بوسیله سرورهای مدیریت اجرایی متمرکز غیرممکن است. بعلاوه، به راحتی می توان مشاهده کرد که خوشه های بسیار بزرگ نیازمند مدیریت تعمیر دائمی (شامل ارتقاء ها) هستند. به بیان ساده تر، چنانچه p احتمال خراب بودن یک سرور بوده و با فرض مستقل بودن خرابی ها، آنگاه برای آنکه خوشه ای با N سرور بدون خرابی حتی یک سرور عمل کند، $(1-p)^N$ خواهد بود. چنانچه p = ۰،۰۰۱ و $N=1000$ ، احتمال عملکرد صحیح تمام سرورها فقط ۳۶ درصد خواهد بود.

در نتیجه، پشتیبانی از خوشه های سرور بسیار بزرگ تقریباً همیشه موردی است. در این رابطه ، از قواعد تجربی زیادی می توان استفاده کرد (Brewer، ۲۰۰۱)، اما روش سیستماتیکی برای کار بر روی موضوع مدیریت سیستم های بسیار بزرگ وجود ندارد. مدیریت خوشه ها هنوز در مراحل اولیه است، اما

[396] Login
[397] Cluster System Management

می توان امیدوار بود که پس از کسب تجربیات کافی در مورد راه حلهای خود مدیریتی ،که در فصل قبل مورد بررسی قرار گرفت، نهایتاً در این موضوع راه خود را بیابد.

مثال: PlanetLab

حال بیایید نگاه دقیقتری به یک سرور خوشه ای تقریباً غیر معمول داشته باشیم . PlanetLab یک سیستم توزیعی براساس همکاری است که در آن ، هر یک از سازمانها یک یا چند کامپیوتر را به جمع صدها گره موجود هدیه می دهند .این کامپیوترها رویهمرفته خوشه سرور تک طبقه ای ایجاد می کنند که در آن تمامی امور دسترسی، پردازش و ذخیره سازی می توانند برروی هریک از گره ها انجام شوند. مدیریت PlanetLab در اغلب موارد لزوماً توزیعی است. پیش از معرفی اصول اساسی، ابتدا ویژگی های اصلی این سیستم توزیعی را تشریح می کنیم (۲۰۰۵، Peterson وگروه همکاران).

در PlanetLab، هر سازمان یک یا چند گره هدیه می کند؛ گره می تواند صرفاً یک کامپیوتر باشد، اگرچه این امکان هست که خوشه ای متشکل از چندین ماشین هم باشد. نحوه سازماندهی گره ها در شکل ۱۵-۳ ارائه شده است . در این رابطه دو مؤلفه مهم وجود دارد(۲۰۰۴، Bavier و گروه همکاران). اول این که، ناظر ماشین مجازی[۳۹۸] (VMM) یک سیستم عامل بهبود یافته Linux است. بهبود یافتگی عمدتاً شامل تنظیماتی برای پشتیبانی از مؤلفه دوم یعنی V سرور ها است. Vسرورهای Linux را می توان بعنوان محیط مجزایی تلقی کرد که گروهی از فرآیندها در آن در حال اجرا هستند. فرآیندهای مربوط به Vسرورهای مختلف *کاملاً* مستقل بوده و نمی توانند مستقیماً در تمام منابع از جمله فایلها، حافظه اصلی و اتصالات شبکه سهیم شوند، چیزی که در بین فرآیندهای در حال اجرا بر روی یک سیستم عامل وجود دارد. بلکه در عوض، Vسرور محیطی متشکل از مجموعه از بسته های نرم افزاری خودی، برنامه ها و امکانات شبکه سازی است. به عنوان مثال، یک V سرور می تواند محیطی فراهم آورد که در آن یکی از فرآیندها درمی یابد که می تواند از ترکیب Python 1.5.2 با وب سرورقدیمی تری از نوع Apache، مثلاً *httpd 1.3.3* ، استفاده کند. در حالیکه سرور دیگری ممکن است از آخرین نسخه های *httpd* و Python پشتیبانی کند. البته اطلاق نام " سرور" به یک V سرور تاحدودی اشتباه است ، چون عملاً فقط جدایی گروه های فرآیندها از یکدیگر را باعث می شود. کمی بعدتر مجدداً به مبحث Vسرورها بازخواهیم گشت.

شکل ۱۵-۳- سازمان اصلی یک گرهPlanetLab (داخل شکل: ماشینهای مجازی تخصیص یافته به کاربر، ماشینهای مجازی مدیریت ممتاز، فرآیند،V سرور ،سیستم عامل بهبود یافته Linux، سخت افزار).

VMM Linux تضمین می کند که Vسرورها از هم جدا باشند: فرآیندهای داخل Vسرور های مختلف به صورتی همروند و مستقل از یکدیگر اجرا می شوند، و هریک از آنها فقط از بسته های نرم

[398] virtual machine monitor

افزاری و برنامه های موجود در محیط خود استفاده می کند. مرزهای جداکننده بین فرآیندها در vسرورهای مختلف جدی هستند. بعنوان مثال، ممکن است دو فرآیند در vسرورهای مختلف ID کاربر یکسانی داشته باشند، اما این به هیچ وجه بمعنای داشتن کاربر واحد نیست. قائل شدن این تفکیک به میزان چشمگیری باعث تسهیل پشتیبانی کاربران از سازمانهای مختلفی خواهد شد که می خواهند از PlanetLab ، مثلاً ، بعنوان بستری برای تجربه برنامه های کاربردی و سیستم های توزیعی کاملاً متفاوت استفاده کنند.

PlanetLab جهت پشتیبانی از چنین تجربیاتی ایده **برش**[۳۹۹] را مطرح می کند که در واقع، مجموعه ای ازvسرورهایی است که هریک روی گره متفاوتی اجرا می شوند. بنابراین، هر برش را می توان خوشه سرور مجازی تصور کرد که بوسیله مجموعه ای از ماشینهای مجازی پیاده می شود. ماشینهای مجازی در PlanetLab بر روی سیستم عامل Linux اجرا می شوند که هسته اصلی سیستم عامل با پیمانه هایی توسعه یافته است.

به دلیل وجود مسائل متعدد، که از مهمترین آنها می توان به ۳ مورد زیر اشاره کرد، مدیریت PlanetLab تبدیل به مسأله ویژه ای شده است:

۱. گره ها به سازمانهای مختلفی تعلق دارند. هر یک از این سازمانها بایستی قادر به تعیین فرد مجاز برای اجرای برنامه های کاربردی بر روی گره های خود بوده ، و همچنین کاربرد منابع را به نحو مناسبی محدود کنند.
۲. گرچه ابزارهای نظارتی مختلفی وجود دارد، اما هریک ترکیب خاصی از سخت افزار و نرم افزار را فرض می کنند. بعلاوه، هر یک به صورت سفارشی برای یک سازمان قابل استفاده است.
۳. برنامه های مربوط به برشهای متفاوت که روی یک گره واحد اجرا می شوند، نباید در کار یکدیگر مداخله کنند. این مورد مشابه استقلال فرآیند در سیستم عامل است.

در ادامه هریک از مسائل فوق را مفصلاً مورد بررسی قرار خواهیم داد.

در مدیریت منابع PlanetLab، نقش کلیدی برعهده **مدیر گره** است. تمامی گره ها از وجود چنین مدیری برخوردار هستند. مدیر گره بوسیله یک vسرور جداگانه اجرا شده و تنها وظیفه آن ایجاد vسرورهای دیگر روی گره تحت مدیریت خود و کنترل تخصیص منابع است. مدیر گره هیچگونه تصمیمی در مورد سیاست ها نمی گیرد و فقط مکانیزمی است برای ایجاد اجزاء اساسی لازم جهت اجرای برنامه روی یک گره مفروض.

پی گیری منابع بوسیله یکی از ویژگی های منبع ، یا بطور خلاصه *rspec* ، انجام می پذیرد. *rspec* بیانگر فاصله زمانی است که در طی آن منابع بخصوص تخصیص می یابند. این منابع شامل فضای دیسک، توصیف گرهای فایل، پهنای باند شبکه داخل محدوده[۴۰۰] و خارج محدوده[۴۰۱] ، نقاط پایانی

[399] slice
[400] Inbound

سطح انتقال، حافظه اصلی، و به کارگیری CPU می باشد. *rspec* از طریق یک شناسه ۱۲۸ بیتی منحصر به فرد جهانی به نام توانایی منبع[۴۰۲] (*rcap*) شناسایی می شود. مدیر گره با در اختیار داشتن *rcap* می تواند *rspec* مربوطه را در جدول محلی جستجو کند.

منابع به برش ها بسته هستند. به بیان دیگر، برای استفاده از منابع باید یک برش ایجاد شود. هر برش با یک **ارائه کننده سرویس**[۴۰۳] - موجودیتی که روی PlanetLab حساب دارد- مرتبط می باشد. بنابراین، هریک از برشها را می توان بوسیله جفت (*principal- id ، slice-tag*) شناسایی نمود، که در آن *principal- id* بیانگر ارائه کننده و *slice-tag* ، شناسه انتخاب شده توسط ارائه کننده است.

برای ایجاد یک برش جدید، گره باید یک **سرویس ایجاد برش**[۴۰۴](**SCS**) را اجرا کند . این سرویس هم ضمن تماس با مدیر گره از او درخواست ایجاد یک vسرور و تخصیص منابع را خواهد داشت. نمی توان مستقیماً از روی شبکه با خود مدیر گره تماس برقرار کرده و به این ترتیب مدیر گروه فقط روی مدیریت منبع محلی تمرکز می کند. SCS هم به نوبه خود درخواستهای ایجاد برش از جانب هرکسی را نمی پذیرد، بلکه فقط برخی از **مسئولین برش**[۴۰۵] مجاز به درخواست ایجاد برش هستند. هریک از مسئولین برش ها حق دسترسی به مجموعه ای از گره ها را دارند. در ساده ترین مدل، فقط یک مسئول برش وجود دارد و او مجاز به درخواست ایجاد برش روی تمامی گره هاست.

برای تکمیل بحث خود در نظر می گیریم که ارائه دهنده سرویس ضمن تماس با مسئول برش ، از او درخواست ایجاد برش در مجموعه ای از گره ها را خواهد کرد. ارائه دهنده سرویس هم ، مثلاً به این دلیل که قبلاً تأیید هویت شده و بعنوان کاربر PlanetLab ثبت شده بود، برای مسئول برش شناخته شده خواهد بود. در عمل،کاربران PlanetLab بوسیله یک سرویس براساس وب با مسئول برش تماس برقرار می کنند. جزئیات بیشتر در این رابطه را در مرجع (۲۰۰۳، Chun and Spalink) مطالعه کنید.

از این روال می توان دریافت که مدیریت PlanetLab از طریق میانجی ها انجام می شود. مسئولین برش یکی از انواع مهم این میانجی ها را تشکیل می دهند. این مسئولین، اعتبارنامه هایی[۴۰۶] را در گره ها برای ایجاد برش دریافت کرده اند. کسب این اعتبارنامه ها به صورت خارج باند و اساساً از طریق تماس با مدیران اجرایی سیستم در سایتهای مختلف صورت می پذیرد . ناگفته پیداست که این فرآیند زمانبر، بوسیله کاربران نهایی(یا به زبانPlanetLab ، ارائه دهندگان سرویس) انجام نخواهد شد.

---

401 outbound
402 resource capability
403 service provider
404 slice creation service
405 slice authorities
406 credentials

علاوه بر مسئولین برش، مسئولین مدیریت[۴۰۷] نیز وجود دارند. در حالیکه مسئول برش فقط روی مدیریت برشها تمرکز می کند، مسئول مدیریت مراقب از گره ها را بر عهده دارد، و بخصوص تضمین می کند که گره های تحت حاکمیت او نرم افزار اصلی PlanetLab را اجرا کرده و تابع قوانین وضع شده توسط PlanetLab باشند. ارائه دهندگان سرویس اطمینان دارند که گره های ارائه شده به وسیله مسئول مدیریت، رفتار مناسبی خواهند داشت.

شکل ۱۶-۳- روابط مدیریتی بین موجودیت های مختلف PlanetLab(داخل شکل: مالک گره، مسئول مدیریت، ارائه دهنده سرویس، مسئول برش).

سازمان فوق الذکر منجر به ساختار مدیریتی شکل ۱۶-۳ می شود که در مرجع (۲۰۰۵، Peterson وگروه همکاران) از نقطه نظر روابط اعتمادی تشریح شده است. این روابط به شرح زیر است:

۱. مالک گره مفروض ،گره خود را تحت تسلط مسئول مدیریت قرار داده و در صورت نیاز، به کارگیری راهم محدود می کند.
۲. مسئول مدیریت نرم افزار لازم را برای افزودن گره به PlanetLab ارائه می کند
۳. ارائه دهنده سرویس ضمن اعتماد به این که مسئول مدیریت گره هایی با رفتار مناسب ایجاد می کند، در آن ثبت نام می نماید.
۴. ارائه دهنده سرویس برای ایجاد برش روی مجموعه ای از گره ها، با مسئول برش تماس برقرار می کند.
۵. مسئول برش بایستی ارائه دهنده سرویس را تأیید هویت نماید.
۶. مالک گره جهت ایجاد برش، ، سرویس ایجاد برش را برای مسئول برش ایجاد نموده و مدیریت منبع را به مسئول برش واگذار می کند.
۷. مسئول مدیریت ایجاد برشها را به مسئول برش واگذار می کتد

این روابط بیانگر مشکل تفویض اختیار به گره ها به صورتی کنترل شده است تا مالک گره بتواند از وجود مدیریتی ایمن و بقاعده اطمینان داشته باشد . مسأله دوم که باید مورد بررسی قرار گیرد، مسأله نظارت است. لازم است با اتخاذ شیوه ای واحد، به کاربران اجازه داده شود تا بر رفتار برنامه های خود در داخل برشی خاص نظارت داشته باشند .

PlanetLab از شیوه ساده ای تبعیت می کند. به این ترتیب که هریک از گره ها مجهز به مجموعه ای از حسگرها بوده و هر حسگر قادر به گزارش دهی اطلاعاتی از قبیل کارکرد CPU، فعالیت دیسک و غیره می باشد. حسگرها ممکن است بسیار پیچیده باشند، اما مسأله مهم آن است که همواره گزارشات خود را براساس هرگره ارائه می دهند. این اطلاعات بوسیله یک سرور وب در اختیار دیگران قرار می

[407] management authorities

گیرد، بطوریکه هریک از حسگرها از طریق درخواستهای ساده HTTP قابل دسترسی هستند (۲۰۰۴، Bavier و گروه همکاران).

هرچند این شیوه نظارتی هنوز در مراحل اولیه خود بسر می برد ، اما می تواند به عنوان مبنایی برای طرح های پیشرفته نظارتی مدنظر قرار گیرد . بعنوان مثال، دلیلی وجود ندارد که نتوان از Astrolabe که در فصل ۲ مورد بحث قرار گرفت، جهت خواندن تجمیع حسگر متراکم در بین چندین گره استفاده نشود.

بعنوان سومین و آخرین نکته در مورد مسائل مدیریتی، یعنی حفاظت از برنامه ها در برابر یکدیگر، باید گفت که PlanetLab از سرورهای مجازی Linux (که به نام vسرورها خوانده می شوند) جهت جدا سازی بین برش ها استفاده می کند. همانطور که قبلاً هم گفته شد، ایده اصلی در یک vسرور، اجرای برنامه های کاربردی در محیط خودشان است. محیط شامل تمامی فایل هایی است که معمولاً در داخل یک ماشین به اشتراک گذاشته می شوند. دستیابی به این جداسازی تقریباً آسان و از طریق فرمان UNIX chroot امکان پذیر است ؛ فرمان مذکور در عمل باعث تغییر ریشه سیستم فایل از محل جستجوی فایلها توسط برنامه های کاربردی می شود . فقط ابرکاربرها[۴۰۸] قادر به اجرای chroot هستند.

البته هنوز ناگفته ها بسیار است. سرورهای مجازی Linux علاوه بر جداسازی سیستم های فایل، اطلاعات معمولاً مشترک در مورد فرآیندها، آدرسهای شبکه ای، کارکرد حافظه و غیره را هم جدا می سازند. در نتیجه، هر ماشین فیزیکی مفروض واقعاً به چندین واحد تقسیم شده و هر واحد در رابطه با یک محیط تمام عیار Linux است که مجزا از بخش های دیگر است. مروری بر سرورهای مجازی Linux را در مرجع (۲۰۰۵) Potzl وگروه همکاران مطالعه کنید.

[408] superuser

۵-۳- مهاجرت کد[۴۰۹]

تا به اینجا، ما بیشتر برروی سیستم های توزیعی تمرکز کرده ایم که ارتباطات آنها منحصر به عبور داده هاست . با این وجود، مواردی هم وجود دارند که عبوردادن برنامه ها حتی در حین اجرای آنها هم، باعث ساده سازی طرح سیستم های توزیعی خواهد شد. در این بخش، نگاه دقیقی به مفهوم واقعی مهاجرت کد خواهیم داشت. کار را با بررسی روشهای مختلف مهاجرت کد آغاز کرده و سپس، به سراغ منابع محلی مورد استفاده در برنامه های مهاجرتی خواهیم رفت . یک مسئله مشکل مهاجرت کد در سیستم های ناهمگن است که در همین بخش بررسی می شود.

۱-۵-۳- روشهای مهاجرت کد

پیش از ورود به مبحث انواع مختلف مهاجرت کد، ابتدا دلایل سودمند بودن مهاجرت کد را بررسی می کنیم.

دلایل مهاجرت کد

سابقاً ، مهاجرت کد در سیستم های توزیعی به شکل **مهاجرت فرآیند**[۴۱۰] انجام می شد و طی آن ،کل فرآیند از یک ماشین به ماشین دیگر انتقال می یافت (۲۰۰۰، Milojicic وگروه همکاران). انتقال یک برنامه در حال اجرا به ماشین دیگر کاری هزینه بر و دقیق است و بهتر است دلیل خوبی برای این کار وجود داشته باشد. دلیل اصلی همواره بهبود کارآیی بوده است. ایده اصلی این است که می توان با انتقال فرآیندها از ماشینهای دارای بار زیاد به ماشنهای کم بار، کارآیی کلی سیستم را افزایش داد. بار را غالباً به صورت طول صف پردازنده یا بهره وری از پردازنده تعریف می کنند، اما شاخصه های کارآیی دیگری هم می توان به این تعریف اضافه کرد.

الگوریتمهای توزیع بار- که تصمیمات مربوط به تخصیص و توزیع مجدد کارها با توجه به مجموعه پردازنده ها براساس آن گرفته می شود- نقش مهمی را در سیستم های محاسباتی سنگین ایفا می کنند. با این وجود، در بسیاری از سیستم های مدرن توزیعی ، بهینه سازی ظرفیت محاسباتی به اندازه مسائلی از قبیل به حداقل رساندن ارتباط، مهم محسوب نمی شود. همچنین، به دلیل ناهمگن بودن بسترهای زیربنایی و شبکه های کامپیوتری، بهبود کارآیی از طریق مهاجرت کد به جای مدلهای ریاضی ، غالباً براساس استدلالهای کمّی استوار است.

بعنوان نمونه، سیستم مشتری- سروری را در نظر بگیرید که سرور آن ، مدیریت پایگاه داده ای عظیمی را برعهده داشته باشد . چنانچه برنامه کاربردی مشتری نیاز به انجام عملیات پایگاه داده ای متعدد بر روی مقادیر بزرگ داده ای داشته باشد، شاید بهتر باشد که بخشی از برنامه کاربردی مشتری به سرور

[409] code migration
[410] process migration

انتقال یافته و فقط نتایج بر روی شبکه ارسال شود. در غیراینصورت، ممکن است شبکه با سیل عظیمی از داده های در حال تبادل از سرور به مشتری مواجه شود. در این حالت، مهاجرت کد بر اساس این فرضیه صورت خواهد پذیرفت که بهتر است داده ها در مجاورت محل استقرار خودشان پردازش شوند .
از همین استدلال می توان برای مهاجرت بخشهایی از سرور به مشتری استفاده نمود. بعنوان مثال، در بسیاری از برنامه های کاربردی تعاملی پایگاه داده ای ، مشتریان باید فرمهایی را پرکنند که بعداً بصورت مجموعه ای از عملیات پایگاه داده ای ترجمه خواهند شد. پردازش این فرم در طرف مشتری و ارسال فرم تکمیل شده برای سرور، ممکن است مانع از عبور تعداد نسبتاً زیادی از پیامهای کوچک در شبکه شود. نتیجه آنکه مشتری با کارآیی بهتری مواجه شده و سرور هم مدت زمان کمتری را صرف پردازش فرم و ارتباطات خواهد کرد.
بعلاوه، پشتیبانی از مهاجرت کد می تواند با بهره گیری از موازی گرایی[۴۱۱] ، اما بدون در نظرگرفتن دقت های معمول در واسطه با برنامه سازی موازی، به بهبود کارآیی کمک کند. بعنوان نمونه می توان به جستجوی اطلاعات در وب اشاره کرد. تقریباً به راحتی می توان پرس و جوی[۴۱۲] جستجو را به شکل برنامه سیار کوچکی به نام **نماینده سیار**[۴۱۳] ،که می تواند از یک سایت به سایت دیگر حرکت کند اجرا نمود. با تهیه چندین کپی از این نوع برنامه ها و ارسال آنها به سایتهای مختلف، ممکن است بتوان در مقایسه با شیوه استفاده از فقط یک برنامه به طور خطی سرعت را افزایش داد.
علاوه بر بهبود کارآیی، دلایل دیگری هم برای پشتیبانی از ایده مهاجرت کد می توان ارائه کرد که مهمترین آن، مسأله انعطاف پذیری است. روش متعارف در ساخت برنامه های کاربردی توزیعی ، تقسیم برنامه کاربردی به چندین بخش، و تصمیم گیری پیش از اجرا در مورد محل مناسب اجرای هر یک از بخشهاست. این روش، بعنوان مثال، منجر به برنامه های کاربردی مشتری- سرور چندطبقه ای مختلفی شده که در فصل دو مورد بحث قرار گرفت.
چنانچه کد بتواند بین چندین ماشین جابجا شود، پیکره بندی پویای سیستم های توزیعی هم امکانپذیر می شود. بعنوان مثال، فرض کنید که یک سرور مفروض واسط استانداردی به یک سیستم فایل را پیاده سازی کرده است. برای فراهم آوردن امکان دسترسی مشتریان دور به سیستم فایل، سرور از یک پروتکل اختصاصی[۴۱۴] استفاده می کند. معمولاً، پیاده سازی واسط سیستم فایل در طرف مشتری، که براساس این پروتکل استوار است ، نیازمند برقراری پیوند با برنامه کاربردی مشتری است. در این روش لازم است که نرم افزار در زمان ایجاد برنامه کاربردی مشتری، موجود و در دسترس مشتری باشد.
یک روش جایگزین آن است که سرور پیاده سازی مورد نیاز مشتری را فقط زمانی که مشتری نیازمند آن است ارائه نماید، یعنی زمانی که مشتری به سرور پیوند می خورد. در این زمان، مشتری به گونه ای

411 parallelism
412 query
413 mobile agent
414 proprietary protocol

پویا این پیاده سازی را دانلود می کند، مراحل اولیه مورد نیاز را می گذراند، و نهایتاً سرور را فرامی خواند. این مطلب در شکل ۱۷-۳ نشان داده شده است. این مدل انتقال پویای کد از یک سایت دور مسلماً نیازمند یک پروتکل استاندارد برای دانلود کد و گذراندن مراحل آغازین است. بعلاوه باید بتوان کد دانلودشده را روی ماشین مشتری اجرا نمود. راه حلهای مختلف برای اینکار در این فصل و در فصلهای بعد بررسی می شوند.

شکل ۱۷-۳- اصل کلی پیکر بندی پویای مشتری جهت برقراری ارتباط با سرور. مشتری ابتدا نرم افزار لازمه را واکشی می نماید و سپس سرور را فرا می خواند(داخل شکل: کد ویژه سرویس طرف مشتری، مشتری، سرور، ۱- مشتری کد را واکشی می کند، ۲- مشتری و سرور با هم ارتباط برقرار می کنند، مخزن کد).

یکی از مزایای مهم مدل دانلود پویای نرم افزار طرف مشتری آن است که دیگر لازم نیست مشتریان برای صحبت با سرورها تمام نرم افزار را ازپیش نصب شده داشته باشند. بلکه می توان در صورت لزوم نرم افزار را انتقال داده و سپس در زمانی که به آن نیازی نباشد آنرا دور ریخت. یک مزیت دیگر این است که در صورت استاندارد بودن واسط ها ، می توان پروتکل مشتری- سرور و پیاده سازی آنرا به هر تعداد که مورد نیاز باشد تغییر داد. این تغییرات تأثیری بر برنامه های کاربردی مشتری که متکی به سرور هستند نخواهد داشت. البته معایبی هم دارد که از مهمترین آنها می توان به مشکلات امنیتی اشاره کرد که در فصل ۹ بحث می شود. اطمینان کوررکورانه از اینکه کد دانلود شده در صورت دسترسی به هارد دیسک محافظت نشده شما، فقط واسط اعلان شده را اجرا خواهد کرد و ناب ترین بخشهای آنرا به ناکجا آباد نخواهد فرستاد، ممکن است همیشه نتایج چندان مطلوبی هم به بار نیاورد.

مدلهای مهاجرت کد

هرچند مهاجرت کد به معنای جابجایی کد در بین ماشینهاست ، این اصطلاح در واقع حوزه بزرگتری را شامل می شود. بطور معمول ، ارتباطات در سیستم های توزیعی به تبادل داده ها بین فرآیندها می پردازد. مهاجرت کد در گسترده ترین حالت، به موضوع جابجایی برنامه ها در بین ماشینها با هدف اجرای این برنامه ها در مقصد انجام می پذیرد. در برخی موارد از جمله مهاجرت فرآیند، وضعیت اجرای یک برنامه، سیگنالهای در انتظار[۴۱۵] و بخشهای دیگر محیط هم باید انتقال داده شوند.

برای درک بهتر مدلهای مهاجرت کد، از چارچوب مشروح در مرجع (۱۹۹۸) Fuggetta وگروه همکاران استفاده می کنیم. براساس این چارچوب، هر فرآیند از سه قطعه تشکیل می شود. *قطعه کد*[۴۱۶] بخش حاوی مجموعه دستورالعملهای تشکیل دهنده برنامه در حال اجرا است. *قطعه منبع* [۴۱۷]شامل مرجع ها به منابع خارجی مورد نیاز در فرآیند از قبیل فایلها، چاپگرها، دستگاهها ، فرآیندهای دیگر و از

[415] pending signals
[416] Code Segment
[417] resource segment

این قبیل است. قطعه آخر یعنی *قطعه اجرا* [۴۱۸]هم جهت ذخیره سازی وضعیت اجرای فعلی فرآیند از جمله داده های خصوصی[۴۱۹]، پشته و، البته، شمارشگر برنامه استفاده می شود.

مهاجرت کد در حالت حداقل خود فقط نوعی **جابجایی ضعیف**[۴۲۰] ارائه می کند. در این مدل، می توان فقط قطعه کد و در صورت لزوم، برخی داده های آغازین[۴۲۱] را انتقال داد. یکی از ویژگی های بارز جابجایی ضعیف آن است که برنامه انتقال یافته همواره از یکی از چندین موقعیت آغازکننده ازپیش تعریف شده آغاز می شود. بعنوان نمونه می توان به کوچک برنامه های[۴۲۲] Java اشاره کرد که همواره کار اجرا را از ابتدا آغاز می کنند. مزیت این رویکرد در سادگی آن است. جابجایی ضعیف نیازمند آن است که فقط ماشین هدف قادر به اجرای کد باشد، و نهایتاً به کد قابل جابجایی منجر می شود. به هنگام بحث در مورد مهاجرت در سیستم های ناهمگن مجدداً به این موضوع بازخواهیم گشت.

برخلاف جابجایی ضعیف، در سیستم هایی که از **جابجایی قوی** [۴۲۳]پشتیبانی می کنند، قطعه اجرا هم قابل انتقال است. ویژگی بارز جابجایی قوی آن است که فرآیند در حال اجرا را می توان متوقف ساخته، به ماشین دیگری انتقال داد و سپس ، کار اجرا را از همانجا که متوقف شده، مجدداً از سر گرفت. مشخص است که هرچند جابجایی قوی بسیار عمومی تر از جابجایی ضعیف است ، اما پیاده سازی آن بسیار مشکل است.

صرف نظر از جابجایی ضعیف و قوی، باید بین مهاجرت آغاز شده از فرستنده و آغاز شده از گیرنده هم تمایز قائل شد. در حالت مهاجرت **آغاز شده از فرستنده** [۴۲۴]، مهاجرت از ماشینی آغاز می شود که کد در آن اقامت داشته یا در حال اجراست. نوعاً ، مهاجرت آغاز شده از فرستنده در هنگام آپلود کردن برنامه ها به کامپیوتر سرور انجام می شود. مثال دیگر، ارسال برنامه جستجو از طریق اینترنت به سرور پایگاه داده ای وب جهت انجام پرس و جو در آن سرور است. در مهاجرت **آغاز شده از گیرنده** [۴۲۵]، اقدام برای مهاجرت کد بوسیله ماشین هدف انجام می شود. کوچک برنامه های Java نمونه ای از این روش است.

مهاجرت آغاز شده از گیرنده ساده تر از مهاجرت آغازشده از فرستنده است. در بسیاری از موارد، مهاجرت کد بین مشتری و سرور انجام شده و مشتری برای مهاجرت آغازگر است. آپلود کردن ایمن کد به سرور، که در مهاجرت آغازشده از فرستنده صورت می پذیرد، غالباً مستلزم آن است که مشتری قبلاً در آن سرور ثبت[۴۲۶] و تأیید [۴۲۷] شده باشد. به بیان دیگر، ازآنجاییکه مشتری به احتمال قوی خواهان

---

418 excecution segment
419 private data
420 weak mobility
421 initialization data
422 applets
423 strong mobility
424 sender-initiated
425 receiver-initiated
426 register

دسترسی به منابع سرور از جمله دیسک آن خواهد بود، لازم است که سرور تمامی مشتری های خود را بشناسد . محافظت از چنین منابعی ضروری است. بالعکس، دانلود کردن کد همانند حالت آغاز شده از گیرنده ، ممکن است بطور ناشناس انجام شود. بعلاوه، سرور معمولاً علاقه ای به منابع مشتری نداشته و مهاجرت کد به سمت مشتری فقط با هدف بهبود کارآیی طرف مشتری انجام می شود. از اینرو ، فقط تعداد معدودی از منابع از قبیل اتصالات شبکه و حافظه نیازمند محافظت هستند. در فصل ۹ مفصلاً راجع به ایمنی در مهاجرت کد بحث خواهیم کرد.

در حالت جابجایی ضعیف، بین اجرای کد مهاجرت کرده توسط فرآیند هدف و یا ایجاد فرآیند دیگر برای اجرای کد تفاوت وجود دارد. بعنوان مثال، کوچک برنامه های Java صرفاً بوسیله جستجوگر وب (فرآیند هدف) دانلود شده و در فضای آدرس جستجوگر وب اجرا می شوند. مزیت روش این است که دیگر نیازی به ایجاد فرآیند دیگری نبوده و بدین ترتیب از ارتباطات در ماشین هدف جلوگیری می شود. اما مهمترین نقطه ضعف این است که فرآیند هدف بایستی در برابر اجراهای سهوی یا مغرضانه کد محافظت شود. یک راه حل ساده آن است که اجازه دهیم تا سیستم عامل با ایجاد فرآیند جداگانه ای جهت اجرای کد مهاجرت کرده، اقدام به مراقبت کند. توجه داشته باشید که این روش مشکلات دسترسی به منابع را حل نمی کند. بنابراین موضوع دسترسی هنوز هم نیازمند بحث و بررسی بیشتر است .

در عوض جا به جایی یک فرآیند در حال اجرا (جا به جایی فرآیند)، جا به جایی قوی را می توان با روش تقلید از دور[۴۲۸] نیز پشتیبانی نمود. برخلاف مهاجرت فرآیند ، در روش تقلید، کپی دقیقی از فرآیند اصلی ایجاد می شود که در همان زمان بر روی ماشین دیگری در حال اجراست. فرآیند مقلد به طور موازی با فرآیند اصلی اجرا می شود. در سیستم های UNIX، تقلید از راه دور از طریق ایجاد فرآیند فرزند و ادامه یافتن آن برروی یک ماشین دور انجام می شود. مزیت مدل تقلید این است که بسیار شبیه آن چیزی است که هم اکنون در بسیاری از برنامه های کاربردی مورد استفاده قرار می گیرد. تنها تفاوت این است که فرآیند مقلد روی ماشین دیگری اجرا می شود. از این نقطه نظر، مهاجرت به روش تقلید روش ساده ای برای بهبود شفافیت توزیع محسوب می شود.روشهای مختلف مهاجرت کد در شکل ۱۸-۳ خلاصه شده است.

شکل ۱۸-۳-روشهای مختلف مهاجرت کد(داخل شکل: مکانیزم جابجایی، جابجایی ضعیف، جابجایی آغاز شده از فرستنده، اجرا در فرآیند هدف، اجرا در یک فرآیند جداگانه، جابجایی آغاز شده از گیرنده، اجرا در فرآیند هدف، اجرا در یک فرآیند جداگانه، جابجایی قوی، جابجایی آغاز شده از فرستنده، مهاجرت فرآیند، تقلید فرآیند، جابجایی آغاز شده از گیرنده، مهاجرت فرآیند، تقلید فرآیند).

---

[427] authenticate
[428] Remote Cloning

۲-۵-۳- مهاجرت و منابع محلی[۴۲۹]

تا به اینجا، در مورد مهاجرت کد و قطعه اجرا صحبت کرده ایم. قطعه منبع نیازمند توجه ویژه ای است. آنچه غالباً مهاجرت کد را تبدیل به امری بسیار مشکل می کند این است که نمی توان قطعه منبع را براحتی و بدون هیچ تغییری همراه با قطعه های دیگر انتقال داد. بعنوان مثال، فرآیندی را فرض کنید که مرجعی[۴۳۰] به یک درگاه TCP خاص دارد و از طریق آن با فرآیندهای(ازراه دور) دیگر ارتباط برقرار می کند. این مرجع در قطعه منبع نگه داشته می شود. در صورتیکه فرآیند به محل دیگری انتقال پیدا کند، باید درگاه قبلی را رها کرده و درگاه جدیدی را در محل مقصد درخواست کند. در مواردی دیگر، انتقال مرجع لزوماً مشکلساز نخواهد بود. بعنوان مثال، ارجاع به یک فایل از طریق یک URL مطلق[۴۳۱]، بدون در نظرگرفتن ماشینی که فرآیند دارای URL در آن اقامت دارد، همچنان معتبر خواهد ماند.

برای درک اثرات مهاجرت کد بر قطعه منبع ، (۱۹۹۸) Fuggetta و گروه همکاران سه نوع پیوند فرآیند به منبع را شناسایی کرده اند. قوی ترین پیوند هنگامی است که یک فرآیند از طریق شناسه[۴۳۲] خود به منبع ارجاع کند. در این حالت، فرآیند دقیقاً نیازمند همان منبع ارجاع داده شده خواهد بود و نه چیز دیگری. بعنوان نمونه ای از این **پیوند توسط شناسه**[۴۳۳] می توان به مواردی اشاره کرد که فرآیند ، جهت ارجاع به یک وب سایت خاص از URL استفاده کند یا هنگامیکه بوسیله آدرس اینترنتی سرور به سرور FTP آن ارجاع می کند. بر پایه همین دیدگاه ارجاعات به نقاط پایانی ارتباطی محلی نمونه دیگری از پیوند توسط شناسه محسوب می شود.

یکی از حالات ضعیف تر پیوند فرآیند به منبع هنگامی است که فقط مقدار منبع مورد نیاز باشد. در این حالت، اجرای فرآیند دستخوش تغییری نخواهد شد اگر منبع دیگری مقداری برابر با مقدار منبع قبلی ارائه کند. یکی از نمونه های **پیوند توسط مقدار**[۴۳۴] هنگامی است که برنامه بر کتابخانه های استانداردی تکیه می کند که مثلاً در برنامه نویسی در C یا Java مورد استفاده قرار می گیرند. این دست کتابخانه ها بایستی همواره بصورت محلی قابل دسترسی باشند ، اما ممکن است محل دقیق آنها در سیستم فایل محلی در سایتهای مختلف متفاوت باشد. برای اجرای مناسب و موفقیت آمیز فرآیند، نه خود فایلها، بلکه محتوای آنها اهمیت دارد.

ضعیف ترین و آخرین شکل پیوند هنگامی است که فرآیند فقط نیازمند منبعی از نوع مشخصی باشد. از نمونه های این **پیوند توسط نوع**[۴۳۵] می توان به ارجاعات به دستگاههای محلی از قبیل صفحه نمایش ها، چاپگرها و ازاین قبیل اشاره کرد.

---

[429] local resources
[430] Reference
[431] absolute
[432] Idetifier
[433] binding by identifier
[434] binding by value
[435] binding by type

برای مهاجرت کد غالباً لازم است که ارجاعات به منابع را تغییر دهیم، اما نمی توانیم بر چگونگی پیوند فرآیند به منبع تأثیر بگذاریم. چرایی و چگونگی تغییر ارجاع به یک منبع بستگی به آن دارد که آیا می توان منبع را بهمراه کد به ماشین هدف انتقال داد یا خیر. به بیان دقیقتر، باید پیوند منبع به ماشین را در نظر گرفته و بین موارد زیر تمایز قائل شویم. **منابع نابسته**[۴۳۶] را می توان براحتی در بین ماشینهای مختلف جابجا کرده و نوعاً فایلهای (داده ای) مرتبط با برنامه باید مهاجرت کنند. هرچند امکان انتقال یا کپی کردن **منبع بسته شده** [۴۳۷] وجود دارد ، این کار هزینه نسبتاً زیادی دارد. بعنوان نمونه های منابع بسته شده می توان به پایگاههای داده ای محلی و وب سایتهای کامل اشاره کرد. هرچند این منابع، در تئوری، به ماشین فعلی خود وابسته نیستند، غالباً انتقال آنها به محیط دیگر امکان پذیر نیست. **منابع ثابت**[۴۳۸] هم به شدت به ماشین یا محیط خاصی متصل بوده و بنابراین قابل انتقال نیستند. این منابع غالباً دستگاههای محلی هستند. مثال دیگری از منابع ثابت نقاط پایانی ارتباط محلی[۴۳۹] هستند.

ترکیب سه نوع پیوند فرآیند به منبع با سه نوع پیوند منبع به ماشین منجر به نه صورت ترکیبی می شود که باید در مهاجرت کد مدنظر قرار گیرند. این نه حالت در شکل ۱۹-۳ نمایش داده شده است.

	پیوند منبع به ماشین			
		نابسته	بسته	ثابت
پیوند فرآیند به منبع	توسط شناسه	MV (or GR)	GR (or MV)	GR
	توسط مقدار	CP (or MV,GR)	GR (or CP)	GR
	توسط نوع	RB (or MV,GR)	RB (or GR,CP)	RB (or GR)

GR (Global Reference) ایجاد یک مرجع جهانی برای تمام سیستم

MV (Move) جابجایی منبع

CP (Copy) کپی کردن مقدار منبع

RB (Rebind) الحاق مجدد فرآیند به منبع قابل دسترسی به صورت محلی

شکل ۱۹-۳- اقداماتی که باید با توجه به ارجاعات به منابع محلی در مهاجرت کد به ماشین دیگر انجام شود.

ابتدا بیایید نگاهی به حالات مختلف پیوند فرآیند به منبع توسط شناسه داشته باشیم . در صورت نابسته بودن منبع، معمولاً بهترین گزینه جابجایی آن بوسیله کد در حال مهاجرت است. با این وجود، هنگامیکه منبع با فرآیندهای دیگر مشترک باشد، یک راه حل ایجاد یک مرجع جهانی[۴۴۰] یعنی مرجعی

436 unattached resources
437 fastened resource
438 fixed resources
439 local communication end points
440 global reference

است که بتواند از مرزهای ماشین عبور کند . یکی از نمونه های چنین مراجعی URL[۴۴۱] ها هستند. در صورتیکه منبع بسته شده یا ثابت باشد، بهترین راه حل بازهم ایجاد مرجع جهانی خواهد بود.
باید بدانیم که ایجاد مرجع جهانی ممکن است مستلزم کارهایی بیش از استفاده از URLها بوده و استفاده از چنین مرجعی می تواند بسیار هزینه بر باشد. بعنوان مثال برنامه ای را فرض کنید که تصاویر با کیفیت بالایی را برای یک ایستگاه کاری مخصوص چندرسانه ای تولید می کند. تولید تصاویر با کیفیت بالا به صورت بی درنگ نیازمند کار محاسباتی زیادی بوده و به همین دلیل ممکن است برنامه به یک سرور محاسباتی کارآ انتقال داده شود. برقراری مرجع جهانی برای ایستگاه کاری چندرسانه ای به معنای ایجاد یک مسیر ارتباطی بین سرور محاسباتی و ایستگاه کاری است. بعلاوه، سرور و ایستگاه کاری هردو نیازمند انجام پردازش های زیادی برای دست یابی به پهنای باند مورد نیاز جهت انتقال تصاویر هستند. در نتیجه، ممکن است انتقال برنامه به سرور محاسباتی، به دلیل هزینه بر بودن زیاد مرجع جهانی، چندان هم ایده خوبی محسوب نشود .
بعنوان نمونه دیگری از مواردی که برقراری مرجع جهانی همواره چندان هم ساده نیست، می توان به مهاجرت فرآیندی اشاره کرد که از نقطه پایانی ارتباط محلی استفاده می کند. در این حالت، ما با منبع ثابتی مواجه هستیم که فرآیند توسط شناسه به آن پیوند دارد. اساساً دو راه حل در این واسطه وجود دارد. یک راه حل آن است که فرآیند پس از مهاجرت، ضمن ایجاد اتصال با ماشین منبع ، فرآیند جداگانه ای را در ماشین منبع نصب کرده و بکمک آن براحتی تمامی پیامهای ورودی به سمت فرآیند مهاجرت کرده پیش برده شود. نقطه ضعف عمده این روش آن است که در صورتی که ماشین منبع دچار اشکال شود، ارتباط با فرآیند مهاجرت کرده ممکن است قطع شود. راه حل دیگر آن است که تمامی فرآیندهایی که قبلاً با فرآیند در حال مهاجرت ارتباط برقرار کرده بودند ، مرجع جهانی خود را تغییر داده و پیامهای جدید را به نقطه پایانی ارتباطی جدید در ماشین هدف ارسال کنند.
در مبحث پیوند به توسط مقدار با شکل متفاوتی مواجه هستیم. ابتدا یک منبع ثابت را در نظر بگیرید. ترکیب این منبع ثابت و پیوند به توسط مقدار هنگامی رخ می دهد که فرآیندی فرض کند که از حافظه مشترک بین فرآیندها می تواند استفاده کند. ایجاد مرجع جهانی در چنین حالتی به این معنای لزوم پیاده سازی شکل توزیعی از حافظه اشتراکی است . در بسیاری از موارد، این راه حل واقعاً قابل اجرا و مؤثر نیست.
منابع بسته شده که براساس مقدارشان ارجاع داده می شوند، نوعاً کتابخانه های زمان اجرا[۴۴۲] هستند . معمولاً، کپی های این دست منابع روی ماشین هدف قابل دستیابی بوده و یا بایستی به نحوی تا پیش از مهاجرت کد، کپی شوند. درصورت زیاد بودن حجم داده های کپی شونده، مثلاً در تهیه فرهنگهای لغت و واژه یابها در سیستم های پردازش متن، ایجاد مرجع جهانی گزینه بهتری خواهد بود.

[441] Universal Resource Locator
[442] runtime

ساده ترین حالت مربوط به منابع نابسته است. به استثنای مواردی که منبع در بین چند فرآیند مشترک باشد، بهترین راه حل کپی کردن (یا انتقال) منبع به مقصد جدید می باشد. اما در حالت استثناء گفته شده ، ایجاد مرجع جهانی تنها گزینه محسوب می شود.

آخرین حالت مربوط به پیوند به توسط نوع است. غیر از پیوند منبع به ماشین، راه حل مشخص پیوند مجدد فرآیند به یک منبع قابل دستیابی به صورت محلی از همان نوع است. تنها در صورت نبود چنین منبعی است که باید منبع اصلی را به مقصد جدید کپی کرده یا انتقال داده، و یا اقدام به ایجاد مرجع جهانی نمود.

۳-۵-۳- مهاجرت در سیستم های ناهمگن[۴۴۳]

تا به اینجا، به طور ضمنی فرض کرده ایم که کد مهاجرت کرده در ماشین هدف به راحتی قابل اجراست . این فرض در بحث در مورد سیستم های همگن[۴۴۴] صدق می کند. اما در مجموع باید گفت که سیستم های توزیعی براساس مجموعه ناهمگنی از بسترها استوار هستند که هریک دارای سیستم عامل و معماری ماشین مختص به خود می باشد. در چنین سیستم هایی، موفقیت مهاجرت مستلزم پشتیبانی از تمامی بسترهاست؛ به این معنا که، بتوان قطعه کد را روی هریک از بسترها اجرا نمود. همچنین باید از وجود قطعه اجرا در هریک از بسترها اطمینان داشت.

مشکلات ناشی از ناهمگن بودن در بسیاری از موارد مشابه مشکلات مربوط به قابلیت جابجایی است . بنابراین تعجبی ندارد اگر بگوییم که راه حلهای آنها هم بسیار مشابه هستند. بعنوان مثال، در پایان دهه هفتاد، یک راه حل ساده برای رفع بسیاری از مشکلات مربوط به جا به جایی Pascal به ماشینهای مختلف، تولید کد میانی مستقل از ماشین برای یک ماشین مجازی انتزاعی بود(۱۹۸۱، Barron). البته، این ماشین باید روی بسیاری از بسترها پیاده می شد، و به اینصورت برنامه های Pascal در هرجایی قابل اجرا می شدند . هرچند این ایده ساده برای چندین سال کاربرد گسترده ای پیدا کرد، هرگز واقعاً بعنوان یک راه حل عمومی برای مشکلات جابجایی در زبانهای دیگر، خصوصاً C، مطرح نشد. حدود ۲۵ سال بعد، ایده مهاجرت کد در سیستم های ناهمگن بوسیله زبانهای اسکریپتی[۴۴۵] و زبانهای دارای قابلیت جابجایی بالا[۴۴۶] از قبیل Java مورد استفاده قرار گرفت. بطور خلاصه، شیوه مورد استفاده در این راه حلها همان شیوه بکار رفته برای انتقال Pascal است . وجه مشترک تمامی راه حل های این گونه، استوار بودن براساس ماشین مجازی (فرآیندی) است که یا مستقیماً کد منبع (مانند زبانهای اسکریپتی) و یا به نوعی کد میانی تولید شده توسط یک کمپایلر (مانند Java) را اجرا می کند. بودن در مکان و زمان مناسب برای توسعه دهندگان زبان های برنامه نویسی بسیار مهم است.

---

[443] heterogeneous systems
[444] homogeneous
[445] scripting languages
[446] highly portable languages

اخیراً اقداماتی برای کاهش میزان وابستگی به زبانهای برنامه نویسی انجام شده است . به خصوص اینکه ، راه حلهایی برای مهاجرت فرآیندها و بلکه برای مهاجرت کل محیط های محاسباتی پیشنهاد شده است. ایده اصلی در این راه حل های جدید آن است که کل محیط را بخش بندی کرده و برای فرآیندها دیدگاه خودشان نسبت به محیط محاسباتی را ایجاد کند.

در صورتیکه بخش بندی[۴۴۷] به طور مناسب انجام شود، می توان بخش را از سیستم زیرین تفکیک کرد و آن را واقعاً به ماشین دیگری مهاجرت داد. به این ترتیب ، مهاجرت عملاً شکلی از جابجایی قوی برای فرآیندها خواهد شد، چون از آن پس می توان آنها را در هر زمان حین اجرا انتقال داده و پس از اتمام مهاجرت، درست از همان محل توقف مجدداً ازسر گرفت. بعلاوه ، می توان بسیاری از پیچیدگی های مربوط به مهاجرت فرآیندها در رابطه با پیوند آنها به منابع محلی را حل نمود، چون این پیوندها در بسیاری از موارد براحتی حفظ می شوند. به این معنی که ، منابع محلی غالباً بخشی از محیطی هستند که فرآیند به آن مهاجرت می کند.

دلایل متعددی می توان برای مهاجرت تمام محیط مطرح کرد؛ اما شاید مهمترین آن چنین باشد که در صورتی که لازم باشد ماشینی به هر دلیل از دور خارج شود، بازهم امکان ادامه عملیات وجود خواهد داشت . بعنوان مثال، فرض کنید که در یک خوشه سرور، مدیر اجرایی سیستم ها تصمیم به خاموش کردن یا تعویض یکی از ماشینها بگیرد، اما نباید فرآیندهای در حال اجرا را متوقف سازد. بلکه در عوض می تواند موقتاً محیطی را متوقف کرده ، آنرا به ماشین دیگر منتقل کند(در آنجا در کنار محیط های موجود دیگر قرار خواهد گرفت)، و مجدداً آنرا از حالت توقف خارج کند. واضح است که این روش مدیریت برای محیط های محاسباتی و فرآیندهای آنها که قرار است در زمان های خیلی طولانی فعالیت کنند بسیار قدرتمند می باشد.

اجازه دهید نمونه ای از مهاجرت ماشینهای مجازی را در نظر بگیریم که در مرجع (۲۰۰۵) Clark وگروه همکاران تشریح شده است. نویسندگان روی مهاجرت بی درنگ یک سیستم عامل مجازی شده[۴۴۸] تمرکز کرده اند که در خوشه ای از سرورها قابل مشاهده است و اتصال محکم[۴۴۹] آنها از طریق یک شبکه محلی مشترک ایجاد می شود. در اینصورت ، مهاجرت با دو مشکل عمده روبرو خواهد بود،یکی مهاجرت کل شکل حافظه و دیگری مهاجرت پیوندها به منابع محلی.

در مورد مشکل اول، در کل سه روش برای مهاجرت وجود دارد(که می توانند ترکیب شوند):

۱. هل دادن صفحات حافظه به ماشین جدید و ارسال مجدد صفحاتی که بعداً تغییر می یابند در حین فرآیند مهاجرت.
۲. متوقف سازی ماشین مجازی فعلی ؛ مهاجرت حافظه ، و آغاز ماشین مجازی جدید.

[447] compartmentalization
[448] Virtualized
[449] Tightly Coupled

۳. ایجاد امکان داخل کشیدن صفحات جدید در صورت نیاز توسط ماشین جدید ، یعنی ایجاد امکان آغاز فوری فرآیندها در یک ماشین مجازی جدید و کپی شدن صفحات حافظه در صورت وجود تقاضا .

چنانچه ماشین مجازی در حین مهاجرت در حال اجرای سرویس زنده ای[۴۵۰] - یا با بیان دیگر، سرویس پیوسته - باشد ، گزینه دوم ممکن است باعث زمان از کار افتادگی غیر قابل قبول سرویس شود. از طرف دیگر، اتخاذ شیوه بر پایه تقاضا که در گزینه سوم ارائه می شود، ممکن است باعث طولانی شدن بیش از حد زمان لازم برای مهاجرت شود. بعلاوه، از آنجایی که انتقال مجموعه کاری فرآیندهای مهاجرت کرده به ماشین جدید بسیار زمانبر است ، احتمالاً با افت کارآیی هم مواجه خواهیم شد .

در روش پیشنهادی(۲۰۰۵) Clark و گروه همکاران از شیوه پیش کپی[۴۵۱] ، یعنی ترکیب گزینه اول با یک فاز توقف و کپی[۴۵۲] درگزینه دوم ، استفاده می شود . با چنین ترکیبی مدت ازکارافتادگی سرویس[۴۵۳] ممکن است به ms ۲۰۰ یا حتی کمتر تقلیل یابد.

در مورد منابع محلی باید گفت چنانچه فقط با یک سرور خوشه ای سروکار داشته باشیم ، مسائل تا حدودی ساده تر خواهد شد. اولاً، از آنجاییکه فقط یک شبکه وجود دارد، فقط نیازمند اعلام پیوند آدرس شبکه- به- MAC جدید خواهیم بود تا مشتریان بتوانند بکمک آن با فرآیندهای مهاجرت کرده در واسط شبکه ای صحیح تماس برقرار کنند. و ثانیاً، اگر بتوان فرض کرد که منبع ذخیره ساز طبقه جداگانه ای را به خود اختصاص می دهد (مطابق آنچه در شکل ۱۲-۳ مشاهده می کنید)، مهاجرت پیوند به فایلها هم ساده تر خواهد شد.

نتیجه و تأثیر نهایی این است که، بجای مهاجرت فرآیندها، عملاً شاهد جابجایی کل سیستم عامل در بین ماشینهای مختلف خواهیم بود .

۶-۳- خلاصه

از آنجایی که فرآیندها مبنای ارتباط بین ماشینهای مختلف را فراهم می کنند، بنابراین نقش اساسی در سیستم های توزیعی برعهده دارند. یکی از مهمترین مسائل در ارتباط با فرآیندها، نحوه سازماندهی درونی آنها و بطور اخص، پشتیبانی یا عدم پشتیبانی از چند نخ کنترل است. کاربرد ویژه نخها در سیستم های توزیعی موجب ادامه استفاده از پردازنده در حین انجام عملیات I/O مسدود شونده است. به این ترتیب، امکان ساخت سرورهای پربازدهی فراهم خواهد شد که چندین نخ را به صورت موازی با هم اجرا کرده و برخی ازآنها ممکن است تا زمان تکمیل I/O دیسک یا ارتباط شبکه ای منتظر و مسدود باقی بمانند.

---

450 live service
451 pre-copy
452 stop-and-copy phase
453 service downtimes

سازماندهی برنامه کاربردی توزیعی به صورت مشتریها و سرورها هم بسیار مفید می باشد . فرآیندهای مشتری غالباً واسط های کاربر را پیاده سازی می کنند که ممکن است از صفحه نمایشهای بسیار ساده تا واسط های پیشرفته متفاوت باشند و می توانند روی اسناد ترکیبی کار کنند . از این گذشته، یکی از اهداف نرم افزار مشتری کسب شفافیت توزیعی از طریق مخفی سازی جزئیات مربوط به ارتباط با سرورها ، محل استقرار فعلی این سرورها و چند نسخه ای یا یک نسخه ای بودن آنهاست. بعلاوه، نرم افزار مشتری تا حدودی مسئول مخفی سازی خرابی ها و بازسازی خرابی هاست.

گرچه سرورها غالباً پیچیده تر از مشتری ها هستند، اما مسائل طراحی محدودی دارند. به عنوان مثال، سرورها ممکن است تکراری یا همروند بوده، یک یا چند سرویس را پیاده سازی کرده و بدون حالت یا دارای خالت باشند. مسائل طراحی دیگر مربوط به آدرس دهی سرویسها و مکانیزم ها جهت توقف سرور پس از صدور درخواست سرویس بوده که احتمالاً در حال پردازش می باشد.

بایستی در سازماندهی سرورها به صورت خوشه توجه زیادی صرف کرد. یکی از اهداف مشترک در این خوشه ها، مخفی سازی سازمان دهی داخلی خوشه از دید دنیای خارج است. به این معنا که سازمان خوشه مخفی در برابر برنامه های کاربردی است. بدین منظور، اغلب خوشه ها فقط از یک نقطه ورودی برای تقدیم پیامها به سرورهای داخل خوشه استفاده می کنند. یکی از مشکلات چالشی در طراحی خوشه ها تعویض شفاف این نقطه ورودی بوسیله راه حلهای توزیعی می باشد.

یکی از موضوعات مهم در سیستم های توزیعی ، مهاجرت کد بین ماشینهای مختلف است. دو دلیل عمده برای پشتیبانی از مهاجرت کد، افزایش کارآیی و انعطاف پذیری است. در مواردیکه ارتباطات هزینه بر باشد، می توان با انتقال برخی از محاسبات از سرور به مشتری و فراهم کردن امکان انجام حداکثر پردازش محلی ممکنه برای مشتری اقدام به کاهش ارتباطات نمود. چنانچه مشتری بتواند به صورت پویا نرم افزار لازم برای ارتباط با یک سرور خاص را دانلود کند، انعطاف پذیری افزایش خواهد یافت. نرم افزار دانلود شده می تواند مخصوص آن سرور باشد، بدون اینکه نیازی باشد که مشتری از قبل آن را نصب کرده باشد.

مهاجرت کد مشکلاتی در واسطه با استفاده از منابع محلی بوجود می آورد . برای حل این مشکلات یا باید منابع را هم مهاجرت داد، یا پیوندهای جدید به منابع محلی ماشین هدف ایجاد شود، و یا از مراجع شبکه ای جهانی استفاده نمود . مشکل دیگر اینکه در مهاجرت کد باید مسأله ناهمگن بودن مورد توجه قرار گیرد. در وضعیت فعلی ، بهترین راه حل در این واسطه استفاده از ماشینهای مجازی است. این ماشین ها ممکن است یا به صورت ماشینهای مجازی فرآیندی باشند که مثلاً درJava بچشم می خورد و یا از ناظرهای ماشین مجازی استفاده نمود که امکان مهاجرت مؤثر مجموعه ای از فرآیندها به همراه سیستم عامل زیرین آنها را فراهم می آورد.

مسائل فصل

۱- در این مسأله باید خواندن فایل با استفاده از سرور فایل تک نخی را با استفاده از سرور چند نخی مقایسه کنید. msec ۱۵ طول می کشد تا یک درخواست را به کار گرفت، آنرا توزیع کرده و پردازش لازم را انجام داد، البته با این فرض که داده های لازم برای اینکار در حافظه پنهان در حافظه اصلی قرار داشته باشند . در صورت نیاز به عملیات دیسک، که در یک سوم موارد هم لازم است ، ۷۵msec دیگر هم برای خوابیدن نخ به رقم بالا اضافه کنید. در صورت تک نخی بودن ، سرور قادر به انجام چه تعداد درخواست در واحد ثانیه خواهد بود؟ در صورت چند نخی بودن چطور؟

۲- آیا محدود کردن تعداد نخها در فرآیند سرور کار معقولانه ای است؟

۳- در متن درس، نمونه ای از سرورهای فایل چند نخی را تشریح و دلایل برتری آن در مقایسه با سرور تک نخی شده و سرور ماشین حالت محدود را ذکر کردیم . آیا مواردی وجود دارد که سرورهای تک نخی بهتر باشند. یک مثال ذکر کنید.

۴- مرتبط ساختن ایستای فقط یک نخ با یک فرآیند سبک وزن ایده چندان مناسبی نیست. چرا؟

۵- داشتن فقط یک فرآیند سبک وزن در هر فرآیند هم چندان ایده مناسبی نیست. چرا؟

۶- طرح ساده ای را شرح دهید که در آن تعداد فرآیندهای سبک وزن موجود به اندازه تعداد نخهای قابل اجرا باشد.

۷- X ، پایانه کاربر را بعنوان میزبان سرور تعیین می کند ، در حالیکه برنامه کاربردی تحت عنوان مشتری خوانده می شود. آیا اینکار صحیح است؟

۸- پروتکل X دچار مشکلات مقیاس پذیری است. چه راه حلی برای این مشکل ارائه می دهید؟

۹- همانطور که در متن درس هم گفتیم، نرم افزار های پنهان[۴۵۴] می توانند با فراخوانی هر یک از نسخه ها از شفافیت چند نسخه ای پشتیبانی کنند. آیا(طرف سرور) یک برنامه کاربردی هم می تواند در معرض فراخوانی های کپی شده قرار گیرد؟

۱۰- ساخت یک سرور همروند از طریق تولید یک فرآیند در مقایسه با سرورهای چند نخی، مزایا و معایبی دارد. چند نمونه از هردو را ذکر کنید.

۱۱- طرحی از یک سرور چند نخی ترسیم کنید که با استفاده از سوکتها بعنوان واسط لایه انتقال با سیستم عامل از پروتکل های متعددی پشتیبانی می کند.

۱۲- چگونه می توان از دور زدن مدیر پنجره به توسط برنامه های کاربردی و ایجاد بهم ریختگی در صفحه جلوگیری کرد؟

۱۳- سروری که ارتباط TCP/IP با یک مشتری دارد بدون حالت یا دارای حالت است؟

۱۴- یک سرور وب را در نظر بگیرید که شامل جدولی است که در آن آدرسهای IP مشتری به آخرین صفحات وبی که دسترسی پیدا کرده، نگاشته شده اند. وقتی مشتری به سرور متصل می شود، سرور در

[454] Proxies

جدول خود به دنبال مشتری می گردد و در صورت یافت شدن، صفحه ثبت شده را باز می گرداند. چنین سروری دارای حالت است یا بدون حالت؟

۱۵- جابجایی قوی در سیستم های UNIX از طریق ایجاد امکان منشعب ساختن فرزند واقع روی یک ماشین دور قابل پشتیبانی است. نحوه کار را توضیح دهید.

۱۶- از شکل ۱۸-۳ می توان دریافت که جابجایی قوی با اجرای کد مهاجرت کرده در فرآیند هدف سازگار نیست. یک نمونه عکس ذکر کنید.

۱۷- فرض کنید فرآیند P نیازمند دسترسی به فایل F باشد که به صورت محلی روی ماشینی قابل دسترسی است که هم اکنون P برروی آن در حال اجراست. در صورتیکه P به ماشین دیگری انتقال پیدا کند، P بازهم نیازمند دسترسی به فایل F خواهد بود . اگر پیوند فایل به ماشین ثابت باشد، ارجاع تمام سیستمی[۴۵۵] به F چگونه قابل اجراست؟

۱۸- به تفصیل شرح دهید که در حالت تقدیم[۴۵۶] TCP ، بسته های TCP به همراه اطلاعات مربوط به آدرس های منبع و مقصد در سرآیندهای مختلف چگونه جریان پیدا می کنند؟

[455] System-wide
[456] Handoff

۴
ارتباطات

ارتباطات ميان فرآيندي قلب تپنده تمامي سيستم هاي توزيعي محسوب مي شود. در واقع، مطالعه سيستم هاي توزيعي بدون بررسي دقيق روشهاي تبادل اطلاعات از طريق فرآيندهاي ماشينهاي مختلف امري بيهوده است. در سيستم هاي توزيعي، ارتباطات همواره بر اساس ردكردن پيام سطح پايين ارائه شده از شبكه زيرين (زيربنايي؟؟؟) استوار است. ارتباطات سريع از طريق ردكردن پيام مشكلتر از به كارگيری عمليات اوليه[۴۵۷] حافظه اشتراكي[۴۵۸] است كه ويژگي روشهاي (بسترهاي؟؟؟؟) غيرتوزيعي محسوب مي شود. سيستم هاي توزيعي مدرن غالباً متشكل از هزاران يا حتي ميليونها فرآيندي است كه در سرتاسر شبكه هاي داراي ارتباطات نامطمئن از جمله اينترنت پراكنده شده اند. تا زمانيكه امكانات ارتباطي ابتدايي شبكه هاي كامپيوتري تعويض و يا جايگزين نشوند، توسعه برنامه هاي كاربردي توزيعي پردامنه تقريباً غيرممكن خواهد بود.
در اين فصل، ابتدا راجع به قوانيني به نام پروتكل بحث مي كنيم كه فرآيندهاي ارتباطي بايستي براساس آن صورت گرفته و روي لايه اي كردن اين پروتكل ها تمركز خواهيم كرد.سپس نظري به سه مدل ارتباطي پركاربرد ،شامل فراخواني روال از راه دور[۴۵۹] (PRC)، ميان افزار پيام گرا[۴۶۰] (MOM) و رشته اي كردن داده ها[۴۶۱] خواهيم داشت. همچنين درمورد مشكل عمومي ارسال داده براي چندين گيرنده، يا چندپخشي[۴۶۲] ،بحث خواهيم كرد.
بعنوان اولين مدل ارتباطات در سيستم هاي توزيعي ،فراخواني روال از راه دور(RPC) را در نظر مي گيريم. هدف از RPC مخفي سازي بخش عظيمي از ظرافتهاي ردكردن پيام است و به همين دليل، براي برنامه هاي كاربردي مشتري- سرور ايده آل مي باشد.
در بسياري از برنامه هاي كاربردي توزيعي ، ارتباطات از الگوي بسيار خشك تبادل مشتري- سرور[۴۶۳] تبعيت نمي كند. در نتيجه، در چنين مواردي بررسي خود پيام مفيدتر خواهد بود. با اين وجود، امكانات ارتباطات سطح پايين شبكه هاي كامپوتري، به دليل عدم شفافيت توزيعي، ازبسياري جهات نامناسب است. يك راه حل، استفاده از مدل صف بندي پيام سطح بالا[۴۶۴] است كه ارتباطات آن تاحد زيادي مشابه سيستم هاي پست الكترونيك صورت مي پذيرد.
اهميت موضوعي ميان افزار پيام گرا(MOM) به حدي است كه مي توان يك مبحث كامل را به آن اختصاص داد.
با ظهور سيستم هاي توزيعي چندرسانه اي مشخص شد كه بسياري از سيستم ها از ارتباطات رسانه هاي پيوسته اي از قبيل رسانه هاي صوتي و شكلي پشتيباني نمي كنند. در واقع لازم است با ايجاد

---

[457] primitive
[458] shared memory
[459] Remote Procedure Calls
[460] Message-Oriented Middleware
[461] data streaming
[462] multicasting
[463] client-server
[464] high-level message-queuing model

رشته، از جريان پيوسته پيامهاي دچار محدوديت زمان بندي مختلف پشتيباني شود. در مورد جريانات در بخشي جداگانه بحث خواهيم كرد.
در پايان،از آنجاييكه دانش ما نسبت به ايجاد امكانات چندپخشي افزايش يافته ، راه حلهاي جديد و بسيار مناسبي هم جهت انتشار داده ها بوجود آمده است . از اينرو ، بخش پاياني فصل را به همين موضوع اختصاص داده ايم.

۱-۴- مباني

پيش از شروع بحث در مورد ارتباطات در سيستم هاي توزيعي، ابتدا برخي از مسائل ريشه اي در زمينه ارتباطات را به طور اجمالي مورد بحث قرار خواهيم داد. در بخش بعد، مختصراً در مورد پروتكلهاي ارتباطات شبكه اي، از جمله آنهايي كه اساس سيستم هاي توزيعي را تشكيل مي دهند، بحث خواهيم كرد. سپس،با دسته بندي انواع مختلف ارتباطات در سيستم هاي توزيعي ، روش متفاوتي را معرفي مي كنيم.

۱-۱-۴- پروتكلهاي لايه اي

در سيستم هاي توزيعي،به دليل عدم وجود حافظه اشتراكي ،تمامي ارتباطات براساس ارسال و دريافت پيامها (ي سطح پايين) استوار است.در صورتيكه فرآيند A بخواهد با فرآيند B ارتباط برقرار كند،ابتدا در فضاي آدرس خود يك پيام ايجاد مي كند.سپس با اجراي فراخوان سيستمي[۴۶۵] ، به سيستم عامل امكان مي دهد تا پيام را از طريق شبكه به B ارسال كند.هر چند اين ايده بسيار ساده مي نمايد، A و B بايستي در مورد معناي بيتهاي ارسال شده به توافق برسند، وگرنه احتمال بروز هرنوع بحراني وجود خواهد داشت.چنانچه A داستان كوتاه تازه اي را به زبان فرانسه ارسال كند كه براساس كد حروف EBCDIدرIBM كدگذاري شده ، ولي B انتظار دريافت ليست موجودي كالاي سوپرماركت نوشته شده به زبان انگليسي و كدگذاري شده در ASCII را داشته باشد،آنگاه ارتباط در حد بهينه نخواهد بود.
لازم است راجع به موارد مختلفي توافق شود، از قبيل اينكه براي ارسال بيت صفر از چه مقدار ولت و براي بيت يک از چه مقدار بايستي استفاده شود؟گيرنده بيت آخر را چگونه شناسايي مي كند؟يا درصورت آسيب ديدگي يا مفقود شدن، پيام چگونه قابل رديابي خواهد بود يا براي يافتن پيام به چه صورت عمل خواهد شد.اعداد، جريانات و ديگر اقلام داده اي چه طولي داشته و چگونه نمايش داده مي شوند؟به طور خلاصه،بايستي در تمامي سطوح و موارد،از جزئيات سطح پايين انتقال بيت گرفته تا جزئيات سطح بالاي نحوه بيان اطلاعات توافق صورت گيرد.
به منظور تسهيل بحث در مورد سطوح و مسائل متعدد مربوط به ارتباطات، مؤسسه بين المللي استاندارد[۴۶۶] (ISO)با ارائه مدل مرجع،اقدام به شناسايي دقيق سطوح مختلف مربوطه نمود . بعلاوه، اسامي استانداردي را براي هريك تعيين و مسئوليت هر سطح را مشخص نموده است.اين مدل، **مدل مرجع ميان اتصالي سيستم هاي**

[465] system call
[466] International Standard Organization

باز[467]- OSI -(Day and Zimmerman،۱۹۸۳) يا به طور خلاصه **ISO OSI** و يا **مدل OSI** ناميده مي شود. لازم به ذكر است كه پروتكل هاي ايجادي بعنوان بخشي از مدل OSI هرگز مورد كاربرد وسيع و گسترده قرار نگرفت و امروزه ديگر به طور كامل منسوخ شده اند. با اين وجود، بررسي اين مدل در درك شبكه هاي كامپيوتري بسيار مفيد خواهد بود. هرچند قصد نداريم در اينجا شرح كاملي از مدل مذكور و اثرات جانبي آن ارائه دهيم ، تعريف مختصر آن خالي از لطف نخواهد بود. براي مطالعه دقيق تر موضوع به Tanenbaum (۲۰۰۳) مراجعه شود.

مدل OSI جهت ايجاد ارتباط بين سيستم هاي باز طراحي شده است. سيستم باز سيستمي است كه آمادگي دارد تا براساس قوانين استاندارد مربوط به قالب، محتوا و معناي پيام هاي ارسال و دريافت شده، با هر سيستم باز ديگري ارتباط برقرار نمايد. اين قوانين تحت عنوان **پروتكل**[468] خوانده و مدون مي شوند . براي آنكه يك گروه كامپيوتر بتوانند در شبكه با هم ارتباط برقرار نمايند، بايستي همگي از پروتكل هاي مشتركي استفاده كنند. در اين رابطه، دو نوع پروتكل با برخي تفاوتها وجود دارد. در **پروتكل هاي اتصال گرا**[469]، فرستنده و گيرنده پيش از تبادل داده ها ، ابتدا به وضوح يك اتصال ايجاد كرده و احتمالاً در مورد پروتكل مورد استفاده خود بحث و چانه زني مي نمايند . در پايان، اتصال بايستي آزاد(متوقف) شود. دستگاه تلفن نمونه اي از سيستم هاي اتصال گرا است. اما در **پروتكل هاي بدون اتصال**[470]، نيازي به ايجاد هيچ تنظيم قبلي وجود ندارد. اولين پيام به محض آماده شدن توسط فرستنده ارسال مي شود . انداختن نامه در صندوق پست نمونه اي از ارتباطات بدون اتصال است. در كامپيوترها، هردو صورت ارتباطات اتصال گرا و بدون اتصال كاربرد دارد.

درمدل OSI، مطابق شكل ۱-۴ ،ارتباط به هفت سطح يا لايه تقسيم مي شود و هر لايه مسئول جنبه خاصي از ارتباط مي باشد. به اين ترتيب، مسأله را مي توان به چندين بخش معنادار تقسيم و هريك را مستقل از بخش هاي ديگر حل نمود. بعلاوه، هر لايه رابطي[471] براي لايه فوقاني خود ايجاد مي كند. اين رابط مجموعه اي از عملياتي است كه رويهم رفته معرف سرويس قابل ارائه توسط لايه به كاربران خود ارائه خواهد بود.

شكل ۱-۴- لايه ها،رابط ها و پروتكلهاي مدل OSI(داخل شكل: برنامه كاربردي،پروتكل برنامه كاربردي، ارائه[472]،پروتكل ارائه،جلسه[473]،پروتكل جلسه، انتقال[474] ، پروتكل انتقال، شبكه[475] ،پروتكل شبكه،پيوند داده[476] ،پروتكل پيوند داده ،فيزيكي[477] ،پروتكل فيزيكي).

---

467 Open Systems Interconnection Reference Model
468 protocol
469 connection-oriented protocols
470 connectionless protocols
471 interface
472 presentation
473 session
474 transport
475 network
476 data link

هنگاميكه فرآيند *A* روی ماشين ۱ قصد برقراري ارتباط با فرآيند *B* روی ماشين ۲ را داشته باشد،پس از ساخت پيام، آنرا به لايه برنامه كاربردي ماشين خود انتقال مي دهد. اين لايه ممكن است بعنوان مثال يك روال كتابخانه اي باشد، ولی به روش ديگري نيز قابل پياده سازی است (مثلاً داخل سيستم عامل يا روي يک پردازشگر شبكه بيروني و غيره). سپس نرم افزار لايه كاربردي ،يک **سرپيام**[۴۷۸] را به بخش اوليه پيام اضافه نموده و پيام را نهايتاً از طريق رابط لايه ۶/۷ به لايه ارائه انتقال مي دهد.لايه ارائه هم متقابلاً سرپيام خود را اضافه و نتيجه را به لايه جلسه ارسال مي كند و الي آخر.برخي لايه ها علاوه بر افزودن سرپيام، پشت بندي[۴۷۹] را هم به انتهاي آن اضافه مي كنند.در پايان،لايه فيزيكي با قرار دادن پيام روي رسانه ارسال فيزيكي، واقعاً پيام را كه ممكن است در اين حالت مشابه شكل ۲-۴ باشد، ارسال مي كند.

شكل ۲-۴-نمونه اي از شكل پيام در شبكه(داخل شكل:سرپيام لايه پيوند داده، سرپيام لايه شبكه، سرپيام لايه انتقال،سرپيام لايه جلسه،سرپيام لايه ارائه، سرپيام لايه برنامه كاربردي).

پيام پس از رسيدن به ماشين۲ ، به سمت بالا می رود، در هر لايه سرپيام مربوطه جدا شده و بازبينی می شود. در نهايت، پيام به گيرنده يعني به فرآيند ۲ مي رسد و اين فرآيند در مسير عكس به آن جواب مي دهد. از اطلاعات داخل سرپيام لايه n جهت پروتكل لايه n هم استفاده مي شود.

براي درك صحيح اهميت پروتكل هاي لايه اي شده، ارتباطات بين دو شركت خطوط هوايي Zippy و تهيه غذاي Mushy Meals را در نظر بگيريد. هر ماه، رئيس خدمات مسافربريZippy از طريق منشي خود با مدير فروش Mushy تماس گرفته و تعداد ۱۰۰۰۰۰ بسته جوجه شفارش مي دهد.سابقاً كار سفارش دهي از طريق پست انجام مي شد.اما به دليل نامناسب بودن خدمات پستي ، منشي ايندو شركت تصميم مي گيرند كه بوسيله پست الكترونيك باهم ارتباط برقرار كنند.چون در پروتكلهاي مورد توافق آنها در مورد انتقال فيزيكي سفارشات بحث شده نه محتواي آنها،منشي ايندو شركت مي توانند بدون ايجاد كوچكترين مزاحمت يا حتي اطلاع رؤساي خود،كار سفارشات را انجام دهند.

به همين ترتيب،رئيس خدمات مسافربري مي تواند به راحتي سفارش جوجه را كنسل كرده و به جاي آن،غذاي ويژه Mushy را سفارش دهد.توجه كنيد كه اين تصميم جديد هيچ ارتباطي با منشي ها ندارد و ممكن است آنها حتي كوچكترين اطلاعي هم از سفارش جديد نداشته باشند. مسأله مهم آن است كه در اين مورد، دو لايه داريم يكي رؤسا و ديگري منشي ها. به علاوه، هر لايه پروتكل خاصي دارد كه مستقل از ديگري قابل تغيير است .دقيقاً همين استقلال مبحث پروتكلهاي لايه اي را به موضوعي جذاب تبديل كرده است . در واقع هر لايه را مي توان بدون كوچكترين اثري بر لايه ديگر و بر اساس آخرين فن آوري روز اصلاح نمود.

---

[477] physical
[478] header
[479] trailer

در مدل OSI فقط دو لايه وجود ندارد ، بلكه مطابق شكل ۱-۴ ، با هفت لايه سروكار داريم. مجموعه پروتكل هاي مورد استفاده در يك سيستم خاص، **مجموعه پروتكل**[۴۸۰] يا **بسته پروتكل**[۴۸۱] ناميده مي شود. مسأله مهم ديگر، تمايز قائل شدن بين *مدل مرجع* و *پروتكلهاي* واقعي آن است. همانطور كه قبلاً هم اشاره كرديم، پروتكل هاى OSI هرگز كاربرد گسترده نيافتند. در حالي كه پروتكل هاي ساخته شده براي اينترنت از قبيل TCP و IP رواج فراواني دارند.در بخش هاي بعد، هر يك از لايه هاي OSI را به نوبت و مختصراً (از پايين به بالا) مورد بررسي قرار خواهيم داد. همچنين، به جاي ارائه نمونه هايي در مورد پروتكل هاي OSI ، هر كجا مناسب باشد به برخي از پروتكل هاي اينترنتي مربوط به هر لايه اشاره خواهيم كرد.

پروتكل هاي سطح پايين تر

بحث را با سه لايه زيرين مجموعه پروتكل OSI آغاز مي كنيم. اين لايه ها در مجموع،كاركردهاي اساسي يك شبكه كامپيوتري را پياده سازی مي كنند.

لايه فيزيكي مسئول ارسال ۰و۱ است. مقدار ولت لازم براي ارسال ۰و۱ ، تعداد بيتهاي قابل ارسال در هر ثانيه و امكان ارسال همزمان در هر دو جهت از جمله مسائل كليدي در لايه فيزيكي هستند. بعلاوه، ابعاد و شكل اتصال دهنده شبكه (plug) و همچنين پين ها و معنی هريك مورد بحث قرار مي گيرد.

پروتكل لايه فيزيكي به استاندارد سازي رابط هاي الكتريكي ، مكانيكي و سيگنال دهي مي پردازد تا هر زماني كه ماشين بيت صفر ارسال مي كند، واقعاً هم به صورت بيت صفر دريافت شود نه بيت يك . استاندارد هاي لايه فيزيكي متعددي(براي رسانه هاي مختلف) ايجاد و ارائه شده كه ازآن جمله مي توان به استاندارد RS-232-C براي خطوط ارتباطي سريال اشاره كرد.

لايه فيزيكي صرفاً مسئول ارسال بيت است. مادامیكه هيچ خطايي انجام نشود، همه چيز روال عادي خود را طي خواهد كرد. اما ، شبكه هاي ارتباطي واقعي در معرض خطا قرار دارند. به همين دليل، براي رديابي و اصلاح اين دست خطاها نيازمند مكانيزم هاي لازم هستيم. در واقع، اين مكانيزم ها وظيفه اصلي لايه پيوند داده است. اين لايه بيت ها را به چندين واحد كه در برخي موارد قاب[۴۸۲] ناميده مي شود، دسته بندي كرده و سپس بر دريافت درست هر قاب نظارت مي كند.

وظيفه لايه پيوند داده، قرار دادن يك الگوي خاص بيت در انتها و ابتداي هر فريم جهت نشانه گذاري آنهاست. همچنين با جمع كردن تمامي بيت هاي قاب به روشي خاص، اقدام به محاسبه **حاصلجمع آزمايشي**[۴۸۳] مي كند. لايه پيوند داده اين حاصلجمع آزمايشي را به قاب ضميمه مي كند.گيرنده پس از دريافت قاب، حاصلجمع آزمايشي را برحسب داده دريافت شده محاسبه نموده و نتيجه را با حاصلجمع آزمايشي مربوط به قاب مقايسه مي كند. در صورت تطابق، قالب

[480] protocol suite
[481] protocol stack
[482] frame
[483] checksum

صحيح بوده و پذيرفته مي شود. در غيراينصورت، گيرنده از فرستنده درخواست ارسال مجدد آنرا مي نمايد. (در بخش سرپيام) به هريك از قالب ها يك عدد ترتيب اختصاص مي يابد تا بتوان هريك را به دقت شناسايي نمود.
در شبكه های محلی (LAN) ، معمولاً نيازي به مكان يابي گيرنده توسط فرستنده وجود ندارد، بلكه صرفاً پيام توسط فرستنده روي شبكه قرار گرفته و گيرنده آنرا اخذ مي كند. ولی در شبكه هاي گسترده، تعداد زيادي ماشين وجود دارد كه هر يك با تعدادی خط ارتباطی به ماشين هاي ديگر وصل مي شود و از اين نظر بسيار شبيه نقشه بزرگي از شهرها و راههاي مواصلاتي بين آنهاست. براي آنكه يك پيام از فرستنده به گيرنده برسد، ممكن است چند گذر[۴۸۴] داشته و در هر گذر از يك خط استفاده كند. مسأله نحوه انتخاب بهترين مسير، **مسيريابي**[۴۸۵] ناميده شده واساساً وظيفه اصلي لايه شبكه است.
اما اين واقعيت كه كوتاه ترين مسير هميشه بهترين نيست، تا حدودي باعث پيچيدگي مسأله مسيريابی مي شود. آنچه در واقع عامل اين مشكل مي باشد، مقدار تأخيرها در يك مسير مفروض است كه آنهم به نوبه خود ، با ميزان ترافيك و تعداد پيام هاي منتظر براي ارسال روي خطوط مختلف ارتباط دارد. بنابراين، تأخير مي تواند در طول زمان تغيير كند. در برخي الگوريتم هاي مسيريابي سعي شده مسأله تطبيق با تغيير بار لحاظ شود؛ در حاليكه در برخي ديگر، تصميمات فقط بر اساس ميانگين هاي بلند مدت گرفته مي شود.
در حال حاضر، پركاربردترين پروتكل شبكه، پروتكل بدون اتصال **IP (پروتكل اينترنت[۴۸۶])** است كه بخش مهمي از مجموعه پروتكل اينترنت محسوب مي شود. **بسته IP**[۴۸۷] (اصطلاح فني مورد استفاده براي پيام در لايه شبكه) را مي توان بدون هيچ تنظيم و تغييري ارسال نمود. هريك از بسته هاي IP مستقل از ديگري به سمت مقصد مسيردهي مي شود. بعلاوه، هيچ مسير داخلي از پيش تعيين و انتخاب نمي شود.

پروتكل هاي انتقال

لايه انتقال بخش آخر چيزي را تشكيل مي دهد كه مجموعه اساسی پروتكل شبكه ناميده مي شود. دليل نامگذاري آن است كه لايه مذكور قادر به انجام كليه سرويس هايي است كه گرچه منطقاً لازمه ساخت برنامه هاي كاربردي شبكه مي باشند، اما در رابط لايه شبكه ارائه نمي شود . به بيان ديگر،لايه انتقال ، شبكه زيرين را براي سازنده برنامه كاربردي قابل استفاده مي نمايد.
ممكن است برخي بسته ها در طي مسير خود از فرستنده به گيرنده مفقود شوند. هر چند برخي برنامه هاي كاربردي قادر به بازسازي و اصلاح خطاي خود هستند، برخي ديگر به دنبال ايجاد يك اتصال قابل اطمينان و بدون عيب هستند.ارائه اين سرويس وظيفه لايه انتقال است . به اين معني كه لايه انتقال بايستي قادر به تحويل

---
[484] hop
[485] routing
[486] Internet Protocol
[487] IP packet

پيام به لايه انتقال باشد و اين تحويل بايستي بدون هيچ خدشه اي به پيام انجام شود.
لايه انتقال به محض دريافت پيام از لايه برنامه كاربردي آنرا به قطعات كوچكي تبديل مي كند. اندازه قطعات بايد به حدي باشد كه ارسال آنها امكانپذير باشد.سپس به هر قطعه يك شماره ترتيب الصاق و ارسال مي شود. بحث در مورد سرپيام لايه انتقال روي موضوعاتي از اين قبيل تمركز مي كند: كدام بسته ها ارسال شده اند،كدام بسته ها دريافت شده اند،گيرنده چه مقدار فضا براي دريافت پيام در اختيار دارد، كدام پيام بايد مجدداً ارسال شود و ساير مواردي از اين دست.
اتصالات انتقال مطمئن را (كه برحسب تعريف اتصال گرا هستند) مي توان توسط خدمات شبكه اتصال گرا يا بدون اتصال با موفقيت كنترل نمود.در حالت اول،تمامي بسته ها به ترتيب صحيح خواهند رسيد(البته اگر به مقصد برسند)، اما در حالت دوم ممكن است يكي از بسته ها مسير ديگري را پيش گرفته و زودتر از بسته اي كه پيش از آن ارسال شده، به مقصد برسد.اين وظيفه نرم افزار لايه انتقال است كه تمامي بسته ها را به صورتي بازگرداند كه گويا اتصال انتقال شبيه يك لوله بزرگ عمل مي كند – پيام در اين لوله قرار گرفته و سالم و بدون كوچكترين آسيبي، به همان ترتيب ورود، خارج مي شود. ايجاد اين رفتار ارتباطي مرتب و منظم يكي از جنبه هاي مهم لايه انتقال است.
پروتكل انتقال اينترنت به نام**TCP (پروتكل كنترل ارسال[488])** خوانده شده و به تفصيل در كتاب (۲۰۰۶)Comer شرح داده شده است. امروزه از تركيب TCP/IP بعنوان يك استاندارد كاربردي در ارتباطات شبكه اي استفاده مي شود. بعلاوه، مجموعه پروتكل اينترنت از يك پروتكل انتقال بدون اتصال به نام **UDP** (پروتكل جهاني ديتاگرم[489]) پشتيباني مي كند كه در واقع همان IP قديمي بعلاوه برخي اضافات جزئي است. در برنامه هاي كاربراني كه نيازي به پروتكل اتصال گرا ندارند، غالباً از UDP استفاده مي شود.
پروتكل هاي انتقالي جديد ديگري هم مرتباً به بازار عرضه مي شوند. بعنوان مثال، جهت پشتيباني از انتقال داده بی درنگ[490]، **پروتكل انتقال بی درنگ[491] (RTP)** تعريف و ارائه شده است. RTP يك پروتكل چارچوبی بوده و قالب بسته ها را جهت داده هاي بي درنگ تعيين و تعريف مي كند. اينكار بدون ايجاد مكانيزم هاي عملي تضمين تحويل پيام صورت مي پذيرد. بعلاوه، پروتكلي جهت نظارت و كنترل بر انتقال داده هاي بسته هاي RTP تعيين مي كند (۲۰۰۳،Schulzrinne و گروه همكاران) .

پروتكل هاي سطح بالاتر

OSI سه لايه ديگر هم در بالاي لايه انتقال مشخص نموده، اما بطور عملي تاكنون فقط لايه برنامه كاربردي استفاده شده است. در واقع، در مجموعه پروتكل اينترنت، هر چيزي كه بالاتر از لايه

[488] Transmission Control Protocol
[489] Universal Datagram Protocol
[490] real-time
[491] Real-time Transport Protocol

انتقال قرار گيرد در اين گروه دسته بندي مي شود. در اين بخش مشاهده خواهيم كرد كه براي سيستم هاي ميان افزاري، هيچيك از دو روش OSI و اينترنت واقعاً مناسب نيستند .
لايه جلسه[۴۹۲] در اصل نسخه تقويت شده لايه انتقال است. اين لايه گفتگو را كنترل مي كند تا بدين وسيله امكان پي گيري طرف گفتگو كننده فراهم آيد. بعلاوه، امكانات همگام سازي را ايجاد مي كند. اين امكانات به كاربر اجازه مي دهد تا در انتقالات طولاني، نقاط آزمايشي[۴۹۳] را درج نمايد تا در صورت بروز خرابي به جاي تكرار مجدد مسير، فقط آخرين نقطه آزمايشي بررسي شود. درعمل، برنامه هاي كاربردي بسيار محدودي به لايه جلسه علاقمند بوده و به ندرت پشتيباني می شوند. اين لايه حتي در مجموعه پروتكل اينترنت هم وجود ندارد. اين در حالي است كه مفهوم جلسه و پروتكلهاي مربوط به آن در روند ارائه راه حل براي ميان افزارها و خصوصاً در هنگام تعريف پروتكلهاي ارتباطات سطح بالاتر نقش مهمي ايفا مي كند.
لايه ارائه، برخلاف لايه هاي زيرين خودكه مسئول دريافت بيت ها و ارسال موثق و مؤثر آنها به گيرنده هستند، مسئول معناي بيتهاست. بطوريكه اغلب پيامها متشكل از زنجيره هاي بيت تصادفي نيستند، بلكه حاوي اطلاعات سازماندهي شده اي از قبيل اسامي افراد، آدرس ها و مقدار موجودي آنها و غيره مي باشد. در لايه ارائه، امكان تعريف گزارشات حاوي موضوعاتي از اين دست وجود دارد. سپس ، فرستنده به گيرنده اطلاع مي دهد كه پيام حاوي گزارشي خاص در قالبي خاص است. اين روش باعث تسهيل ارتباط ماشين هاي مختلف يا اجزاي داخلي مختلف مي شود.
هدف اوليه طراحي لايه برنامه كاربردي OSI شمول مجموعه اي از برنامه هاي كاربردي استاندارد شبكه از قبيل پست الكترونيك، انتقال فايل و شبيه سازي ترمينال بود. اما امروزه تبديل به ظرفي براي گنجاندن تمامي برنامه هاي كاربردي و پروتكل هايي شده كه در لايه های زيرين نمی گنجند. از نقطه نظر مدل OSI، تمامي سيستم هاي توزيعي ذاتاً و صرفاً برنامه هاي كاربردي هستند.
آنچه در اين ميان لحاظ نشده، تمايز آشكار بين برنامه هاي كاربردي ، پروتكلهاي كاربردگرا و پروتكلهاي عام منظوره است. بعنوان مثال، **پروتكل تبادل فايل** اينترنت[۴۹۴] **(FTP)** (Horowitz and ،1997 Lunt؛ ۱۹۹۸ ،Postel and Reynolds) پروتكلي جهت تبادل فايلها بين ماشين مشتري و سرور تعريف مي كند. اين پروتكل را نبايد با برنامه ftp يكي دانست؛ برنامه مذكور برنامه كاربردي كاربر نهايي جهت انتقال فايل ها است كه تصادفاً پروتكل FTP اينترنت را اجرا مي كند.
بعنوان نمونه ديگري از انواع پروتكل هاي كاربردگرا مي توان به **پروتكل انتقال ابرمتن**[۴۹۵] **(HTTP)** اشاره كرد(۱۹۹۱، Fielding و گروه همكاران) كه جهت مديريت و هدايت انتقال صفحات وب ازراه دور

[492] session layer
[493] checkpoint
[494] File Transfer Protocol
[495] Hypertext Transfer Protocol

طراحي شده است. اين پروتكل بوسيله برنامه هاي كاربردي از قبيل جستجوگرهاي وب و سرورهاي وب اجرا مي شود. با اين وجود، HTTP امروزه در سيستم هايي هم كه ذاتاً ارتباطي با وب ندارند، به كار برده مي شود. بعنوان مثال، مكانيزم فراخوانی شيء از راه دور Java از HTTP جهت درخواست فراخوانی اشياء از راه دوري استفاده مي كند كه تحت پشتيباني ديواره آتش قرار دارند (Sun Microsystems، ۲۰۰۴).

پروتكل هاي عام منظوره متعدد ديگري هم وجود دارند كه براي بسياري از برنامه هاي كاربردي مناسب هستند، اما نمي توان آنها را در رديف پروتكل هاي انتقال قرار داد، بلكه بيشتر بعنوان پروتكل هاي ميان افزار تلقي مي شوند. به همين دليل بحث در مورد آنها را به بعد موكول مي كنيم.

پروتكل هاي ميان افزار

ميان افزار ،كاربردي است كه (در اغلب موارد) منطقاً در لايه برنامه كاربردي قرار مي گيرد، اما حاوي تعداد زيادي از پروتكلهاي عام منظوره اي است كه لايه ای مخصوص به خود و مستقل از كاربردهای مشخص را طلب مي كنند. قابل ذكر است كه بين پروتكلهاي ارتباطات سطح بالا و پروتكل هاي مربوط به ايجاد سرويسهاي مختلف ميان افزاري تفاوت وجود دارد.

پروتكل هاي متعددي جهت پشتيباني از سرويسهاي ميان افزاري وجود دارد. بعنوان مثال، همانطور كه در فصل۹ هم خواهيم گفت، روشهاي مختلفي براي تصديق هويت، يعني اثبات هويت ادعا شده، وجود دارد. پروتكل هاي تصديق با هيچيك از برنامه هاي كاربردي وابستگی خاص نداشته و به همين دليل مي توان آنها را به عنوان يك سرويس عمومي در سيستم ميان افزاري تلفيق نمود. به همين ترتيب ،آندسته از پروتكل هاي صدور مجوز كه فرآيندها و كاربران تصديق شده به كمك آنها فقط به منابعی كه مجوز آن را دارند دسترسي پيدا مي كنند، تمايل دارند كه ماهيت عمومي و مستقل از برنامه كاربردي داشته باشند.

بعنوان نمونه ديگر، تعدادي از پروتكل هاي تعهد توزيعي[۴۹۶] را در فصل ۸ بررسي خواهيم كرد. پروتكل هاي تعهد اثبات مي كند كه در گروهي از فرآيندها، يا تمامي فرآيندها يك عمليات خاص را انجام مي دهند يا اينكه اصلاً هيچ عملياتي انجام نمي شود. اين پديده **اتمي بودن**[۴۹۷] هم خوانده مي شود و كاربرد گسترده اي در تبادل ها[۴۹۸] يافته است. همان طور كه بعداً خواهيم ديد، علاوه بر تبادل ها ، برنامه هاي كاربردي ديگري از قبيل برنامه هاي تحمل خرابي هم مي توانند از پروتكل هاي تعهد توزيعي بهره مند شوند.

بعنوان آخرين نمونه، پروتكل هاي قفل توزيعي را در نظر بگيريد كه مي توان براساس آن و به كمك مجموعه اي از فرآيندها كه در ماشين هاي مختلف توزيع شده اند، از يك منبع در برابر دسترسي هاي همزمان محافظت نمود. در فصل ۶ تعدادي از اين پروتكل ها را بررسي خواهيم كرد. اين نيز مثالی از پروتكلی است كه از آن مي

[496] distributed commit protocols
[497] atomicity
[498] transaction

توان جهت پياده سازی يك سرويس عمومي ميان افزاري استفاده نمود، در عين حال بسيار مستقل از هر برنامه كاربردي خاصي است. پروتكل هاي ارتباطات ميان افزاري از سرويس هاي ارتباطات سطح بالا پشتيباني مي كنند. بعنوان مثال، در دو بخش بعد راجع به پروتكل هايي بحث خواهيم كرد كه باعث مي شود تا فرآيند به روشي بسيار شفاف ، رويه مفروضي را فراخواني يا شيء روي ماشين دوري را فراخواني كند . به همين ترتيب، سرويس هاي ارتباطي سطح بالايي هم براي ايجاد و همگام سازي جريانات[۴۹۹] جهت انتقال داده هاي زمان دار وجود دارد كه بعنوان نمونه در برنامه هاي كاربردي چندرسانه اي استفاده دارند. مثلاً برخي سيستم هاي ميان افزاري خدمات چندپخشي قابل اطميناني را ارائه مي دهد كه به هزاران گيرنده پراكنده در سرتاسر يك شبكه وسيع توسعه يابد.
برخي از پروتكل هاي ارتباطات ميان افزار را مي توان به لايه انتقال هم نسبت داد، اما ممكن است دلايل خاصي براي نگه داشتن آنها در سطحي بالاتر وجود داشته باشد. بعنوان مثال، سرويس هاي چندپخشي مطمئنی كه مقياس پذيري را تضمين مي كند فقط در صورتي قابل پياده سازی هستند كه نيازمندی های برنامه كاربردي را در نظر بگيرند. در نتيجه، يك سيستم ميان افزاري ممكن است پروتكل هاي مختلف (قابل تنظيم) را ارائه دهد كه هرچند هريك با استفاده از پروتكلهاي انتقالي مختلفي پياده سازی شده، يك رابط واحد را ارائه مي دهد.

شكل ۳-۴- مدل مرجع تغييريافته براي ارتباطات شبكه اي شده (داخل شكل: برنامه كاربردي،پروتكل برنامه كاربردي،ميان افزار، پروتكل ميان افزار، انتقال، پروتكل انتقال، شبكه، پروتكل شبكه، پيوند داده، پروتكل پيوند داده، فيزيكي، پروتكل فيزيكي، شبكه).

استفاده از اين روش در لايه بندي منجر به توليد مدل مرجع نسبتاً سازگاري جهت ارتباطات(مطابق شكل ۳-۴) مي شود. در قياس با مدل OSI، به جاي لايه ارائه و جلسه از يك لايه ميان افزاري جداگانه استفاده شده كه حاوي پروتكل هاي مستقل از برنامه كاربردي است. اين پروتكل ها متعلق به لايه هاي زيرين پيشتر گفته شده نيست. سرويس هاي اصلي انتقال ممكن است بدون هيچ اصلاح و تغييري بعنوان يك سرويس ميان افزاري هم ارائه شوند. اين روش تا حدودي مشابه ارائه UDP در لايه انتقال است. به همين ترتيب، سرويس هاي ارتباطات ميان افزار ممكن است حاوي سرويسهاي ردكردن پيام قابل قياس با سرويس هاي لايه انتقال باشند.
در ادامه فصل، روي چهار سرويس ارتباطات ميان افزار سطح بالا يعني فراخواني روال از راه دور(RPC)، سرويس هاي صف بندي پيام، پشتيباني از ارتباطات پيوسته رسانه اي از طريق جريانات، و چندپخشي ها تمركز خواهيم كرد. اما پيش از آن ، معيارهاي كلي ديگري براي تمايز ارتباطات (ميان افزاري) وجود دارد كه ذيلاً در مورد آنها بحث خواهيم كرد.

۲-۱-۴- انواع ارتباطات

جهت درك روش هاي مختلف ارتباطي كه ميان افزارها براي برنامه هاي كاربردي ايجاد مي كنند، بهتر است مطابق شكل ۴-۴ ميان افزار را به عنوان يك سرويس افزوده در محاسبه مشتري- سرور

[499] streams

تصور نمود. براي مثال، يك سيستم پست الكترونيك را درنظر بگيريد. بطور كلي، هسته سيستم تحويل پست را مي توان يك سرويس ارتباطي ميان افزاري تلقي نمود. هر ميزبان با اجراي نماينده كاربر، به او امكان مي دهد تا پست الكترونيك خود را ساخته، ارسال و دريافت نمايد. نماينده كاربر ارسال كننده پست الكترونيك را به سيستم تحويل پست انتقال داده و انتظار دارد كه آنهم متقابلاً و در نهايت پيام پستي را به دريافت كننده مورد نظر تحويل دهد. به همين ترتيب، نماينده كاربر طرف دريافت كننده، به سيستم تحويل پيام پستي متصل شده و پيامهاي ورودي احتمالي را چك مي كند. اين پيامها در صورت وجود ، به نماينده كاربر انتقال داده مي شوند تا كاربر آنها را مشاهده كرده و بخواند.

شكل ۴-۴- نمايش ميان افزار بعنوان يك سرويس مياني(توزيع شده) در ارتباطات سطح برنامه كاربردي(داخل شكل: همزمان سازي در هنگام ارائه درخواست، همگام سازي در هنگام تحويل(ارائه؟؟؟؟؟) درخواست، همگام سازي پس از پردازش توسط سرور، وقفه در انتقال ، تجهيزات ذخيره سازي، سرور، پاسخ، زمان).

سيستم پست الكترونيك نمونه بارزي از ارتباطات پايدار است. در **ارتباطات پايدار**[500]، پيام تسليمي جهت ارسال بوسيله ميان افزار ارتباطي ذخيره شده و مادامیكه ميان افزار اقدام به انتقال آن به گيرنده نكند، در همانجا باقي خواهد ماند. در اين حالت، مطابق شكل ۴-۴ ، پيام توسط ميان افزار در يك يا چند وسيله ذخيره سازي، ذخيره مي شود. در نتيجه، ديگر لازم نيست تا برنامه كاربردي ارسال كننده پس از تسليم پيام هم به اجرا ادامه دهد. همچنين لزومي ندارد كه پس از تسليم پيام، برنامه كابردي دريافت كننده در حال اجرا باشد.

بالعكس، در **ارتباطات ناپايدار**[501]، پيام فقط تا زماني توسط سيستم ارتباطي ذخيره مي شود كه برنامه كاربردي دريافت و ارسال در حال اجرا باشند. به بيان دقيق تر، براساس شكل ۴-۴، در صورت وقفه در روند ارسال يا غيرفعال شدن گيرنده، ميان افزار ديگر قادر به تحويل پيام نبوده و پيام حذف خواهد شد. غالباً تمامي سرويس هاي ارتباط سطح انتقال فقط ارتباطات ناپايدار را ارائه مي دهند. در اين حالت، سيستم ارتباطي شامل مسيرياب هاي ذخيره – و- ارسال متداول است. چنانچه يك مسيرياب نتواند پيام را به مسيرياب ديگر يا به ميزبان مقصد تحويل دهد، پيام را حذف خواهد كرد.

روش ارتباط علاوه بر دو حالت پايدار و ناپايدار فوق ، ممكن است ناهمگام[502] و يا همگام[503] هم باشد. ويژگي بارز **ارتباط ناهمگام**[504] آن است كه فرستنده، آناً پس از تسليم پيام خود، به كار ادامه مي دهد. بدين معنا كه پيام به محض تسليم فوراً(و به طور موقت) توسط ميان افزار ذخيره مي شود. در **ارتباط همگام**[505] ، مادامي كه

[500] persistent communication
[501] transient communication
[502] asynchronous
[503] synchronous
[504] asynchronous communication
[505] synchronous communication

درخواست فرستنده پذیرفته نشود، فرستنده مسدود باقي خواهد ماند. به طور مشخص سه نقطه برای همگام سازی وجود دارد. در حالت اول فرستنده تا زمانی مسدود می ماند که میان افزار اطلاع دهد که انتقال این درخواست را انجام مي دهد. درحالت دوم، فرستنده ممکن است تا زمان تحویل درخواست خود به دریافت کننده مورد نظر، کار همگام سازي را ادامه مي دهد. در حالت سوم، همگام سازي با انتظار فرستنده براي پردازش کامل درخواست آن يعني تازمانيکه دريافت کننده پاسخي را بازگرداند، انجام مي شود.
در عمل، انواع روش ها از دیدگاه پایداری و همگامی ممکن است با هم ترکیب شوند که از جمله معروفترین آنها مي توان به ترکیب حالت پایدار و همگام جهت تسلیم درخواست اشاره کرد. این طرح در بسیاري از سیستم هاي متعارف صف بندي پیام استفاده مي شود و از این رو در بخش هاي بعد مورد بحث قرار خواهد گرفت. به همین ترتیب، ترکیب ارتباطات ناپایدار و همگام سازي پس از پردازش کامل درخواست بطور گسترده استفاده خواهد شد. این طرح با فراخواني روال از راه دور همخواني دارد که ذیلاً به ان اشاره خواهیم کرد.
علاوه بر پایداری و همگامی، لازم است از تفاوت ارتباطات گسسته و جریانی هم صحبت کنیم . تا به اینجا همه مثال ها در دسته ارتباطات گسسته قرار داشتند. بطوریکه طرفین با پیام با یکدیگر ارتباط برقرار کرده و هر پیام یك واحد کامل اطلاعاتي را در بر داشت. در حالیکه در جریان، پیام هاي متعدد یکي پس از دیگري ارسال شده و بر اساس ترتیب ارسال یا رابطه اي زماني به یکدیگر مرتبط مي شوند. در قسمت بعد مفصلاً راجع به این موضوع بحث خواهیم کرد.

۲-۴- فراخواني روال از راه دور (RPC)

بسیاري از سیستم هاي توزیعي بر اساس تبادل آشکار پیام بین فرآیندها استوار هستند. با این وجود، رویه هاي *send* و *receive* نمی توانند ارتباط را به هیچ وجهی پنهان سازند و این ویژگي تأثیر مهمی در به دست آوردن شفافیت دسترسي در سیستم هاي توزیعي دارد. این اشکال مدتهاست که شناسایي شده، اما تلاش چنداني براي رفع آن صورت نگرفته بود تا اینکه Birrell and Nelson با ارائه مقاله اي در سال ۱۹۸۴ روش کاملاً متفاوتي را براي مسأله ارتباط ارائه دادند. هر چند ایده آنها (در ظاهر) بسیار ساده مي نماید ، اثر جانبي ظریفي دارد. در این بخش، مفهوم ، اجرا، مزایا و معایب آنرا بررسي خواهیم کرد.
به طور خلاصه،Birrell and Nelson پیشنهاد کردند که برنامه ها بتوانند رویه هاي ماشین هاي دیگر را فراخواني نمایند. هنگامي که فرآیند ماشین *A* اقدام به فراخواني روال ماشین *B* مي کند، فرآیند فراخواننده در *A* معلق شده و اجراي رویه فراخواني شده روي ماشین *B* انجام مي شود. اطلاعات را مي توان به صورت پارامتري از فراخواننده به فراخوان شده انتقال داد و به شکل نتیجه رویه بازگرداند. بعلاوه، برنامه نویس به هیچ وجه قادر به

مشاهده ردکردن پیام ها نخواهد بود. این روش به نام **فراخواني روال از راه دور** یا **RPC** خوانده مي شود.
هرچند ایده اساسي بسیار ساده و بي نقص به نظر مي رسد، مشکلات ظریفي در این میان وجود دارد. به عنوان پیش زمینه به خاطر داشته باشید از آنجایي که روال هاي فراخواني و فراخوان شده روي ماشین هاي مختلف اجرا مي شوند، فضاهاي آدرسی مختلفی دارند که باعث پیچیدگی می شود. پارامترها و نتایج نیز بایستی ردوبدل شوند که اینهم، درصورت تفاوت ماشین ها ، باعث افزایش پیچیدگی خواهد شد . نهایتاً هر یک از دو ماشین می توانند دچار شکست شوند و حالت های مختلف شکست مسائل مختلفی را ایجاد می کنند.
اما غالب این مسائل قابل حل بوده و RPC شیوه پرکاربردي است که شالوده بسیاري از سیستمهاي توزیعي را تشکیل مي دهد.

۱-۲-۴- عملیات اصلي RPC

بحث خود را با فراخواني روال مرسوم آغاز کرده و سپس توضیح مي دهیم که چگونه مي توان خود فراخواني را به صورت اجزاء مشتري و سرور تجزیه نمود که هر یك روي ماشین هاي مختلف قابل اجراست.

فراخواني روال مرسوم

براي درك نحوه عملکرد RPC ،درك کامل عملکرد فراخواني روال مرسوم (یعني فقط یك ماشین) اهمیت خاصي دارد. یک فراخواني در C را به این صورت فرض کنید:

count =read (fd ، buf، nbytes)

که در آن *fd* عدد صحیح بیانگر فایل، *buf* رشته اي از کاراکترهایي است که داده ها درآن خوانده مي شود و *nbytes* عدد صحیح دیگري است که بیانگر تعداد بایتهاي خواندني است. چنانچه از برنامه اصلي فراخواني شود، پشته[۵۰۶] پیش از فراخوانی به صورت شکل (الف) ۵-۴ خواهد بود. بر اساس شکل (ب) ۵-۴ ، فراخواننده جهت فراخواني، پارامترها را بر روی پشته به صورت آخرین ورود / اولین خروج[۵۰۷] قرار خواهد داد. (دلیل آنکه کمپایلرهاي C پارامترها را به ترتیب عکس وارد مي کنند مربوط به *printf* است. چون به این ترتیب، *printf* قادر خواهد بود تا اولین پارامتر خود یعني زنجیره قالب[۵۰۸] را مکان یابي نماید). پس از اتمام اجراي روال *read*، مقدار بازگشت در یك ثبات[۵۰۹] قرار داده شده ، آدرس بازگشت حذف مي شود و کنترل را به فراخواننده برمي گرداند. سپس ، فراخواننده پارامترها را از پشته جدا کرده و پشته را به وضعیت اصلي پیش از فراخواني بازمي گرداند.

شکل ۵-۴- الف) رد کردن پارامتر فراخواني روال محلي: پشته پیش از فراخواني read. ب) پشته در حین فعال بودن روال فراخوان شده (داخل شکل: الف) متغیرهاي محلي برنامه اصلي، نشانگر پشته[۵۱۰]، ب) متغیرهاي محلي برنامه اصلي، آدرس بازگشت، متغیرهاي محلي read)

[506] stack
[507] Last in / First out (LIFO)
[508] format string
[509] registerer
[510] stack pointer

چند نکته را باید خاطرنشان نمود. در درجه اول، در C پارامترها ممکن است **فراخواني توسط مقدار**[511] يا **فراخواني توسط مرجع**[512] باشند. همانطور که در شکل (ب) ۵-۴ مشاهده مي کنيد، پارامتر مقداري از قبيل *fd* يا *nbytes* به پشته کپي مي شوند. از نظر روال فراخواني شده، پارامتر مقدار صرفاً يك متغير محلي مقداردادە شده است. روال فراخواني شده مي تواند آنرا تغييردهد، اما چنين تغييراتي هيچ تأثيري بر مقدار اوليه در طرف فراخواننده نمي گذارد.

هر پارامتر مرجع در C ، نشانگري به يك متغير(يعني آدرس متغير) است، نه به مقدار آن. در حالت فراخواني *read*، از آنجاييکه آرايه ها[513] همواره بوسيله مرجع در C رد کرده مي شوند، پارامتر دوم يك پارامتر مرجع خواهد بود. آنچه واقعاً به پشته وارد مي شود، آدرس آرايه کاراکتری است. چنانچه روال فراخوانده شده از اين پارامتر جهت ذخيره سازي در آرايه کاراکتری استفاده کند ، قطعاً آرايه را در روال فراخواني شده اصلاح خواهد کرد. همچنانکه بعداً خواهيم ديد، تفاوت بين فراخواني توسط مقدار و فراخواني توسط مرجع اهميت خاصي در RPC دارد.

مکانيزم ردکردن پارامتر ديگري هم به نام **فراخواني توسط کپي/بازيابي**[514] وجود دارد که در C فاقد کاربرد است. اين مکانيزم شامل کپي کردن متغير به پشته توسط فراخواننده است (که از اين نظر مشابه فراخواني توسط مقدار مي باشد). متغير پس از خاتمه فراخواني کپي و بازگردانده مي شود و روي مقدار اوليه فراخواننده رونويسي مي شود. در اغلب موارد، تأثير ايجاد شده دقيقاً مشابه تأثير فراخواني توسط مرجع است. در برخي موارد، از جمله زمانيکه پارامتر واحد چندين مرتبه در ليست پارامترها قرار داده مي شود، مفهوم ديگري براي خواندن آن مشاهده مي شود. اما در بسياري از زبانها از مکانيزم فراخواني توسط کپي/ بازيابي استفاده نمي شود.

تصميم گيري راجع به روش مناسب ردکردن پارامتر غالباً بر عهده طراحان زبان بوده و ويژگي ثابت زبان محسوب مي شود. اين امر گاهي اوقات وابسته به نوع داده رد کرده مي باشد. بعنوان مثال، همچنانکه بعداً خواهيم ديد، در C اعداد صحيح و ديگر انواع غيربرداري[515] همواره توسط مقدار رد مي شوند ، در حاليکه آرايه ها ،توسط مرجع. برخي از کمپايلرهاي Ada براي پارامترهاي **in out**، از کپي/ بازيابي استفاده مي کنند، اما در برخي ديگر از فراخواني توسط مرجع استفاده مي شود. نحوه تعريف زبان امکان انتخاب يکي از اين دو حالت را فراهم مي کند که البته معني را هم تا حدودي پيچيده و مبهم مي کند.

مشتري مجازی[516] و سرور مجازي[517]

[511] call by value
[512] call by reference
[513] array
[514] copy/restore
[515] scalar
[516] client stub

ايده زيربنايي RPC اين است كه فراخواني روال از راه دور حتي الامكان به صورت فراخواني روال محلي به نظر برسد. به بيان ديگر، RPC بايد شفاف باشد و روال فراخواني نبايد از اجراي روال فراخواننده شده روي ماشين ديگر اطلاع داشته باشد و برعكس. فرض كنيد كه بخواهيم برنامه اي بعضي از داده ها از يك فايل را بخواند. برنامه نويس براي كسب داده ، فراخوان ***read*** را در كد خود وارد مي كند. در سيستم هاي متعارف (تك پردازشگري) ، روال ***read*** توسط پيوند دهنده[٥١٨] از كتابخانه استخراج و داخل برنامه مورد نظر درج مي شود. اين روال كوتاه غالباً از طريق فراخواني يك فراخوان سيستمي معادل براي ***read*** پياده سازي مي شود. به بيان ديگر، روال ***read*** نوعي رابط بين كد كاربر و سيستم عامل محلي است.

مطابق شكل (ج) ٥-۴، اگرچه ***read*** اقدام به فراخوان سيستمي مي نمايد، ولي به همان صورت عادي يعني با قراردادن پارامترها روي پشته فراخوانده خواهد شد. به اين ترتيب، برنامه نويس كوچكترين اطلاعي از اين عمل زيركانه ***read*** پيدا نخواهد كرد.

RPC شفافيت خود را به روشي مشابه كسب مي كند. در صورتيكه ***read*** واقعاً روال از راه دور باشد، (مثلاً روالي كه روي ماشين سرور اجرا خواهد شد)، نسخه متفاوتي از ***read*** به نام **مشتري مجازي** در كتابخانه قرار داده خواهد شد. اين نسخه هم مشابه ***read*** اصلي با استفاده از زنجيره فراخواني شكل (ج) ٥-۴ فراخوانده مي شود . همچنين مشابه ***read*** اصلي سيستم عامل محلي را فراخواني مي كند. تنها تفاوت اين است كه برخلاف اصل خود از سيستم عامل تقاضاي گرفتن داده را نخواهد كرد، بلكه در عوض پارامترها را به صورت پيام بسته بندي كرده و درخواست مي كند كه پيام مطابق شكل ۶-۴ به سرور ارسال شود. پس از فراخواني ***send***، مشتري مجازي ***receive*** را فراخواني كرده و تا زمان رسيدن پاسخ، خود را بلوكه مي كند.

شكل ۶-۴- اصل كلي RPC بين برنامه مشتري و سرور(داخل شكل:مشتري، انتظار دريافت نتيجه، فراخواني روال از راه دور، بازگشت از فراخواني، درخواست، پاسخ، سرور، فراخوانی روال محلي و بازگشت نتايج، زمان).

با رسيدن پيام به سرور، سيستم عامل سرور آنرا به سرور مجازي انتقال مي دهد. سرور مجازي در طرف سرور معادل يك **مشتري مجازي** است. به بيان ديگر ، قطعه اي از يك كد است كه درخواستهاي ورودي از سرتاسر شبكه را به فراخوان هاي روال محلي تبديل مي كند. معمولاً سرور مجازي ***receive*** را فراخواني نموده و تا رسيدن پيامهاي ورودي بلوكه باقي مي ماند. سرور مجازي پارامترهاي پيام را باز كرده ، سپس مطابق معمول (شكل ٥-۴) روال سرور را فراخواني مي نمايد. از نقطه نظر سرور، اين وضعيت مشابه فراخواني مستقيم سرور بوسيله مشتري است – پارامترها و آدرس بازگشت همگي روي پشته اي قرار دارند كه به آن متعلق بوده و همه چيز حالت عادي خواهد داشت. سرور كار خود را انجام داده و سپس نتيجه را به صورت معمول به فراخوان بازميگرداند. بعنوان

---

[517] server stub
[518] linker

مثال، در مورد *read* ، سرور بافری[519] را که به توسط پارامتر دوم به آن اشاره شده است را پر از داده می نماید، این بافر داخلي سرور مجازي خوهد بود.
پس از تکمیل فراخواني، هنگامیکه سرور مجازي کنترل را مجدداً در دست مي گیرد، نتیجه (بافر) را در قالب یك پیام بسته بندي کرده و *send* را فراخواني مي کند تا آنرا به مشتري بازگرداند. معمولاً پس از آن ، سرور مجازي مجدداً *receive* را فراخواني مي کند تا منتظر درخواست ورودي بعد بماند.
پس از آنکه پیام به ماشین مشتري بازگردانده مي شود، سیستم عامل مشتري مشاهده مي کند که به فرآیند مشتري (یا مشتري مجازي) آدرس دهي شده است (توجه داشته باشید که از نظر سیستم عامل ایندو تفاوتي باهم ندارند). پیام به بافر درحال انتظار کپي شده و فرآیند مشتري آزاد مي شود. مشتري مجازي پیام را بازبیني کرده، نتیجه را باز کرده، آنرا به فراخواننده اش کپي و به همان صورت معمول بازمي گرداند . پس از فراخواني *read* ، هنگامي که فراخواننده کنترل را مجدداً در اختیار مي گیرد ، فقط از وجود داده ها اطلاع دارد و کوچکترین اطلاعي ندارد که در عوض سیستم عامل محلی، اینکار از راه دور انجام شده است.
اما این ناآگاهي مشتري، در واقع راز ظرافت و جذابیت کلي طرح است . تا به اینجا ، سرویس هاي از راه دور از طریق ایجاد فراخواني هاي روال عادي(یعني محلي) قابل دسترس بودند، نه از طریق فراخواني *send* و *receive*.. همان گونه که جزئیات انجام فراخوان سیستمی در کتابخانه های متعارف مخفی هستند، تمامي جزئیات مربوط به ردکردن پیام در دو روال کتابخانه قرار داده مي شوند .
بطور خلاصه، فراخواني روال از راه دور طي مراحل زیر انجام مي شود:

۱. روال مشتري، مشتري مجازي را به طور معمول فرا مي خواند.
۲. مشتري مجازي پس از ساخت پیام، سیستم عامل محلي را فرامي خواند.
۳. OS(سیستم عامل) مشتري پیام را به OS ماشین دور ارسال مي کند.
۴. OS ماشین دور پیام را به سرور مجازي مي دهد.
۵. سرور مجازي پارامتر ها را باز کرده و سرور را فراخواني مي کند.
۶. سرور کار خود را انجام داده نتیجه را به سرور مجازي بازمي گرداند.
۷. سرور مجازي نتیجه را به شکل پیام بسته بندي نموده و OS محلي خود را فرا مي خواند.
۸. OS سرور پیام را به OS مشتري ارسال مي کند.
۹. OS مشتري پیام را به مشتري مجازي مي دهد.
۱۰. مشتري مجازي نتیجه را باز و آنرا به مشتري بازمي گرداند.

[519] buffer

نتيجه خالص مراحل فوق ،تبديل فراخواني محلي توسط روال مشتري به مشتري مجازي، به فراخواني محلي به روال سرور است، بدون آنكه مشتري يا سرور حتي از مراحل مياني يا وجود شبكه كوچكترين اطلاعي داشته باشند.

۲-۲-۴ – رد كردن پارامتر[۵۲۰]

وظيفه مشتري مجازي، گرفتن پارامترهاي مربوط به خود ، بسته بندي آنها به شكل پيام و ارسال به سرور مجازي است. اينكار برخلاف ظاهر سهل و ساده خود داراي پيچيدگي هاي خاصي است. در اين بخش، نگاهي به برخي از مسائل مرتبط با ردكردنپارامتر در سيستم هاي RPC خواهيم داشت.

رد كردن پارامترهاي مقدار

بسته بندي پارامترها به شكل پيام، **مرتب سازي پارامتر**[۵۲۱] ناميده مي شود. براي نمونه يك روال از راه دور ساده مثل اضافه كردن (j ، i) را در نظر بگيريد. در اين حالت، دو پارامتر عدد صحيح *j* و *i* گرفته شده و حاصلجمع رياضي آنها بصورت نتيجه بازگردانده مي شود (البته در عمل هيچكس چنين روال ساده از راه دوري را انجام نمي دهد، ولی برای مثال مناسب است). فراخواني اضافه كردن در سمت چپ شكل ۷-۴ (در بخش فرآيند مشتري) نمايش داده شده است. مشتري مجازي ايندو پارامتر را گرفته و آنها را به ترتيب گفته شده به پيام تبديل مي كند. سپس، نام يا شماره روالي را كه قرار است فراخواني شود، در پيام قرار مي دهد. از آنجاييكه امكان دارد سرور چندين فراخوان را پشتيبانی نمايد، فراخوان مورد نظر بايد مشخص شود.

شكل ۷-۴- مراحل مربوط به انجام محاسبه از راه دور در RPC (داخل شكل: ماشين مشتري، فرآيند مشتري، OS مشتری، فراخواني روال توسط مشتري، مشتري مجازي، سرور مجازي، مشتري مجازي پيام را ايجاد مي كند، ماشين سرور، فرآيند سرور، سرور مجازي فراخواني محلي add را انجام مي دهد، سرور مجازي پيام را باز مي كند، OS سرور پيام را به سرور مجازي تحويل مي دهد).

وقتي پيام به سرور مي رسد، سرور مجازي پس از بررسي ، روال لازم را شناسايي و فراخواني مناسب را انجام مي دهد . چنانچه سرور روال هاي از راه دور ديگري را هم پشتيباني كند، ممكن است سرور مجازي –برحسب فيلد اول پيام- از جمله سوئيچ برای انتخاب روال فراخوانده شده استفاده كند. فراخواني واقعي از سرور مجازي به سرور مشابه فراخواني مشتري اصلي است، تنها تفاوت در اينجاست كه پارامترها، متغيرهايي هستند كه از پيام وارده آغاز مي شوند. پس از اتمام كار سرور، كنترل مجدداً در اختيار سرور مجازي قرار مي گيرد. همچنين نتيجه عودت داده شده توسط سرور را اخذ و آنرا به شكل پيام دسته بندي مي كند. اين پيام مجدداً به ماشين مشتري ارسال و توسط آن باز مي شود. مشتري مجازي نتيجه را استخراج و مقدار را به روال مشتري در حال انتظار باز مي گرداند.

مادامي كه ماشين هاي مشتري و سرور يكسان بوده و تمامي پارامترها و نتايج از انواع اسكالر از قبيل اعداد صحيح،

[520] parameter passing
[521] parameter marshaling

کاراکترها و منطقی باشند، این مدل به خوبي عمل خواهد کرد. اما در يك سيستم بزرگ توزيعي، معمولاً چندين نوع ماشين وجود دارد و هر ماشين غالباً از روش خاصي براي نمايش اعداد، کاراکترها و ديگر اقلام داده اي استفاده مي کند. بعنوان مثال، در کامپيوترهاي بزرگ IBM از کد EBCDIC برای حروف و در کامپيوترهاي شخصي IBM از کد ASCII استفاده مي شود. در نتيجه، مطابق طرح ساده شده شکل ۷-۴ امکان انتقال يك پارامتر کاراکتري از مشتري کامپيوتر شخصي IBM به سرور کامپيوتر بزرگ IBM وجود ندارد. حتي در صورت انتقال، کاراکتر توسط سرور به صورت نادرستي ترجمه خواهد شد.

نمايش اعداد صحيح (متمم عدد يك در مقايسه با متمم عدد دو) و اعداد مميز شناور هم ممکن است با مشکلاتي مشابه همين انجام شود. مشکل بزرگتر اين است که برخي ماشينها از جمله Intel Pentium ، بايتها را از راست به چپ شماره گذاري مي کنند ، در حاليکه برخي ديگر از قبيل Sun SPARK از جهت ديگر براي شماره گذاری استفاده مي کنند. با الهام از داستان سفرهاي گاليور و درگرفتن جنگ بين سياستمداران بر سر شکستن تخم مرغ در اين کتاب، فرمت Intel **سرکوچك**[۵۲۲] و فرمت SPARK **سربزرگ**[۵۲۳] ناميده مي شود(Cohen،۱۹۸۱). بعنوان مثال، يك روال را با دو پارامتر يکي به صورت عدد صحيح و ديگري يك زنجيره چهار حرفي در نظر بگيريد. هر پارامتر نيازمند يك کلمه ۳۲ بيتي است. شکل (الف) ۸-۴ بخش پارامتري پيام ساخته شده توسط مشتري مجازي درIntel Pentium را نمايش مي دهد. کلمه اول حاوي پارامتر عدد صحيح ، يعني ۵، و کلمه دوم حاوي رشته "JILL " است.

شکل ۸-۴- الف) پيام اصلي درPentium .ب) پيام پس از دريافت در SPARK .ج) پيام پس از تبديل.اعداد ريز داخل مربعها بيانگر آدرس هر بايت است.

از آنجاييکه پيام ها بايت به بايت (در عمل، بيت به بيت) در سرتاسر شبکه انتقال داده مي شود، اولين بايت ارسالي اولين بايت رسيده خواهد بود. در شکل (ب) ۸-۴ نمايش داده ايم که پيام شکل (الف) ۸-۴ در صورت دريافت بوسيله SPARK چگونه بنظر خواهد رسيد. برخلاف تمامي تراشه هاي Intel که بايت از راست شماره گذاري مي شود(بايت مرتبه پايين) ،SPARK اينکار را از بايت صفر در سمت چپ(بايت مرتبه بالا) شروع مي کند. اگر سرور مجازي پارامترها را به ترتيب در آدرسهاي ۰و۴ بخواند، عدد صحيحي برابر با (۵×۲^{24}) ۸۳۸۸۶۰۸۰ و زنجيره "JILL " بدست خواهد آمد. يك روش آشکار اما متأسفانه نادرست، معکوس کردن ساده ترتيب بايت هاي هر کلمه پس از دريافت است که منجر به توليد شکل (ج) ۸-۴ خواهد شد. براي اين حالت، عدد صحيح ۵ و رشته "LLIJ " را در نظر بگيريد . مشکل اين است که اعداد صحيح از طريق مرتب سازي متفاوتی نسبت به رشته ها معکوس شده اند، اما رشته ها بدون تغيير باقي مي مانند. بدون در اختيار داشتن اطلاعات کافي

[522] little endian
[523] big endian

درمورد ماهيت رشته و عدد صحيح، جبران اشتباهات غيرممكن خواهد بود.

ردكردن پارامترهاي مرجع

سؤال اين است كه نشانگرها[524] ،يا به طور كلي ، مرجع ها[525] چگونه رد مي شوند؟ در پاسخ بايد گفت كه اينكار بسيار مشكل يا در بعضي موارد بطور كلي غيرممكن است. به خاطر داشته باشيد كه يك نشانگر فقط در داخل فضاي آدرس فرآيندي معنا دارد كه در آن استفاده شده است. مجدداً مثال خواندن را بياد بياوريد كه در آن چنانچه پارامتر دوم (آدرس بافر) در مشتري ۱۰۰۰ باشد، نمي توان فقط عدد ۱۰۰۰ را به سرور انتقال داد و انتظار نتيجه خوبي را هم داشت. ممكن است آدرس ۱۰۰۰ در سرور در وسط متن برنامه قرار داشته باشد. يك راه حل ساده آن است كه تمام نشانگرها و پارامترهاي مرجع را ممنوع كرد. اما اهميت ايندو المان به حدي است كه ممنوع سازي آنها هم واقعاً بسيار غيرمنطقي و هم غيرلازم است. در مثال خواندن (*read*)، مشتري مجازي مي داند كه پارامتر دوم اشاره به رشته اي از كاراكترها دارد. تصور كنيد كه بزرگي و ابعاد رشته هم مشخص باشد. بنابراين راهكار احتمالي، كپي كردن رشته در پيام و ارسال آن به سرور است. در اينصورت سرور مجازي مي تواند با يک نشانگر به رشته ، سرور را فراخواني نمايد. اين امر حتي در صورتي هم كه نشانگر داراي مقدار عددي متفاوت از پارامتر دوم *read* باشد، امكان پذير است. تغييراتي كه سرور با استفاده از نشانگر ايجاد مي كند (مانند ذخيره داده ها در داخل آن) تأثير مستقيمي بر بافر پيام داخل سرور مجازي مي گذارد. وقتي كار سرور به پايان رسيد، مي توان پيام اصلي را به مشتري مجازي بازگردانده و سپس آنرا مجدداً به مشتري كپي نمود. در نتيجه، روش كپي/ بازيابي جايگزين فراخواني توسط مرجع مي شود. هرچند كاملاً متناظر نيست، ولی در بسياري از موارد عملكرد قابل قبولي دارد.

اثربخشي اين مكانيزم را مي توان از طريق بهينه سازي مي توان دوچندان كرد. چنانچه مجازي ها بدانند كه بافر پارامتر ورودي به سرور است يا خروجي از آن، مي توان يكي از كپي ها را حذف نمود. چنانچه آرايه ورودي به سرور باشد(بعنوان مثال در فراخواني نوشتن) ديگر نيازي به كپي برای برگشت آن نيست. اما چنانچه خروجي باشد، لازم نيست كه در مرحله اول فرستاده شود.

در پايان لازم به ذكر است كه هرچند اكنون قادر به اداره اشاره گرها به آرايه ها و ساختارهاي ساده هستيم، هنوز راه حلي براي حالت های عام يعني اداره اشاره گر به ساختارهاي داده اي دلخواهي از قبيل يك گراف پيچيده ارائه نشده است. برخي از سيستم ها تلاش مي كنند تا با ردكردن واقعي اشاره گر به سرور مجازي و توليد كد ويژه در روال سرور برای استفاده از نشانگر بر اين مشكل فائق آيند. بعنوان مثال، مي توان براي تأمين داده ارجاع داده شده، درخواست را به مشتري بازگرداند.

توصيف پارامتر و توليد مجازي ها

[524] pointer
[525] reference

از آنچه تاكنون توضيح داده شد به راحتي مي توان دريافت كه مخفي سازي فراخواني روال از راه دور مستلزم توافق فراخواننده و فراخوان شده در مورد قالب پيام تبادلي است. بعلاوه بايستي در مواردي از قبيل ردكردن ساختارهاي پيچيده داده اي از مراحل يكسانی تبعيت كنند. به بيان ديگر، طرفين RPC بايستي از يك پروتكل واحد تبعيت كرده و درغيراينصورت RPC عملكرد صحيحي نخواهد داشت.

براي نمونه ، روال شكل (الف) ۹-۴ را در نظر بگيريد. اين روال داراي سه پارامتر ، يك كاراكتر، يك عدد صحيح مميز شناور و آرايه اي متشكل از پنج عدد صحيح است. با اين فرض كه يك كلمه از چهار بايت تشكيل مي شود، در پروتكلRPC تصريح شده كه كاراكتر را در اولين بايت سمت راست كلمه(سه بايت بعدي را خالي بگذارد) ، عدد مميز شناور را به صورت يك كلمه كامل و آرايه را بعنوان گروهي از كلمات با طول برابر با رشته انتقال دهيم و پيش از اينها هم كلمه اي به عنوان بيانگر طول ارسال شود (شكل (ب) ۹-۴) . براساس اين قوانين، مشتري مجازي foobar مي داند كه بايد از قالب شكل (ب)۹-۴ استفاده كرد و سرور مجازي مي داند كه پيامهاي ورودي از foobar بايستي در قالب شكل (ب) ۹-۴ صدق كند.

شكل ۹-۴- الف) يك روال ب) پيام متناظر روال(داخل شكل: متغيرهاي محلي foobar)

هرچند تعريف فرمت پيام يكي از ابعاد لازم پروتكل RPC است، اما كافي نمي باشد. بعلاوه لازم است كه مشتري و سرور در مورد ارائه ساختارهاي ساده داده اي از قبيل اعداد صحيح، كاراكترها، اعداد جبر بولي و غيره هم توافق داشته باشند. بعنوان مثال، ممكن است در پروتكل تصريح شود كه اعداد صحيح در متمم دو، كاراكترها در استاندارد يوني كد ۱۶ بيتي و اعداد مميز شناور در فرمت ۷۵۴# استاندارد IEEE نمايش داده شده و همه چيز در سركوچك ذخيره شود. با داشتن اين اطلاعات اضافي مي توان پيامها را به وضوح تفسير نمود.

پس از آنكه قوانين كدگذاري تا بيت آخر مشخص شدند، فقط لازم است كه فراخواننده و فراخوان شده در مورد تبادل واقعي پيامها به توافق برسند. بعنوان مثال ، ممكن است توافق شود كه از سرويس انتقال اتصال گرايي مثل TCP/IP استفاده شود. روش ديگر، استفاده از يك سرويس ديتاگرام نامطمئن است. سپس به مشتري و سرور اجازه داد تا بتوان بعنوان بخشي از پروتكل RPC ، يك طرح كنترل خطا اجرا كنند. در عمل از راهكارهاي متفاوتی استفاده مي شود.

پس از تعريف كامل و دقيق پروتكلRPC، بايستي مشتري و سرور اجرا شود. خوشبختانه مجازي هاي يك پروتكل واحد با روال هاي مختلف، غالباً فقط از نظر رابط با برنامه هاي كاربردي تفاوت دارند. يك رابط از مجموعه اي از روال ها تشكيل شده كه بوسيله مشتري فراخواني شده و توسط سرور اجرا می شود. رابط معمولاً به همان زبان برنامه نويسي قابل دسترسي است كه سرور و مشتري به آن نوشته مي شوند(هرچند لزومي به انجام آن نيست). به بيان ساده

تر، رابط ها غالباً بوسيله **زبان تعريف رابط**[526] **(IDL)** تعريف مي شوند. سپس، اين رابط به همراه رابط هاي مناسب زمان كمپايل يا زمان اجرا، به صورت مشتري مجازي و سرور مجازي كمپايل مي شوند. در عمل اثبات شده كه استفاده از زبان تعريف رابط به نحو چشمگيري باعث تسهيل برنامه كاربردي مشتري – سرور بر اساس RPC ها مي شود. از آنجايي كه ساخت كامل مشتري و سرور مجازي امري ساده است، تمامي سيستم هاي ميان افزاري براساس RPC جهت حمايت از توسعه برنامه هاي كاربردي، يك IDL ارائه مي دهند. همچنانكه در بخشهاي بعدي همين فصل مشاهده خواهيم كرد، در برخي از موارد استفاده از IDL الزامی است.

۳-۲-۴- RPC ناهمگام

مشابه فراخواني هاي روال معمول، هنگامي كه مشتري روال دوري را فرامي خواند، تا زمان بازگشت پاسخ مسدود باقي خواهد ماند. اين رفتار انعطاف ناپذير درخواست/پاسخ هنگامي غير ضروري مي نمايد كه هيچ نتيجه اي بازگشت داده نشده و فقط منجر به مسدود كردن مشتري شود؛ در حاليكه مي توانست پيشروي كرده و پس از درخواست فراخواني روال از راه دور كار مفيدي انجام دهد. از نمونه مواردي كه غالباً نيازي به انتظار براي دريافت پاسخ نيست مي توان به انتقال پول از يك حساب به حساب ديگر ، افزودن ليست به پايگاه داده اي ، راه اندازي خدمات ازراه دور، پردازش دسته و غيره اشاره كرد.

سيستم هاي RPC جهت پشتيباني از چنين مواردي امكاناتي به نام **RPC هاي ناهمگام**[527] ارائه مي دهند و بدين ترتيب، پس از صدور درخواست RPC ، مشتري فوراً فعال مي شود. در RPC هاي ناهمگام، سرور به محض دريافت درخواست RPC پاسخي را براي مشتري ارسال كرده و سپس روال درخواست شده را فرا مي خواند. اين پاسخ براي مشتري به عنوان تصديقي است كه سرور قصد پردازش RPC را دارد. مشتري به محض دريافت تصديق سرور، مسدود شدن را پايان داده و شروع به كار خواهد كرد. شكل (ب) ۴-۱۰ نحوه تقابل مشتري- سرور را در RPC هاي ناهمگام نمايش مي دهد. براي سهولت مقايسه، در شكل (الف) ۴-۱۰ رفتار عادي درخواست/پاسخ نمايش داده شده است.

RPC هاي ناهمگام علاوه بر اين درحالتي مي توانند مفيد باشند كه پاسخ بازگردانده خواهد شد،اما مشتري آمادگي براي انتظار كشيدن و غيرفعال ماندن در اين فاصله زماني ندارد. بعنوان مثال، مشتري ممكن است بخواهد آدرس هاي شبكه ای مجموعه اي از ميزبان هايي را كه انتظار مي رود در آينده تماس بگيرند، پيش واكشی كند. منطق حكم مي كند كه در چنين مواردي ارتباط بين مشتري وسرور از طريق دو RPC ناهمگام مطابق شكل ۴-۱۱ سازماندهي شود. ابتدا مشتري سرور را فراخواني مي كند تا ليستي از اسامي ميزبانان قابل جستجو را ارائه دهد. پس از تصديق دريافت ليست توسط سرور، مشتري به كار خود ادامه مي دهد. فراخواني دوم توسط سروري انجام مي شود كه مشتري را براي تحويل آدرسهاي يافت شده

[526] Interface Definition Language
[527] asynchronous RPCs

فراخواني مي كند. تركيب دو RPC ناهمگام در بعضي موارد به نام **RPC همگام معوق**[528] هم ناميده مي شود.
لازم به ذكر است كه در برخي از انواع RPC هاي ناهمگام ، مشتري فوراً پس از ارسال درخواست به سرور، همچنان به اجرا ادامه مي دهد. به بيان ديگر، مشتري ديگر منتظر تصديق پذيرش درخواست سرور باقي نمي ماند. ما چنين RPCهايي را **RPCهاي يكطرفه**[529] مي ناميم. مشكل اين روش آن است كه درصورتی كه قابليت اطمينان تضمين نشود، مشتري بطور قطع نمي داند كه آيا درخواست او پردازش خواهد شد يا نه. در فصل ۸ راجع به اين مسائل بحث خواهيم كرد. به همين ترتيب، در RPC هاي همگام معوق، مشتري مي تواند به جاي اين كه به سرور اجازه دهد نتيجه را بعداً به صورت فراخوان به مشتری ارسال كند، در مورد آماده بودن نتيجه از سرور پرس و جو كند.
شكل ۱۰-۴- الف) تقابل بين مشتري و سرور درRPC متداول.ب) تقابل با استفاده از RPC ناهمگام(داخل شكل: انتظار براي نتيجه،مشتري، روال فراخواني از راه دور، بازگشت از فراخواني،درخواست، پاسخ، سرور، زمان، روال فراخواني محلي و بازگشت نتايج،انتظار براي پذيرش،مشتري،روال فراخواني از راه دور،بازگشت از فراخواني، درخواست، پذيرش درخواست، سرور، زمان، روال فراخواني از راه دور).
شكل ۱۱-۴- تبادل مشتري و سرور از طريق دو RPC ناهمگام (داخل شكل: انتظار براي نتيجه، مشتري، روال فراخواني از راه دور، بازگشت از فراخواني، درخواست، پذيرش درخواست، سرور، زمان، روال فراخواني از راه دور، وقفه مشتري، بازگشت نتايج، اعلان وصول، زمان، فراخواني مشتري بوسيله RPC يكطرفه).

۲-۴-۲- مثال: DCE PRC

از فراخواني هاي از راه دور بطور گسترده بعنوان مبناي سيستم هاي توزيعي و ميان افزاري استفاده شده است. در اين بخش، نگاه دقيقتري به يكي از سيستم هاي ويژه RPC خواهيم داشت. اين سيستم كه **محيط محاسبه توزيعي (DCE)**[530] ناميده ميشود، توسط OSF[531] ابداع و ارائه شد، اما امروزه بيشتر از آن بعنوان گروه باز[532] ياد مي شود. آوازه DCE RPC به اندازه برخي ديگر از سيستم هاي RPC از جمله Sun RPC نيست. با اين وجود، DCE PRC می تواند نماينده ديگر سيستم هايPRC باشد و در سيستم مبناي مايكروسافت از ويژگي هاي آن جهت محاسبه توزيعي، DCOM ، استفاده شده است(۱۹۹۸، Eddon and Eddon). بحث خود را با تعريف كوتاهي از DCE شروع كرده و سپس وظايف اصليDCE RPC را شرح مي دهيم. براي مطالعه اطلاعات فني دقيقتر در مورد نحوه ساخت برنامه هاي كاربردي براساس RPC به كتاب (۱۹۹۹) Stevens مراجعه كنيد.

مقدمه اي بر DCE

[528] deferred synchronous RPC
[529] one-way RPC
[530] distributed computing environment
[531] Open Software Foundation
[532] open group

از آنجاييكه DCE جهت اجرا بعنوان لايه انتزاعي بين سيستم هاي عامل موجود (شبكه) و برنامه هاي كاربردي توزيعي طراحي شده، مي توان آنرا يك سيستم ميان افزاري واقعي تلقي نمود. هر چند اين سيستم براي UNIX طراحي شده بود، امروزه براي بسياري از سيستم عامل هاي مهم از جمله نسخه هاي ويندوز و VMS و همچنين سيستم هاي عامل كامپيوترهای شخصی وجود دارد. هدف آن است كه خريدار بتواند مجموعه اي از ماشين هاي موجود را گرفته ،نرم افزار DCE را به آن افزوده و سپس برنامه توزيعي را اجرا نمايد. تمامي اين مراحل بايستي بدون ايجاد كوچكترين مشكلي براي برنامه هاي كاربردي موجود (غيرتوزيعي) انجام شود. هرچند قسمت اعظم بسته DCE در فضاي كاربر اجرا مي شود، در برخي پيكربندي ها بايستي يك قطعه (بخشي از سيستم فايل توزيعي) را به هسته اضافه نمود. گروه باز فقط كد منبع را مي فروشد و فروشندگان اين كد را در سيستم هاي خود به كار مي گيرند.

مدل زيربنايي برنامه نويسی DCE ها، يك مدل مشتري- سرور است كه در بخش قبل بررسي شد. فرآيندهاي كاربر، جهت دسترسي به سرويس هاي ازراه دور ايجادي توسط فرآيندهاي مشتري، بعنوان مشتري ايفاي نقش مي كنند. برخي از اين سرويس ها بخشي از خودDCE هستند. اما برخي ديگر متعلق به برنامه هاي كاربردي بوده و توسط سازندگان برنامه هاي كاربردي نوشته مي شوند. تمامي ارتباطات بين مشتري ها و سرورها از طريق RPC ها انجام مي شود. برخي از سرويس ها جزء تشكيل دهنده DCE محسوب مي شوند. **سرويس فايل توزيعي**[٥٣٣] يك سيستم فايل جهاني است كه روش شفاف و واحدي را براي دسترسي به تمام فايل هاي درون سيستم ايجاد مي كند. اين سرويس يا در بر روی سيستم هاي فايل داخلي ميزبان ايجاد می شود و يا به جاي آنها مورد استفاده قرار مي گيرد. سرويس فهرست[٥٣٤] جهت رديابي محل تمامي منابع موجود در سيستم بكار برده مي شود. اين منابع شامل ماشين ها، چاپگرها، سرورها، داده ها و ساير مواردي است كه ممكن است هريك از نظر جغرافيايي در گوشه اي از جهان قرار داشته باشند. سرويس فهرست به فرآيند امكان مي دهد تا بدون توجه به محل استقرار منبع، آنرا جستجو نمايد مگر آنكه خود فرآيند تمايل به دانستن مكان منبع داشته باشد. **سرويس امنيتي**[٥٣٥] امكان محافظت از تمامي انواع منابع را فراهم مي آورد تا فقط افراد تأييد شده و مجاز امكان دسترسي داشته باشند. و نهايتاً **سرويس زمان توزيعي**[٥٣٦] موجب همگامي جهاني تمامي ساعتهاي ماشين هاي مختلف مي شود. همانطور كه در فصل بعد خواهيم گفت، اطلاع از زمان جهاني باعث هماهنگي سيستم هاي توزيعي خواهد شد.

اهداف DCE RPC

اهداف سيستم DCE PRC تا حدودي مانند مواد قبل است. اول و مهمتر از همه اينكه سيستم PRC به مشتري امكان مي دهد تا صرفاً با فراخواني يك روال محلي به سرويس از راه دور موردنظر دسترسي پيدا كند. اين رابط باعث مي شود تا برنامه هاي مشتري(

[533] distributed file service
[534] directory service
[535] distributed service
[536] distributed time service

يعني برنامه هاي كاربردي) به روشي ساده و آشنا براي اغلب برنامه نويسان نوشته شود. همچنين موجب تسهيل اجراي حجمهاي بالايي از كد موجود در يك محيط توزيعي بدون نياز به اعمال تغييرات زياد (يا حتي بدون هيچ تغييري) مي شود.

سيستم PRC وظيفه دارد تا تمامي جزئيات را از مشتريان و تا حدودي از سرورها هم مخفي نگه دارد. بدين منظور، سيستم PRC مي تواند بطور اتوماتيك سرور درست را مكان يابي نموده و سپس ارتباط بين نرم افزار مشتري و سرور را راه اندازي كند (اين ويژگي غالباً **متصل شدن**[537] ناميده مي شود). همچنين مي تواند كار تبادل پيام را در هردو طرف هدايت كرده و در صورت نياز (مانند مواردي كه پارامترها آرايه بزرگي را تشكيل مي دهد) آنها را تجزيه و مجدداً سرهم بندي مي كند. در نهايت، سيستمPRC مي تواند بطور اتوماتيك كار تبديلات نوع داده بين مشتري و سرور ، حتي در مواردیكه روي معماري هاي مختلف اجرا و ترتيب بايت مختلفي داشته باشد، را انجام دهد.

در اثر قابليت سيستم PRC در مخفي سازي جزئيات، ميزبان ها و مشتری ها به طور قابل توجهی مستقل از يكديگر هستند؛ بطوريكه مي توان مشتري را به يك زبان مثلJava و سرور را به زبان ديگري از قبيل C نوشت و بالعكس. سرور و مشتري مي توانند روي پايگاه های سخت افزاري مختلف اجرا و از سيستم هاي عامل مختلف استفاده نمايند. بعلاوه، بدون هيچگونه دخالتي از جانب مشتري يا سرور، از انواع پروتكل هاي شبكه و نمايش هاي داده پشتيباني نمود.

نوشتن مشتري و سرور

سيستم DCE RPC از اجزاي مختلفي از قبيل زبان، كتابخانه، برنامه های محافظ[538] ، برنامه هاي كمكي و بسياري ديگر تشكيل شده است. اين اجزاء به كمك هم امكان نوشتن مشتري و سرور را فراهم مي كنند. در اين بخش، هر يك از اجزاء و نحوه ارتباط آنها با يكديگر را تشريح خواهيم كرد. كل فرآيند نوشتن و استفاده از مشتري و سرور PRC در شكل ۱۲-۴ خلاصه شده است.

شكل ۱۲-۴- مراحل نوشتن مشتري و سرور در DCE RPC (داخل شكل: رابط تعريف فايل، كمپايلر، كد مشتري، مشتري مجازي، سرپيام، سرور مجازي، كد سرور، كمپايلر C،كمپايلر C، كمپايلر C، فايل object مشتري، فايل object مشتري مجازي، فايل object سرور مجازي، فايل object سرور، پيوند دهنده، كتابخانه زمان اجرا، كتابخانه زمان اجرا، پيوند دهنده، باينری مشتري، باينری سرور).

در يك سيستم مشتري- سرور،عاملي به نام زبان تعريف رابط ياIDL باعث اتصال تمام اجزاء مي شود . اين چسب قوي امكان اعلان روال[539] را به صورتي بسيار مشابه نمونه هاي اوليه توابع[540] ANSI C فراهم مي آورد. بعلاوه، فايل هاي IDL ممكن است حاوي تعاريف نوع[541] ، اعلان هاي ثابت[542] وديگر اطلاعات لازم جهت مرتب سازي صحيح

[537] binding
[538] daemon
[539] procedure declaration
[540] function prototypes
[541] type definitions
[542] constant declaration

پارامترها و استخراج نتايج[543] باشند. در حالت ايده آل، تعريف رابط بايستي حاوي تعريفي رسمي از وظيفه روال ها هم باشد. اما چنين تعريفي در حال حاضر بالاتر از دانش و توان موجود بوده و به همين دليل، در تعريف رابط فقط نحوه اداي فراخواني ها ارائه ميشود نه معناي آنها. در بهترين حالت، نويسنده مي تواند با افزودن چند كامنت[544] عملكرد روال ها را تشريح نمايد.
يكي از المان هاي بسيار مهم هر فايلIDL ، وجود شناسه اي كاملاً منحصر به فرد براي رابط مورد نظر است. اين شناسه توسط مشتري در اولين پيام RPC ارسال شده و سرور صحت آنرا تأييد مي نمايد. بدين ترتيب، چنانچه مشتري سهواً به سرور اشتباه يا حتي نسخه قديمي تري از سرور درست وصل شود، سرور خطا را تشخيص داده و مانع از اتصال خواهدشد.
تعاريف رابط و شناسه هاي منحصر به فرد در DCE ارتباط نزديكي باهم دارند. همانطور كه در شكل ۱۲-۴ به نمايش درآمده، غالباً اولين قدم در نوشتن برنامه كاربردي مشتري/ سرور، فراخواني برنامه *uuidgen* و درخواست از آن براي توليد پيش نمونه فايلIDL حاوي شناسه رابط است. اين شناسه بايستي در بين تمامي رابط هاي قبلي توليدشده بوسيله *uuidgen* منحصر به فرد باشد. اين يگانگي با كد كردن محل و زمان ايجاد در داخل آن تضمين مي شود. كد مذكور شامل يك عدد دودويي ۱۲۸ بيتي است كه در فايلIDL به صورت رشته ASCII مبناى شانزده نمايش داده مي شود.
مرحله بعد، ويرايش فايل IDL و قراردادن اسامي روال هاي از راه دور و پارامترهاي آنها در فايل است. بايد بدانيم كه RPC كاملاً شفاف نيست- بعنوان مثال، مشتري وسرور نمي توانند متغيرهاي جهاني مشتركي داشته باشند – اما قوانين IDL مانع از بيان ساختارهاي فاقد پشتيباني مي شود.
پس از تكميل فايل IDL، كمپايلر IDL جهت پردازش فراخواني مي شود. در خروجي كمپايلر IDL سه فايل به چشم مي خورد:

۱. فايل سرپيام (بعنوان مثال، *interface.h* به زبان C)
۲. مشتري مجازي[545]
۳. سرور مجازي[546]

فايل سرپيام حاوي شناسه منحصر به فرد، تعاريف نوع، تعاريف ثابت ها و پيش نمونه هاي تابع است كه بايستي(به كمك *#include*) در هر دو كد سرور و مشتري قرار داده شوند. مشتري مجازي حاوي رويه هاي واقعي است كه توسط برنامه مشتري فراخواني خواهد شد. اين رويه ها مسئول جمع آوري و بسته بندي پارامترها به شكل پيام خروجي و سپس فراخواني سيستم زمان اجرا[547] جهت ارسال آن است. مشتري مجازي هم به بازكردن پاسخ و برگرداندن مقادير به مشتري مي پردازد. سرور مجازي حاوي روال هايي است كه پس از رسيدن پيام جديد، بوسيله سيستم زمان اجرا روي ماشين سرور

[543] unmarshal
[544] comment
[545] client stub
[546] server stub
[547] run-time system

فراخواني مي شود. اين روال ها متقابلاً روال هاي واقعي سرور را فراخواني مي كند كه مسئول انجام اينكار است.
در مرحله بعد، كد سرور و مشتري توسط نويسنده برنامه كاربردي نوشته مي شود. سپس ايندو به همراه دو روال مجازي كمپايل مي شوند. آنگاه، كد مشتري حاصله و فايلهاي object مشتري مجازي بوسيله كتابخانه زمان اجرا به هم متصل و باينرى قابل اجرايي را براي مشتري ايجاد مي كند. به همين ترتيب، كد سرور و سرور مجازي كمپايل و به هم متصل و باينري سرور را بوجود مي آورد. در زمان اجرا، سرور و مشتري راه اندازي مي شوند تا برنامه كاربردي هم واقعاً اجرا شود.

اتصال مشتري به سرور

براي آنكه مشتري بتواند سروري را فراخواني نمايد، سرور بايستي ثبت و آماده پذيرش فراخواني هاي ورودي باشد. ثبت سرور امكان رديابي و اتصال به آنرا توسط مشتري فراهم مي آورد. مكان يابي سرور در دو مرحله انجام مي شود:

۱. مكان يابي ماشين سرور
۲. مكان يابي سرور (يعني فرآيند صحيح) روي اين ماشين

مرحله دوم كمي تخصصي تر از مرحله اول است. به طور خلاصه، لازم است مشتري جهت ايجاد ارتباط با سرور، يك نقطه پاياني[۵۴۸] را در ماشين سرور شناسايي و پيام ها را به آن ارسال نمايد. در DCE، جدولي حاوي جفتهاي(سرور، نقطه پاياني) بوسيله فرآيندي به نام **محافظ DCE** [۵۴۹] در هر ماشين سرور نگهداري مي شود. سرور بايستي پيش از ورود درخواستهاي جديد، از سيستم عامل تقاضاي تعيين نقطه پاياني را نموده و سپس اين نقطه به كمك محافظ DCE در جدول نقطه پاياني ثبت و حفظ مي شود.
همچنين سرور با فراهم آوردن آدرس ماشين سرور و نامي كه بتوان سرور را تحت آن جستجو نمود، در سرويس دايركتوري ثبت مي شود.
سپس اتصال مشتري به سرور مشابه شكل ۱۳-۴ انجام مي شود.
فرض كنيد كه مشتري بخواهد به سرور ويدئويي ملحق شود كه به صورت محلي تحت نام */local/multimedia/video/movies* شناخته مي شود . مشتري اين نام را به سرور دايركتوري انتقال داده و آن هم آدرس شبكه ماشين اجراكننده سرور ويدئو را باز مي گرداند. سپس مشتري به سراغ فرآيند محافظ DCE ماشين (كه نقطه پاياني شناخته شده اي دارد) رفته و از آن تقاضا مي كند كه نقطه پاياني سرور ويدئو را در جدول نقطه پاياني خود جستجو نمايد و بازگرداند. با داشتن اين اطلاعات مي توان RPC را اجرا نمود. در RPCهاي بعدي ديگر نيازي به اين جستجو نيست. بعلاوه، DCE به مشتريان امكان مي دهد تا در صورت لزوم، جستجوهاي پيشرفته تري را براي يافتن سرور مناسب انجام دهند. همچنين در مواردي كه امنيت يا انسجام داده اي اهميت داشته باشد، از RPC اين استفاده ميشود.

شكل ۱۳-۱۴- اتصال مشتري به سرور در DCE(داخل شكل:ماشين دايركتوري، سرور دايركتوري، جستجوي سرور، ثبت سرويس، ماشين مشتري، مشتري، انجام

[548] end-point
[549] DCE daemon

RPC،تقاضاي نقطه پاياني، ماشين سرور، سرور، محافظDCE ، ثبت نقطه پاياني،جدول نقطه پاياني) .

اجراي RPC

RPC واقعي به صورت شفاف و به طور معمول انجام مي شود. مشتري مجازي به كمك پروتكل انتخابي در زمان اتصال، پارامترها را براى كتابخانه زمان اجرا مرتب و جهت انتقال آماده مي كند. هنگاميكه پيام به طرف سرور مي رسد، برحسب نقطه پاياني تعيين شده در پيام ورودي به سمت سرور صحيح مسيردهي مي شود .كتابخانه زمان اجرا پيام را به سرور مجازي ردكرده و آنهم پارامترها را مرتب نموده و سرور را فرا مي خواند. سپس پاسخ در جهت معكوس بازگردانده مي شود.

DCE امكانات معنايى مختلفى را ارائه مى كند. در حالت پيش فرض[550] يعني معناى **حداكثر يك مرتبه**[551]، هيچ فراخواني حتي در صورت خرابي سيستم هم بيش از يك مرتبه انجام نمي شود. اين ويژگي در عمل باعث مي شود تا چنانچه سرور در حين RPC خراب و سپس به سرعت ترميم شود، مشتري مجدداً عمليات را تكرار نخواهد كرد زيرا بيم آن مى رود كه قبلاً يک بار انجام شده باشد.

در روش ديگر مي توان روال مورد نظر را (در فايل IDL) تحت عنوان" **احتمال تكرار**"[552] نشانه گذاري كرد. در اينصورت مي توان آنرا بدون هيچ اشكالي چندين مرتبه تكرار نمود. بعنوان مثال، خواندن بلوک مشخصى از يك فايل را مي توان تا رسيدن به نتيجه مطلوب بارها تكرار كرد. در صورتيكه RPC احتمال تكرار به دليل خرابي سرور با شكست مواجه شود، مشتري بايستي تا زمان راه اندازي مجدد سرور صبر كرده و سپس مجدداً تلاش كند. معناهاى ديگري هم وجود دارند(اما به ندرت استفاده مي شوند) كه از آن جمله مي توان به انتشارRPC در تمام ماشينهاي روي شبكه محلي اشاره كرد. در فصل ۸ در مبحث مربوط به RPC به هنگام شكست[553] مجدداً در مورد اين معناها براي RPC صحبت خواهيم كرد.

۳-۴-ارتباطات پيام گرا

فراخواني هاي روال از راه دور[554] و احضار شيء از راه دور[555] در مخفي سازي ارتباطات در سيستم هاي توزيعي نقش دارند ؛ به اين معني كه باعث ارتقاء شفافيت دسترسي مي شوند. متأسفانه هيچ سيستمي براي همه موارد مناسب نيست . در برخي موارد كه نمي توان اطمينان داشت كه طرف دريافت كننده در زمان صدور درخواست در حال اجراست ، لزوم وجود سرويس هاي جايگزين بيش از پيش آشكار مي شود. به همين ترتيب، ماهيت ذاتاً همگام RPCها – كه باعث بلوكه شدن مشتري تا زمان پردازش درخواست او مي شود- لازم است گاهي با چيز ديگري جايگزين شود.

---

[550] default
[551] at-most-once
[552] idempotent
[553] failure
[554] remote procedure calls
[555] remote object invocations

اين چيز ديگر همان پيام دهي است. در اين بخش، روي ارتباطات پيام گرا در سيستم هاي توزيعي تمركز كرده و به همين منظور ابتدا نگاه دقيق تري به مفهوم رفتار همگام سازي و اثرات جانبي آن خواهيم داشت. سپس راجع به سيستم هاي پيام دهي بحث خواهيم كرد كه در آنها فرض بر اين است كه طرفين در زمان ارتباط در حال اجرا نيز هستند . در پايان، انواعي از سيستم هاي صف بندي پيام را مطالعه مي كنيم كه به فرآيندها امكان مي دهند تا حتي در صورتي هم كه طرف مقابل در زمان شروع ارتباط در حال اجرا نباشد، اطلاعات را تبادل نمايند.

۱-۳-۴-ارتباطات ناپايدار پيام گرا

بسياري از سيستمهاي توزيعي و برنامه هاي كاربردي مستقيماً بر روي مدل ساده پيام گراي ارائه شده توسط لايه انتقال ساخته شده اند. جهت درك بهتر مفهوم و اهميت سيستم هاي پيام گرا بعنوان بخشي از راه حلهاي ميان افزاري ، ابتدا راجع به پيام دهي از طريق سوكتهاي لايه انتقال بحث خواهيم كرد.

سوكتهاي بركلي

مسأله استانداردسازي رابط لايه انتقال بسيار مورد توجه قرار گرفته تا برنامه نويسان بتوانند به كمك مجموعه اي از عمليات اوليه[۵۵۶] ساده، از كل مجموعه پروتكلهاي پيام دهي استفاده نمايند. بعلاوه ،استفاده از رابطهاي استاندارد موجب تسهيل انتقال برنامه كاربردي مورد نظر به ماشين ديگر مي شود.
به عنوان نمونه، مختصراً راجع به **رابط سوكتها**[۵۵۷] به همان صورتي كه در سالهاي ۱۹۷۰ در UNIX بركلي ارائه شد، بحث مي كنيم. يكي ديگر از رابط هاي مهم ،**XTI** يا **رابط انتقالي X باز**[۵۵۸] است كه قبلاً به نام رابط لايه انتقال(TLI) خوانده و توسط AT&T ابداع شده است. سوكت ها و XTI از نظر مدل برنامه نويسي شبكه اي بسيار مشابه، ولي از نظر مجموعه عمليات اوليه متفاوت هستند.
از نظر مفهومي، **سوكت** در واقع نقطه پاياني ارتباط بوده و برنامه كاربردي مي تواند داده هايي را كه بايد به شبكه زيرين ارسال شود، در آن بنويسد و داده هاي ورودي را از آن بخواند. سوكت انتزاعي از ارتباط واقعي برروي نقطه پاياني است كه توسط سيستم عامل محلي براي يك پروتكل انتقال خاص مورد استفاده قرار مي گيرد . ذيلاً روي عمليات اوليه سوكتTCPتمركز مي كنيم كه در شكل ۱۴-۴ ارائه شده است.
سرورها در غالب موارد چهار عمل اوليه را ، به همان ترتيب داده شده، اجرا مي كنند. وقتي عمل اوليه *سوكت* راخوانده مي شود، فراخواننده به ازاي يك پروتكل انتقال خاص ، يك نقطه پاياني ارتباطي جديد هم ايجاد مي كند. از نظر داخلي، ايجاد چنين نقطه پاياني ارتباطي به اين معناست كه سيستم عامل محلي منابع را جهت فضادادن به پيامهاي ارسالي و دريافتي براي آن پروتكل خاص رزرو مي كند.
عمل اوليه *bind* يك آدرس محلي را به سوكت جديدالاحداث مربوط مي كند. بعنوان مثال، سرور بايستي آدرس IP ماشين خود را به همراه

[556] primitive
[557] sockets interface
[558] X/ open Transport Interface

يك شماره درگاه (احتمالاً مشهور) به سوكت وصل كند. وصل كردن براي سيستم عامل به اين
معناست كه سرور قصد دارد پيام ها را فقط روي آدرس و درگاه مورد نظر دريافت كند. عمل اوليه ***listen*** فقط در ارتباطات اتصال گرا فراخواني مي شود. اين فراخواني غيربلوكه است و به سيستم عامل محلي امكان مي دهد تا بافركافي را براي حداكثر تعداد مورد نظر اتصالاتي كه فراخواننده قصد پذيرش دارد، رزرو كند.

عمل اوليه	معني
سوكت	ايجاد يك نقطه پاياني ارتباط جديد
Bind	آدرس محلي جديد را به يك سوكت ضميمه كن
Listen	تمايل خود را براي پذيرش اتصالات جديد اعلام كن
Accept	تا زمان رسيدن درخواست اتصال جديد، فراخواننده را بلوكه كن
Connect	بطور به صورتي پويا براي ايجاد اتصال تلاش كن
Send	برخي داده ها را روي اتصال ارسال كن
Receive	برخي داده ها را روي اتصال دريافت كن
Close	اتصال را آزاد كن

شكل ۱۴-۴-عمليات اوليه سوكت براي TCP/IP

فراخواني عمل اوليه ***accept*** باعث بلوكه شدن فراخواننده تا زمان رسيدن درخواست اتصال مي شود. پس از رسيدن درخواست، سيستم عامل محلي سوكت جديدي با همان ويژگي هاي سوكت اصلي ايجاد كرده و آنرا به فراخواننده عودت مي دهد. اين روش به سرور امكان مي دهد تا بعنوان مثال، از طريق ايجاد اتصال جديد، فرآيندي را ايجاد كند كه بعداً ارتباط واقعي را هدايت خواهد كرد . در اين حين، سرور مي تواند به عقب بازگردد و منتظر درخواست اتصال ديگر روي سوكت اصلي باقي بماند.
حال بياييد نگاهي به طرف مشتري بيندازيم. در اينجا هم ابتدا بايد با استفاده از عمل اوليه *سوكت* ، يك سوكت ايجاد شود؛ اما از آنجاييكه سيستم عامل مي تواند پس از ايجاد اتصال، بصورتي پويا درگاه را تخصيص دهد، ديگر نيازي به اتصال آشكار سوكت به آدرس محلي نيست. عمل اوليه ***connect*** باعث تعيين آدرسي از لايه انتقال توسط فراخواننده مي شود كه درخواست اتصال بايد به آن ارسال شود. مشتري تا زمان ايجاد موفقيت آميز اتصال، بلوكه شده و پس از آن طرفين مي توانند به كمك عمليات اوليه ***connect*** و ***receive*** اقدام به تبادل اطلاعات نمايند. در نهايت اينكه بستن(پايان) اتصال در حين استفاده از سوكتها متقارن بوده و با فراخواني عمل اوليه ***close*** توسط هر دو طرف مشتري و سرور امكان پذير مي شود. الگوي كلي مشتري و سرور در ارتباطات اتصال گرا با استفاده از سوكتها در شكل ۱۵-۴ نمايش داده شده است. جزئيات

مربوط به برنامه ريزي شبكه ها با استفاده از سوكت ها و ديگر رابط ها در محيط UNIX را در كتاب(۱۹۹۸) Stevens مطالعه نماييد.

رابط رد كردن پيام(MPI)

همگام با ظهور مولتي كامپيوترهاي پربازده، سازندگان در پي ارائه عمليات اوليه پيام گرايي هستند كه بتوانند به كمك آنها برنامه هاي كاربردي بسيار بهره ور را به راحتي بنويسند. به اين معنا كه عمليات اوليه بايد(جهت تسهيل ايجاد برنامه كاربردي) در سطح انتزاع قابل قبولي بوده و پياده سازي آنها سربار حداقلي ايجادكند. سوكت ها به دو دليل فاقد كارآيي تصور مي شدند. اولاً ، به دليل پشتيباني از عمل هاي اوليه ساده ***send*** و ***receive***، از نظر انتزاع درحد نادرستي قرارداشتند. ثانياً، هدف از طراحي سوكتها ايجاد ارتباط در سرتاسر شبكه ها با استفاده از مجموعه هاي پروتكلي عام منظوره اي از قبيل TCP/IP بود. به همين دليل براي پروتكل هاي اختصاصي طرح ريزي شده براي شبكه هاي ميان ارتباطي پرسرعت، از قبيل آنهايي كه در خوشه هاي سرور[559] با كارآيي بالا مورد استفاده قرار مي گيرند، مناسب نمي باشند.در نتيجه، اين نوع پروتكل ها نيازمند وجود رابطي هستند كه بتواند با ويژگيهاي پيشرفته تري از قبيل اشكال مختلف بافر و همگام سازي عمل كند.

شكل ۱۵-۴-الگوي ارتباطات اتصال گرا با استفاده از سوكتها(داخل شكل:سرور،سوكت،bind،listen، accept، read، write، close، نقطه همگام سازي، ارتباطات،سوكت،bind، write، read، close، مشتري)

نتيجه اين شد كه اغلب شبكه هاي ميان ارتباطي[560] و مولتي كامپيوترهاي پربازده همراه با كتابخانه هاي ارتباط اختصاصي عرضه مي شدند. اين كتابخانه ها گنجينه عظيمي از عمل هاي اوليه ارتباطي سطح بالا وغالباً مؤثر ارائه ميكردند. البته تمامي كتابخانه ها به صورت دوجانبه ناسازگار بوده و از اينرو سازندگان برنامه هاي كاربردي با مشكل قابليت انتقال[561] مواجه شدند.

لزوم عدم وابستگي به سخت افزار و سكوي خاص منجر به تعريف استانداردي براي ردكردن پيام شد كه به نام **رابط ردكردن پيام**[562] يا **MPI** نام گرفت. MPI ، براي برنامه هاي كاربردي موازي طراحي شده و به همين دليل جهت ارتباطات ناپايدار سفارشي شده است. همچنين، فرض شده كه خرابي هاي جدي از قبيل شكست فرآيند يا بخشهاي شبكه حياتي بوده و به همين دليل نيازمند احياي اتوماتيك نيستند.

در MPI فرض بر اين است كه ارتباطات در داخل گروه شناخته شده اي از فرآيندها رخ مي دهد. به هر گروه يك شناسه نسبت داده مي شود. همچنين به هر يك از فرآيندهاي داخل گروه يك شناسه (محلي) تعلق مي گيرد. بنابراين جفت (ID گروه ، ID فرآيند) منحصراً يك منبع يا مقصد پيام را شناسايي نموده و به جاي آدرس لايه انتقال مورد استفاده قرار مي گيرد. ممكن است در محاسبه اي، گروههاي

[559] server clusters
[560] interconnection networks
[561] portability
[562] Message Passing Interface

فرآيند متعدد و احتمالاً متداخلي موجود باشد كه همگي به صورت همزمان در حال اجرا باشند.
در هسته MPI ، عمليات اوليه پيام رساني جهت پشتيباني از ارتباطات ناپايدار قرار دارند كه مشهودترين آنها در شكل ۱۶-۴ به طور خلاصه ارائه شده است.
ارتباطات ناپايدار همزمان بوسيله عمل هاي اوليه ***MPI-bsend*** پشتيباني مي شود. معمولاً هنگاميكه فرستنده پيامي را براي تبادل تسليم مي كند، اين پيام ابتدا به يك بافر در سيستم زمان اجراي[۵۶۳] MPI كپي مي شود. پس از آنكه پيام كپي شد، فرستنده به كار خود ادامه مي دهد. سيستم زمان اجراي محلي MPI پيام را از بافر محلي خود جدا كرده و به محض اينكه گيرنده عمل اوليه ***receive*** را فراخواني نمود، اقدام به تبادل مي كند.

عمليات اوليه	معني
MPI_ bsend	پيام خروجي را به يك بافر ارسال محلي ضميمه كن.
MPI_ send	پيامي را ارسال و تا زمان كپي شدن به يك بافر يا محلي صبر كن.
MPI_ ssend	پيامي را ارسال و تا زمان آغاز دريافت صبر كن .
MPI_ sendrecv	پيامي را ارسال و منتظر جواب بمان.
MPI_ isend	منبع را به پيام خروجي انتقال و ادامه بده.
MPI_ issend	منبع را به پيام خروجي انتقال و تا زمان آغاز دريافت منتظر بمان.
MPI_ recv	پيامي را دريافت كن؛و صورت عدم وجود پيام آنربلوكه كن.
MPI_ irecv	بررسي كن آيا پيام جديدي وجود دارد يا نه ،اما آنرا بلوكه نكن.

شكل ۱۶-۴- تعدادي از مشهودترين عمليات اوليه ردكردن پيام در MPI

همچنين يك عمل ارسال مسدود به نام ***MPI_ send*** وجود داردكه هم معناي آن وابسته به پياده سازي آن است. عمليات اوليه ***MPI_ send*** مي تواند يكي از ايندو كار را انجام دهد؛ **الف)** مسدود كردن فراخواننده تا زماني كه پيام در سيستم زمان اجراي MPI در طرف فرستنده كپي مي شود و **ب)** مسدود شدن فراخواننده تا زماني كه دريافت كننده عمليات دريافت را آغاز كند . ارتباط همگام به گونه اي كه فرستنده تا زماني كه درخواستش براي پردازش پذيرفته شود ، مسدود شود، به توسط عمل اوليه ***MPI_ ssend*** تأمين مي شود. در نهايت قوي ترين شكل ارتباط همگام نيز پشتيباني مي شود؛ يعني هنگامي كه يك فرستنده ***MPI_ sendrecv*** را فراخواني مي كند، درخواستي به دريافت كننده ارسال كرده و تا زمان بازگشت پاسخ

[563] runtime system

مسدود باقي مي ماند . اساساً، اين عمل اوليه متناظر با يك PRC عادي است.
MPI_ send و *MPI_ ssend* انواع ديگري دارند كه مانع از كپي شدن پيامها از بافر كاربر به بافر داخلي سيستم زمان اجراي MPI محلي ميشوند. اين انواع متناظر با شكلي از ارتباط همگام هستند. بوسيله *MPI_ isend* ، فرستنده نشانگر به پيام را رد كرده و پس از آن سيستم زمان اجراي MPI اجراي ارتباط را به انجام مي رساند . فرستنده بي درنگ ادامه مي دهد. جهت جلوگيري از رونويسي پيام پيش از پايان ارتباط، MPI عمليات اوليه اي را جهت بررسي تكميل يا حتي بلوكه كردن در صورت لزوم ، ارائه مي دهد. در اينجا هم مانند حالت *MPI_send* ، اين كه پيام واقعاً به گيرنده منتقل شده باشد يا اين كه بوسيله سيستم زمان اجرا صرفاً به يك بافركپي شده باشد ، نامشخص باقي مي ماند .
به همين ترتيب، در*MPI _ issend* ، فرستنده فقط نشانگر را به سيستم زمان اجراي MPI رد مي كند. هنگامي كه سيستم زمان اجرا بيان مي كند كه پيام را پردازش نموده فرستنده مطمئن مي شود كه گيرنده پيام را پذيرفته و بر روي آن كار مي كند.
عمليات *MPI _recev* ، براي دريافت پيام فراخواني مي شود؛ همچنين تا زمان رسيدن پيام، فراخواننده را مسدود مي كند. يك نوع ناهمگام هم به نام *MPI_ irecv* وجود دارد كه بوسيله آن گيرنده براي پذيرش پيام اعلان آمادگي مي كند.گيرنده مي تواند صحت و سقم واقعي رسيدن پيام را بررسي كرده و يا اينكه تا زمان رسيدن پيام مسدود شود.
معاني عمل هاي اوليه ارتباط MPI هميشه صريح و روشن نيستند، بطوريكه عمل هاي اوليه مختلف را مي توان بعضاً بدون داشتن تأثير بر صحت برنامه ، بجاي ديگري استفاده نمود. دليل واقعي پشتيباني از تعداد انبوهي از اشكال ارتباطاتي اين است كه به پياده سازي هاي سيستم هاي MPI امكانات كافي براي بهينه سازي كارآيي داده شود. در يونان باستان، اعضاي سنا خود را تنها و بالاترين اجماع قابل قبول مي دانستند، به همين دليل در همه تصميم گيري ها دخالت مي كردند. MPI براي برنامه هاي كاربردي موازي با كارآيي بالا طراحي شده است و همين امر باعث سهولت درك تنوع آن در عمل هاي اوليه ارتباطي مي شود . اطلاعات بيشتر در مورد MPI را در كتاب Gropp (۱۹۹۸) وگروه همكاران مطالعه كنيد. مرجع كاملي با بيش از ۱۰۰كاربرد در MPI با شرح تفصيلي آنها در كتاب Snir (۱۹۹۸) و گروه همكاران و همچنين Gropp (۱۹۹۸a)ذكر شده است.

۲-۳-۴-ارتباطات پايدار پيام گرا[564]

حال نوبت به بررسي گروه مهمي از سرويسهاي ميان افزار پيام گرا مي رسدكه غالباً به نام **سيستم هاي صف بندي پيام**[565] يا **ميان افزار پيام گرا**[566] **(MOM)**خوانده مي شود. سيستم هاي صف بندي پيام

[564] Message-Oriented Persistent Communication
[565] message-queuing systems
[566] Message-Oriented Middleware

پشتيباني قوي از ارتباطات ناهمگام پايدار ارائه مي كنند. بطور خلاصه اين سيستم ها ظرفيت ذخيره سازي ميان مدتي را براي پيام ها ايجاد مي كنند ، بدون آنكه حتي لازم باشد فرستنده يا گيرنده حين تبادل پيام فعال باقي بمانند. يكي از موارد مهم تفاوت اين نوع سيستم ها با سوكتهاي بركلي و MPI آن است كه سيستم هاي صف بندي پيام نوعاً جهت پشتيباني از تبادلات پيامي مورد استفاده قرار مي گيرند كه ممكن است به جاي چندين ثانيه يا هزارم ثانيه ، چند دقيقه طول بكشند. ابتدا روش كلي سيستم هاي صف بندي پيام را توضيح داده و اين بخش را با مقايسه آنها با سيستم هاي متعارف تر، علي الخصوص سيستم هاي پست الكترونيك اينترنت، به پايان مي رسانيم.

مدل صف بندي پيام

ايده اصلي تشكيل دهنده سيستمهاي صف بندي پيام آن است كه برنامه هاي كاربردي با درج كردن پيامها در صف هاي خاصي با هم ارتباط برقرار كنند. حتي اگر سرور مقصد در هنگام ارسال پيام بالا نباشد، اين پيامها بازهم روي يك سري از سرورهاي ارتباطي ارسال شده و نهايتاً به مقصد تحويل داده مي شوند . در عمل اكثر سرورهاي ارتباطي مستقيماً به يكديگر متصل هستند . در اصل، هر برنامه كاربردي صف خاص خود را دارد و ديگر برنامه هاي كاربردي مي توانند به آن پيام ارسال كنند. هر صف فقط بوسيله برنامه كاربردي مربوط به آن قابل خواندن است . اما اين امكان هم وجود دارد كه براي چندين برنامه كاربردي يك صف مشترك وجود داشته باشد.

يكي از جنبه هاي مهم سيستم هاي صف بندي پيام آن است كه به فرستنده فقط تضمين داده مي شودكه پيام او نهايتاً در صف گيرنده درج خواهد شد. هيچ تضميني در مورد زمان و يا حتي اينكه پيام واقعاً خوانده خواهد شد (كاملاً وابسته به رفتار گيرنده است) ارائه نمي شود.

اين معاني ارتباطاتي را امكان پذير مي كنند كه از نظر زماني ارتباط سستي دارند . بنابراين ، ديگر لزومي ندارد كه گيرنده در حين ارسال پيام به صف خود در حال اجرا باشد. همچنين ، لزومي ندارد كه فرستنده در لحظه اي كه پيام او توسط گيرنده گرفته مي شود، فعال باشد. فرستنده و گيرنده مي توانند كاملاً به طور مستقل از يكديگر اجرا شوند . در واقع، هنگامي كه پيامي در صف قرار داده مي شود، فارغ از اينكه فرستنده و گيرنده مربوطه در حال اجرا باشد ، تا زمان برداشته شدن در همانجا باقي خواهد ماند . بر همين اساس ، چهار تركيب مختلف نسبت به حالات اجراي فرستنده و گيرنده بوجود مي آيد كه در شكل ۱۷-۴ ديده مي شود.

شكل ۱۷-۴-چهار نوع تركيب براي ارتباطات داراي پيوند سست در استفاده از صف ها (داخل شكل:الف)فرستنده در حال اجرا-گيرنده در حال اجرا-ب)فرستنده در حال اجرا-گيرنده غير فعال-ج)فرستنده غير فعال-گيرنده در حال اجرا-د)فرستنده غير فعال-گيرنده غيرفعال).

در شكل ۱۷-۴(الف)، فرستنده و گيرنده در كل زمان مخابره پيام در حال اجرا هستند.در شكل ۱۷-۴ (ب)، فقط فرستنده در حال اجراست و گيرنده غير فعال، به اين معني كه در اين حالت تحويل پيام غير ممكن است. با اين وجود، فرستنده بازهم قادر به ارسال پيامها مي باشد. تركيب فرستنده غير فعال و گيرنده در حال اجرا

در شكل ۱۷-۴ (ج) نمايش داده شده است. در اين حالت،گيرنده قادر به خواندن پيام هاي ارسالي براي خود است، اما لزومي ندارد كه فرستنده هاي مربوط به آنها هم در حال اجرا باشند. نهايتاً در شكل ۱۷-۴ (د) مشاهده مي كنيم كه سيستم، حتي در صورت غير فعال بودن گيرنده و فرستنده هم، قادر به ذخيره سازي (و احتمالاً تبادل) پيام هاست.

اصولاً ، پيام ها مي توانند حاوي هر نوع داده اي باشند. تنها نكته مهم از بعد ميان افزاري ، آدرس دهي مناسب پيام ها است. در عمل، آدرس دهي از طريق ايجاد يك نام منحصر به فرد در تمامي سيستم براي صف مقصد انجام مي شود. ممكن است در بعضي موارد، برروي اندازه پيام محدوديت وجود داشته باشد ؛ گرچه سيستم زيرين مي تواند عمل تجزيه و سرهم بندي مجدد پيام هاي بزرگ را به گونه اي شفاف (مخفي) براي برنامه هاي كاربردي انجام دهد .يكي از اثرات اين روش آن است كه رابط اصلي ارائه شده براي برنامه هاي كاربردي مي تواند مشابه شكل ۱۸-۴ بسيار ساده شود .

عمليات اوليه	معني
Put	پيام را به صف مورد نظر ضميمه كن
Get	تا زمانيكه صف مورد نظر خالي باشد، مسدود شو و پيام اول را بردار
Poll	پيامهاي صف مورد نظر را چك كرده و پيام اول رابردار . هرگز مسدود نشو.
Notify	عمل كننده اي نصب كن تا زماني كه پيام درصف مورد نظر قرار داده شود ، فراخواني شود

شكل ۱۸-۴-رابط اصلي يك صف در يك سيستم صف بندي پيام

عمل هاي اوليه ***Put*** بوسيله فرستنده جهت ردكردن پيامي كه قرار است به صف خاصي ضميمه شود، به سيستم زيرين فراخواني مي شود. همانطور كه قبلاً هم گفتيم، اين فراخواني غير انسدادي است. عمل اوليه ***Get*** فراخواني انسدادي است كه يك فرآيند تأييد شده مي تواند بوسيله آن پيامي را كه بيشترين مدت در صف بوده را بردارد. فرآيند فقط در صورت خالي بودن صف قابل انسداد است. انواع مختلف اين فراخواني امكان جستجوي پيامي خاص رادر صف، مثلاً براساس اولويت يا هماهنگي با يك الگو امكانپذير مي كند. گونه غير انسدادي بوسيله عمل اوليه ***Poll*** ارائه مي شود. در صورت خالي بودن صف يا چنانچه نتوان پيام خاصي را پيداكرد، فرآيند فراخواني به كار خود ادامه مي دهد.

در نهايت اينكه، اغلب سيستم هاي صف بندي به فرآيند امكان مي دهد تا عمل كننده اي[۵۶۷] به نام *كاركرد بازفراخواني*[۵۶۸] ايجاد كند تا هر زماني كه پيام در صف قرار داده مي شود، احضار شود. از بازفراخواني ها همچنين مي توان جهت آغاز اتوماتيك فرآيندي براي برگيري پيام ها از صف در زماني استفاده نمود كه فرآيند ديگري فعال نباشد . اين روش غالباً بوسيله يك برنامه محافظ در طرف گيرنده پياده سازي مي شود. اين محافظ دائماً صف را از نظر

[567] handler
[568] callback function

وجود پيامهاي جديدالورود بررسي و به نحو مناسب آنها را اداره مي كند .

معماري عمومي سيستم هاي صف بندي پيام

اجازه دهيد نگاه دقيقتري به ظاهر كلي يك سيستم صف بندي پيام داشته باشيم. يكي از اولين محدوديت ها اين است كه پيام ها را مي توان فقط در صف هايي قرار دادكه محلي فرستنده باشند؛ يعني صف هاي همان ماشين يا حداكثر روي ماشيني نزديك و برروي همان LAN ، بطوريكه به كمك يك RPC به نحو مناسب قابل دسترسي باشد.چنين صف بندي **صف منبع**[٥٦٩] خوانده مي شود. به همين ترتيب ،پيامها فقط از طريق صف هاي محلي قابل خواندن هستند. با اين وجود، پيامي كه در صف قرار مي گيرد حاوي توصيف **صف مقصدي**[٥٧٠] است كه بايد به آن انتقال داده شود . از وظايف سيستم صف بندي پيام آن است كه صف هايي براي فرستنده وگيرنده ايجادكرده و مراقب تبادل پيامها از منبع خود به صف مقصد آنها باشد.

لازم است بدانيم كه مجموعه صف ها در سرتاسر ماشين هاي مختلفي توزيع مي شوند. در نتيجه، براي آنكه يك سيستم صف بندي پيام قادر به انتقال پيام ها باشد، بايد نگاشت صف ها به مكانهاي شبكه را در اختيار داشته باشيم . همانطور كه در شكل ١٩-٤ مشاهده مي كنيد، در عمل اين بدان معناست كه بايد حاوي پايگاه داده اي (احتمالاً توزيع شده اي) از **اسامي صف ها**[٥٧١] به مكان هاي شبكه باشد. توجه كنيد كه چنين نگاشتي[٥٧٢] كاملاً مشابه استفاده از سيستم نام دامنه[٥٧٣](DNS) در پست الكترونيك اينترنت است. بعنوان مثال، براي ارسال پيام به آدرس منطقي *پست* steen@cs.vu.nl ، سيستم پستي براي يافتن آدرس *شبكه اي*(يا IP)سرور پستي گيرنده ، DNS را جستجو خواهد كرد تا پيام انتقال داده شود.

شكل ١٩-٤-رابطه بين آدرس دهي سطح صف و آدرس دهي سطح شبكه (داخل شكل:فرستنده ،لايه صف بندي، سيستم عامل محلي،جستجوي آدرس سطح انتقال صف،آدرس سطح صف،پايگاه داده اي جستجوي آدرس، گيرنده ،لايه صف بندي، سيستم عامل محلي،شبكه ،آدرس سطح انتقال)

مديريت صف ها بر عهده مديران صف است. غالباً مدير صف با برنامه كاربردي كه در حال دريافت و ارسال پيام است،مستقيماً تبادل دارد. با اين وجود، برخي مديران صف بعنوان مسير ياب ،يا **رله**[٥٧٤] عمل مي كنند؛ يعني پيام هاي ورودي را به ديگر مديران صف انتقال مي دهند . به اين ترتيب، يك سيستم صف بندي پيام ممكن است تدريجاً تبديل به يك **شبكه فوقاني**[٥٧٥] كامل سطح برنامه كاربردي در بالاي شبكه كامپيوتري موجود شود . اين روش مشابه اولين پياده سازي MBone روي اينترنت است كه در آن فرآيندهاي كاربر به عنوان مسيرياب چندپخشي پيكربندي مي شوند. همانطور كه مشاهده خواهيم كرد، اهميت چند پخشي از طريق شبكه هاي فوقاني هنوز هم به قوت خود باقي است.

---

[569] source queue
[570] destination queue
[571] queue names
[572] mapping
[573] Domain Name System
[574] relay
[575] overlay network

به چند دليل رله ها مي توانند مفيد باشند. بعنوان مثال، در بسياري از سيستم هاي صف بندي پيام هنوز هيچ سرويس نامگذاري عمومي وجود ندارد كه قادر به نگهداري پوياي نگاشت هاي صف به مكان باشد. در حال حاضر، توپولوژي شبكه صف بندي ايستا بوده و هر مدير صف بايد مستقلاً كپي از نگاشت صف- به- مكان در اختيار داشته باشد. نگفته پيداست كه در سيستم هاي صف بندي مقياس وسيع ، اتخاذ اين روش مي تواند باعث بروز مشكلات مديريتي شبكه شود.
يك راه حل براي اين مشكل استفاده از تعداد معدودي از مسيرياب هايي است كه از توپولوژي شبكه اطلاع داشته باشند. هنگاميكه فرستنده *A* پيامي را براي مقصد *B* در صف محلي خود قرار مي دهد، مطابق شكل ۲۰-۴ ، اين پيام ابتدا به نزديكترين مسيرياب مثلاً به *R1* منتقل مي شود.سپس، مسيرياب پيام را به سمت *B* ارسال مي كند.به عنوان مثال، *R1* با بررسي نام *B* متوجه مي شود كه پيام بايد به مسيرياب *R2* انتقال داده شود. فقط هنگامي مسيرياب ها به روزرساني مي شوند كه صف يا صف هايي اضافه يا حذف شوند و هريك از مديران صف هاي ديگر بايد فقط از محل نزديكترين مسيرياب اطلاع داشته باشد.
بنابراين رله ها غالباً مي توانند به ساخت سيستم هاي مقياس پذير صف بندي پيام كمك كنند. با اين وجود، همزمان با رشد سيستمهاي صف بندي پيام ،پيكربندي دستي شبكه ها غيرقابل مديريت خواهد شد. تنها راه حل ممكنه براي مشكل آن است كه همانند شبكه هاي كامپيوتري، از روش هاي مسيريابي پويا استفاده كنيم. تعجب آور است كه چنين روشهايي هنوز در برخي از سيستمهاي متداول صف بندي پيام نهفته نشده اند .
دليل ديگر براي استفاده از رله ها اين است كه پردازش ثانويه پيام ها را امكانپذير مي نمايد . بعنوان مثال، ممكن است لازم باشد به دلايل امنيتي يا جهت تحمل خرابي ها، پيامها ثبت شوند. يكي از انواع خاص رله كه در بخش بعد مورد بحث قرار خواهيم داد رله اي است كه به صورت دروازه عمل مي كند، يعني پيامها را تبديل به قالب قابل فهم براي گيرنده مي كند.
سخن آخر اينكه از رله ها مي توان جهت چند پخشي استفاده نمود.در اينصورت، هر پيام ورودي در هر صف ارسال قرار داده خواهد شد.

شكل ۲۰-۴-سازمان بندي كلي يك سيستم صف بندي پيام با مسيرياب ها(فرستنده A ،برنامه كاربردي،دريافت صف، ارسال صف،پيام،برنامه كاربردي،برنامه كاربردي،مسيرياب،برنامه كاربردي،گيرنده B).

واسطه هاي پيام[576]

يكي از موارد مهم كاربرد سيستمهاي صف بندي پيام ، تلفيق برنامه هاي كاربردي فعلي و جديد به صورت يك سيستم اطلاعات توزيعي منسجم و منفرد است. چنين تلفيقي الزام مي كند كه برنامه هاي كاربردي قادر به درك پيام هاي دريافتي باشند. در عمل، اين ويژگي باعث ميشود تا فرستنده پيام هاي ارسالي خود را در قالبي همانند قالب گيرنده قرار دهد.
يكي از معايب اين روش آن است كه در صورتيكه برنامه كاربردي به سيستم افزوده شود كه نيازمند قالب پيام جداگانه اي باشد،

[576] message brokers

تمامي گيرنده هاي احتمالي بايد تنظيم شوند تا امكان توليد قالب مورد نظر فراهم شود.
روش ديگري كه در پروتكل هاي متعارف شبكه مورد استفاده قرار مي گيرد، توافق بر سر يك قالب مشترك پيام است. متأسفانه، اين روش غالباً در مورد سيستم هاي صف بندي پيام صدق نمي كند. مشكل ناشي از سطح انتزاع عمل اين سيستم هاست. قالب پيام مشترك فقط در صورتي معنادارو قابل استفاده است كه مجموعه فرآيندهايي كه از آن قالب استفاده مي كنند، واقعاً به اندازه كافي باهم اشتراك داشته باشند. در صورت تفاوت فاحش بين مجموعه برنامه هاي كاربردي كه يك سيستم اطلاعات توزيعي را تشكيل مي دهند (كه غالباً همينطور هم هست)، بهترين قالب مشترك همان زنجيره بايتهاست.
هرچند تعداد كمي قالب هاي پيام مشترك براي برخي دامنه هاي برنامه كاربردي تعريف شده است ، ولي روش كلي و توصيه شده يادگيري نحوه برخورد با قالب هاي مختلف و ايجاد روشهايي جهت انجام تبديلات به ساده ترين صورت ممكنه است. در سيستم هاي صف بندي پيام، اين دست تبديلات بوسيله برخي گره هاي موجود در شبكه صف بندي به نام واسطه هاي پيام انجام مي شود. واسطه هاي پيام در سيستمهاي صف بندي پيام بعنوان يك دروازه سطح برنامه كاربردي عمل مي كنند. هدف اصلي اين واسطه ها، تبديل پيامهاي ورودي به صورتي قابل فهم براي برنامه كاربردي مقصد است. توجه داشته باشيد كه مطابق شكل ۲۱-۴ ، واسطه پيام در سيستم صف بندي پيام صرفاً بعنوان يك برنامه كاربردي ديگر عمل مي كند. به بيان ديگر، واسطه پيام غالباً جزء لازم صف بندي تلقي نمي شود.
شكل ۲۱-۴-سازمان بندي كلي واسطه پيام در يك سيستم صفبندي پيام(داخل شكل:مشتري منبع،واسطه پيام،منبع حاوي برنامه هاو قوانين تبديل،برنامه واسطه،لايه صف بندي،مشتري مقصد،شبكه).
واسطه پيام مي تواند براي پيام ها مانند يك مبدل عمل كند. بعنوان مثال، فرض كنيد كه يك پيام ورودي حاوي جدولي از پايگاه داده اي باشد كه در آن ركوردها بوسيله يك حائل *پايان ركورد* خاص تفكيك شده و حوزه هاي[۵۷۷] داخل ركورد طول ثابت و شناخته شده اي دارند. چنانچه برنامه كاربردي مقصد انتظارحائل بين ركورد ديگري را داشته و بعلاوه انتظار داشته باشد كه اين حوزه ها از طولهاي مختلفي برخوردار باشند، از واسطه پيام مي توان جهت تبديل پيامها به قالب مقصد موردنظر استفاده نمود.
در موارد پيشرفته تر، واسطه پيام ممكن است به عنوان دروازه سطح برنامه كاربردي عمل نمايد كه از نمو نه هاي آن مي توان به دروازه هاي مسئول تبديل بين برنامه كاربردي دو پايگاه داده اي مختلف اشاره كرد. در اينصورت، همواره نمي توان تضمين كرد كه تمامي اطلاعات داخل پيام ورودي واقعاً به چيزي مفيد براي پيام خروجي تبديل شوند.
با اين وجود، معمولاً از واسطه پيام براي **تلفيق برنامه كاربردي شركتي[۵۷۸](EAI)** استفاده مي شود كه در فصل يك مفصلاً مورد بحث قرار گرفت.در اين حالت، واسطه به جاي (صرفاً) تبديل پيامها، مسئول هماهنگ سازي برنامه هاي كاربردي بر اساس پيامهاي مبادله

[577] fields
[578] enterprise application integration

شده است.در اين مدل كه **انتشار/ اشتراك**[579] نام دارد، پيام ها توسط برنامه هاي كاربردي به شكل *انتشاري* ارسال مي شوند. در حالت خاص، اين مدل ممكن است پيامي را در مورد موضوع X انتشار داده و سپس به واسطه ارسال كنند. بنابراين، برنامه هاي كاربردي كه نسبت به موضوع X علاقه مند بوده و يا به عبارت ديگر، در آن پيامها مشترك شده اند، اين پيام ها را از واسطه دريافت مي كنند. اشكال پيشرفته ديگري هم از واسطه گري وجود دارد كه بحث در مورد آنها را به فصل ۱۳ موكول مي كنيم.
در دل واسطه پيام، منبع قوانين و برنامه هايي قرار دارد كه مي توان به كمك آنها يك پيام از نوع T1 را به شكل T2 تبديل نمود. مشكل اينكار تعريف قوانين و ايجاد برنامه هاست . اغلب محصولات واسطه پيام با ابزارهاي توسعه پيشرفته ارائه مي شوند، اما در نهايت مجموعه اطلاعات منبع بايد از كارشناسان خبره دريافت شود . ذيلاً نمونه اي از محصولات تجاري ذكر مي شود كه به اشتباه ارائه دهنده "هوشمندي" تلقي مي شوند ، اما واقعيت اين است كه هوش تنها در مغز كارشناسان خبره قرار دارد .

نكته اي در مورد سيستم هاي صف بندي پيام

با بررسي مجدد آنچه تاكنو ن درمورد سيستم هاي صف بندي پيام گفته شد، به نظر مي رسد كه اين سيستم ها مدتهاست در سرويس هاي پست الكترونيك پياده سازي شده و در حال كاربرد هستند. سيستم هاي پست الكترونيك غالباً از طريق مجموعه اي از سرورهاي پستي پياده سازي مي شوند كه پيام ها را به نمايندگي از جانب كاربران روي ميزبان هايي كه مستقيماً به سرور متصل هستند، ذخيره و ارسال مي كنند . از آنجايي كه سيستم هاي پست الكترونيك مي توانند مستقيماً از سرويس هاي انتقال زيرين استفاده كنند، مسيريابي كاربرد ندارد. به عنوان مثال، در پروتكل پست اينترنت يا SMPT (Postel،۱۹۸۲) ارسال پيام از طريق ايجاد اتصال TCP مستقيم به سرور پست مقصد انجام مي پذيرد.
وجه تمايز ويژه سيستم هاي پست الكترونيك از سيستم هاي صف بندي پيام آن است كه هدف اصلي سيستم هاي پست الكترونيك، ايجاد پشتيباني مستقيم براي كاربران نهايي است. از همين رو مي توان بعنوان مثال توضيح داد كه چرا برخي از برنامه هاي كاربردي گروه افزار مستقيماً بر اساس سيستم پست الكترونيك استوار شده اند (Khoshafian and Buckiewicz،۱۹۹۵). بعلاوه، سيستم هاي پست الكترونيك از ويژگي هاي بسيار خاصي از قبيل فيلترينگ خودكار پيام، پشتيباني از پايگاههاي پيشرفته داده اي(مثلاً جهت بازيابي آسان پيام هاي قبلاً ذخيره شده) و غيره برخوردار هستند.
هدف سيستم هاي عمومي صف بندي پيام فقط حمايت از كاربران نهايي نيست. يك هدف مهم از ايجاد آنها برقراري امكان ارتباطات پايدار در بين فرآيندهاست، مستقل از آنكه آيا فرآيند در حال اجراي برنامه كاربردي كاربر است يا دسترسي به پايگاه داده اي را برعهده دارد و يا به انجام محاسبات مشغول است .اين شيوه باعث مي شود تا سيستم هاي صف بندي پيام در قياس با سيستم هاي پست الكترونيكي از مجموعه ويژگي هاي متنوع تري برخوردار باشند . بعنوان مثال، غالباً نيازي نيست كه سيستم هاي پست الكترونيك

[579] publish/ subscribe

براي يك كاربرد عام ، تحويل تضميني پيام، اولويت بندي پيام، امكانات ثبت وقايع، چند پخشي مؤثر، تعادل بار ، تحمل خطا و از اين قبيل را ارائه دهد .
بنابراين، سيستم هاي صف بندي پيام عام منظوره داراي كاربردهاي گسترده اي از قبيل پست الكترونيك، جريان كار[580]، گروه افزار و پردازش دسته مي باشند. با اين وجود، همچنان كه قبلاً هم گفتيم، مهمترين زمينه كاربرد آنها تلفيق مجموعه (احتمالا بسيار پراكنده اي) از پايگاههاي داده اي و برنامه هاي كاربردي به شكل يك سيستم اطلاعاتي است(Hohpe and Woolf ، ۲۰۰۴). بعنوان مثال، ممكن است لازم باشد پرس وجويي كه در چندين پايگاه داده اي انجام مي شود، به چندين زير مجموعه تقسيم و به پايگاههاي داده اي مختلف ارسال شود. سيستم هاي صف بندي پيام ابزار اصلي را براي بسته بندي هريك از زيرپرس وجوها به شكل پيام و مسيردهي آن به پايگاه داده اي مناسب ايجاد مي كنند. ديگر امكانات ارتباطي كه در اين بخش مورد بحث قرار گرفت، چندان براي اين كار مناسب نيستند.

۳-۳-۴-مثال:سيستم صف بندي پيام WebSphere

جهت درك بهتر نحوه عملكرد سيستم هاي صف‌بندي پيام ، اجازه دهيد نگاهي به يكي از سيستم هاي خاص يعني سيستم صف بندي پيام بيندازيم كه بخشي از محصول IBM WebSphere محسوب مي شود. اين محصول كه سابقاً MQSeries ناميده مي شد، امروزه **MQ WebSphere** خوانده مي شود. مستندات فراواني در ارتباط با MQ WebSphere وجود دارد ، اما ذيلاً فقط اصول كلي اصلي را ذكر مي كنيم. بسياري از جزييات مربوط به معماري شبكه هاي صف‌بندي پيام را مي توانيد در (IBM ، ۲۰۰۵ b ، ۲۰۰۵d) مطالعه كنيد. برنامه نويسي شبكه هاي صف بندي پيام را نمي توان يك شبه ياد گرفت و راهنماي برنامه نويسيMQ (IBM، ۲۰۰۵ a) شاهد خوبي دال بر اين مدعاست كه از نظريه تا عمل راه فراوان است.

شرح كلي

معماري اصلي شبكه صف بندي MQ ،كه نمونه اي از آن را در شكل ۴-۲۲ مشاهده مي كنيد ، صريح و قابل فهم است. تمامي صف ها تحت مديريت **مديران صف**[581] قرار دارند. مدير صف مسئول برداشتن پيامها از صف هاي ارسال مربوطه و پيش بردن آنها به سمت ديگر مديران صف است. بعلاوه، مدير صف مسئول هدايت پيامهاي ورودي از طريق برداشت آنها از شبكه زيرين و سپس ذخيره سازي هر پيام در صف ورودي مناسب است. جهت درك بهتر مفهوم پيام رساني لازم است بدانيد كه حداكثر سايز پيش فرض براي يك پيام ۴ مگابايت است كه تا ۱۰۰MB هم قابل افزايش مي باشد. در يك صف معمولاً ۲ GB داده وجود داردكه بسته به سيستم عامل زيرين، اين رقم به راحتي قابل افزايش است.
مديران صف به صورت دوتايي از طريق **كانالهاي پيام**[582] متصل هستند كه انتزاعي ازاتصالات سطح انتقال هستند. يك كانال پيام، اتصال

[580] workflow
[581] queue managers
[582] message channels

مطمئن و يك جهتي بين مدير صف ارسال و دريافت است كه پيامهاي صف بندي شده از طريق آن انتقال مي يابند. بعنوان مثال، يك كانال پيام بر اساس اينترنت به صورت اتصال TCP اجرا مي شود. هر يك از دو سر كانال پيام توسط **نماينده كانال پيام**[583] **(MCA)** مديريت مي شود. MCA ارسال اصولاً وظيفه اي غير از بررسي صف هاي ارسال از نظر وجود پيام، تبديل آن به شكل بسته سطح انتقال و ارسال آن از طريق اتصال به MCA دريافت مربوطه ندارد. به همين ترتيب، يكي از وظايف اصلي MCA دريافت، گوش دادن براي ورود بسته ، باز كردن آن و سپس ذخيره پيام باز شده در صف مناسب است.

شكل۲۲-۴- سازمان بندي كلي يك سيستم صف بندي پيام IBM (داخل شكل:مشتري ارسال،برنامه،رابط MQ ،مجازي، RPC(همگام)،جدول مسيردهي،مديرصف،صف ارسال،سرورمجازي،شبكه محلي،صف دريافت مشتري،مدير صف، سرور مجازي،مشتري دريافت،برنامه،مجازي ،شبكه شركتي،ردكردن پيام(ناهمگام)،به سمت ديگر مديران صف دور).

مديران صف را مي توان به همان برنامه كابردي متصل نمود كه صف را براي آن مديريت مي كنند. در اين حالت، صف ها از برنامه كاربردي به توسط يك رابط استاندارد مخفي مي شوند، اما مي توان آنها را به نحو مؤثري بوسيله برنامه كاربردي دستكاري نمود. در يك سازمان بندي ديگر، مديران صف و برنامه هاي كاربردي روي ماشين هاي جداگانه اجرا مي شوند. در اين حالت، همان رابطي به برنامه كاربردي ارائه مي شود كه در صورت قرارگرفتن مدير صف روي آن ماشين ارائه خواهد شد. در اينصورت ، رابط به عنوان ميانجي[584] اجرا مي شود كه با استفاده از ارتباط همگام متعارف براساس RPC با مدير صف ارتباط پيدا مي كند. به اين ترتيب، MQ مدلي را كه بتوان فقط به صف هاي محلي يك برنامه كاربردي دسترسي پيدا كرد، حفظ مي كند .

كانالها

يكي از اجزاء مهم و اصلي MQ ،كانالهاي پيام است. هر كانال پيام دقيقاً داراي يك صف ارسال است و پيامهاي ارسالي به طرف ديگر از آن انتخاب خواهند شد. انتقال در داخل كانال فقط در صورتي امكانپذير است كه هردو MCA دريافت و ارسال آن برقرارو در حال اجرا باشند. غير از راه اندازي دستي MCA ها ،روشهاي متعدد ديگري براي راه اندازي كانال وجود دارد كه ذيلاً راجع به آنها بحث خواهيم كرد.

يك روش آن است كه برنامه كاربردي با فعال كردن MCA ارسال يا دريافت ، سه كانال طرف خود را فعال كند . با اين وجود، به دليل كاهش شفافيت، اين روش چندان جالب و مفيد نيست. روش بهتر براي آغاز MCA ارسال آن است كه صف ارسال كانال را به صورتي پيكربندي كنيم كه در صورتي كه پيام براي اولين بار در صف قرار مي گيرد، يك تريگر[585] فعال شود. تريگر به عاملي متصل است كه MCA ارسال را آغاز مي كند تا بتواند پيامها را از صف ارسال بردارد.

---

[583] message channel agent
[584] proxy
[585] trigger

روش ديگر، آغاز MCA روي شبكه است. بخصوص ، در صورت فعال بودن يكي از طرفين كانال، طرف مذكور مي تواند با ارسال پيام كنترل از طرف ديگرMCA درخواست آغاز نمايد. اين پيام كنترل به محافظي ارسال مي شود كه به آدرس معروفي روي همان ماشين گوش مي دهد.
توقف كانالها بطور اتوماتيك و پس از گذشت مدت زماني معين كه هيچ پيام جديدي در صف ارسال انداخته نشود، انجام مي پذيرد.
هر MCA داراي مجموعه اي از ويژگي هاي مربوطه است كه رفتار كلي كانال را تعيين مي كند. برخي از اين ويژگي ها در شكل ۲۳-۴ درج شده است. مقادير ويژگي MCA ارسال و دريافت بايستي سازگار بوده و احتمالاً پيش از برقراري كانال مورد توافق قرار گيرند. بعنوان مثال، هر دو MCA ها بايد آشكارا از يك پروتكل انتقال واحد پشتيباني كنند. يك نمونه از ويژگي هاي قابل بحث آن است كه آيا پيام ها بايد به همان ترتيب قرارگيري در صف ارسال، تحويل داده شوند يا خير. چناچه يك MCA خواستار تحويل به صورت FIFO باشد، ديگري هم بايد با آن هماهنگ باشد. يك نمونه از مقادير ويژگي قابل توافق ، حداكثر طول پيام است كه حداقل مقدار اعلام شده بوسيله دو MCA انتخاب مي شود.

ويژگي	شرح
نوع انتقال	تعيين كننده پروتكل انتقال مورد استفاده است
تحويل FIFO	بيان مي كند كه پيامها بايد به همان ترتيت ارسال تحويل داده شوند
طول پيام	حداكثر طول يك پيام
شمارش دفعات تلاش براي راه اندازي	بيانگر حداكثر دفعات تلاش براي آغاز يك MCA راه دور است
دفعات تكرار تحويل	حداكثر دفعاتيكه MCA تلاش مي كند تا پيام دريافتي را در صف قرار دهد

شكل ۲۳-۴- برخي از ويژگي هاي مربوط به نمايندگي هاي كانال پيام

انتقال پيام

براي انتقال پيام از يك مدير صف به مدير(احتمالاً دور) صف ديگر،لازم است كه هر پيام حاوي آدرس مقصد خود باشد كه بدين منظور از يك سرپيام استفاده مي شود. هر آدرس در MQ شامل دو بخش است. بخش اول شامل نام مدير صفي است كه پيام بايد به آن تحويل داده شود. بخش دوم نام صف مقصد تحت مديريت آن است كه پيام بايد به آن ضميمه شود.
لازم است علاوه بر آدرس مقصد، مسيري هم كه پيام بايد طي كند مشخص شود. تعيين مسير از طريق ارائه نام صف ارسال محلي انجام مي پذيرد كه پيام بايد به آن ضميمه شود. بنابراين لزومي ندارد كه مسير كامل در پيام قيد شود. به ياد داشته باشيد كه هر كانال پيام دقيقاً حاوي يك صف ارسال است. با ذكر اينكه پيام بايد به كدام صف ارسال ضميمه شود، مي توان به نحو مؤثري اقدام به تعيين مدير صفي نمود كه پيام بايد به آن ارسال شود.

در اغلب موارد، مسيرها به صورتي واضح در جدول مسيريابي در داخل مدير صف ذخيره مي شوند. ثبت در جدول مسيريابي به صورت جفتي (QM مقصد، Q ارسال) انجام مي پذيرد كه در آن، *QM مقصد* نام مديرصف مقصد و*Q ارسال* نام صف ارسال محلي است كه پيام مربوط به مدير آن صف بايد بدان ضميمه شود.(ورودي به جدول مسيريابي در نام مستعار[۵۸۶] خوانده مي شود).

اين احتمال وجود دارد كه يك پيام پيش از رسيدن به مقصد خود از چندين مدير صف رد كند. هرزماني‌كه پيام تحت اين شرايط توسط مدير صف مياني دريافت شود ، مدير صرفاً نام مدير صف مقصد را از عنوان پيام استخراج كرده و در جدول مسيرها به دنبال صف ارسال محلي مي گردد كه پيام بايد به آن ضميمه شود.

لازم به ذكر است كه هر مدير صف داراي يك نام منحصر به فرد در تمامي سيستم است كه به نحو مؤثري بعنوان شناسه آن مدير صف مورد استفاده قرار مي گيرد. مشكل استفاده از چنين اسامي آن است كه جايگزيني مدير صف يا تغيير نام آن بر تمامي برنامه هاي كاربردي كه پيام ها به آن ارسال مي شود ، تأثير خواهد گذاشت. مي توان با استفاده از يك **نام مستعار محلي**[۵۸۷] براي اسامي مدير صف، اين دست مشكلات را تاحدودي حل و فصل نمود . نام مستعار تعريف شده داخل مدير صف *M1* نام ديگري براي مدير صف *M2* است، اما فقط براي برنامه هاي كاربردي رابط*M1* قابل استفاده است. نام مستعار استفاده از يك نام(منطقي) واحد براي صف را، حتي در صورت تغيير مدير آن صف هم امكان‌پذير مي كند. تغيير نام مدير صف مستلزم تغيير نام مستعار آن در تمام مديران صف است. اما ، كوچكترين تغييري در برنامه هاي كاربردي ايجاد نخواهد شد.

شكل ۲۴-۴- سازمان بندي كلي يك شبكه صف بندي پيام MQ با استفاده از جداول و نام هاي مستعار مسيردهي(داخل شكل:جدول نام مستعار،جدول مسيردهي، جدول نام مستعار،جدول مسيردهي،جدول نام مستعار،جدول مسيردهي،جدول مسيردهي).

اصل كلي استفاده از جداول مسيردهي و نام مستعار در شكل ۲۴-۴ ارائه شده است. بعنوان مثال، مي توان با استفاده از نام مستعار محلي *LA1* ،برنامه كاربردي متصل به مدير صف *QMA* را به يك مدير صف دور ارجاع داد. مدير صف ابتدا در جدول نام مستعار ، مقصد واقعي را جستجو خواهد كرد تا مدير صف *QMC* را پيدا كند. مسير به سمت *QMC* درجدول مسيريابي يافت مي شود و همين بيان مي دارد كه پيام هاي مربوط به *QMC* بايد به صف خروجي *SQ1* ضميمه شوند كه از آن جهت انتقال پيام ها به مدير صف *QMB* استفاده شده است. مدير صف *QMB* هم از جدول مسيريابي جهت ارسال پيام به *QMC* استفاده مي كند.

اتخاذ اين روش جهت تعيين مسير و نام مستعار منجر به برنامه نويسي رابطي مي شود كه ساده بوده و به نام **رابط صف پيام**[۵۸۸] **(MQI)** خوانده مي شود. مهمترين عمليات اوليه MQI در شكل ۲۵-۴ به طور خلاصه ارائه شده اند.

---

[586] alias
[587] local alias
[588] Message Queuing Interface

عمليات اوليه	شرح
MQopen	يك صف (احتمالاً دور) را باز مي كند
MQclose	يك صف را مي بندد
MQput	پيامي را در صف باز شده قرار مي دهد
MQget	پيامي را از يك صف (محلي)مي گيرد

شكل۲۵-۴-عمل هاي اوليه موجود در رابط صف‌بندي پيام

جهت قراردادن پيامها در صف ، برنامه كاربردي عمل اوليه *MQopen* را فرامي خواند و يك صف مقصد را دريك مدير صف خاص مشخص مي كند . مدير صف را مي توان با استفاده از نام مستعار موجود به صورت محلي نامگذاري كرد. دور يا محلي بودن صف مقصد براي برنامه كاربردي كاملاً مخفي است. همچنين فراخواني *MQopen* هنگامي صورت مي گيرد كه برنامه كاربردي بخواهد پيام ها را از صف محلي خود بگيرد . براي خواندن پيام هاي ورودي، فقط صفهاي محلي را مي توان باز كرد. پس از دسترسي به صف و پايان برنامه كابردي ،لازم است با فراخواني *MQclose* اقدام به بستن آن نمود.
با استفاده از *MQput* و *MQget* به ترتيب امكان نوشتن و خواندن پيام ها از صف فراهم مي شود. در مجموع، برداشتن پيام از صف بر اساس اولويت بندي صورت مي پذيرد. پيام هاي داراي اولويت واحد بر اساس قاعده اولين ورود -اولين خروج برداشته مي شوند، به اين معنا كه پيامي كه طولاني ترين انتظار را داشته ، زودتر از همه خارج مي شود. بعلاوه امكان درخواست پيام هاي خاص وجود دارد. در پايان بايد گفت كه MQ امكاناتي را براي سيگنال دهي به برنامه هاي كاربردي به محض ورود پيامها ارائه داده و به اين ترتيب ديگر لازم نيست كه برنامه كاربردي دائماً صف پيام مورد نظر را براي يافتن پيامهاي جديد جستجو كند.

مديريت شبكه هاي فوقاني

تا به اينجا مي توان نتيجه گرفت كه يكي از جنبه هاي مهم سيستم هاي مديريت MQ ، اتصال مديران مختلف صف به يك شبكه فوقاني پايدار است. بعلاوه، لازم است كه اين شبكه در طول زمان نگهداري شود . در شبكه هاي كوچك ، اين نگهداري فقط مستلزم اعمال مديريتي متوسطي است. اما هنگامي كه از همين صف بندي پيام جهت تلفيق و تشكيل سيستم هاي بزرگ موجود استفاده مي شود، اوضاع تا حدودي پيچيده مي شود.
يكي از مشكلات مهم MQ آن است كه شبكه هاي فوقاني بايد به صورت دستي مديريت شوند. چنين مديريتي علاوه بر كانال سازي بين مديران صف ، شامل پركردن جداول مسيردهي هم مي باشد. نگفته پيداست كه گسترش اين وظيفه بسيار طاقت فرساست. متأسفانه، پشتيباني مديريتي از سيستم هاي MQ فقط به گونه اي پيش مي رود كه يك مدير اجرايي[۵۸۹] هر نوع ويژگي ممكن را قرار داده و هر پيكربندي قابل تصور را اعمال نمايد . با اين وجود، لازم است

[589] administrator

كانال ها و جداول مسيردهي بازهم به صورت دستي حفظ و نگهداري شوند.
مؤلفه **تابع كنترل كانال**[590] قلب مديريت فوقاني است كه منطقاً بين نماينده هاي كانال پيام جاي دارد. اين مؤلفه به اپراتور امكان مي دهد تا دقيقاً برآنچه برسر دو نقطه انتهايي كانال مي رود، نظارت كند. علاوه بر ايجاد كانالها و جداول مسير يابي ، اين مؤلفه براي مديريت مديران صفي استفاده مي شود كه ميزبان نماينده هاي كانال پيام هستند. از يك نظر، اين روش مديريت فوقاني بسيار شبيه مديريت سرورهاي خوشه اي در مواردي است كه فقط از يك سرور مديريت اجرايي استفاده مي شود. در اين حالت، سرور فقط يك پوسته دور[591] براي هريك از ماشين هاي داخل خوشه به همراه چند عمليات جمعي معدود جهت هدايت گروه هاي ماشين ها ارايه مي كند. با اين وجود، چنانچه به دنبال حوزه جديدي براي يافتن راهكار مشكلات جدي و مهم مي گرديد، كار برروي مديريت سيستم هاي توزيعي فرصت بسيار خوبي در اختيارتان خواهد گذاشت.

۴-۴-ارتباطات جريان گرا[592]

تا به اينجا، مسأله ارتباطات روي تبادل واحدهاي اطلاعات كمابيش مستقل و كامل تمركز يافته بود . ازنمونه هاي آن مي توان به درخواست احظار روال، پاسخ به درخواست احضار روال و تبادلات پيام بين برنامه هاي كاربردي از قبيل سيستم هاي صف بندي پيام اشاره كرد. ويژگي بارز اين نوع ارتباطات آن است كه نقطه زماني انجام ارتباطات كاملاً فاقد اهميت است. بعلاوه، سرعت عملكرد سيستم، خواه بسيار بالا و خواه بسيار پايين،كوچكترين تأثيري بر صحت كار نخواهد داشت.
اما در برخي انواع ارتباطات مسأله زمان نقش كليدي ايفا مي كند. بعنوان مثال تصور كنيد كه يك جريان صوتي به صورت رشته اي از نمونه هاي ۱۶ بيتي ساخته شده باشد كه هريك بيانگر دامنه موج صوت برحسب مدولاسيون كد پالس[593] (PCM) باشد. همچنين تصور كنيد كه اين جريان صوتي بيانگر كيفيت يك CD است، به اين معنا كه موج صوتي اصلي تحت فركانسHz ۴۴۱۰۰ نمونه گيري شده است. براي توليد صوت اصلي،لازم است كه نمونه هاي داخل جريان صوتي به همان ترتيب قرارگيري در جريان اجرا شوند ، اما فواصل زماني دقيقاً ۱/۴۴۱۰ ثانيه اي هم در نظر گرفته شود. اجرا تحت سرعتهاي مختلف، نسخه اشتباهي از صوت اصلي ايجاد خواهد كرد.
مسأله مورد بحث در اين بخش آن است كه يك سيستم توزيعي بايد چه امكاناتي جهت تبادل اطلاعات وابسته به زمان از قبيل جريانات صوتي و شكلي ارائه دهد. پروتكلهاي شبكه اي مختلف در ارتباط با ارتباطات جريان گرا در كتاب(۲۰۰۱، Halsall) بررسي شده اند (Steinmetz and Nahrstedt ،۲۰۰۴). تعريف كلي از مسائل چندرسانه اي ها مطرح كرده كه بخشي از آن در رابطه با ارتباط جريان گرا است .

[590] channel control function
[591] remote shell
[592] stream-oriented communication
[593] Pulse Code Modulation

پردازش پرس وجو روي جريانات داده اي درکتاب (Babcock ،۲۰۰۲) و گروه همکاران مورد بحث قرار گرفته است.

۱-۴-۴-پشتيباني از رسانه هاي پيوسته

پشتيباني از تبادل اطلاعات وابسته به زمان غالباً در قالب پشتيباني از رسانه هاي پيوسته مطرح مي شود. منظور از رسانه ابزارهايي است که جهت انتقال اطلاعات مورد استفاده قرار مي گيرد. اين ابزارها شامل رسانه ذخيره سازي و ارسال، رسانه ارائه از قبيل مانيتور و غيره است . يکي از مهم ترين انواع رسانه ها نحوه ***ارائه***اين اطلاعات يا به بيان ديگر، نحوه کدي کردن اطلاعات در يك سيستم کامپيوتري است . براي انواع مختلف اطلاعات از روشهاي ارائه مختلفي استفاده مي شود.به عنوان مثال، متن معمولاً به صورت ASCII يا يوني کد[۵۹۴] و تصاوير در قالبهاي مختلفي از جمله GIF يا JPEG کد مي شوند. جريانات صوتي را مي توان در يك سيستم کامپيوتري مثلاً با استفاده از PCM به صورت نمونه هاي ۱۶بيتي تبديل کرد.

در **رسانه هاي(ارائه) پيوسته**[۵۹۵] ، روابط زماني بين اقلام مختلف داده اي در تفسير درست معنا ومفهوم واقعي داده اهميت اساسي دارد. مثال مربوط به توليد موج صوتي بوسيله اجراي جريان صوتي را به خاطر بياوريد. بعنوان نمونه ديگر، مسأله حرکت را ذکر مي کنيم . حرکت را مي توان بوسيله مجموعه اي از تصاويري القاءکرد که بايد در فواصل زماني*T* يکنواخت (معمولاً ۴۰-۳۰ ميلي ثانيه براي هرشکل) نمايش داده شوند. نمايش صحيح علاوه بر نمايش عکس ها در ترتيب درست، مستلزم حفظ فرکانس ثابت *1/T* شکل در هر ثانيه است.

در **رسانه هاي (ارائه) گسسته**[۵۹۶]، برخلاف نوع پيوسته، روابط زماني بين اقلام داده اي اهميت اساسي را در تفسير درست داده ها *ندارند*. از نمونه رسانه هاي گسسته مي توان به نمايش متن و تصاوير ثابت و همچنين فايلهاي کد شيء و کد قابل اجرا اشاره کرد.

جريان داده اي

غالباً سيستم هاي توزيعي جهت تبادل اطلاعات وابسته به زمان از **جريانات داده اي**[۵۹۷] پشتيباني مي کنند.جريان دادهاي در واقع رشته اي از واحدهاي داده است .جريانات داده اي را مي توان در هردو نوع رسانه هاي پيوسته و گسسته مورد استفاده قرار داد. از نمونه هاي جريانات داده اي گسسته(بايت گرا) مي توان به لوله هاي UNIX[۵۹۸]يا اتصالات TCP/IP اشاره کرد. اجراي فايلهاي صوتي غالباً نيازمند ايجاد جريان داده اي پيوسته بين فايل و وسيله صوتي است.

مسأله زمانبندي در جريانات دادهاي پيوسته نقش کليدي ايفا مي کند. براي داشتن ويژگي هاي زماني ،معمولاًموضوع تفاوت بين روشهاي

---

[594] unicode
[595] continuous (representation) media
[596] discrete (representation) media
[597] data streams
[598] UNIX pipes

مختلف انتقال مطرح مي شود. در **روش انتقال ناهمگام**[599] ، اقلام داده جريان يكي پس از ديگري ارسال مي شوند؛ اما محدوديت زماني ديگر وجود ندارد . اين ويژگي غالباً در جريانات داده اي گسسته هم صدق مي كند. بعنان مثال، يك فايل را مي توان به صورت جريان داده اي ارسال كرد. اما اين كه چه زماني هر قلم به پايان مي رسد ، فاقد اهميت است .

در **روش انتقال همگام**[600] ، به ازاي هر واحد در جريان داده يك حداكثر تأخير انتها به انتها[601] تعريف مي شود. انتقال واحد داده با سرعتي بسيار بيشتر از حداكثر تأخير تحمل شده فاقد اهميت است . بعنوان مثال، يك حسگر درجه حرارت تحت سرعت مشخصي نمونه گيري كرده و از طريق شبكه به اپراتور انتقال مي دهد. در اين حالت، ممكن است لازم باشد كه زمان انتشار انتها به انتها در شبكه حتماً كمتر از فاصله زماني بين برداشت نمونه ها باشد، اما انتشار نمونه ها تحت سرعتي بسيار بيشتر از سرعت لازم هم مشكل زا نخواهد بود.

در **روش تبادل تك زماني**[602] ، واحدهاي داده اي بايد درست سروقت ارسال شوند . به اين معني كه تبادل داده در معرض نوعي حداكثر و حداقل تأخير انتها به انتها قرار دارد كه به نام لرزش (تأخير) محدودشده معروف است. از آنجاييكه روش تبادل تك زماني نقش كليدي در ارائه شكل و صوت ايفا مي كند، خصوصاً در سيستم هاي توزيعي چندرسانه اي اهميت ويژه اي دارد. در اين فصل، فقط روي جريانات داده اي تك زماني با استفاده از مخابره تك زماني تمركز كرده و به همين دليل از اين پس فقط جرياناتناميده مي شود.

جريانات ممكن است ساده يا پيچيده باشند.**جريان ساده**[603] متشكل از فقط يك زنجيره داده است ، درحاليكه **جريان پيچيده**[604] متشكل از چندين جريان ساده مرتبط به نام زيرجريان[605] است. ارتباط بين زيرجريانات در يك جريان پيچيده غالباً وابسته به زمان نيز مي باشد. بعنوان مثال، صوت استريو را مي توان بوسيله يك جريان پيچيده متشكل از دو زير جرياني كه دايماً همگام مي شوند، انتقال داد . به بيان ديگر، واحدهاي داده هر جريان را بايد به صورت جفت در ارتباط انتقال داد تا تأثير استريو حفظ شود. نمونه ديگري از جريانات پيچيده انتقال فيلم است. چنين جرياني ممكن است متشكل از يك جريان شكلي بهمراه دو جريان جهت انتقال صداي فيلم به صورت استريو و جريان چهارم هم حاوي زيرنويسي براي افراد ناشنوا يا ترجمه به زباني غير از زبان شكل باشد. در اينجا هم همگام سازي زيرجريانات اهميت دارد. در صورت ناديده گرفتن همگام سازي، بازتوليد فيلم غيرممكن خواهد بود . بعداً به اين موضوع برمي گرديم.

---

[599] asynchronous transmission mode
[600] synchronous transmission mode
[601] end-to- end delay
[602] isochronous transmission mode
[603] simple stream
[604] complex stream
[605] substream

از نقطه نظر سيستم هاي توزيعي، پشتيباني از جريانات مستلزم وجود المانهاي مختلفي است . براي سهولت كار در اينجا روي پيوسته سازي داده هاي ذخيره شده در مقايسه با پيوسته سازي داده هاي زنده تمركز مي كنيم. در حالت اخير، داده در زمان واقعي گرفته شده و از طريق شبكه به گيرنده ها ارسال مي شود. تفاوت عمده بين اين دو آن است كه پيوسته سازي داده هاي زنده فرصت كمتري براي تنظيم جريان در اختيار قرار مي دهد. بنابراين با پيروي از Wu(۲۰۰۱) و گروه همكاران مي توان طرح كلي معماري مشتري- سرور جهت پشتيباني از جريانات پيوسته چندرسانه اي را مطابق شكل ۲۶-۴ ترسيم نمود.
معماري كلي ارائه شده چند مسأله مهم و قابل بحث را مطرح مي كند. در درجه اول، داده هاي چندرسانه اي، خصوصاً شكل و تا حدودي صوت ، بايد حدالامكان فشرده شوند تا بدين ترتيب ظرفيت ذخيره سازي لازم و خصوصاً ظرفيت شبكه كاهش يابد . نكته مهم تر از نقطه نظر ارتباطات ،كنترل كيفيت انتقال و مسائل همگام سازي است كه در بخش بعد مورد بحث قرار گرفته است.

شكل ۲۶-۴-معماري كلي پيوسته سازي داده هاي چندرسانه اي ذخيره شده در يك شبكه(داخل شكل:داده هاي چندرسانه اي فشرده شده،سرور چندرسانه اي،كنترل QoS،مشتري،همگام سازي جريان،رمزگشايي جريان،رمزگشايي جريان،كنترل QoS ،شبكه).

۲-۴-۴-جريانات و كيفيت سرويس

نياز هاي زماني(و ديگر نيازهاي غير كاركردي) غالباً به نام نيازهاي **كيفيت سرويس[۶۰۶](QoS)** خوانده مي شوند. اين نيازها بيان مي كنند كه مثلاً براي تأمين روابط زماني در يك رشته ، سيستم توزيعي و شبكه زيرين چگونه بايد باشند. QoS، در جريانات داده اي پيوسته عمدتاً روي نيازهاي متعهد به زمان بودن ، حجم و قابليت اطمينان تأكيد مي كند. در اين بخش، نگاه دقيقتري به QoS و نقش آن در ايجاد جريان خواهيم داشت.
راجع به نحوه بيان QoS موردنياز مطالب بسياري گفته شده است (بعنوان مثال مراجعه شود به Jin and Nahrstedt، 2004). اين نيازها در برنامه كاربردي، در بسياري از موارد تبديل به چند خصوصيت مهم به ترتيب زير ميشود(۲۰۰۱، Halsall):

۱. نرخ بيت لازم جهت انتقال داده.
۲. حداكثر تأخير تا زمان ايجاد جلسه(يعني هنگاميكه يك برنامه كاربردي قادر به آغاز ارسال داده شود).
۳. حداكثر تاخير انتها به انتها (يعني چه مدت طول مي كشد تا يك واحد داده بتواند به گيرنده برسد).
۴. واريانس حداكثر تأخير، يا لرزش[۶۰۷] تأخير .
۵. حداكثر تأخير سفر دوسويه[۶۰۸] .

لازم به ذكر است كه همانطور كه مثلاً در كتاب (۲۰۰۴، Steinmetz and Nahrstadt) هم توضيح داده شده است، بهبودهاي زيادي در اين مشخصات قابل اعمال هستند . با اين وجود، وقتي در مورد

[606] Quality of Service
[607] jitter
[608] round-trip

ارتباطات جريان گرا براساس محموله پروتكل اينترنت بحث مي كنيم ، بايد اين حقيقت را بپذيريم كه ارتباط بر اساس سرويس بسيار ساده اي است، يعني سرويس ديتاگرام بهترين تلاش: IP . هنگامي كه در اجرا به مشكلي برمي خوريم، يعني حالتي كه در اينترنت بسيار به چشم مي خورد، ويژگي IP به پياده سازي پروتكل امكان مي دهد تا هر زمان كه لازم بداند بسته ها را حذف كند . تمام يا بسياري از سيستم هاي توزيعي كه از ارتباطات جريان گرا پشتيباني مي كنند، اكنون برروي محموله پروتكلي اينترنت ساخته شده اند . بنابراين درمورد ويژگي هاي QoS ناگفته ها بسيار است (در واقع، هرچند IP تاحدودي از QoS پشتيباني مي كند، اما به ندرت پياده سازي مي شود).

اعمال QoS

با اين فرض كه سيستم زيربنايي فقط يك سرويس بهترين تلاش ارائه مي دهد ، سيستم توزيعي مي تواند **عدم** كيفيت سرويس را تا حدامكان مخفي نمايد. خوشبختانه، بدين منظور از مكانيزمهاي مختلفي مي توان استفاده نمود.

اولاً، وضعيت بهتر از چيزي است كه تا اينجا ترسيم شده است. بعنوان مثال، اينترنت بوسيله **سرويس هاي متمايز**[609] خود، ابزاري جهت تمايز بين دسته بندي هاي مختلف داده ايجاد مي كند. ميزبان فرستنده اساساً قادر است تا بسته هاي خروجي را به عنوان متعلقات يكي از انواع دسته بندي ها علامتگذاري كند، بعنوان نمونه گروه **ارسال پرشتاب**[610] مشخص مي كند كه يك بسته بايد بوسيله مسيرياب فعلي با در نظرگرفتن اولويت مطلق ارسال شود (David،۲۰۰۲و گروه همكاران). بعلاوه، در دسته بندي **ارسال تأييد شده**[611] ، ترافيك به چهار زيردسته و سه روش براي حذف بسته ها در صورت مسدود بودن شبكه، تقسيم مي شود . بنابراين ارسال تأييدشده به نحو مؤثري مجموعه اي از اولويتهاي قابل الصاق به بسته ها راتعريف كرده و به همين دليل، به برنامه هاي كاربردي امكان مي دهد تا بين بسته هاي حساس به زمان و بسته هاي غيرمهم تمايز قائل شوند.

علاوه بر اين راه حل هاي مربوط به شبكه، سيستم هاي توزيعي مي توانند در رساندن داده ها به گيرنده ها هم مفيد واقع شوند. هرچند معمولاً ابزارهاي زيادي براي اين منظور وجود ندارد، يك روش بسيار مفيد استفاده از بافر جهت كاهش لرزش است. اصل حاكم بر اين روش بسيار ساده بوده و در شكل ۲۷-۴ نمايش داده شده است. با اين فرض كه بسته ها در هنگام مخابره روي شبكه با واريانس خاصي تأخير مي كنند، توسط گيرنده تا حداكثر زمان ممكنه در بافر ذخيره مي شوند. اين امر به گيرنده امكان مي دهد تا با اين فرض كه همواره بسته هاي كافي به بافر وارد شده، بسته ها را تحت نرخ منظمي به برنامه كاربردي انتقال دهد و با همان نرخ پخش شود .

---

[609] differentiated services

[610] expedited forwarding

[611] assured forwarding

شكل ۲۷-۴- استفاده از بافر جهت كاهش لرزش (داخل شكل: بسته از منبع خارج ميشود،بسته به بافر مي رسد،بسته از بافر حذف مي شود،زمان در بافر ،شكاف در اجراي مجدد ،زمان بر حسب ثانيه).

البته ممكن است با اشكالاتي هم مواجه شويم، همانگونه كه در بسته شماره ۸ در شكل ۲۷-۴ ديده مي شود . ابعاد بافرگيرنده متناظر با ۹ ثانيه عبور بسته به برنامه كاربردي است. متأسفانه، رسيدن بسته شماره ۸ به گيرنده ۱۱ ثانيه طول كشيد كه در اين فاصله زماني بافر كاملاً تخليه شده است . اين امر باعث شكاف در اجراي مجدد برنامه كاربردي مي شود. تنها راه حل ممكنه افزايش ابعاد بافر مي باشد . نقطه ضعف آشكار آن است كه در اين صورت، تأخير آغاز پخش داده هاي موجود در بسته ها توسط برنامه كاربردي" دريافت" هم افزايش خواهد يافت .

روش هاي ديگري هم در اين زمينه وجود دارد. با توجه به اينكه در مورد يك سرويس زيربناي بهترين تلاش بحث مي كنيم ، امكان گم شدن بسته ها وجود دارد . جهت جبران اين نقيصه در كيفيت سرويس، بايد از تكنيكهاي تصحيح خطا استفاده كنيم(۲۰۰۰، Wah و گروه همكاران؛۱۹۹۸، Perkins و گروه همكاران). درخواست از فرستنده براي ارسال مجدد بسته مفقوده معمولاً امكان پذير نيست، بنابراين بايد از روش **تصحيح خطاي رو به جلو** [۶۱۲]**(FEC)** استفاده شود. يك روش معروف ،كدگذاري بسته خروجي به گونه اي است كه دريافت k بسته از n بسته براي ساخت k بسته صحيح كفايت كند.

مشكل هنگامي ايجاد مي شود كه يك بسته حاوي چندين قالب صوتي و تصويري باشد. در نتيجه هنگامي كه بسته اي مفقود مي شود، عملاً دريافت كننده در هنگام اجراي قالبها به شكاف بزرگي برخورد مي كند . اين مشكل را مي توان تا حدودي با قراردادن نوبتي قالبها ، مطابق شكل۲۸-۴ (ب) ، از بين برد . به اين ترتيب، در صورت مفقود شدن يك بسته، شكاف حاصله در ميان قالبهاي پي در پي در زمان توزيع مي شود . اما توجه داشته باشيد كه اين روش در قياس با روشهاي غيرنوبتي نيازمند بافر بزرگتري براي گيرنده است . بعنوان مثال، در شكل۲۸-۴ ،گيرنده براي اجراي چهار قالب اول بايد چهار بسته تحويل دهد، درحاليكه در روش ارسال غير نوبتي اين تعداد به يك كاهش مي يابد.

شكل ۲۸-۴-تاثير مفقود شدن بسته در الف)ارسال غيرنوبتي و ب)ارسال نوبتي(داخل شكل: الف)بسته مفقود شده،ارسال شده،تحويل داده شده،شكاف ناشي از قالب هاي مفقود شده،بسته مفقودشده ،ب) ارسال شده،تحويل داده شده،قالب هاي مفقود شده).

۳-۴-۴- همگام سازي جريان

يكي از مسائل مهم در ارتباط با سيستم هاي چندرسانه اي اين است كه جريانات مختلف كه احتمالاً به شكل جريان پيچيده هستند، به صورت دوطرفه همگام مي شوند. همگام سازي جريانات به معني حفظ روابط زماني بين جريانات است . دو نوع همگام سازي وجود دارد . ساده ترين شكل همگام سازي بين يك جريان داده اي گسسته و يك جريان داده اي پيوسته است . براي نمونه، نمايش يك اسلايد را روي وب در نظر بگيريد كه بوسيله صوت پشتيباني شده است. هر اسلايد به شكل يك جريان داده اي گسسته از سرور به مشتري انتقال

[612] forward error correction

مي يابد. در عين حال، مشتري بايد (بخش مشخصي از) جريان صوتي را اجرا كند كه با اسلايد حاضر هماهنگ بوده و از سرور هم پيش واكنش شده باشد . در اين حالت، جريان صوتي بايد با ارائه اسلايدها همگام شود.
نوع پركاربردتر همگام سازي بين جريانات داده اي پيوسته انجام مي شود . يكي از نمونه ها در زندگي روزمره نمايش فيلمي است كه در آن جريان شكلي بايد با صوت همگام شده و معمولاًبه نام همگام سازي لبي[613] خوانده مي شود . نمونه ديگري از همگام سازي ها، اجراي يك جريان صوتي استريو متشكل از دو زيرجريان –يك جريان به ازاي هر كانال- است . اجراي مناسب و موفقيت آميز مستلزم آن است كه هر دو زيرجريان كاملاً همگام باشند، بطوريكه وجود تفاوت بيش از µsec ۲۰ ممكن است باعث ازدست رفتن تأثير استريو شود.
همگام سازي در سطح واحدهاي داده اي تشكيل دهنده جريان رخ مي دهد . به بيان ديگر، دو جريان فقط در بين واحدهاي داده اي قابل همگام سازي هستند. درك واقعي يك واحد داده اي تا حد زيادي بستگي به سطح انتزاع انتخابي براي مشاهده جريان داده دارد . براي درك بهتر اين موضوع، مجدداً مثال جريان صوتي(تك كانالي) با كيفيت CD را بياد بياوريد. دربهترين حالت، اين نوع جريان به صورت رشته اي از نمونه هاي ۱۶ بيتي به نظر مي رسد. تحت فركانس نمونه گيري ۴۴۱۰۰ Hz ، همگام سازي با ديگر جريانات صوتي مي تواند ، در تئوري، تقريباً هر µsec ۲۳ رخ دهد . براي ارتقاء كيفيت استريو، همگام سازي در اين حد واقعاً ضروري است .
بااين وجود، هنگامي كه همگام سازي بين يك جريان صوتي و يك جريان تصويري را براي همگام سازي لبي در نظر مي گيريم، همگام سازي در گام هاي بلندتري انجام مي شود . همانطور كه قبلاً هم گفتيم، فريم هاي ويدئويي بايد تحت سرعت Hz ۲۵ يا بيشتر اجرا شوند . با انتخاب استاندارد پركاربرد NTSC ۲۹/۹۷ Hz ، مي توانيم نمونه هاي صوتي را در واحدهاي منطقي دسته بندي كنيم كه در طي زمان اجراي فريم ويدئو (msec ۳۳) ادامه پيدا كند. بنابراين ، تحت فركانس Hz ۴۴۱۰۰ نمونه گيري صوت ، ابعاد يك واحد داده صوتي مي تواند(با اين فرض كه هر نمونه ۱۶ بيت است)، به اندازه۱۴۷۰ نمونه يا ۱۱۷۶۰ بايت باشد. در عمل ، واحدهاي بزرگتر تا ۴۰ يا حتي msec ۸۰ هم امكان پذير است(۱۹۹۶، Steinmetz) .

مكانيزم هاي همگام سازي

اجازه دهيد در مورد نحوه انجام همگام سازي صحبت كنيم .لازم است دو موضوع را از هم تفكيك كنيم: **الف)**مكانيزم هاي اصلي جهت همگام سازي دو جريان و **ب)** توزيع مكانيزم ها در محيط شبكه .
مكانيزم هاي همگام سازي را مي توان در لايه هاي انتزاع متعدد و مختلف تصور نمود . در پايين ترين سطح، همگام سازي به صورت آشكار با عمليات روي واحدهاي داده اي جريانات ساده انجام مي پذيرد. اين مطلب در شكل ۲۹-۴ نمايش داده شده است . به طور خلاصه ، فرآيندي وجود دارد كه به سادگي عمليات خواندن و نوشتن

[613] lip synchronization

را روي جريانات ساده مختلف انجام داده و تضمين مي كند كه اين عمليات با محدوديتهاي همگام سازي و زمانبندي خاص مطابقت دارد.
شكل ۲۹-۴-اصل كلي همگام سازي آشكار روي واحدهاي داده سطح(داخل شكل:ماشين گيرنده،برنامه كاربردي،روال خواندن دو واحد داده صوتي به ازاي هر واحد داده شكلي،جريان ورودي،شبكه).
بعنوان مثال فيلمي را در نظر بگيريد كه به صورت دو جريان ورودي ارائه شده است. جريان شكل حاوي تصاوير كم كيفيت متراكم نشده۲۴۰×۳۲۰ پيكسلي است كه هر پيكسل با يك بايت كد شده و در نتيجه واحدهاي داده تصويري ۷۶۸۰۰ بايتي را بوجود آورده است . فرض كنيد كه قرار است تصاوير تحت ۳۰ Hz يا به صورت يك شكل در هر ۳۳ msec نمايش داده شود. فرض شده است كه جريان صوتي حاوي نمونه هاي صوتي است كه در گروه هاي واحدهاي ۱۱۷۶۰ بايتي قرار گرفته و مطابق شرح فوق، هر يك متناظر با ۳۳msec صوت مي باشد. چنانچه روال ورودي قادر به هدايت۲/۷ MB/sec باشد ، همگام سازي لبي به راحتي با اجراي نوبتي خواندن شكل و خواندن نمونه هاي بلوك صوتي هر ۳۳ ثانيه يكبار امكان پذير است.
نقطه ضعف روش آن است كه برنامه كاربردي به طور كامل مسئول اجراي همگام سازي مي باشد و در عين حال از امكانات سطح پايين براي اينكار برخوردار است. روش بهتر ارائه رابطي براي برنامه كاربردي است كه امكان كنترل بهتر جريانات و وسيله ها را فراهم آورد . به مثال قبل باز مي گرديم . فرض كنيد كه نمايش شكل رابط كنترلي دارد كه بوسيله آن مي توان نرخ نمايش شكل را تعيين نمود . بعلاوه ، رابط امكاناتي براي ثبت يك عمل كننده قابل تعريف بوسيله كاربر ارائه مي دهد كه به محض ورود k شكل جديد، فراخواني مي شود. وسيله صوتي هم رابطي مشابه همين ارائه مي دهد. با داشتن اين رابط هاي كنترلي، نويسنده برنامه كاربردي مي تواند يك برنامه ساده نظارتي متشكل از دو عمل كننده - يعني يكي به ازاي هر جريان- بنويسد ؛ ايندو با همكاري هم چك مي كند كه آيا جريان صوت و شكل به اندازه كافي همگام شده اند يا نه و در صورت نياز، سرعت اجراي واحدهاي صوت و شكل را هماهنگ مي كند.
مثال اخير در شكل ۳۰-۴ ارائه شده و نمونه اي از سيستم هاي ميان افزاري چندرسانه اي است. به طور خلاصه، ميان افزار چندرسانه اي مجموعه اي از رابط ها را براي كنترل جريانات صوتي و شكلي از قبيل رابط هاي مربوط به كنترل وسيله ها از قبيل مانيتورها، دوربين ها، ميكروفون ها و غيره ارائه مي دهد . هر وسيله و جريان رابط هاي سطح بالاي خاص خود از قبيل رابط هاي مربوط به مطلع كردن برنامه كاربردي در هنگام وقوع رخدادي خاص را دارد. از اين رابط ها بعداً جهت نوشتن عمل كننده ها براي همگام سازي جريان ها استفاده شده است. نمونه هايي از اين دست رابط ها در كتاب (۱۹۹۸) Stefani آمده است .
شكل ۳۰-۴- اصل كلي همگام سازي كه در رابط هاي سطح بالا پشتيباني مي شود(داخل شكل:ماشين گيرنده،برنامه كاربردي،برنامه كاربردي به ميان افزار مي گويد كه با جريانات ورودي چگونه برخورد كند،كنترل چندرسانه اي بخشي از ميان افزار است،لايه ميان افزار،جريان ورودي،شبكه).

توزيع مكانيزم هاي ميان افزاري مسأله قابل توجه ديگري است. اولاً، طرف دريافت كننده يك جريان پيشرفته كه از چندين زيرجريان نيازمند همگام سازي تشكيل شده ، بايد دقيقاً از وظيفه خود اطلاع داشته باشد. به بيان ديگر، *توصيف همگام سازي* كامل آن به صورت محلي موجود باشد. روش معمول اين است كه اطلاعات به صورت ضمني با مالتي پلكسي كردن[614] جريانات مختلف به يك جريان واحد كه شامل تمامي واحدهاي داده اي از جمله واحدهاي داده اي مربوط به همگام سازي باشد، انجام پذيرد.

در جريانات MPEG از روش اخير جهت همگام سازي استفاده مي شود . استانداردهايMPEG **(گروه متخصصين شكل متحرك)**[615]مجموعه اي از الگوريتم ها را براي فشرده سازي صوت و شكل ارائه مي دهد. استانداردهايMPEG متنوع هستند. بعنوان مثال، MPEG-2 اساساً براي فشرده سازي تصوير با كيفيت پخش از Mbps ۶ تا ۴ طراحي شده است. در اين شيوه، تعداد نامحدودي از جريانات پيوسته و گسسته را مي توان در يك جريان واحد قرار داد . هر جريان ورودي، ابتدا به جرياني از بسته ها تبديل مي شود كه براساس ساعت سيستم kHz ۹۰ ،مهر زماني قابل قابل حمل است . سپس اين جريانات به **جريان برنامه اي**[616] مالتي پلكس مي شوند كه متشكل از بسته هايي با طول متغير است ، ولي همگي داراي مبناي زماني واحدي هستند. طرف گيرنده با استفاده از مهرهاي زمان هر بسته بعنوان مكانيزم پايه براي همگام سازي بين جرياني، اقدام به جداسازي جريان ها مي كند.

مسأله مهم ديگر اين است كه آيا همگام سازي بايد در طرف گيرنده انجام شود يا در طرف فرستنده . چنانچه همگام سازي توسط فرستنده صورت پذيرد، جريان هاي داراي انواع مختلف واحدهاي داده اي را مي توان در يك جريان واحد قرار داد . مجدداً جريان صوتي استريو را در نظر بگيريد كه به ازاي هر كانال، از دو زير جريان تشكيل شده است.يك روش، انتقال مستقل هر جريان به گيرنده است و اينكه اجازه دهيم گيرنده، نمونه ها را به صورت جفت آگاه همگام كند. مشخص است كه چون ممكن است هر زيرجريان دچار تأخيرهاي مختلفي شود، احتمالاً همگام سازي به سختي انجام خواهد شد. روش بهتر، قرار دادن هردو زيرجريان در يك جريان در طرف فرستنده است. جريان حاصله شامل واحدهاي داده اي متشكل از جفت نمونه هاست كه به ازاي هر كانال يك نمونه وجود دارد . اكنون گيرنده بايد صرفاً واحد داده اي مورد نظر را خوانده و سپس آنرا بين جريان چپ و راست تقسيم كند. در اينصورت تأخير درهر دو كانال برابر خواهد بود.

۵-۴-ارتباط چند پخشي[617]

يكي از موضوعات مهم در ارتباطات سيستم هاي توزيعي ، پشتيباني از ارسال داده به چند گيرنده يا همان ارتباط چند پخشي است. سالهاست كه اين موضوع در حوزه پروتكل هاي شبكه قرار گرفته و پيشنهادات متعددي براي راه حل هاي سطح انتقال و سطح شبكه

[614] multiplexing
[615] Motion Picture Experts Group
[616] program stream
[617] multicast communication

پياده سازي و ارزيابي شده اند(Janic،۲۰۰۵؛ Obraczka،۱۹۹۸). يك مسأله مهم در تمامي راه حل ها ، ايجاد مسيرهاي ارتباطي جهت انتشار اطلاعات بوده است. در عمل، تحقق اين ايده مستلزم تلاش هاي مديريتي فراوان و در بسياري موارد ، دخالت هاي انساني بود . بعلاوه، مادامي كه تمامي پيشنهادات روي يك موضوع واحد همگرايي نداشته باشند، ISPها تمايل چنداني به پشتيباني از چندپخشي نخواهند داشت(Diot ،۲۰۰۰ وگروه همكاران).
همگام با ظهور فن آوري نظير به نظير و بخصوص ، مديريت فوقاني ساخت يافته ، راه براي ايجاد مسيرهاي ارتباطي هموارتر شد. ازآنجايي كه راه حل هاي نظير به نظير نوعاً در لايه كاربردي اجرا مي شوند، روشهاي چندپخشي سطح برنامه كاربردي متعددي معرفي شده اند. در اين بخش، نظري به اين تكنيك ها خواهيم داشت.
ارتباط چندپخشي به روشهاي ديگري غير از ايجاد مسيرهاي ارتباطي آشكار هم قابل انجام است. همانطور كه در اين بخش هم خواهيم ديد، انتشار اطلاعات براساس شايعه[۶۱۸] روشهاي ساده اي (هرچند غالباً با اثربخشي كمتر)را جهت چندپخشي ايجاد مي كند.

۱-۵-۴-چندپخشي در سطح كاربردي

ايده اصلي در چندپخشي سطح كاربردي اين است كه گره ها به شكل يك شبكه فوقاني سازماندهي شده و سپس جهت انتشار اطلاعات به اعضاي خود بكار برده شوند. يكي از مشاهدات مهم اين است كه مسيرياب هاي شبكه در عضويتهاي گروهي سهيم نمي شوند. در نتيجه، اتصالات بين گره ها در شبكه فوقاني ممكن است از پيوندهاي فيزيكي متعددي عبور كنند و به همين دليل، پيام هاي مسيريابي در داخل شبكه فوقاني ممكن است ، در قياس با آنچه بوسيله مسيردهي سطح شبكه بوجود مي آيد بهينه نباشند.
يكي از مسائل بسيار مهم در طراحي ، ساخت شبكه فوقاني است. به طور خلاصه، دو روش در اين رابطه وجود دارد(El-Sayed،۲۰۰۳). در حالت اول، ممكن است گره ها مستقيماً به شكل يك درخت سازماندهي شود كه در اينصورت فقط و فقط يك مسير(فوقاني) در بين هر جفت گره وجود خواهد داشت. در حالت دوم ممكن است گره ها به صورت يك شبكه مشبك[۶۱۹] سازماندهي شوند كه در آن ، هر گره داراي چندين همسايه بوده و در كل بين هر جفت گره چندين مسير وجود خواهد داشت . تفاوت عمده بين اين دو روش آن است كه حالت اخير مقاوم تر است؛ يعني چنانچه يكي از اتصالات بشكند (مثلاً در اثر خرابي گره) ، باز هم اين امكان وجود دارد كه بدون نياز به سازماندهي مجدد و فوري كل شبكه فوقاني، اقدام به انتشار اطلاعات نمود.
براي درك بهتر موضوع، اجازه دهيد طرح نسبتاً ساده تر ساخت درخت چندپخشي در Chord را در نظر بگيريم كه در فصل ۲ شرح داده شد. اين طرح كه براي اولين بار در Scribe (Castro،۲۰۰۲ و گروه همكاران) پيشنهاد شد، طرح چندپخشي سطح برنامه كاربردي است كه در بالاي Pastry (Rowstorn and Druschel،۲۰۰۱) ساخته شده است. Pastry يك سيستم نظير به نظير بر اساس DHT مي باشد.

---

[618] gossip-based
[619] mesh network

تصور کنید که گره مفروض قصد آغاز جلسه چندپخشي را داشته باشد. بدین منظور، یك شناسه چندپخشي ، مثلاً *mid* را ایجاد مي کند که صرفاً یك کلید ۱۶۰ بیتي تصادفاً انتخاب شده است. سپس *succ (mid)* را جستجو کرده که در واقع گره مسئول کلید موردنظر است و آنرا ارتقاء مي دهد تا تبدیل به ریشه درخت چندپخشي شودکه جهت ارسال داده ها به گره هاي مورد علاقه استفاده خواهد شد.گره *P* جهت پیوستن به درخت، فقط اقدام به اجراي عملیات LOOKUP(mid) مي نماید. به این ترتیب، یك پیغام جستجو با درخواست براي پیوستن به گروه چندپخشي *mid* از *P* به *succ (mid)* مسیردهي خواهد شد. همانطور که قبلاً هم گفته شد، راجع به خود الگوریتم مسیردهي به تفصیل در فصل ۵ صحبت خواهیم کرد.

درخواست پیوستن در طي مسیر خود به سمت ریشه، از گره هاي متعددي عبور خواهد کرد.فرض کنید که ابتدا به گره *Q* برسد. چنانچه *Q* تا پیش از این لحظه با درخواست پیوستن به *mid* روبرو نشده باشد، تبدیل به **پیش برنده**[620] براي آن گروه خواهد شد. در اینجا،*P* فرزند *Q* خواهد شد و *Q* به پیش بردن درخواست پیوستن به ریشه ادامه خواهد داد. چنانچه گره بعدي ریشه، مثلاً *R* هم پیش برنده گروه نباشد، تبدیل به پیش برنده شده و پس از ثبت *Q* بعنوان فرزند خود ، به پیش بردن درخواست پیوستن ادامه خواهد داد.

از طرف دیگر، چنانچه *Q* (یا*R*) از قبل پیش برنده *mid* باشد، آنهم پیش برنده قبلي (یعني به ترتیب،*P* یا *Q*) را بعنوان فرزند خود ثبت مي کند، اما دیگر نیازي به ارسال درخواست پیوستن به ریشه وجود نخواهد داشت؛ چون *Q* (یا *R*) از قبل عضو درخت چندپخشي هستند.

گره هایي از قبیل *P* که آشکارا درخواست پیوستن به درخت چندپخشي را نموده اند، بر اساس تعریف پیش برنده نیز خوانده مي شوند. نتیجه طرح این است که ما یك درخت چندپخشي در شبکه فوقاني با دو نوع گره ایجاد مي کنیم : گره هاي صرفاً پیش برنده که به عنوان کمك رسان[621] عمل مي کنند، و گره هایي که آنها هم پیش برنده هستند اما صراحتاً خواستار پیوستن به درخت هم مي باشند .

حال چندپخشي ساده مي شود: یك گره با اجراي مجدد عملیات LOOKUP(mid) ، صرفاً یك پیام چندپخشي را به ریشه درخت ارسال کرده و سپس مي توان این پیام را به تمام درخت گسیل نمود.

اشاره مي کنیم که تشریح سطح بالاي چندپخشي در Scribe شباهت چنداني به طرح اصلي آن ندارد. بنابراین از خوانندگان مشتاق درخواست مي شود براي مطالعه تفصیلي تر به کتاب Castro (۲۰۰۲) و گروه همکاران مراجعه کنند.

ساخت شبکه فوقاني

با توجه به شرح سطح بالاي فوق الذکر، به راحتي مي توان دریافت که هرچند ساخت درخت به تنهایي در حالي که گره ها به شکل شبکه فوقاني سازماندهي شده اند مشکل نیست، ولي ساخت درخت مفید و مؤثر مسأله اي جدا و متفاوت است. توجه داشته باشید که تا به

[620] forwarder
[621] helper

اينجا، براي انتخاب گره هايي كه در ساخت درخت مشاركت مي كنند، هيچ معيار كارآيي لحاظ نشده است. به اين معنا كه اينكار فقط بر اساس مسيردهي (منطقي) پيام ها از طريق شبكه فوقاني انجام شده است .

شكل ۳۱-۴-روابط بين پيوندها در يك شبكه فوقاني و مسيرهاي سطح شبكه اي واقعي(داخل شكل: ميزبان نهايي،مسيرياب،شبكه فوقاني، اينترنت).

براي درك مشكل مطرح شده، نگاهي به شكل ۳۱-۴ مي اندازيم كه در آن مجموعه كوچكي از چهار گره به شكل يك شبكه فوقاني ساده سازماندهي شده و گره A ريشه درخت چندپخشي را تشكيل مي دهد. هزينه هاي حركت در يك پيوند فيزيكي هم ارائه شده است . مشاهده مي كنيد كه هرزمانيكه A پيامي را به گره هاي ديگر چندپخشي مي كند، اين پيام دوبار از هريك از پيوندهاي$< B،Rb >$ ، $< Ra،Rb >$ ، $< Rc،Rd>$ و $< D،Rd >$ عبور خواهد كرد. در صورتي كه به جاي ايجاد پيوند شبكه فوقاني از B به D،پيوند را از A بهC ايجاد مي كرديم، كارآيي شبكه فوقاني افزايش مي يافت. تحت چنين پيكربندي ديگر نيازي به پيمودن دومرتبه اي پيوندهاي$< Ra،Rb >$ و $<Rc،Rd >$ وجود نخواهد داشت .

كيفيت درخت چندپخشي سطح برنامه كاربردي غالباً براساس سه معيار مختلف يعني فشار پيوند[۶۲۲]،كشيدگي[۶۲۳] و هزينه درخت[۶۲۴] برآورد مي شود. **فشار پيوند** به ازاي هر پيوند تعريف شده و شمارش مي كند كه يك بسته، چندبار از پيوند مورد نظر عبور خواهد كرد(۲۰۰۲، Chuو گروه همكاران). فشار پيوند بيشتر از يك به اين معناست كه هرچند يك بسته ممكن است در سطح منطقي در امتداد دو اتصال مختلف ارسال شود، بخشي از اين اتصالات ممكن است واقعاً متناظر با يك پيوند فيزيكي واحد بوده كه نمونه اي از آنرا در شكل ۳۱-۴ مشاهده كرديم .

كشيدگي يا **جريمه تأخير نسبي**[۶۲۵] **(RDP)** حاصل تقسيم تأخير بين دو گره در شبكه فوقاني بر تأخيري است كه آندو گره در شبكه زيرين تجربه خواهند كرد. بعنوان مثال، در شبكه فوقاني مفروض، پيام ها براي عبور از B بهC مسير$B\rightarrow Rb \rightarrow Ra\rightarrow Rc\rightarrow C$ راطي مي كند كه در مجموع ۵۹ واحد هزينه دارد. با اين وجود، پيام ها در شبكه زيرين در امتداد مسير $B\rightarrow Rb\rightarrow Rd\rightarrow Rc\rightarrow C$ با هزينه كلي ۴۷ واحد مسيردهي مي شوند كه در نهايت باعث ايجاد كشيدگي۱/۲۲۵ مي شود. مشخص است كه در ساخت شبكه هاي فوقاني، هدف به حداقل رساندن كشيدگي كلي يا به بيان ديگر، ميانگين RDP برآورد شده روي تمام جفت گره هاست .

در نهايت، **هزينه درخت** يك مقياس جهاني بوده و غالباً با به حداقل رساندن جمع هزينه هاي پيوند ارتباط دارد. بعنوان نمونه، در صورتي كه هزينه يك پيوند، تأخير بين دو گره نهايي آن فرض شود، آنگاه بهينه سازي هزينه درخت تبديل به پيداكردن درخت

---

[622] link stress
[623] stretch
[624] tree cost
[625] Relative Delay Penalty

فاصله زماني مينيمم[626] خواهد شد كه در آن، مجموع زمان انتشار اطلاعات به همه گره ها حداقل است.
براي ساده سازي مطالب ، فرض كنيد كه يك گروه چندپخشي داراي گره شناخته شده اي است كه گزارش گره هايي را نگهداري مي كند كه به درخت پيوسته اند. هنگامي كه يك گره جديد درخواست پيوستن مي دهد ، با اين **گره قرارملاقات**[627] تماس برقرار مي كند تا ليست(احتمالاً تقريبي) اعضا را بدست آورد. هدف از اينكار انتخاب بهترين عضوي است كه بتواند بعنوان ولي گره جديد در درخت عمل كند. اما چه كسي انتخاب خواهد شد؟ راههاي فراوان و پيشنهادات مختلفي وجود دارند كه غالباً راه حل هاي بسيار متفاوتي را پي مي گيرند.
بعنوان مثال، يك گروه چندپخشي با يك منبع واحد را درنظر بگيريد. در اين حالت، انتخاب بهترين گره به راحتي انجام خواهد شد: در واقع همان منبع است (چون در اين حالت مي توانيم مطمئن باشيم كه كشيدگي برابر با ۱ خواهد بود). به اين ترتيب ، يك توپولوژي ستاره اي مطرح كرده ايم كه منبع در ميان آن قرار گرفته است. اگر چه ساختار ساده اي است ، ولي به راحتي مي توان تصور كرد كه منبع دچار اضافه بار خواهد شد. به بيان ديگر، انتخاب گره غالباً به صورتي انجام خواهد شد كه فقط گره هايي انتخاب شوند كه k همسايه يا كمتر داشته باشند- k در واقع پارامتر طراحي است. اين محدوديت به شدت باعث پيچيدگي الگوريتم ايجاد درخت مي شود؛ چون ممكن است يكي از راه حل هاي مناسب پيشنهادي شامل پيكربندي مجدد درخت فعلي باشد.
Tan (۲۰۰۳) و گروه همكاران در كتاب خود مرور و ارزيابي جامعي بر راه حل هاي مختلف اين مشكل ارائه داده اند. براي درك بهتر، بياييد نگاه دقيق تري به يك خانواده خاص به نام **درختهاي تغيير**[628] (Helder and Jamin ،۲۰۰۲) بيندازيم . ايده اساسي بسيار ساده است. فرض كنيد كه اكنون يك درخت چندپخشي با يك منبع واحد بعنوان ريشه داريم. در اين درخت، گره P مي تواند با قطع پيوند ولي فعلي خود و ايجاد اتصال با گره ديگر، ولي را تغيير دهد. تنها محدوديت در تغيير پيوندها آن است كه ولي جديد ديگر هرگز نمي تواند عضو زيردرخت با ريشه P باشد(چون اينكار باعث بخش بندي درخت و ايجاد حلقه خواهد شد) و بعلاوه ولي جديد ديگر داراي فرزندان بلافصل زيادي نخواهد بود. مورد اخير لازم است تا بار پيام هاي ارسالي توسط هريك از گره ها محدود شود.
تصميم گيري راجع به تغيير ولي تابع ضوابط مختلفي است. يكي از ساده ترين آنها بهينه سازي مسير به سمت منبع است كه به نحو مؤثري، باعث به حداقل رساندن تأخير در هنگامي مي شود كه قرار است پيامي چندپخشي شود. بدين منظور، هر گره به صورت منظم اطلاعات روي گره هاي ديگر را دريافت مي كند(يكي از روشهاي خاص اينكار را ذيلاً شرح خواهيم داد). در اينجا، گره مي تواند بررسي كند كه آيا گره ديگر از نظر تأخير در طول مسير به سمت منبع

[626] minimal spanning tree
[627] rendezvous node
[628] switch trees

ولي بهتري خواهد بود و در صورت مثبت بودن، كار تغيير را آغاز خواهد كرد.
معيار ديگر اين است كه آيا تأخير به ولي احتمالي ديگر كمتر از ولي فعلي خواهد بود . اگر هر يك از گره ها اين ضابطه رارعايت كنند، آنگاه كل تأخير درخت حاصله در حالت ايده آل بايستي حداقل باشد. به بيان ديگر، اينكار نمونه اي از بهينه سازي هزينه درخت است كه در بالا شرح داده شد. با اين وجود، براي ساخت چنين درختي نيازمند اطلاعات بيشتري هستيم. اما همچنانكه مشاهده خواهيم كرد، همين طرح ساده ابتكاري منطقي است كه تخمين خوبي از يك درخت حداقل فاصله زماني ارائه مي دهد.
بعنوان نمونه، حالتي را درنظر بگيريد كه در آن گره P اطلاعات همسايگان ولي خود را دريافت مي كند. توجه داشته باشيد كه در اين حالت همسايه ها شامل جد P به همراه خواهر و برادرهاي ولي P مي باشد . بنابراين گره P مي تواند تأخيرهاي هر يك از اين گره ها را ارزيابي كرده و در نهايت، گره داراي كمترين تأخير مثلاً Q را بعنوان ولي جديد خود انتخاب كند. بدين منظور، درخواست تغيير را براي Q ارسال مي كند. جهت جلوگيري از ايجاد حلقه در اثر همزماني درخواست هاي تغيير، گره داراي درخواست تغيير معوق پردازش تمامي درخواستهاي وارده را رد خواهد كرد. در واقع، اينكار باعث مي شود كه فقط تغييرات كاملاً مستقل را بتوان بصورت همزمان اجراكرد. بعلاوه، P اطلاعات كافي در اختيار Q خواهد گذاشت تا Q بتواند نتيجه بگيرد كه هردو گره ولي واحدي داشته يا اينكه ، Q جد است.
يكي از مشكلات مهمي كه تا به اينجا بحثي از آن به ميان نيامد، خرابي گره[629] است. در مورد درخت هاي تغيير، يك راه حل ساده پيشنهاد مي شود: هروقت كه گرهي از خرابي ولي خود اطلاع پيدا كند، خود را به ريشه متصل مي كند. در اين حالت، پروتكل بهينه سازي مطابق معمول پيش خواهد رفت و در نهايت، گره را در نقطه مناسبي در درخت چندپخشي قرار خواهد داد. تجربيات ذكرشده در كتاب Helder and Jamin (۲۰۰۲) بيان مي كند كه درخت منتج واقعاً نزديك به درخت فاصله زماني مينيمم است.

۲-۵-۴- انتشار داده ها بر اساس شايعه

يكي از روشهاي فوق العاده مهم در انتشار اطلاعات براساس *رفتار اپيدميك*[630] انجام مي شود. بر اساس نحوه شيوع بيماري ها در بين انسان ها، محققان از مدتها پيش تحقيق برروي استفاده از تكنيك هاي ساده جهت توسعه و گسترش اطلاعات در سيستم هاي توزيعي پردامنه را آغاز نموده اند. هدف اصلي اين **پروتكل هاي اپيدميك**[631] پخش سريع اطلاعات در بين مجموعه هاي بزرگي از گره ها با استفاده از اطلاعات صرفاً محلي است. به بيان ديگر، براي هماهنگ سازي انتشار اطلاعات، هيچ جزء مركزي وجود ندارد.
جهت تشريح اصول كلي اين دست الگوريتم ها، فرض مي كنيم كه تمامي به روزآوري ها براي يك آيتم داده اي فقط در يك گره آغاز مي شود. به اين ترتيب، به راحتي مي توان از تداخل نوشتن- نوشتن

[629] node failure
[630] epidemic behaviour
[631] epidemic protocols

اجتناب كرد . مطلب زير براساس يك مقاله كلاسيك اثر Demers و گروه همكاران در سال ۱۹۸۷ درمورد الگوريتم هاي اپيدميك ارائه شده است. يكي از بررسي هاي اخير در مورد انتشار اطلاعات اپيدميك را مي توانيد در كتاب (۲۰۰۴) Eugsterv و گروه همكاران مطالعه كنيد.

مدل هاي انتشار اطلاعات

همانطور كه از نام آن بر مي آيد، الگوريتم هاي اپيدميك براساس تئوري اپيدمي ها استوار است كه به مطالعه شيوع بيماري هاي عفوني مي پردازد. در مورد سيستم هاي توزيعي پردامنه، اين الگوريتم ها به جاي پراكندن بيماري ها، باعث پراكندن اطلاعات مي شوند. تحقيق در مورد اپيدمي ها در سيستم هاي توزيعي با هدف كاملاً متفاوتي دنبال شده است. هرچند سازمان هاي بهداشتي نهايت تلاش خود را براي جلوگيري از شيوع بيماريهاي عفوني در بين گروه هاي بزرگ انساني انجام مي دهند، طراحان الگوريتم هاي اپيدميك در سيستم هاي توزيعي تلاش مي كنند تا تمامي گره هاي داراي اطلاعات جديد را با سرعت هرچه تمامتر "عفوني" كنند.

با استفاده از اصطلاح هاي مورد استفاده در بيماري هاي اپيدميك، گرهي كه بخشي از يك سيستم توزيعي محسوب مي شود، در صورت داشتن اطلاعاتي كه بخواهد به گره هاي ديگر انتشار يابد، **عفوني**[۶۳۲] و گرهي كه تا كنون اين اطلاعات را نديده است ، **در معرض**[۶۳۳] ناميده مي شود. در نهايت ، گره به روزآوري شده اي كه قصد يا توان توزيع داده هاي خود را ندارد، **حذف شده**[۶۳۴] ناميده مي شود. البته در اينجا فرض بر اين است كه مي توان مابين داده هاي قديم و جديد تفاوت قائل شد ، مثلاً به اين دليل كه داده ها مهر زماني خورده يا نسخه بندي شده اند . از اينرو گفته مي شود كه گره ها به روزآوري ها را هم توزيع مي كنند.

يكي از مدل هاي پخش معروف ، مدل **ضدآنتروپي**[۶۳۵] است. در اين مدل، يك گره P گره ديگرQ را به صورت تصادفي انتخاب كرده و سپس به روزآوري ها را با Q مبادله مي كند. سه روش براي تبادل به روزآوري ها وجود دارد:

۱. P صرفاً به روزآوري هاي خود را در Q هل مي دهد[۶۳۶] .
۲. P فقط به روزآوري هاي خود را از Q مي كشد[۶۳۷].
۳. P و Q به روزآوري هاي خود را براي هم مي فرستند(يعني روش هل دادن- كشيدن).

در زمان انتشار سريع به روزآوري ها ، تنها روش نامناسب از بين سه روش فوق ، به روزآوري هاي هل دادن است . براي درك موضوع به اين موارد توجه كنيد. اولاً) در يك روش صرفاً براساس هل دادن، به روزآوري ها فقط از طريق گره هاي عفوني قابل پخش است . با اين وجود، چنانچه تعداد گره هاي زيادي عفوني شوند، احتمال اينكه هريك گره در معرضي را انتخاب نمايد، نسبتاً ناچيز

---

632 infected
633 susceptible
634 removed
635 anti-entropy
636 push
637 pull

است. در نتيجه، اين احتمال وجود دارد كه يك گره خاص فقط به اين دليل كه توسط گره عفوني انتخاب نشده ، براي مدت زمان طولاني در معرض باقي بماند.
بالعكس، روش براساس كشيدن در صورتي بهترين كارآيي را خواهد داشت كه تعداد گره هاي زيادي عفوني شده باشند. در اين حالت، پخش به روزآوري ها اساساً بوسيله گره هاي در معرض انجام مي شود . احتمال زيادي وجود دارد كه چنين گرهي با يك گره عفوني تماس برقرار كرده و بعداً به روزآوري ها را درون خود كشيده و آن را هم عفوني كند.
مي توان نشان داد كه در صورت عفوني شدن فقط يك گره ، به روزآوري ها با استفاده از هر يك از اشكال ضدآنتروپي به سرعت در تمامي گره ها انتشار خواهند يافت ، هرچند روش كشيدن- هل دادن بازهم بهترين روش خواهد بود (Jelasity ،٢٠٠٥ وگروه همكاران). يك **دور**[٦٣٨] به صورت يك فاصله زماني تعريف مي شود كه در آن هر گره حداقل يك مرتبه براي اولين بار اقدام به تبادل به روزآوري ها با يك گره تصادفاً انتخاب شده ديگر مي نمايد. بنابراين مي توان نشان داد كه تعداد دورها براي انتشار به روزآوري مفروض در تمام گره ها $O(\log(N))$ طول مي كشد كه در آن N تعدادگره ها در سيستم است. در واقع بيان مي كند كه انتشار به روزآوري ها به سرعت و مهم تر از آن، بصورت مقياس پذير انجام مي شود.
يكي از نمونه هاي خاص اين روش **انتشار شايعه**[٦٣٩] يا به بيان ساده تر **شايعه پراكني**[٦٤٠] است . در اين روش اگر گره P به تازگي براي آيتم داده اي x به روزآوري شده باشد، با گره اختياري ديگر يعنيQ تماس برقرار كرده و تلاش مي كند تا به روزآوري را به Q هل دهد. اما اين امكان وجود دارد كه Q پيش از آن به وسيله گره ديگري به روزآوري شده باشد. در اين حالت، P ممكن است مثلاً با احتمال $1/k$ تمايل خود را به پخش بيشتر از دست دهد. به بيان ديگر، از آن به بعد حذف خواهد شد.
شايعه پراكني از بسياري جهات مشابه زندگي روزمره ماست. وقتي باب اخبار داغي براي مطرح كردن با ديگران داشته باشد، ممكن است با زنگ زدن به دوست خود آليس همه چيز را به او بگويد. آليس هم مثل باب ممكن است دوستان نزديكي داشته باشد كه بخواهد اخبار را به آنها بگويد . با اين وجود، ممكن است بعد از زنگ زدن به يكي از دوستان خود به نام چاك و اطلاع ازاينكه او قبلاً اخبار را شنيده است، از زنگ زدن به بقيه دوستان خود و خبررساني به آنها منصرف شود. چون شنيدن اخبار دست دوم هيچ لذتي براي كسي ندارد.
بنظر مي رسد شايعه پراكني يكي از بهترين راه ها براي پخش سريع اخبار باشد . با اين وجود، نمي توان تضمين كرد كه تمامي گره ها واقعاً به روزآوري شوند (Demers،١٩٨٧ و گروه همكاران). مي توان اثبات كرد كه در صورت تعدد گره هايي كه در اپيدميك ها مشاركت مي كنند، كسر s از گره ها كه از قافله به روزآوري عقب

[638] round
[639] rumor spreading
[640] gossiping

يا به بيان علمي، در معرض باقي مانده ، در معادله زير صدق مي كنند:

$$s=e^{-(k+1)(1-s)}$$

شكل ۳۲-۴ ، $ln(s)$ را بعنوان تابعي از k نمايش مي دهد. بعنوان مثال، اگر $ln(s) = -4.97$ و $k=4$ ، در آنصورت s كمتر از ۰,۰۰۷ خواهد بود ، به اين معنا كه كمتر از ۰/۷% گره ها در معرض باقي مي مانند. در هر صورت، براي تضمين به روزآوري قطعي اين گره ها هم نيازمند مقياس هاي خاصي هستيم. تركيب دو حالت ضدآنتروپي و شايعه پراكني اين آرزو را محقق مي كند.

شكل۳۲-۴- رابطه بين كسر s گره هاي به روزآوري نشده و پارامتر k در شايعه پراكني خالص. نمودار $ln(s)$ را به صورت تابعي از k نمايش مي دهد.

يكي از مزاياي عمده الگوريتم هاي اپيدميك ، مقياس پذيري[۶۴۱] آنهاست. اين ويژگي ناشي از اين واقعيت است كه تعداد همگامي هاي بين فرآيندها در قياس با ديگر روشهاي انتشار نسبتاً ناچيز است. Marzullo and Lin (۱۹۹۹) نشان داده اند كه بهتر است در سيستمهاي حوزه گسترده، توپولوژي واقعي شبكه را در نظرگرفت تا كسب نتايج بهتر تضمين شود. در روش پيشنهادي آنها ، گره هايي كه به تعداد بسيار معدودي از گره هاي ديگر متصل هستند، به احتمال نسبتاً زيادي تماس گرفته مي شوند. فرض زيربنايي آن است كه چنين گره هايي پل ارتباط با ديگر اجزاي دور شبكه هستند؛ بنابراين، بايد در اولين فرصت ممكنه با آنها تماس ايجاد نمود. اين روش اصطلاحاً به نام **شايعه پراكني جهت دار**[۶۴۲] خوانده شده و انواع مختلفي دارد.

مشكل فوق اشاره به فرض مهمي دارد كه در اغلب راه حل هاي اپيدميك صدق مي كند، به اين صورت كه يك گره مي تواند به صورت تصادفي گره ديگري را براي پراكندن شايعه به آن انتخاب كند . به اين معنا كه بطور كلي، كل مجموعه گره ها بايد براي تمام اعضا شناخته شده باشد. در سيستم هاي بزرگ، اين فرضيه هرگز صدق نمي كند.

خوشبختانه، لزومي به دراختيار داشتن چنين ليست كاملي از اعضا وجود ندارد. همانطور كه در فصل ۲ هم گفتيم، داشتن نماي تقريبي كه كمابيش دائماً به روزآوري مي شود، باعث سازماندهي مجموعه گره ها در يك گراف تصادفي خواهد شد. با به روزآوري مرتب نماي تقريبي هر گره، انتخاب تصادفي ديگر تبديل به امري ساده خواهد شد.

حذف داده ها

الگوريتم هاي اپيدميك براي انتشار به روزآوري ها بسيار مناسب هستند. با اين وجود، اثر جانبي عجيبي هم دارند: انتشار *حذف* يك آيتم داده اي مشكل مي شود. اين اشكال ناشي از اين واقعيت است كه حذف يك آيتم داده اي باعث تخريب تمامي اطلاعات روي آن آيتم مي شود. در نتيجه، با حذف يك آيتم داده اي از يك گره، گره نهايتاً كپي هاي قديمي از آيتم داده اي را دريافت كرده و آنها

[641] scalibility
[642] directional gossiping

را بعنوان به روزآوري روي چيزي كه قبلاً نداشته تفسير خواهد كرد.

ترفند اين است كه حذف يك آيتم داده اي را فقط بعنوان به يك روزآوري ديگر ثبت نموده و گزارشي از اين حذف نگهداري كرد. به اين ترتيب، كپي هاي قديمي جديد و تازه وارد محسوب نشده و صرفاً بعنوان نسخه هايي تلقي مي شوند كه بوسيله عمليات حذف به روزآوري شده اند. ثبت حذف توسط انتشار **گواهي هاي فوت**[۶۴۳] انجام مي شود.

البته مشكل گواهي هاي فوت آن است كه بايد در نهايت پاك شوند و در غيراينصورت، هر گره تدريجاً پايگاه داده اي محلي عظيمي از اطلاعات تاريخي درمورد آيتمهاي داده اي حذف شده ايجاد خواهد كرد كه ديگر قابل استفاده نخواهند بود. Demers (۱۹۸۷) و گروه همكاران پيشنهاد مي كنند كه از چيزي به نام گواهي هاي فوت خاموش[۶۴۴] استفاده شود. تمامي گواهي هاي فوت پس از ايجاد مهرزماني مي خورند. اگر بتوان تصور كرد كه به روزآوري ها تحت مدت زماني محدودي به همه گره ها پراكنده مي شوند، آنگاه مي توان گواهي هاي فوت را پس از گذشت مدت زمان حداكثر حذف نمود.

با اين وجود، براي تهيه تضمين هاي محكم درمورد پراكنده شدن واقعي حذف شده ها به همه گره ها ، تعداد بسيار معدودي از گره ها گواهي فوت خاموش دارند كه هرگز دور انداخته نمي شوند. فرض كنيد كه گره P به ازاي آيتم داده اي x چنين گواهي دارد. چنانچه تحت هر احتمالي به روزآوري قديميx به P برسد، آنگاه P صرفاً با پراكنده كردن گواهي فوت مجدد به ازاي x واكنش خواهد داد.

كاربردها

به عنوان خاتمه بحث بياييد نگاهي به برخي از كاربردهاي جالب پروتكل هاي اپيدميك بيندازيم . تا به اينجا انتشار به روزآوري ها را ذكر كرديم كه احتمالاً پركاربردترين موارد كاربرد محسوب مي شود. همچنين، در فصل ۲ شرح داديم كه چگونه ارائه اطلاعات مكان يابي راجع به گره ها مي تواند در ايجاد توپولوژي هاي خاص كمك كند. در همين راستا، از شايعه پراكني مي توان براي كشف گره هايي استفاده نمود كه چند پيوند خروجي حوزه وسيع دارند، و سپس از شايعه پراكني جهتدار به صورتي كه در بالا گفته شد، استفاده كرد.

يكي ديگر از موارد مهم كاربرد پروتكل هاي اپيدميك ، جمع آوري يا درواقع ، انباشتن اطلاعات است(۲۰۰۵، Jelasity و گروه همكاران). حالت فرضي تبادل اطلاعات زير را درنظر بگيريد. هر گره مفروض i ابتدا يك عدد اختياري مثلاً x_i را انتخاب مي كند. هنگاميكه گره i با گره j تماس برقرار مي كند ، هريك مقدار خود را به صورت زير به روزآوري مي كند:

$$x_i, x_j \leftarrow (x_i + x_j)/2$$

مشخص است كه پس از اين تبادل، i و j هردو مقدار واحدي خواهند داشت. در واقع، به راحتي مي توان مشاهده كرد كه در نهايت تمامي گره ها يك مقدار واحد يعني ميانگين تمامي مقاديراوليه را خواهند داشت. سرعت انتشار در اينجا هم نمايي است.

[643] death certificates
[644] dormant death certificate

محاسبه ميانگين چه كاربردي دارد؟ حالتي را در نظر بگيريد كه در آن تمامي گره هاي i، x_i را برابر با صفر قرار داده اند، به استثناي x_1 كه برابر با يك است .

$$if i = 1 \begin{cases} 1 \\ 0 \end{cases} \leftarrow x_i$$
$$\{if i \rangle 1$$

اگر N گره وجود داشته باشند، آنگاه نهايتاً ميانگين هر گره بصورت $1/N$ محاسبه خواهد شد. در نتيجه، هر گره i مي تواند ابعاد سيستم را براساس رابطه $1/x_i$ بدست آورد. از اين اطلاعات مي توان به تنهايي جهت تنظيم پوياي پارامترهاي مختلف سيستم استفاده نمود. بعنوان مثال، ابعاد نماي تقريبي[645] (يعني تعداد همسايگاني كه هر گره گزارش آنها را نگهداري مي كند) قاعدتاً بستگي به تعداد كل گره هاي مشاركت كننده خواهد داشت. اطلاع از اين رقم موجب مي شود تا يك گره بتواند به صورتي پويا ابعاد نماي تقريبي خود را تنظيم نمايد. در واقع، اين عامل را مي توان بعنوان يكي از ويژگي هاي خودمديريتي تلقي نمود.

محاسبه ميانگين ممكن است مخصوصاً هنگامي مشكل شود كه گره ها بطور منظم به سيستم پيوسته و آنرا ترك كنند . يك راه حل عملي براي اين مشكل ، معرفي نقاط عطف زماني[646] است. اگر فرض كنيم گره شماره ۱ ايستا باشد، ، هرازگاهي يك نقطه عطف زماني جديد را آغاز خواهد كرد. هنگاميكه گره i براي اولين بار با يك نقطه عطف زماني مواجه مي شود، متغير x_i خود را برابر با صفر قرار داده و مجدداً شروع به محاسبه ميانگين خواهد كرد.

البته، نتايج ديگر را هم مي توان محاسبه نمود. بعنوان مثال، به جاي آنكه گره ثابت (x_1) كار محاسبه ميانگين را آغاز كند، به راحتي مي توان گره تصادفي را با استفاده از رابطه زير انتخاب نمود. هرگره i در ابتدا x_i را روي عدد اختياري از فاصله يكسان مثلاً [۱ و ۰] قرار خواهد داد و همچنين آنرا بطور دائم بصورت m_i ذخيره مي كند. در اثر تبادل بين گره هاي i و j ، هر يك مقادير خود را بصورت زير تغيير مي دهند :

$$x_i , x_j \leftarrow max(x_i , x_j)$$

هر گره i كه براي آن $m_i < x_i$ باشد، در رقابت برسر آغازگر بودن جهت شروع محاسبه ميانگين بازنده خواهد بود. در پايان، فقط يك برنده وجود خواهد داشت . البته، هرچند به راحتي مي توان راجع به بازنده شدن يك گره نتيجه گيري كرد، اما تصميم گيري درباره برنده شدن آن بسيار دشوارتر است، زيرا نمي توان مطمئن بود كه تمامي نتايج وارد شده اند . راه حل اين است كه خوش بين باشيم: يعني يك گره هميشه خود را برنده بداند مگر اينكه خلاف آن ثابت شود. در اينحال فقط بايد متغيري را كه براي محاسبه ميانگين استفاده نموده، مجدداً روي صفر تنظيم كند. توجه داشته باشيد كه

[645] partial view
[646] epoch

ممکن است تا به اینجا محاسبات مختلف و متعددي (که در مثال ما محاسبه حداکثر و محاسبه میانگین است) به طور همروند درحال اجرا باشند.

۴-۶ – خلاصه

در اختیارداشتن امکانات قدرتمند و انعطاف پذیر براي ارتباطات بین فرآیندها براي تمامي سیستم هاي توزیعي نوعي ضرورت محسوب مي شود. در کاربردهاي متعارف شبکه، ارتباطات غالباً بر اساس عملیات اولیه سطح پایین عبور پیام استوار است که بوسیله لایه انتقال ارائه مي شوند. یکي از مسائل مهم در سیستم هاي میان افزاري، پیشنهاد سطح بالاتري از انتزاع است. این امر باعث مي شود تا بیان ارتباطات بین فرآیندها در مقایسه با پشتیباني هاي ارائه شده براي رابط لایه انتقال آسان تر شود.

یکي از پرکاربردترین انتزاعات مورداستفاده ، فراخواني روال از راه دور(RPC) است. RPC بطور خلاصه به این معناست که یك سرویس بوسیله روالي پیاده سازي مي شود که بدنه آن در سرور اجرا مي شود. فقط امضاي روال یعني نام روال به همراه پارامترهاي آن، به مشتري ارائه مي شود. هنگامي که مشتري روال را فراخواني مي کند، پیاده سازي طرف مشتري که مجازي نامیده مي شود، اقدام به بسته بندي مقادیر پارامتر به شکل یك پیام و ارسال آن به سرور مي کند. این سرور، روال واقعي را فراخواني کرده و نتایج را مجدداً به شکل پیام بازمي گرداند. مجازي مشتري مقادیر نتیجه را از پیام بازگشتي استخراج کرده و آنرا به برنامه کاربردي مشتري در حال فراخواني باز مي گرداند.

RPCها امکانات ارتباطي همگام را ارائه مي کنند که بوسیله آن ، مشتري تا زمان ارسال پاسخ بوسیله سرور مسدود مي شود. هرچند انواعي از مکانیزم ها وجود دارند که باعث انعطاف پذیري این مدل مي شوند، مي توان نتیجه گرفت که مدل هاي عام منظوره سطح بالاي پیام گرا غالباً مناسب تر هستند.

در مدل هاي پیام گرا، مسأله مهم پایداري یا عدم پایداري ارتباطات و همچنین همگام و ناهمگام بودن آنهاست . بطور خلاصه ، ارتباط پایدار به این معناست که پیامي که براي مخابره تحویل داده شده، تا زمان تحویل بوسیله سیستم ارتباطي ذخیره خواهد شد. به بیان دیگر، براي موفقیت کار تبادل، لازم نیست که گیرنده و فرستنده سرپا و در حال اجرا باشند. در ارتباط ناپایدار ، هیچ نوع امکانات ذخیره سازي ارائه نمي شود و گیرنده باید آمادگي داشته باشد تا پیام ها را به محض ارسال آنها بپذیرد.

در ارتباط ناهمگام ، به فرستنده امکان داده مي شود تا فوراً پس از تحویل پیام براي ارسال ، و احتمالاً حتي پیش از ارسال آن ، به کار ادامه دهد . در حالي که در ارتباط همگام ، فرستنده حداقل تا زمان دریافت پیام، مسدود مي شود. در روش دیگر، فرستنده ممکن است تا زمان وقوع تحویل پیام و یا حتي تازمانیکه گیرنده، مشابه RPCها، واکنش نشان دهد، مسدود شود.

مدل هاي میان افزاري پیام گرا غالباً ارتباط ناهمگام پایدار ارائه داده و در مواردي استفاده دارند که RPCها دیگر مفید نباشند. از این مدل ها غالباً جهت کمك به تلفیق مجموعه هاي (بسیار پراکنده) پایگاه هاي داده اي به شکل سیستم هاي اطلاعاتي

پردامنه استفاده مي شود. كاربردهاي ديگر مدل هاي ميان افزاري شامل پست الكترونيك و جريان كار مي باشد.

يكي از اشكال بسيار متفاوت ارتباطات ، جريان سازي است كه در آن مسأله اين است كه آيا دو پيام متوالي رابطه زماني دارند يا نه. در جريانات داده اي پيوسته، به ازاي هر پيام يك تأخير انتها به انتهاي حداكثر تعريف مي شود. بعلاوه، لازم است كه پيامها براساس حداقل تأخير انتها به انتها ارسال شوند. از نمونه هاي اين نوع جريانات داده اي پيوسته مي توان به جريانات تصويري و صوتي اشاره كرد. تعيين دقيق روابط زماني يا انتظاري كه از زيرسيستم ارتباطات زيرين از نظر كيفيت خدمات مي رود، غالباً امري مشكل است. يكي از فاكتورهاي پيچيده نقش لرزش است. حتي در صورت قابل قبول بودن ميانگين كارآيي هم، تغييرات در زمان تحويل ممكن است منجر به غيرقابل قبول شدن كارآيي شود.

در نهايت اينكه ، يكي از مهم ترين انواع پروتكل هاي ارتباطي در سيستم هاي توزيعي، چندپخشي است. در اين پروتكل ها، ايده اصلي انتشار اطلاعات از يك فرستنده به چندين گيرنده است. راجع به دو پروتكل مختلف بحث نموديم. در حالت اول، چندپخشي با ايجاد يك درخت از فرستنده به گيرنده ها ايجاد شود. باتوجه به دانش كامل فعلي ما در مورد نحوه سازماندهي در سيستم هاي نظير به نظير ، راه حل هايي براي ايجاد پوياي درخت ها بصورت غيرمتمركز ايجاد شده است.

يكي ديگر از انواع مهم راه حل هاي انتشار مربوط به پروتكل هاي اپيدميك است. اين پروتكل ها هرچند بظاهر بسيار ساده مي نمايند، در واقع بسيار مقاوم هستند. پروتكل هاي اپيدميك علاوه بر انتشار پيام ها، به نحوي مفيد و مؤثر براي انباشت اطلاعات در يك سيستم توزيعي بزرگ هم مورد استفاده قرار گيرند.

مسائل فصل

۱- در بسياري از پروتكل هاي لايه اي، هر لايه سرپيام خاص خود را دارد. مسلماً بهتر است كه به جاي داشتن چندين سرپيام جداگانه، يك سرپيام واحد در هر پيام قرار داده شده و تمام كنترل ها داخل آن قرار گيرد. چرا اينطور نيست؟

۲- چرا سرويس هاي ارتباط لايه انتقال معمولاً براي ساختن برنامه هاي كاربردي توزيع شده نامناسب هستند؟

۳- يك سرويس چندپخشي مطمئن به فرستنده امكان مي دهد تا با اطمينان پيام مورد نظر را به مجموعه اي از گيرنده ها انتقال دهد. آيا چنين سيستمي متعلق به لايه ميان افزاري است يا بايد بخشي از يك لايه سطح پايين تر تلقي شود؟

۴- روال *incr* را با دو پارامتر عدد صحيح در نظر بگيريد. اين روال باعث اضافه شدن عدد يك به هرپارامتر مي شود. حال فرض كنيد كه با يك متغير واحد *incr (i، i)* دو مرتبه فراخواني شود. اگر *i* در ابتدا صفر باشد، پس از استفاده از فراخواني توسط مرجع چه مقداري خواهد داشت ؟ همين مسأله را با استفاده از كپي/بازيابي حل كنيد.

۵- در C ساختاري به نام union وجود دارد كه طي آن فيلد يك ركورد (كه در C ، struct ناميده مي شود) مي تواند هريك از چندين مقدار مختلف را در خود نگه دارد. در زمان اجرا، بطور

قطع نمي توان تعيين كرد كه كداميك موجود است . آيا اين ويژگي C اثراتي بر روال فراخواني از راه دور دارد؟ توضيح دهيد.
۶- يك روش براي انجام تبديلات پارامتر در سيستمهاي RPC آن است كه هر ماشين پارامترها را در قالب ارائه داخلي خود ارسال نموده و ماشين ديگر در صورت نياز ، كار تبديل را انجام دهد. سيستم داخلي را مي توان به كمك يك كد در بايت اول بيان كرد. با اين وجود، از آنجاييكه مكان يابي بايت اول در كلمه اول مشكل اصلي است ،آيا اين روش كار مي كند؟
۷- فرض كنيد كه يكي از مشتري ها ، RPC ناهمگامي به يك سرور را فراخواني مي كند و سپس منتظر باقي مي ماند تا سرور با استفاده از RPC غيرهمزمان ديگر نتيجه را عودت دهد. آيا اين روش مشابه آن است كه به مشتري اجازه دهيم يك RPC عادي را اجرا كند؟ در صورت تعويض RPC هاي ناهمگام با RPC هاي همگام چه اتفاقي خواهد افتاد؟
۸- به جاي اينكه به سرور اجازه دهيم كه خود را در محافظي ثبت كند، مانند آنچه در DCE انجام مي شود ، مي توان هميشه يك نقطه پاياني واحد را به آن نسبت داد. سپس از اين نقطه پاياني مي توان براي ارجاع به اشياء در فضاي آدرس سرور استفاده نمود. نقطه ضعف عمده اين طرح پيشنهادي چيست؟
۹- آيا تمايز قائل شدن بين RPC هاي ثابت و پويا مفيد خواهد بود؟
۱۰- توضيح دهيد كه در صورت استفاده از سوكت ها، ارتباطات بدون اتصال در بين مشتري و سرور به چه صورت انجام خواهد شد؟
۱۱- تفاوت بين عمليات اوليه MPI_bsend و MPI_isend درMPI را توضيح دهيد.
۱۲- فرض كنيد كه فقط از عمليات اوليه ارتباط ناهمگام ناپايدار شامل فقط يك عمل اوليه دريافت ناهمگام مي توانيد استفاده كنيد. عمل اوليه مربوط به ارتباط *همگام* ناپايدار را چگونه پياده سازي خواهيد كرد؟
۱۳- فرض كنيد كه فقط از عمليات اوليه ارتباط همگام ناپايدار مي توانيد استفاده كنيد. عمل اوليه مربوط به ارتباط *ناهمگام* ناپايدار را چگونه پياده سازي خواهيد كرد؟
۱۴- آيا اجراي ارتباط ناهمگام پايدار بوسيله RPCها امكانپذير هست؟
۱۵- در متن گفتيم كه براي شروع اتوماتيك فرآيندي براي آوردن پيام ها از يك صف ورودي، غالباً از محافظي استفاده مي شود كه بر صف ورودي نظارت مي كند. اجراي متفاوتي را ذكر كنيد كه در آن از محافظ استفاده نشود.
۱۶- جداول مسيردهي در IBM WebSphere و همچنين در بسياري از ديگر سيستم هاي صف بندي پيام، به صورت دستي پيكربندي مي شوند. روش ساده اي را براي انجام اتوماتيك اينكار ذكر كنيد.
۱۷- در ارتباط پايدار، گيرنده غالباً داراي بافر محلي است كه مي توان در صورتي كه گيرنده در حال اجرا نباشد، پيام ها را در آن ذخيره نمود. براي ايجاد چنين بافري ، بايد حتماً ابعاد آنرا هم تعيين و تخصيص نمود. درباره ترجيح دادن اين روش و همچنين برعليه مشخص ساختن اندازه بافر بحث كنيد .

۱۸- توضيح دهيد كه چرا ارتباط همگام ناپايدار داراي مشكلات ذاتي مقياس پذيري است و چه راه حلي براي آنها پيشنهاد مي كنيد؟
۱۹- نمونه اي ذكر كنيد كه در آن، چندپخشي براي پخش جريانات داده اي گسسته هم مفيد و مورد استفاده باشد.
۲۰- فرض كنيد كه در يك شبكه حسگر ، درجه حرارت هاي اندازه گيري شده بوسيله حسگر مهر زماني ندارند، اما فوراً به اپراتور ارسال مي شوند. آيا صرف تضمين تأخير انتها به انتهاي حداكثر كافي خواهد بود؟
۲۱- در صورتي كه مجموعه اي از كامپيوترها به شكل يك حلقه (منطقي يا فيزيكي) سازماندهي شده باشند، چگونه مي توان تأخير انتها به انتها حداكثر را تضمين نمود؟
۲۲- در صورتي كه مجموعه اي از كامپيوترها به شكل يك حلقه (منطقي يا فيزيكي) سازماندهي شده باشند، چگونه مي توان تأخير سربه سري حداقل را تضمين نمود؟
۲۳- باوجودي كه چندپخشي از نظر فني قابل اجراست ، حمايت چنداني از اجراي آن در اينترنت صورت نگرفته است. راه حل اين مشكل در مدلهاي شغلي حقيقي يافته مي شود. يعني اينكه هيچ كس نمي داند چگونه مي توان از چندپخشي بعنوان وسيله اي براي كسب درآمد استفاده كرد. چه طرحي پيشنهاد مي كنيد؟
۲۴- معمولاً، درخت هاي چندپخشي سطح برنامه كاربردي با توجه به كشيدگي بهينه سازي مي شوند كه براساس مقدار تأخيرها و تعداد پرشها اندازه گيري مي شود. موردي را ذكر كنيد كه استفاده از اين معيار سنجش مي تواند باعث ايجاد درخت هاي بسيار نامناسب شود.
۲۵- در هنگام جستجو براي فايل ها در يك سيستم نظير به نظير ساخت نيافته ، ممكن است محدودسازي جستجو به گره هاي داراي فايل هاي مشابه فايل هاي مورد نظر باعث تسهيل كار شود. توضيح دهيد كه شايعه پراكني چگونه مي تواند به پيداكردن اين گره ها كمك كند؟