بسم الله الرحمن الرحیم

اصول مهندسی کارایی نرم افزار

مسعود آراسته

نعیم اصفهانی

وحید رحیمیان

محمد هادی فروغمند اعرابی

تابستان ۱۳۸۵

این صفحه مخصوصا خالی رها شده است

چکیده

یکی از مباحث مهم در زمینه کارایی سیستم‌های کامپیوتری، توجه به اصول و تکنیک‌های مهندسی نرم‌افزار در طراحی سیستم می‌باشد. این تکنیک‌ها به ویژه در سیستم‌های بزرگ با معماری کارگزار–مشتری، نقش کلیدی را در عملکرد مناسب سیستم ایفا می‌نمایند. در این گزارش به بررسی این اصول و تکنیک‌ها پرداخته می‌شود.

پس از اشاره مختصری به اهمیت طراحی نرم‌افزار، به توصیف کارایی به عنوان یک هدف طراحی پرداخته خواهد شد. پس از آن چندین اصل و تکنیک بنیادی که برای کمینه کردن بار کاری، بالا بردن بازدهی، مکان‌دهی مناسب مولفه‌های نرم‌افزار، به اشتراک گذاری، و انجام پردازش موازی به کار گرفته می‌شوند مورد بررسی قرار خواهد گرفت. در انتها نیز اشاره مختصری به کاربردهای اصول ذکر شده در سیستم‌های با معماری کارگزار–مشتری خواهیم داشت.

این صفحه مخصوصا خالی رها شده است

فهرست مطالب

این صفحه مخصوصا خالی رها شده است

بخش ۱ کارایی در مهندسی نرم‌افزار

معمولا در مهندسی نرم‌افزار عاملی مانند کارایی به عنوان یک هدف دست دوم دیده می‌شود و مستقیما به آن پرداخته نمی‌شود؛ معمولا توصیه بر این بوده که اول سیستم را به گونه‌ای بنویس که کار کند، سپس تغییرش بده تا درست کار کند و در نهایت کارایی آن را بالا ببر. این در حالتی است که در برخی موارد این عامل را باید به عنوان یک عنصر اساسی دید و از همان اول طراحی در نظر گرفت وگرنه نرم‌افزار درستی طراحی نمی‌شود.

در این‌جا نرم‌افزار هم مثل سخت افزار می‌ماند؛ چطور است که اگر از ابتدا طراحی یک سیستم برای سرعت درگاه اطلاعاتی خاص صورت نگیرد پس از پیاده‌سازی با هر قدر تغییر هم به آن نمی‌رسیم و باید سیستم را از اول بسازیم، برای نرم‌افزار هم همین‌طور است. البته نرم‌افزار به این غیرقابل انعطافی نیست ولی عدم طراحی درست، هزینه‌های خاص خود را دارد.

این است که در برخی موارد کارایی را بعدی اساسی از معماری برخی نرم‌افزارها می‌بینند. در ادامه اصول اولیه‌ی ساخت سیستم‌ها، اصول مهندسی نرم‌افزار و مهندسی کارایی نرم‌افزار را می‌بینیم و ربط این اصول را به اختصار توضیح می‌دهیم. مهندسی نرم‌افزار در یک سطح منطقی و سطح بالا سعی بر راحت‌تر کردن بعضی عملیات استفاده‌ی مجدد و نگهداری دارد. این تصمیمات بر کارایی نرم‌افزار تاثیر دارد. در مهندسی کارایی نرم‌افزار اثرات فیزیکی این تصمیم‌گیری‌های سطح بالا را بررسی می‌کنیم.

۱–۱ اصول اولیه

یک سری از اصول اساسی در مهندسی نرم‌افزار مطرح است که اولویت بالایی دارند و اهمیت رعایت آن‌ها از هر اصلی بالاتر است. یکی از این اصول سادگی[1] و تمیزی یک نرم‌افزار است. اگر یک نرم‌افزار با معماری سازگار و ساده و تمیز داشته باشیم فهم کد و توانایی نگهداری آن را بالا برده‌ایم. طبق این اصل خصوصیات اضافی را که موضوعیتی در نرم‌افزار ندارند و تنها بر پیچیدگی آن می‌افزایند را حذف می‌کنیم و در نظر داریم که سادگی چیزی نیست که به صورت خودکار بدست آید بلکه باید برای آن انرژی گذاشت. سادگی در اثر مرور زمان و درگیری با اصول دیگر دچار چالش می‌شود. باید حواسمان جمع

[1] Simplicity

باشد که سادگی به خودی خود از اهمیت بالایی برخوردار است و آن را از دست ندهیم. سادگی باید در طراحی باشد و همان طراحی ساده را به صورت هوشمندانه پیاده سازی کرد. به طور مثال طراحی اولیه‌ی یک سیستم که از الگوریتم مرتب‌سازی استفاده می‌کند ربطی به پیاده‌سازی آن ندارد. در عین حال که می‌توانیم یک طراحی ساده داشته باشیم می‌توانیم یک پیاده‌سازی هوشمندانه داشته باشیم.

یکی دیگر از خصوصیات اولیه و اساسی در مهندسی نرم‌افزار کامل[1] بودن است؛ به این معنی که وقتی یک خصوصیت را پیاده سازی می‌کنیم باید آن را به صورت کامل پیاده‌سازی کنیم به گونه‌ای که در تمام محدوده‌ی تعیین شده عمل کرده و جوابگو باشد.

آخرین خصوصیت پایه‌ای صوری بودن[2] است. این اصل بسیار بدیهی است و هدفش این است که بیان کند کار یک سیستم نباید غیرقابل پیش‌بینی باشد بلکه باید یک پایه‌ی ریاضی داشته باشد تا بتوان بر روی آن حساب کرد. عملا نرم‌افزاری که عمل‌کرد آن مشخص نیست قابل اتکا نیست.

۱-۲ اصول و تکنیک‌های مهندسی نرم‌افزار

اصول مهندسی کارایی نرم‌افزار بر پایه‌ی اصول و تکنیک‌های مهندسی نرم‌افزار سوار می‌شوند و برای درک آن‌ها بررسی اصول و تکنیک‌های مهندسی نرم‌افزار گام ابتدایی است. بنابراین اصول و تکنیک‌های مهندسی نرم‌افزار را مرور می‌کنیم. البته اصول مهندسی نرم‌افزار نیز خود بر پایه‌ی اصول اولیه‌ی ذکر شده در بخش قبل بنا می‌شوند.

این اصول عبارتند از تجرید[3]، تجزیه[4] و اختفا[5]. در اصل تجرید می‌خواهیم قسمت‌های مهم از موضوع را ببینیم و جزئیات نامرتبط را از محدوده‌ی دیدمان خارج کنیم. به طور مثال در طراحی یک درخت از ابتدا تمام درخت را با تمام جزئیات نمی‌بینیم بلکه اول کلیتی از آن را می‌کشیم و به مرور زمان آن را تکمیل می‌کنیم.

---

[1] Completenes

[2] Formality

[3] Abstraction

[4] Decomposition

[5] Hiding

اصل تجزیه به ما کمک می‌کند که مساله را با شکستن به یک سری زیر مساله و حل تک تک آن‌ها حل کنیم. اگر دوباره همان مثال طراحی را در نظر بگیریم، برای کشیدن صورت اجزای مختلفی را در نظر می‌گیریم که ابتدا ارتباط کلی این اجزا (گوش، بینی و ...) را طراحی کرده و سپس هرکدام را جداگانه و با دقت بیش‌تر طراحی می‌کنیم.

اصل اختفا به ما می‌گوید این اجزا باید از هم مستقل بوده و از درون یک‌دیگر بی‌اطلاع باشند. این اصل را در دنیای واقعی به خوبی مشاهده می‌کنیم. در دنیا موجودات از درون هم مطلع نیستند و ارتباطات مشخصی با هم دارند و عملا درون خود را از موجودات دیگر مخفی می‌کنند. زبان‌های شیئ‌گرا از همین ایده‌ی اساسی الهام گرفته‌اند.

تکنیک‌های مهندسی نرم‌افزار که در سطح بعد و ادامه‌ی اصول مهندسی نرم‌افزار بیان می‌شوند عبارتند از بهبود[1]، استقلال[2] و تمرکز[3].

در تکنیک بهبود که از دو اصل تجرید و تجزیه استفاده می‌کند بحث بر سر این است که نرم‌افزار به گونه‌ای طراحی می‌شود که در مرور زمان کامل شده و بهبود می‌یابد. این نیازمند دو اصل گفته شده می‌باشد چون هم باید مساله را کلی دید و در سطح مجرد طراحی را انجام داد که به مرور زمان راه بر تکامل سیستم وجود داشته باشد. از طرفی سیستم باید حتما به قسمت‌های جدا از هم شکسته شود تا بتواند این‌گونه گسترش را داشته باشد.

در استقلال هدف ما این است که اجزای نرم‌افزار از هم جدا باشند و وابستگی بین یک‌دیگر نداشته باشند. همان‌طور که مشخص است این اصل بر پایه‌ی دو اصل تجزیه و اختفا بنا می‌شود. عملا برای این‌که دو قسمت نرم‌افزار از هم جدا باشند نیازمند این هستیم در ابتدا اجزایی وجود داشته باشند و سپس این قسمت‌ها از درون یک‌دیگر با خبر نباشند تا به هم وابسته نشوند. عملا این اجزا باید درون خود از هم‌دیگر مخفی کنند.

در نهایت تکنیک تمرکز که بر پایه‌ی تمام اصول و تکنیک‌های دیگر مطرح می‌شود می‌خواهد به این سمت برود که کل اجزای مرتبط با هم در یک گروه و در کنار هم قرار گیرند تا اصل هم‌محلی هم برقرار شود.

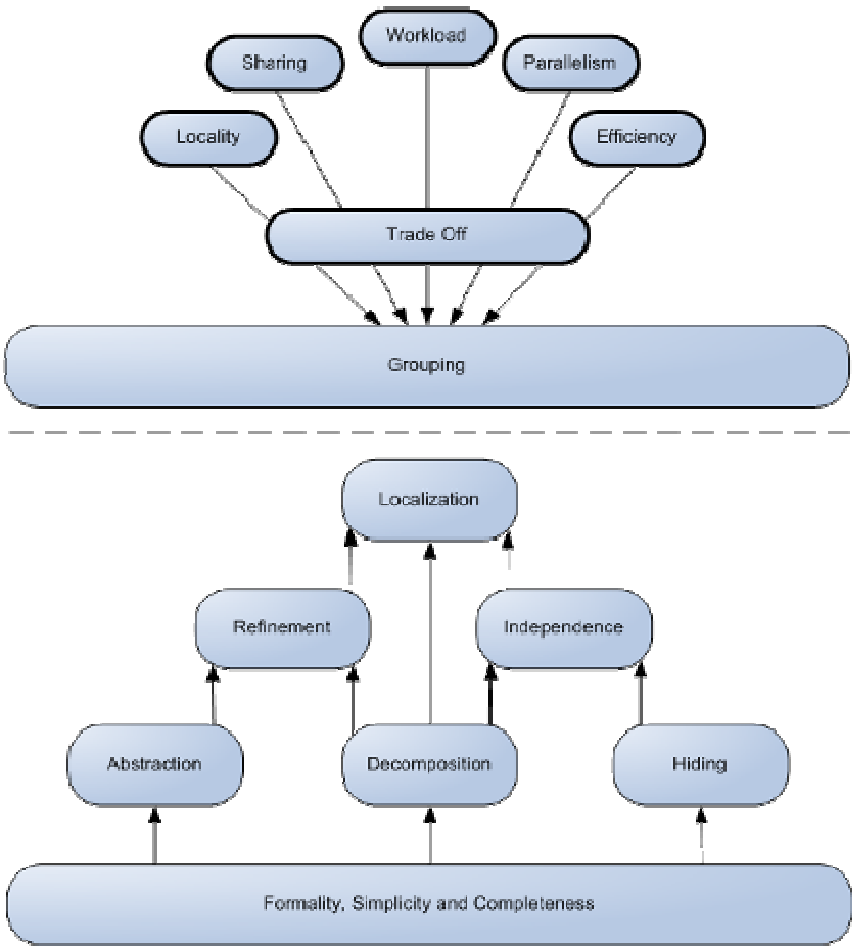
---

[1] Refinement

[2] Independence

[3] Localization

این اصل به گونه‌ای هم از لحاظ منطقی و هم از لحاظ فیزیکی محلی برای تامل دارد. در دید فیزیکی اگر چیزهای مرتبط در نزدیکی هم باشند، کارایی آن‌ها بسیار بالا می‌رود. دیگر هزینه‌های ما معطوف به خود کار می‌شود و در گیر هزینه‌های مربوط به جابجایی و ... نمی‌شویم. از طرف دید منطقی می‌خواهیم به یک سری از اهداف همان‌طوری که توضیح داده شد برسیم و برای تامین این اهداف کنار هم بودن اجزای یک بخش بسیار مناسب است. به طور مثال یکی از دلایل موفقیت تکنولوژی شیئ‌گرا اعمال همین اصل کنار هم بودن اجزای مرتبط است. عملا این اصل که از همه‌ی اصول دیگر تاثیر می‌پذیرد محل ارتباط بین اصول مهندسی نرم‌افزار و مهندسی کارایی نرم‌افزار است. در شکل زیر ارتباط اصول گفته شده و ارتباط آن‌ها با اصول مهندسی کارایی نرم‌افزار را می‌بینیم سپس اصول مهندسی کارایی نرم‌افزار را به طور مختصر معرفی می‌کنیم.



شکل ۱ – اصول مهندسی نرم‌افزار و مهندسی کارایی نرم‌افزار و ارتباط آن‌ها

۱–۳ ارتباط اصول کارایی و اصول نرم‌افزار

تصمیم‌هایی که در مهندسی کارایی نرم‌افزار گرفته می‌شود نیز در نهایت بر اصل تمرکز تاثیر شدیدی دارند. تاثیر این تصمیمات به دلیل تاثیراتی است که در گروه‌بندی می‌گذارند. اصولا کارایی به انتخاب درست گروه‌بندی وابسته است. اصول مهندسی نرم‌افزار باعث ایجاد ارتباطات منطقی در سطح طراحی می‌شوند که این ارتباطات در سطح پیاده‌سازی و کارایی به ارتباطات فیزیکی منجر می‌شوند. اصولا ارتباطاتی که از سطح بالاتر می‌آیند را نمی‌توان حذف کرد به عبارتی در طراحی فیزیکی و بحث کارایی اصولا این

ارتباطات و وابستگی‌ها زیادتر می‌شوند و از آن‌ها کاسته نمی‌شود. در مهندسی کارایی نرم‌افزار می‌خواهیم عناصر مرتبط به هم در کنار هم ارتباطی کارا با هم داشته باشند. به طور مثال در بحث طراحی پایگاه داده‌ها، اگر بخواهیم کارایی را در نظر بگیریم باید ستون‌های مرتبط‌تر را در کنار هم قرار دهیم تا جداولی بسازیم که کارایی پرسش و پاسخ روی پایگاه داده را بالاتر ببرد.

تناقضی که ممکن است در این‌جا رخ دهد این است که اصولا توصیه بر این است که طراحی از پیاده‌سازی جدا باشد تا هرکدام بتوانند به تنهایی تغییر کنند. این اصل تحت عنوان عدم وابستگی به یک سکوی[1] خاص مطرح می‌شود.

این حرف به طور کلی درست است. اما در زمان‌هایی که کارایی به صورت اساسی مورد نیاز است باید از ابتدا و در طراحی‌های منطقی آن را مد نظر داشته باشیم. به همین دلیل است که طراحی برای کارایی به ساختار زیرین مانند سیستم عامل وابسته است. حتی اگر خود سیستم عامل را در نظر بگیریم، می‌بینیم که همیشه قسمت‌هایی از آن با کد اسمبلی که شدیدا وابسته به سکو است نوشته می‌شود و هدف این کار نیز در نظر گرفتن کارایی است. پس دیده می‌شود که برای بعضی از نرم‌افزارها اصولا نمی‌توان این ارتباط را حذف کرد. عملا گفته می‌شود طراح منطقی یک طراح فیزیکی بی‌حوصله است. این امر به این موضوع اشاره می‌کند که درست است که طراح منطقی خود را درگیر جزئیات طراحی فیزیکی نمی‌کند ولی از ابعاد مختلف آن بی‌اطلاع نیست. به عبارتی این طراح خود با این جزئیات آشناست ولی تنها از دور دستی بر آتش دارد.

۱-۴ اصول کارایی نرم‌افزار

در ادامه به توضیح هرکدام از اصول کارایی مهندسی نرم‌افزار می‌پردازیم. این اصول را قبل از این در شکل ۱ مشاهده کردیم و در ادامه پس از توضیح مختصری از هر اصل، در بخش‌های بعدی هر اصل به صورت مجزا شرح داده می‌شود.

- اصل بار کاری

در این اصل سعی بر این است که بار کاری کل سیستم کمینه شود.

[1] Platform

- اصل کارایی

در این اصل می‌خواهیم نسبت کار مفید به سربارها را بیش‌تر کنیم.

- اصل تمرکز

در این اصل به در کنار هم قرار گرفتن و گروه‌بندی کارهای مرتبط می‌پردازیم.

- اصل به اشتراک گذاری

در این اصل می‌بینیم که چگونه می‌توان منابع را به اشتراک گذاشت ولی گلوگاهی[1] در سیستم ایجاد نکرد.

- اصل توازی

در این اصل می‌بینیم که کی مزایای موازی سازی بر هزینه‌های آن تفوق دارد و بهتر است از آن استفاده شود.

- اصل سبک سنگین کردن

در این اصل ارتباط اصول پیشین را بررسی می‌کنیم. مشخص است که اصول فوق در جاهایی باهم تناقض دارند و باید از هر کدام به نسبتی استفاده شود تا بهترین نتیجه‌ی ممکن را بدست آورد.

[1] Bottle-Neck

این صفحه مخصوصا خالی رها شده است

بخش ۲ اصل بار کاری[1]

بار کاری به صورت "هزینه در هر اجرا × تعداد اجراها" تعریف می‌شود. هزینه‌های واحد یا خرد هزینه‌ها هزینه‌هایی هستند که پایه‌ی هزینه‌های بعدی را تشکیل می‌دهند؛ عملا کارهایی که پایه‌های سیستم را تشکیل می‌دهند و معمولا توابع و چیزهای کوچکی هستند را شامل می‌شوند. به طور مثال مکانیزم دسترسی به فایل‌ها توسط سیستم عامل یا فراخوانی سرویس‌های درون سیستم‌عامل از این هزینه‌ها هستند. برای کاهش هزینه‌های اجرا باید از این هزینه‌ها شروع کرد. به عبارتی اگر مواظب پول خردها باشیم اسکناس‌ها مراقب خود هستند!!

در این‌جا تکرار لزوما به معنای یک حلقه نیست. منظور هزینه‌ای است که به صورت مکرر و چندباره به ما القا می‌شود؛ مثلا ورود و خروج تابع‌ها هزینه‌ای است که در کل سیستم پخش است و تکرار می‌شود. این عمل یک بار پیاده‌سازی شده ولی در کل سیستم بارها مورد استفاده قرار گرفته است. بنابراین تکرار استفاده از این قسمت از سیستم هم در هزینه‌ی کلی ما نقش دارد.

برای کاهش دادن هزینه‌ها ابتدا باید نرم‌افزار را بررسی کرد که خاصیت اضافی و زائد نداشته باشد. این خصوصیات می‌توانند در اثر زیاده‌روی در نرم‌افزاری کردن دامنه‌ی کاربرد وارد نرم‌افزار شده باشند. پس از حذف خصوصیات اضافی از نرم‌افزار، به مرحله‌ی فنی‌تر که همان کد است می‌رسیم؛ سعی می‌کنیم کدهای باقی‌مانده که عملیات را انجام می‌دهند را بهینه کنیم؛ عملا پس از حذف خصوصیات زائد برنامه‌ی مانده را برای کارایی تنظیم می‌کنیم.

۲–۱ حذف خصوصیات زائد

این خصوصیات اصولا خصوصیاتی را تشکیل می‌دهند که در اثر زیاده روی برای گسترش نرم‌افزار در سازمان وارد نرم‌افزار شده اند. به طور مثال لزومی ندارد که برای وارد کردن اطلاعات اولیه‌ی سیستم که یک‌بار انجام می‌شود از یک راه‌کار پردازش تصویر بر مبنای تشخیص حروف[2] استفاده شود. در این‌جا می‌توان از راه حل انسانی استفاده کرد و اطلاعات را به صورت دستی وارد کرد.

---

[1] Workload

[2] OCR

برای حذف خصوصیات زائد ابتدا باید آن‌ها را شناسایی کنیم. و برای این کار باید سئوالاتی مشابه سئوالات زیر را مطرح کرد:

- هدف خصوصیت چیست و چه سودی دارد ؟
- کی فعال می شود؟
- چند وقت یک بار فعال می‌شود؟
- مشکلی که در صورت نبودن آن رخ می‌دهد چیست؟
- منابع محاسباتی لازم برای آن چیست؟
- چه مقدار از منابع را مصرف می‌کند؟
- کار کامپیوتر در آن چیست؟

به این ترتیب می‌توان تشخیص داد که اصولا لزومی دارد که این خصیصه در داخل نرم‌افزار باشد یا می‌توان آن را خارج کرد و در حیطه‌ی نرم‌افزار در نظر نگرفت. دقیقا مثل همان مثال زده شده. وقتی می‌بینیم یک عمل فقط یک بار در کل عمر برنامه استفاده می‌شود و منابع مالی فراوانی را استفاده می‌کند و توسط انسان بهتر قابل انجام است، تصمیم می‌گیریم که آن را از حیطه‌ی نرم‌افزار خارج کنیم. این عمل همیشه به این راحتی قابل تشخیص نیست بلکه معمولا به بررسی بیش‌تر و سبک سنگین کردن نیاز دارد. اصولا بحث ما در این قسمت این است که باید ابتدا از تعریف سیستم شروع کرد و سیستم از آن‌جا بهینه شود و محدوده‌ی عمل آن تعیین گردد. عملا بهینه کردن قسمتی از نرم‌افزار که قرار نیست استفاده‌ی چندانی شود دور ریختن هزینه‌ها و منابع است. پس در این قسمت ابتدا محلی را که ارزشی چه از نظر عمل‌کردی و چی از نظر بهبود کارایی ندارد را شناخته و از تحلیل‌های بعدی خود کنار می‌گذاریم.

۲-۲ تنظیم برای سادگی و کارایی

پس از حذف خصوصیات زائد قسمتی از نرم‌افزار که می‌ماند را برای سادگی و کارایی تنظیم می‌کنیم. در این راستا ابتدا هزینه‌ی چیزهای بی‌خودی را از سیستم حذف می‌کنیم سپس تکرارها را بهینه می‌کنیم و در نهایت دسترسی‌ها به حافظه را بهبود می‌بخشیم.

چیزهای بی‌خودی عبارتند از مجموعه چیزهایی که بدون تاثیر در جواب هزینه‌ها را تغییر می‌دهند و بالا می‌برند. در این زمینه معمولا بهبودهای خیلی بدیهی می‌تواند به ما کمک کند. بحث‌های مربوط به درخط[1] تعریف کردن توابع در زبان‌های برنامه نویسی این‌جا کاربرد دارد. به این ترتیب برای استفاده از یک تابع کوچک دیگر لازم نیست هربار آن را صدا کرد بلکه کافی است که کد آن در قسمت صدا شده کپی شود. این نوع بهبودها اگرچه بدیهی هستند ممکن است در عین حال دچار مشکل شوند. اگر همین مثال توابع برخط را در نظر بگیریم، ممکن است زیاده روی در استفاده از آن باعث شود اندازه‌ی کد برنامه از حدی بیش‌تر شود و خود درگیر مکانیزم‌های صفحه‌بندی[2] و مدیریت حافظه‌ی سیستم عامل شود که هزینه‌ای اضافی را به سیستم تحمیل می‌کند. این نوع بهبودها معمولا به استانداردهای سازمان نیز می‌پردازند؛ به طور مثال عدم طراحی مناسب استانداردی برای نوع داده‌ها در پایگاه داده در سازمان باعث می‌شود نوع یک داده در هرکجای سیستم یک جور باشد و عملا برای کار کردن بین زیرسیستم‌ها باید به صورت مرتب تبدیل نوع انجام شود، حال آن‌که با یک طراحی مناسب می‌توان از ابتدا جلوی این مشکل را گرفت. در بعضی جاها هم این مشکلات از تنبلی برنامه‌نویس در سیستم می‌ماند مثلا برای گرفتن اطلاعات از پایگاه داده‌ها تنها اطلاعات مورد نیاز را باید گرفت و تمام اطلاعات را درخواست نکرد؛ اما برنامه نویسان معمولا در ابتدای کار خود همه‌ی اطلاعاتی که در مورد آن موضوع دارند (چه در آینده لازم شود و چه اصلا استفاده نشود) را از پایگاه می‌گیرند[3] و سپس کارهای خود را انجام می‌دهند. که این باعث تحمیل یک هزینه‌ی اضافی به سیستم می‌شود.

در ادامه باید تکرارها بهینه شوند. کوچکترین بهبودی در آن‌ها بسیار مهم است چون این بهبود در هزینه به تعداد انجام آن‌ها لحاظ شده و به سود ما می‌شود. هرچه تعداد تکرار یک امر بیش‌تر شود این بهبود کوچک تاثیر بیش‌تری خواهد داشت و عملا آن کار داغ‌تر می‌شود و کاندیدای بهتری برای بهبود است. به طور مثال اگر در پایگاه داده‌ها چندین جدول به تعداد زیادی با هم پیوند[4] زده می‌شوند به‌تر است که آن‌ها را غیر نرمال کرده تا از صرف هزینه پرهیز کنیم. البته بدیهی است که بدی‌های افزونگی حاصل را باید پذیرفت. معمولا این دست از بهبودها کوچک ولی هوشمندانه هستند و تاثیر بسیار زیاد و درخور توجهی

---

[1] Inline

[2] Paging

[3] Select *

[4] Join

در سیستم دارند. برای مثال اگر بهبودی در حد یک دهم هزینه‌ی موجود در زمان فراخوانی توابع، دسترسی به حافظه و بعضی عملیات پایه‌ای رخ دهد، اثر این بسیار بزرگ خواهد بود چون این یک دهم بهبود در کل تعداد استفاده‌ها ضرب می‌شود.

اصولا در هر بهبودی ما در گیر خوبی‌ها و بدی‌های آن هستیم که باید بین آن‌ها سبک سنگین کنیم. این توانایی معمولا از طریق تجربه کسب می‌شود. البته بعضی از این تجربه‌ها کلاسه بندی شده‌اند و به صورت یک سری تمرین خوب[1] در آمده‌اند. از آن جمله است همین بحث غیر نرمال‌سازی در پایگاه داده‌ها. باید توانایی این تحلیل‌ها و استفاده از اطلاعات موجود را نیز داشت.

معمولا هر وقت بحث مدیریت حافظه پیش می‌آید، ناخودآگاه ذهن به سمت عملیات نهان‌سازی[2] می‌رود که یکی از تکنیک‌های معروف در این زمینه است. بحث حافظه‌های ستون مبنا و سطر مبنا که معمولا هر مهندس نرم‌افزاری با آن روبرو شده است در این‌جا قابل مطرح شدن است. کافی است کل یک آرایه با ابعاد بزرگ را به دو ترتیب چاپ کنید. در روش اول ابتدا یک سطر را کامل چاپ کرده سپس به سراغ سطر بعد می‌رویم، در روش دوم همین کار را برای ستون‌ها انجام می‌دهیم. بر حسب این‌که مکانیزم مدیریت حافظه در پیاده‌سازی زبان چگونه باشد زمان اجرا فرق می‌کند. دلیل این امر این است که در پیاده سازی یا اطلاعات را به صورت سطری نگهداری می‌کنیم یا ستونی و در این‌جاست که بحث محلی بودن دسترسی‌ها به حافظه نیز مطرح می‌شود. عملا با هربار خواندن از حافظه یک بلوک از اطلاعات خوانده می‌شود که اگر در این بلوک همان اطلاعاتی باشد که ما در مرحله‌های بعد نیازمندیم ما در مراحل بعد دیگر هزینه‌ی خواندن از حافظه را پرداخت نمی‌کنیم.

۲-۳ برنامه‌های سازمانی[3] و محاسباتی

معمولا برنامه‌ها به دو رده تقسیم می‌شوند. نرم‌افزارهای محاسباتی که هدف در آن‌ها انجام بهینه‌ی یک محاسبه است و نرم‌افزارهای سازمانی که هدف در آن‌ها دسترسی به اطلاعات است.

---

[1] Best Practice

[2] Caching

[3] Enterprise

اصولا برای برنامه‌های محاسباتی حجم محاسبات است که کار اصلی سیستم را تشکیل می‌دهد و هدف پیدا کردن یک راه حل برای یک مشکل است، این برنامه‌ها معمولا اجرای طولانی مدت دارند تا به جواب برسند و کاربران آن معمولا همگن و کم هستند. در این سیستم‌ها هرگونه بهبود به ما کمک می‌کند که زمان اجرای نرم‌افزار را کاهش دهیم.

در مقابل برنامه‌های سازمانی بر پایه‌ی اطلاعات هستند و سعی بر مدیریت اطلاعات زیاد دارند. هدف اصلی در این برنامه‌ها امکان دسترسی به اطلاعات توسط کاربران می‌باشد و تراکنش‌های هر کاربر کوتاه و سریع است معمولا دسته‌ی کاربران این سیستم‌ها بسیار متنوع است و از نظر تعداد زیاد هستند. مشکل اساسی در این سیستم‌ها رفع نیازهای یک سازمان با در اختیار گذاشتن کنترل شده‌ی اطلاعات آن است. در ادامه در جدولی دو نوع سیستم را با هم مقایسه می‌کنیم:

	محاسباتی	سازمانی
نوع بار کاری	محاسبات زیاد	اطلاعات زیاد
هدف	جوابگویی به مشکلات	دسترسی به اطلاعات
مدت اجرا	اجرای طولانی مدت	تراکنش های کوتاه مدت
تعداد کاربران	کم، همگن	زیاد، متنوع
هدف عملیات	کاهش زمان اجرا	رسیدن به اهداف سازمان

جدول ۱ – مقایسه‌ی سیستم‌های محاسباتی و سازمانی

تاکید ما بیش‌تر بر نوعی از سیستم‌های سازمانی است که بر اساس معماری کارگزار و مشتری عمل می‌کنند. در این سیستم‌ها نوع خاصی از نگرانی‌ها و شرایط وجود دارند که با هم آن‌ها را بررسی می‌کنیم.

۲-۴ برنامه‌های سازمانی[1] بر اساس مدل کارگزار و مشتری

معمولا قسمت زیادی از این برنامه‌ها شامل کد خارج سازمان در قالب قسمت‌های خارجی است. به طور مثال میزبان برنامه‌ها[2] و پایگاه داده‌ها قسمت‌هایی هستند جدا از منطق اصلی برنامه ولی بسیاری از عملیات سیستم در قالب آن‌ها انجام می‌شود. برای این اجزا باید اطلاعاتی از کارایی‌شان داشت. معمولا این اطلاعات تحت قالب گزارش‌هایی منتشر می‌شود. در مورد باید بسیار جدی بود چون تا حد زیاد کارایی سیستم ما وابسته به کارایی این اجزاست. معمولا عدم انتشار گزارش کارایی یک محصول این چنینی دلیل بر مشکل داشتن آن سیستم است.

در این سیستم‌ها عملیات پایه‌ای بسیار پرهزینه هستند به طور مثال این عملیات پایه‌ای شامل دسترسی به پایگاه داده ها و انجام تراکنش‌هاست که به خودی خود هزینه‌ی بالایی دارند. بنابراین باید در استفاده از آن‌ها صرفه جویی بیش‌تری انجام داد.

این برنامه‌ها معمولا برنامه های محدود به ورودی و خروجی هستند نه پردازش (همان طور که در جدول ۱ هم مشاهده شد). پس عملیاتی که به گونه‌ای رد و بدل شدن اطلاعات را کم‌تر کند برای ما بهتر است. مثلا عملیات فشرده سازی اطلاعات در این سیستم‌ها می‌تواند بسیار به ما کمک کند.

منابع این برنامه‌ها معمولا منابع راه دور هستند و به دلیل تاخیر شبکه نسبت به حافظه محل فیزیکی منبع نیز در کارایی تاثیر زیادی دارد. مشابه این ملاحظات را در بحث مدیریت حافظه داشتیم ولی دامنه‌ی این بحث در این‌جا بسیار گسترده‌تر از آن بحث است.

از آن‌جا که کارگزار یک عنصر مشترک بین تعداد زیادی از کاربران است وابستگی به تنظیمات کارگزار بسیار بالا می‌رود. تعداد کاربران نقش مهمی را در این کارایی دارد. این در حالی است که در سیستم‌های محاسباتی تنها یک کاربر وجود دارد و سعی می‌شود همان نیاز کاربر بهینه شود ولی در این سیستم‌ها کارایی قسمت کاربر خیلی مهم نیست و کارگزار اهمیت بیش‌تری پیدا می‌کند.

---

[1] Enterprise

[2] Application Server

این سیستم‌ها از آن‌جا که کارهای متنوعی انجام می‌دهند کارایی نرم‌افزار را پایین‌تر می‌آورند. به طور مثال در یک کارگزار ممکن است در یک لحظه چندین نوع کد در حال اجرا باشد. در مقابل در سیستم‌های محاسباتی معمولا تنها یک نوع کار انجام می‌شود و می‌توان سیستم را برای همان یک نوع کار بهینه کرد.

۲-۵ اعمال اصل بار کاری

در اصل بارکاری معمولا گروه بندی‌های مختلف را در نظر می‌گیرند. به طور مثال درشت‌دانگی انجام عملیات توسط سیستم را در نظر می‌گیریم. برای هر نوع گروه‌بندی با توجه به نوع سیستم، یکی از انواع بدست آوردن بار کاری را انتخاب کرده، بار کاری در حالات مختلف را برای سیستم بدست می‌آوریم. تکنیک‌های بدست آوردن بار کاری می‌توانند از شبیه‌سازی باشند تا مدل کردن توسط شبکه‌های صف. در نهایت برای حالات مختلف با توجه به بار کاری بدست آمده تحلیل‌های لازم را انجام می‌دهیم و در نهایت یکی از گزینه‌ها را انتخاب می‌کنیم.

به طور مثال بای سادگی یک برنامه‌ی واسط کاربر را درنظر می‌گیریم. که خود بخش‌هایی دارد و هر بخش زیر بخش‌هایی دارد و ... در این سیستم اطلاعات هر زیر بخش را می‌توان در درشت دانگی‌های مختلف استخراج و بارگزاری کرد. در یک انتها می‌توان اطلاعات کل نرم‌افزار به یک باره استخراج کرد و وارد واسط کاربر کرد و در انتهای دیگر می‌توان اطلاعات هر کنترل را جداگانه و در موقع نیاز استخراج کرد. با توجه به انتخاب‌های مختلف هزینه‌های پایه و تکرار آن‌ها را بدست می‌آوریم و تحلیل می‌کنیم. یکی از راه حل‌ها از بقیه بهتر می‌شود و آن را انتخاب می‌کنیم. به طور مثال در جدول زیر این عملیات را برای یک سیستم انجام داده ایم.

نوع	روش پیاده سازی	تکرار	هر I/O	کل I/O
۱	بارگزاری تمام چیزها در ابتدا	۱	۱۵۰۰	۱۵۰۰
۲	بارگزاری وقتی وارد قسمت آن‌ها می شویم	۱,۵	۶۰۰	۹۰۰
۳	بارگزاری وقتی برنامه ی مربوطه استفاده می شود	۵	۱۸۰	۹۰۰

۴	بارگزاری وقتی مجموعه پنجره مربوطه باز شود	۱۰	۶۰	۶۰۰
۵	بارگزاری وقتی پنجره مربوطه باز می شود	۶۰	۲۰	۱۲۰۰

جدول ۲ – درشت دانگی‌های مختلف و هزینه‌ی آن‌ها در یک واسط کاربر

همان‌طور که مشاهده می‌شود ردیف چهارم با توجه به بارکاری های محاسبه شده بهترین انتخاب است. شایان ذکر است که این امر نشان می‌دهد که لزوما بهترین انتخاب در یکی از دو انتها نیست. اطلاعات در مورد بارهای کاری مختلف برای این نوع از تحلیل ضروری است و بدون آن نمی‌توان تحلیل را انجام داد.

تمام مباحث مطرح شده تا کنون به ما کمک می‌کنند که در تحلیل از نوع تحلیل جدول ۲ موفق‌تر عمل کنیم.

این صفحه مخصوصا خالی رها شده است

بخش ۳ اصل کارآیی

هر برنامه از بخش‌هایی تشکیل شده‌است که به هنگام اجرای برنامه تکرار می‌شوند. و هر قسمت از برنامه به منظور اجرا نیاز دارد عملیاتی به منظور آماده شدن برای اجرا انجام دهد. این عملیات هزینه دارد. بسیاری از اوقات هنگامی که چندین بار یک قسمت از برنامه اجرا می‌شود، قسمت آماده‌سازی نیز به همان تعداد اجرا می‌شود در حالی‌که می‌توان تنها یک بار قسمت آماده‌سازی را اجرا کرد. علاوه بر هزینه‌های آماده‌سازی گاهی اوقات بعد از اتمام کار هزینه‌هایی برای جمع کردن کار نیز پرداخته می‌شود که می‌توان آن‌ها را نیز به اشتراک گذاشت.

در این مورد مثال مشهوری وجود دارد که پخت شش کیک به صورت مجزا بسیار بیشتر از یک بار پخت شش کیک با هم به طول می‌انجامد. یعنی سرباری که هر دفعه برای آماده‌سازی محیط پخت کیک و تمیز کردن آن پس انجام عملیات فراهم می‌شود لازم نیست شش بار تکرار شود.

اصل کارآیی بدین معناست که برای بهتر کردن کارآیی نرم‌افزار می‌توان مقدار سرباری که برای انجام یک کار هدر می‌رود را کمینه کرد. برای مثال هزینه‌ی آماده‌سازی یا هزینه‌ی خاتمه از جمله سربارهای معمول هستند. پس

تعریف: اصل کارآیی یعنی کمینه‌کردن سربار نسبت به کار مفید.

۳–۱ فنون اصل کارآیی

به منظور اعمال اصل کارآیی می‌توان از چهار فن گروه‌بندی بهره برد. مهم‌ترین فنی که برای اصل کارآیی به کار می‌آید استفاده از گروه‌بندی است. در ادامه هر کدام از فنون اعمال اصل کارآیی توضیح داده خواهند شد. البته قبل ارائه‌ی این فنون مثالی از به کار گیری این اصل می‌آوریم تا فضای مطلب مشخص شود:

اطلاعات بر روی شبکه به صورت بسته‌های اطلاعاتی فرستاده می‌شوند؛ هر بسته برای ارسال سربار اطلاعاتی را به همراه خود می‌برد که شامل اطلاعات مورد نیاز لایه‌های شبکه برای شکستن، ارسال و شناخت خطای آن است. اگر احتمال خطا پایین باشد می‌توان این بسته‌ها را بزرگتر کرد تا نسبت سربار که اندازه‌ی ثابتی دارد به کل اطلاعات کم‌تر شود و استفاده‌ی ما از شبکه کاراتر باشد.

۳-۱-۱ گروه‌بندی ایستای مولفه‌های مشابه

ایستا به این معناست که قبل از زمان اجرا این گروه‌بندی طراحی شده و به هنگام اجرا قابل تغییر نیست. در این قسمت مشابه بودن را از نظر الگوی استفاده در نظر گرفته‌ایم. به این معنا که مولفه‌هایی از برنامه که به صورت معمول با یکدیگر استفاده می‌شوند را در کنار یکدیگر قرار دهیم. که این هزینه‌ی دسترسی به مولفه‌هایی که معمولا نیاز به دسترسی آن‌ها با یکدیگر است را سریع‌تر می‌کند.

برای مثال:

- واسط کاربر: در واسط کاربر می‌توان منوها و پنجره‌هایی که به صورت معمول با هم مورد استفاده قرار می‌گیرند را در یک پنجره یا به راحتی قابل دسترسی قرار دهیم. اگر برای انجام عملیات مشابه روی یک سری از اطلاعات، کاربر مجبور نباشد برای هر بار انجام عملیات کار اضافه‌ای انجام دهد عملا ما آن کارها را به گروه‌بندی برای او فاکتورگیری کرده‌ایم تا فقط یک بار انجام شود و به ازای هربار عملیات هزینه‌ی تکرار شونده را پرداخت نکند. این هزینه‌ی تکرار شونده در این‌جا همان گشتن در بین منوها و انتخاب کار مورد نظر است.
- ناهنجارسازی پایگاه داده: در شاخه‌ی پایگاه داده‌ها توصیه به هنجارسازی و نرمال کردن رابطه‌ها می‌باشد. به این ترتیب جلوی بسیاری از مشکلات از جمله افزونگی‌ها گرفته می‌شود. اما کار به همین‌جا ختم نمی‌شود، برای بازیابی اطلاعات مربوط به یک شخص باید بین چندین جدول هنجار شده ارتباط برقرار کرد و آن‌ها را پیوند زد تا به اطلاعات کامل دسترسی پیدا کرد. حال اگر این عمل تکرار شود عملا ما برای هربار دسترسی به مجموعه‌ای از اطلاعات هزینه‌ی سربار را می‌پردازیم. در مقابل می‌توان این داده‌هایی را که به صورت معمول با یکدیگر از پایگاه داده استخراج می‌شوند را در یک رابطه قرار دهیم و به صورت ایستا عملیات پیوند را که هربار اجرا می‌شود را انجام دهیم. در این صورت وقتی یک بلاک از داده از پایگاه داده دریافت شد تمام اطلاعات لازم برای پردازش داده‌ی مورد نظر دریافت شده و لازم به اتصال رابطه‌ها به یکدیگر نمی‌باشد. با این کار اطلاعاتی را که با هم نزدیکی و شباهت دارند را به صورت ایستا در قالب یک رابطه کنار هم قرار داده‌ایم. البته این بدان معنا نیست که خوبی‌های هنجارسازی را نادیده بگیریم بلکه تنها در جاهایی که کارایی مورد نیاز منوط به ناهنجار سازی است آن را انجام می‌دهیم.

- استفاده از مخزن داده[1]: برای نگهداری اطلاعات قسمت‌های برنامه‌های مختلف سازمان می‌توان به دو طریق عمل کرد. هر برنامه در کنار خود یک پایگاه داده‌ی کوچک داشته باشد و کارهای خود را انجام دهد و یا یک مخزن داده برای کل سازمان به راه انداخت که به طور جدا نگهداری شود. در این صورت دیگر نیاز نیست که هرکدام از پایگاه‌های داده‌ی جزئی نگهداری شوند بلکه کل اطلاعات در مخزن داده‌ها نگهداری می‌شود.

مشکل اصلی که در این فن وجود دارد ایجاد تبعیض بین الگوهای متفاوت استفاده از مولفه‌هاست. برای مثال در صورتی که دو برنامه متفاوت از یک پایگاه داده استفاده کنند، این ناهنجارسازی را تنها می‌توان با استفاده از الگوی استفاده‌ی برخی از آن برنامه‌ها انجام داد و این ناهنجارسازی به ضرر برنامه‌های دیگر با الگوی استفاده متفاوت خواهد بود. همان‌طور که مشخص است این جور تصمیم‌گیری‌ها مانند تیغ دولبه هستند که از یک طرف به افزایش کارایی کمک می‌کنند و از طرف دیگر آن را کاهش می‌دهند. در این بین باید با توجه به ساختار برنامه‌ها، حالتی را انتخاب کرد که در مجموع کارایی بهبود یافته و به نفع کل سیستم باشد.

۳-۱-۲ گروه‌بندی پویای مولفه‌های مشابه

در این فن مولفه‌های مشابه (باز هم از نظر الگوی استفاده) به هنگام اجرا تشخیص داده شده و در یک گروه قرار می‌گیرند. این فن معمولا زیاد استفاده می‌شود.

برای مثال:

- بافر کردن و نهان‌سازی: به هنگام اجرا نهان‌سازی داده‌ها بر اساس الگوی استفاده که به هنگام اجرا استخراج می‌شود قابل انجام است. مثلا در DNS ها این روش بسیار معمول است. با این کار دیگر لازم نیست هزینه‌ی سربار آوردن اطلاعات از روی دیسک و یا یافتن آدرس یک کامپیوتر را دوباره پرداخت کنیم. این موضوع در سیستم‌های پایگاه داده‌ها با نهان سازی پرسش‌های SQL، در مرورگرهای اینترنت تحت عنوان یک فایل که اطلاعات گرفته شده را نگهداری می‌کند و دوباره ارائه می‌دهد دیده می‌شود.

[1] Data Warehouse

- بسته‌بندی: وقتی یک پرسش SQL نتیجه‌ی بزرگی را دربرداشته باشد به ازای هر پیامی که بر روی شبکه فرستاده می‌شود هزینه‌ی سربار ثابتی خواهیم داشت. این هزینه‌ها معمولا در سه جا به ما تحمیل می‌شوند: هزینه‌ی پردازش بر روی کارگزار، هزینه‌ی انتقال بر روی شبکه و هزینه‌ی پردازش بر روی مشتری. از آن‌جا که این هزینه‌ها متناسب با تعداد پیغام‌های رد و بدل شده بر روی شبکه است، فرستادن بسته‌های بزرگ‌تر اطلاعات که باعث کم شدن تعداد بسته‌ها می‌شود باعث کم شدن این هزینه‌ی سربار می‌شود. به همین دلیل بعضی از پایگاه داده‌های مشهور این امکان را دارا می‌باشند.
- ارسال اطلاعات به محل عملیات: برخی مواقع به‌تر است اگر مشتری با اطلاعات مربوط به یک موجودیت زیاد کار دارد، اطلاعات را به سمت او فرستاد و پس از انجام عملیات اطلاعات از او پس گرفت. با این کار اطلاعات تنها دو بار جابه‌جا می‌شود و این کار زمانی مناسب است که قرار باشد اطلاعات چند بار رد و بدل شود. البته این کار مشکلات خودش را نیز دارد. از جمله‌ی این مشکلات بحث قفل کردن اطلاعات و مدیریت به روز رسانی اطلاعات است.

۳-۱-۳ گروه‌بندی ایستای مولفه‌های غیرمشابه

در این فن مولفه‌های غیر مشابه با یکدیگر قبل از اجرا در یک گروه قرار می‌گیرند. مولفه‌های غیر مشابه به معنی مولفه‌هایی است که از نظر ماهیتی با یکدیگر تفاوت دارند.

برای مثال:

- ترجمه‌ی کد: در هنگام ترجمه[1] وقتی کد و داده کنار یکدیگر و در یک گروه قرار می‌گیرند مثالی از این فن را تشکیل داده‌اند. ترجمه خود به چهار دسته تقسیم می‌شود: ترجمه‌ی ایستا[2]، ترجمه‌ی پویا[3]، تفسیر[4] و ترجمه‌ی در زمان نیاز[5]. در ترجمه‌ی ایستا کل کد را به یک

[1] Compile
[2] Static Compilation
[3] Dynamic Compilation
[4] Interpretation
[5] Just In Time Compilation

برنامه‌ی قابل اجرا تبدیل می‌کنیم، این برنامه را بر روی دیسک ذخیره کرده و در آینده آن را اجرا می‌کنیم. در ترجمه‌ی پویا این عمل در زمان اجرا انجام می‌شود و حداکثر کاری که انجام می‌شود این است که آن را نهان سازی می‌کند تا در صورت دسترسی در آینده‌ی نزدیک موجود باشد؛ در این حالت چیزی ذخیره نمی‌شود. در تفسیر برنامه‌ی مفسر خط به خط کد را در زمان تفسیر اجرا کرده و دیگر خبری از ترجمه نیست. در نهایت ترجمه‌ی در زمان نیاز که مخلوطی از روی‌کردهای ذکر شده است و جدیدا محبوبیت یافته کدهایی که تا یک مرحله ترجمه شده‌اند را در زمان اجرا تبدیل به کد قابل اجرا می‌کند. روشن است که در ترجمه‌ی ایستا تمام هزینه‌ها یک بار پرداخته می‌شود ولی تفسیر نیز خوبی خاص خود را دارد. با استفاده از تفسیر زمان پاسخ کدی که دائما در حال تغییر است را بسیار کم می‌کنیم.

- رویه‌های ذخیره‌شده در پایگاه داده: که این نیز نوعی ترجمه یا نیمه ترجمه است که می‌توان بهینه‌سازی‌های لازم را هنگام ترجمه و قبل از اجرا انجام داد. در این‌جا هم هزینه‌های لازم برای تبدیل کد را در همان اول پرداخته‌ایم و برخی تکه‌های ناهمگن را به صورت ایستا به هم متصل نموده‌ایم تا هربار از ترجمه و انتقال پرسش‌ها بر روی شبکه جلوگیری نماییم.
- پایگاه داده شیئ‌گرا: بهترین مثال برای این فن پایگاه داده‌ی شیئ‌گرا است. زیرا که در این پایگاه داده، داده‌هایی که به یک موضوع مربوطند ولی ماهیتا خصوصیات متفاوت آن شیئ را بیان می‌کنند در کنار یکدیگر قرار می‌گیرند. این روش دقیقا برعکس پایگاه‌داده‌ی رابطه‌ای است که داده‌های ماهیتا مشابه را که مربوط به اشیاء متفاوت است در یک گروه قرار می‌دهد. در پایگاه داده‌های شیئ‌گرا ما عمل ناهنجار سازی در سطح اشیا را انجام می‌دهیم و به این ترتیب اطلاعات ماهیتا ناهمگن ولی مرتبط در کنار هم به صورت ایستا قرار می‌گیرند.

۳-۱-۴ گروه‌بندی پویای مولفه‌های غیرمشابه

در این فن ما به ایجاد پویای رابطه‌ی موقتی بین مولفه‌های غیر مشابه می‌پردازیم. نمونه‌ی این کار را در کامپایلرهای جدید داریم و از آن به انعقاد تاخیری[1] یاد می‌کنیم.

برای مثال:

[1] Late Binding

- دروازه‌ی SQL و انتخاب مسیر دسترسی: برای اجرای پرسش SQL باید مسیر دسترسی به فایل‌ها و نحوه‌ی آن با توجه به اندازه‌ی جدول‌ها و خصوصیات دیگر موجود در کاتالوگ سیستم تعیین شود. برای این کار باید پرسش را به مسیر دسترسی تبدیل کرد. دو روش برای این کار موجود است. در روش پویا باید در زمان درخواست این تصمیم‌گیری انجام شود در حالی که در روش ایستا این پرسش تبدیل شده است که به سیستم می‌رسد و دیگر نیازی به تبدیل نیست. روش اول برای پرسش‌هایی که هرچند وقت یک بار توسط کاربر انجام می‌شود استفاده می‌شود حال آن‌که روش دوم برای پرسش‌هایی که مکررا و توسط برنامه‌ها پرسیده می‌شود مورد استفاده قرار می‌گیرد. برخی موارد پرسش ما این‌قدر پیچیده است که بین چندین پایگاه داده رد و بدل می‌شود برای این منظور از دروازه‌های SQL استفاده می‌شود. در این فن هنگام اجرای دستور SQL آن را دروازه‌ی SQL می‌دهند و واسط بر اساس داده‌ها و شناختی که از پایگاه‌های داده‌ی متفاوت دارد، در مورد گروه‌بند نحوه‌ی اجرای دستور در پایگاه‌های داده‌ی متفاوت تصمیم‌گیری می‌کند. در این تصمیم‌گیری عواملی وجود دارند که باعث می‌شوند این تصمیم ایستا باشد؛ به طور مثال وقتی بین دو پایگاه داده‌ها پیوند زده می‌شود مهم است که اطلاعات کدام یک به دیگری داده شود و این تنها وابسته به حجم اطلاعات خواهد بود که تنها در زمان اجرا مشخص است.
- وراثت پویا: در صورت استفاده از وراثت پویا مشکل چندین بار بارگذاری کد مربوط به اجداد اشیاء وجود دارد. برای حل این مشکل یکی از راه‌ها این است که کد مربوط به هر کدام از اجداد به همراه کد خود شیئ به عنوان کد اصلی در نظر گرفته شود و در زمان ترجمه به همراه آن ترجمه شود. در این حالت در صورت تغییر کد مربوط به پدر لازم است کد تمام فرزندان نیز دوباره ترجمه شده (علاوه بر کد پدر) بنابراین نمی‌توان در جاهایی که تغییر در کد زیاد است از این روش استفاده کرد. برای این منظور از روش‌های دیگر استفاده می‌کنند و پویایی ارتباط پدر و فرزند در ارث‌بری را بالا می‌برند. مثلا اگر یک سری کلاس برای خروجی دادن یک صفحه‌ی HTML داشته باشیم و لوگوی شرکت در کلاس پدر باشد با تغییر آن به صورت اتوماتیک و بلافاصله این لوگو در تمام صفحات تغییر خواهد کرد.

۳-۱-۵ فنون دیگر گروه‌بندی

به عنوان چند فن فرعی می‌توان از فنون زیر نام برد:

- تکرار جداول مرجع

در برخی از سیستم‌ها برای انجام برخی از کارها باید به یک جدول رجوع کرد تا مثلا تعرفه‌های مربوط به یک موضوع را دانست. این جدول‌ها معمولا ثابت هستند و در جاهای متعددی استفاده می‌شوند. برای بالا بردن کارایی سیستم توصیه می‌شود که این جداول در نزدیک دست کاربردان نگهداری شوند تا هربار هزینه‌ی سربار دریافت اطلاعات آن‌ها از کارگزار پرداخته نشود.

- نمایه‌گذاری[1]

یکی دیگر از کارهایی که در پایگاه‌های داده مرسوم است نمایه‌گذاری ستون‌هایی است که برپایه‌ی آن‌ها پرسش انجام می‌شود. به این ترتیب در زمان جستجو هزینه‌ی کم‌تری برای یافتن ردیف مورد نظر پرداخته می‌شود. در این تکنیک ما با صرف هزینه و زمان قبل از انجام جستجو و با فاکتورگیری هزینه‌های سربار قبل از پرسش آن را برای یک بار پرداخته‌ایم.

- پردازش دسته‌ای در مقابل برخط

ماهیت برخی از نرم‌افزارها برخط است ولی در مورد برنامه‌هایی که برخط نیستند می‌توان از پردازش دستورها به صورت دسته‌ای بهره برد. یک دسته از درخواست‌ها مانند مجموعه‌ای از درخواست‌های برخط تصادفی نیست بلکه معمولا درخواست‌های درون یک دسته نزدیکی‌های زیادی با هم دارند و می‌توان با ترکیب آن با مکانیزم‌هایی مانند نهان سازی کارایی خوبی را بدست می‌آوریم.

- تکرار داده

[1] Indexing

در سیستم‌های توزیع شده که چندین سیستم به صورت هماهنگ سعی در حل کردن یک مساله دارند معمولا باید به گونه‌ای اطلاعات را بین آن‌ها به اشتراک گذارد. یک راه استفاده از یک مرکز برای این به اشتراک گذاری است که به دلیل پایین آوردن تحمل‌پذیری خطا از آن پرهیز می‌شود. یکی از روش‌هایی که معمولا پیش گرفته می‌شود تکرار اطلاعات است. این روش از تکرار جداول مرجع کلی‌تر است و باعث می‌شود کلیه‌ی اطلاعات در دسترس کاربر باشد و ارتباط او با کارگزار کم‌تر شود. البته تکرار داده‌ها مشکلات خودش را دارد که از جمله‌ی آن‌ها الگوریتم‌های هماهنگ سازی توزیع شده است.

- نقاط میانی (checkpoint)

در بسیاری از عملیات‌های بزرگ می‌توان این کارها را به چندین واحد شکست. در نتیجه در صورت شکست در یکی از واحدها دیگر لازم نیست کل کار را دوباره تکرار کرد بلکه تنها قسمتی که دچار مشکل شده است را دوباره تکرار می‌کنیم. در این روش ما نسبت به هزینه‌ی سربار ناشی از خطا بیمه شده‌ایم.

- ردیابی فعالیت‌ها و واقعه‌نگاری

برای ردیابی مشکلات و حل آن‌ها معمولا برنامه‌ها اتفاقاتی که در روند اجرای آن‌ها می‌افتد را به گونه‌ای به اطلاع کاربر خود می‌رسانند. در این برنامه‌های منتشر شده باید به اینکه تا چه جزئیاتی را واقعه‌نگاری می‌کنند، توجه داشت. چرا که واقعه‌نگاری‌ای که بیش از حد ریزدانه باشد کارآیی را تا حد زیادی کم می‌کند و این به آن دلیل است که ورودی و خروجی‌ها یکی از کندترین واحدهای یک کامپیوتر هستند.

این صفحه مخصوصا خالی رها شده است

بخش ۴ اصل تمرکز

اجزای سیستم به تنهایی عمل‌کرد سیستم را مشخص نمی‌کنند بلکه چگونگی ارتباط و هماهنگی آن‌ها نیز در مشخص کردن این عمل‌کرد موثر است. برای این‌که این ارتباط و هماهنگی وجود داشته باشد باید اجزای سیستم به هم جور باشند و اصطلاحا به هم چفت شوند. اصل تمرکز با مفهوم هماهنگی محصول با کاربرد به کار می‌رود. این اصل اصلی بسیار کلی است و در زمینه‌های متنوعی قابل اعمال است. هر جایی که نیازی برای محصول مفروض است، می‌توان هماهنگی محصول با آن نیاز را در قالب اصل تمرکز مورد بررسی قرار داد. البته این اصل محدود به محصول نمی‌شود و حتی ارتباط بین اجزای محصول را نیز می‌تواند در بر گیرد.

این اصل برپایه‌ی دو ایده‌ی اصلی قرار گرفته است: نزدیکی[1] و الگوی استفاده. منظور ما از نزدیکی در این‌جا مشابه و نزدیک بودن انتظارات و سرویس ارائه شده توسط یک زیر بخش یا خود محصول است. الگوی استفاده بیان می‌کند که طراحی محصول باید متناسب با بارکاری که قرار است بر آن تحمیل شود در نظر گرفته شود.

اصل تمرکز را می‌توان هنگام طراحی محصول مد نظر قرار داد. شناخته شده‌ترین مورد از اصل تمرکز موضوع هماهنگی منابع با کاربرد است. یعنی مقدار و نوع منابع باید با توجه به کاربردی که هر کدام از منابع دارند طراحی شود. این موضوع شامل اندازه‌ی حافظه، نوع پردازنده و بسیاری از مواردی می‌شود که در کارآیی محصول کلی موثرند. در این اصل ما می‌خواهیم قرارگیری فیزیکی منابع به نحوی باشد که جوابگوی نیازمندی‌های منطقی سیستم باشد.

در صورتی که بخواهیم جایگاه اصل تمرکز در روند طراحی را بیان کنیم می‌توانیم به این روش عمل کنیم که ابتدا طراحی محصول بر اساس کسب و کار انجام می‌گیرد. سپس محیطی که محصول در آن اجرا می‌شود طراحی می‌شود. و سپس از بین انتخاب‌های موجود منابع و اندازه‌ی آن‌ها محاسبه شده و طراحی می‌گردند. به طور خلاصه در اصل تمرکز ما اجزا را بر مبنای نوع استفاده‌ی آن‌ها گروه‌بندی می‌کنیم تا جوابگوی نیازمندی‌هایی که توسط بارکاری بر آن تحمیل می‌شود باشد.

---

[1] Closeness

همان‌طور که در تعریف اصل تمرکز تاکید شد، اعمال اصل تمرکز بر اساس کاربرد هر کدام از اجزاء و منابع می‌باشد. در نتیجه از جمله پیش‌نیازهای اعمال اصل تمرکز شناخت کاربر و کاربرد هر کدام از منابع می‌باشد. برای این‌که در اعمال این اصل موفق باشیم باید ابتدا نیازمندی‌ها و نوع استفاده‌ی کاربر از منابع را بشناسیم تا از آن به بارکاری که بر روی سیستم می‌رود برسیم و در نهایت با توجه به این بار کاری سیستم را طراحی و اجزای آن را مشخص کنیم. برای این کار باید در مورد هر عملیات به سه سئوال زیر جواب بدهیم:

- چرا این عملیات برای سازمان مهم است؟
- هر عملیات خاص چگونه استفاده خواهد شد؟
- هر عملیات کی و چند بار فراخوانی خواهد شد؟

اصل تمرکز را می‌توان از چهار جهت دید که عبارتند از:

- تمرکز مکانی که به نزدیکی مکانی می‌پردازد
- تمرکز زمانی به نزدیکی در زمان می‌پردازد
- تمرکز درجه که به نزدیکی و شباهت در ظرفیت می‌پردازد
- تمرکز تاثیر که نزدیکی و شباهت در هدف را مد نظر دارد

۴-۱ تمرکز مکانی[1]

تمرکز مکانی به معنای نزدیک کردن مکانی کنش‌های مرتبط می‌باشد. یعنی کنش‌هایی که به صورت معمول در الگوی استفاده‌ی مشترکی مورد استفاده قرار می‌گیرند، مکانا نزدیک به یکدیگر قرار بگیرند. این بحث شباهت نزدیکی با اصل چسبندگی که در مهندسی نرم‌افزار مطرح می‌شود دارد. به طور خلاصه این اصل می‌گوید: عناصری که به هم کار می‌کنند و نیاز است که با هم مورد استفاده قرار بگیرند باید در کنار هم قرار گیرند.

[1] Spatial Locality

این موضوع را می‌توان از دیدگاه اصل کارایی هم دید؛ اگر تعدادی از عناصر بیش‌تر از یک بار با هم استفاده شوند با گروه‌بندی آن‌ها از هزینه‌ی سربار تحمیل شده جلوگیری می‌کنیم.

اگر از زاویه‌ی دیگر به این اصل نگاه کنیم، خواهیم دید که نزدیک کردن عناصر مرتبط مستلزم دور کردن عناصر غیر مرتبط از هم می‌باشد و این همان چفت شدگی است که در بخش ۱ مورد بررسی قرار گرفت. عملا همان‌طوری که گفته شد ارتباط اصول مهندسی کارایی نرم‌افزار و اصول مهندسی نرم‌افزار در این‌جا هویدا می‌شود. یک جزء از یک سیستم توزیع شده وقتی در به‌ترین حالت عمل می‌کند که در کنار عناصر مرتبط با خود باشد و از عناصر غیر مرتبط با خود هیچ اطلاعی نداشته باشد.

۴-۱-۱ تمرکز مکانی در عمل

در کامپیوتر نزدیک‌تر به معنای سریع‌تر است پس تاثیر تمرکز مکانی در کارایی به وضوح مشخص است؛ در سیستم‌های مبتنی بر کارگزار/مشتری این تاثیر حتی بیش‌تر است، از آن‌جا که در این سیستم‌ها توزیع شدگی بالا می‌رود – همان‌طور که می‌دانیم توزیع شدگی دشمن تمرکز مکانی است و به عبارتی توزیع شدگی طراحی‌هایی که بر روی یک کامپیوتر مرکزی خوب عمل می‌کردند را تبدیل به طراحی‌هایی می‌کند که در سیستم توزیع شده خوب کار نمی‌کند – بنابراین باید در معماری جدید مطرح شده تمرکز مکانی را نیز بگنجانیم. در عمل می‌توان از خطوط راهنمای زیر کمک گرفت تا به این هدف رسید:

- کمینه کردن ارتباطات بین کارگزار و مشتری

شبکه‌ها معمولا کندترین و خطا دارترین اجزای یک سیستم هستند و بنابراین بدترین کارایی و تاخیر در سیستم متعلق به آن‌هاست. پس ما باید تا می‌توانیم ارتباط بین آن‌ها و نقل و انتقال اطلاعات در این بین را کاهش دهیم.

- داده‌های مرتبط مجاور و در یک دستگاه نگه‌داری شوند

در یک سیستم اطلاعاتی توزیع شده مشخصا مهم‌ترین معیار برای کارایی محل اطلاعات است. اگر اطلاعاتی که منطقا با هم مرتبط هستند را به گونه‌ای از هم جدا کنیم، هرچقدر هم که پردازش را بهینه کنیم نمی‌توانیم از جابجایی اطلاعات بر روی شبکه و کاهش کارایی

جلوگیری کنیم. البته در این دسته بندی باید نیاز برنامه‌های مختلف در نظر گرفته شود و حالتی که بهینه است برای دسته بندی و محل قرارگیری اطلاعات تعیین شود.

- استفاده از معماری منطق توزیع شده

به این معنا که در صورتی که معماری مورد استفاده‌ی ما به صورت کارگزار/مشتری بود و شامل بخش‌های نمایش، منطق نمایش، منطق داده و داده می‌شد. قسمت‌های مرتبط با نمایش (نمایش و منطق نمایش) در مشتری و بقیه در قسمت کارگزار قرار بگیرند. در برابر این حالت انتخاب‌های دیگری نیز داریم که کارایی پایینی دارند؛ مثلا قرار دادن همه‌ی بخش به جز داده بر مشتری و داده‌ها بر مشتری و یا قراردادن همه‌ی بخش‌ها بر روی کارگزار و تنها مدیریت نمایش بر مشتری باعث ارتباط فراوان بین لایه‌های معماری بر روی شبکه می‌شود و در نتیجه کارایی را کاهش می‌دهد.

- استفاده از رویه‌های ذخیره‌شده پایگاه داده

با این‌که کارایی این رویه‌ها محدود به برخی از برنامه‌ها می‌شود ولی نوع شکل‌گیری آن‌ها (قرار گرفتن عملیاتی که باید انجام شود، در کنار اطلاعاتی که عملیات بر روی آن‌ها انجام می‌شود) نمونه‌ای از تمرکز مکانی هستند و جلوی نقل و انتقال اطلاعات بی‌مورد بر روی شبکه را می‌گیرند. این رویه‌ها اگر از قبل ترجمه شده باشند از هزینه‌ی سرباری که در اصل کارایی بحث شد نیز جلوگیری می‌کنند.

- استفاده از پایگاه داده شیء گرا

شیء همان‌طور که قبلا نیز بیان شد نمایانگر یک موجودیت از دنیای خارج است و مجموعه‌ای است از خصوصیات متفاوت که در یک حوزه‌ی مشترک که موجودیت شیء می‌باشد به هم ربط دارند. این موجودیت که خصوصیات حول آن جمع می‌شوند با توجه به یک یا چند برنامه تعریف می‌شوند. پس مشخص است که اشیا ذاتا تمرکز مکانی دارند. بنابراین استفاده از پایگاه داده‌ی شیءگرا باعث می‌شود اطلاعات فیزیکی مرتبط در مجاورت

هم قرار گیرند و درنتیجه به صورت خودکار تمرکز مکانی برقرار می‌شود. از طرفی این کار با توجه به اصل کارایی جلوی هزینه‌های سربار اضافی را می‌گیرد.

- قسمت‌بندی داده

هرکدام از کاربران پایگاه داده‌ها معمولا به زیرمجموعه‌ای از اطلاعات موجود در پایگاه دسترسی پیدا می‌کنند. وقتی کاربران با توجه به محل جغرافیایی، ناحیه‌ی استفاده و الگوی خرید تقسیم‌بندی می‌شوند هوشمندانه به نظر می‌رسد که از این تقسیم‌بندی برای اطلاعات نیز استفاده کنیم. به این ترتیب بارکاری هر کارگزار و رد و بدل شدن اطلاعات بر روی شبکه تا حد زیادی کاهش می‌یابد.

۴-۲ تمرکز زمانی[1]

تمرکز زمانی به نزدیک بودن از نظر زمانی برمی‌گردد. خیلی مواقع رعایت تمرکز مکانی، تمرکز زمانی را نیز نتیجه می‌دهد ولی مواقعی نیز هستند که این نتیجه مستقیما به دست نمی‌آید و موانعی بر سر آن است. به طور مثال موارد زیر را در نظر می‌گیریم:

- اگر تمرکز ایستا را در نظر بگیریم، تنها یک نوع تمرکز را می‌توانیم به طراحی خود اعمال کنیم. در این صورت نمی‌توانیم نوع دیگری از گروه‌بندی و تمرکز را اعمال کنیم.
- برخی اوقات محدودیت بر روی اندازه‌ی گروه و یا برخی محدودیت‌های فیزیکی دیگر نوع خاصی از گروه‌بندی را به ما تحمیل می‌کنند.
- برخی تصمیم‌گیری‌های منطقی که شاید کاملا بی‌ربط به کارایی باشند تاثیراتی بر روی گروه‌بندی دارند. این موضوع در بخش ۱ مورد بحث قرار گرفت.

[1] Temporal Locality

بنابراین می‌بینیم که نزدیکی مکانی لزوما باعث تمرکز زمانی شود. به عبارتی اگر استفاده نوع دیگری از گروه‌بندی را طلب کند تمرکز زمانی است که با دادن اصولی اضافه بر نزدیکی مکانی، تضمین می‌کند که کارهای مرتبط در یک گروه و پشت سر هم انجام گیرند.

۴-۲-۱ تمرکز زمانی در عمل

برای بیان این نوع تمرکز دو محدوده از سیستم‌ها مبتنی بر معماری کارگزار/مشتری را مورد بررسی قرار می‌دهیم:

- طراحی واسط کاربر

اگر در هنگام طراحی واسط کاربر، پنجره‌ها با استفاده از اصل تمرکز مکانی طراحی شده باشند، برنامه به چندین بخش تقسیم می‌شود و در صورت ورود به هر بخش منوهای آن بخش را پی می‌گیریم. مثلا برای یک برنامه‌ی مالی منوهای مربوط به انجام عملیات برداشت پول پشت سر هم قرار می‌گیرند.

تا این‌جا بر اساس اصل تمرکز مکانی پیش آمدیم و از این که کلافی از منوها را داشته باشیم به‌تر است. در ادامه حالتی را فرض کنیم که مشتری علاقه‌مند به دیدن وجه حساب و یا اطلاعات دیگری در زمان وارد کردن وجه دریافتی باشد؛ در این طراحی باید کاری تا کنون انجام داده را لغو کرده و دوباره رفته وجه را دیده و دوباره برگردد کارهای انجام شده را انجام دهد تا به وضعیت قبلی برسد. این‌جاست که تمرکز زمانی به کار می‌آید. با استفاده از تمرکز زمانی منوهایی که ممکن است در یک زمان پشت سرهم قرار گیرند نیز به هم راه خواهند داشت و برای انجام عملیات دیگر لازم نیست کل مسیر دوباره طی شود.

- پایگاه داده‌ی متحد[1]

برخی از برنامه‌هایی که با چندین پایگاه داده کار می‌کنند برای انجام برخی عملیات پیچیده که همه‌ی پایگاه‌ها را در بر می‌گیرد باید در زمان اجرا به همه‌ی آن‌ها دسترسی داشته باشند.

[1] Federated Databases

این پایگاه‌ها ممکن است با توجه به اصل تمرکز مکانی خوب طراحی شده و از هم دور باشند ولی نیاز ما در این‌جا باعث می‌شود اطلاعات مختلف و بی‌ربط به هم را در یک زمان و در یک جا بخواهیم. با استفاده از اصل تمرکز زمانی راه حلی برای این مساله وجود دارد و آن این که زیر مجموعه‌ای از اطلاعات هر کدام از بخش‌ها را در یک پایگاه داده‌ی محلی آورده تا در زمان استفاده بلافاصله در دسترس باشند و کارایی را کاهش ندهند. البته به روز نگهداری این پایگاه‌ها نیز خود مساله‌ای است که باید حل شود.

۴-۳ تمرکز درجه[1]

تمرکز درجه به معنای تناسب اندازه‌ی هر کدام از منابع مورد استفاده به الگوی استفاده‌ی آن می‌باشد. برای تامین این اصل باید نیازمندی‌های ذخیره‌ی اطلاعات، ارسال اطلاعات و توان پردازشی که توسط بارکاری اعمال می‌شود با منابع هم‌خوانی داشته باشد.

معمولا وقتی بحث از تناسب منابع می‌شود ذهن به سراغ منابع سخت‌افزار – پردازنده، حافظه، دیسک و ... – می‌رود در حالی که منابع دیگری نیز وجود دارند که آن‌ها هم به همین اندازه از اهمیت برخوردار هستند؛ این منابع که منابع منطقی نام دارند شامل منابعی سطح بالاتر مانند قفل‌ها، روندهای اجرا، بافرها و فضای آدرس‌دهی می‌شوند. اجزای نرم‌افزاری بر روی منابع منطقی نیز محدودیت می‌گذارند. اگر برای سرویس دادن به یک بارکاری منابع منطقی مورد نیاز به تعداد فراهم نشوند نمی‌توان به آن بارکاری سرویس مورد نظر را ارئه داد. اصولی که در این بخش مطرح می‌شوند همان قدر که منابع سخت افزار مربوط می‌شوند به این منابع منطقی نیز مرتبط هستند.

تمرکز درجه اصلی اساسی برای تنظیم کارایی سیستم‌های مبتنی بر معماری کارگزار/مشتری می‌باشند به این دلیل که سیستم‌های تجاری از این نوع به منابع متنوعی نیازمند هستند. نمی‌توان یک نرم‌افزار نوشت و انتظار داشت که نیازمندی‌های آن (پردازشی، فایلی، حافظه‌ای و ...) ثابت باشد و این نرم‌افزار بتواند هرکجا که بود اجرا شود.

[1] Degree Locality

۴-۳-۱ تمرکز درجه در عمل

برای اطمینان از تطابق طراحی و نیازهای نرم‌افزار باید نیازمندی‌های مورد نظر را تحلیل کرده و بارکاری مربوطه را استخراج کنیم. برای تخمین صحیح توانایی مشتری، کارگزار و نیز شبکه باید دید درستی از بارکاری و ارتباطات موجود داشت.

هنگام اعمال تمرکز درجه باید در نظر داشت که باید اوج بار کاری در مدت استفاده از منابع در نظر گرفته شود. تنها در نظر گرفتن حالت میانگین بار کاری به عنوان ملاک باعث می‌شود در حالت‌های اوج بار کاری سرویس‌دهنده به کلی از کار بیافتد. همچنین رشد بار کاری را نیز در این مورد باید مد نظر داشت. بار کاری در حالت اوج در این زمان برابر است با بار کاری متوسط در آینده بنابراین ما باید بتوانیم بارکاری اوج را جواب دهیم تا در آینده‌ی نزدیک توانایی جوابگویی به نیازهای متوسط را داشته باشیم و توانایی مقیاس‌پذیری[1] را داشته باشیم.

۴-۴ تمرکز تاثیر[2]

تمرکز تاثیر عبارت است از نزدیکی هدف‌ها. این نوع تمرکز که کلی‌ترین نوع تمرکز و شامل تمام موارد فوق می‌باشد به معنی هماهنگی نیازها با محصول می‌باشد. به عبارت دیگر کارکرد تمام سیستم به صورت مطلوب و به خوبی با هم کار می‌کند تا نیاز کاربران را رفع کند.

[1] Scalability

[2] Effectual Locality

در مجموع اگر بخواهیم در نموداری رابطه‌ی انواع تمرکز را نشان بدهیم، نموداری متناسب با شکل ۲ خواهیم داشت.

شکل ۲: نمودار سلسله مراتب تمرکزها

درست است که تمرکز اصولا در مورد جنبه‌ی فیزیکی سیستم و نرم‌افزار مطرح می‌شود؛ اما تا ما شناخت دقیقی از طراحی منطقی سیستم و ارتباط آن با طراحی فیزیکی نداشته باشیم نمی‌توانیم سیستمی طراحی و پیاده‌سازی کنیم که نیازمندی‌های مورد نظر ما را جوابگو باشد، عملا تمرکز تکنیکی است که هم در سطح منطقی سیستم و هم در سطح فیزیکی آن وارد می‌شود.

در سیستم‌های توزیع شده معمولا سیستم به لایه‌هایی شکسته می‌شود که با هم کار می‌کنند. تمرکز اصولا به کل سیستم اعمال می‌شود و تضمین می‌کند که تمام قسمت‌ها و لایه‌های سیستم متناسب هم هستند و انتظاری که هرکدام از دیگری دارد تامین می‌شود. تنها در صورتی تمرکز تاثیر در سیستمی وجود دارد که تک تک عناصر سیستم به صورت مجزا به درستی کار کنند و همچنین کلیه‌ی آن‌ها در تعامل با یکدیگر جواب‌گوی نیازها و بار کاری باشند.

یکی از نمونه‌های تمرکز تاثیر قانون اینمون است که در مورد تفکیک بارهای کاری صحبت می‌کند. فرض کنیم تمرکز تاثیر به ما می‌گوید که باید بارکاری خاصی را با منابع پردازشی خود پشتیبانی کنیم. این بارکاری، توان عملیاتی بالا و زمان پاسخ کوتاهی را نیاز دارد. مشخص است که نمی‌توان در همین سیستم

یک سیستم پشتیبانی از تصمیم با حجم پرسش‌های بالا داشت. مشکل از آن‌جا ناشی می‌شود که در این ترکیب دو نوع بارکاری وجود دارد؛ بارکاری اول شامل درخواست‌های کوچک ولی زیاد است که پاسخ سریع نیاز دارند، درحالی که بارکاری دوم شامل درخواست‌های بزرگ ولی کم تعداد است. هرچقدر هم که سعی کنیم که این دو نوع کار را با هم مخلوط کنیم یکی از بارهای کاری صدمه می‌بینند. این در صورتی است که دو بار کاری از هم جدا باشند حال اگر مکانیزم‌های قفل دهی و ... را نیز در نظر بگیریم خواهیم دید که اگر این دو بار کاری روی هم تاثیر داشته باشند شرایط سخت‌تر هم می‌شود.

قانون اینمون می‌گوید برای جلوگیری از این مشکلات باید بارکاری که نیاز به زمان پاسخ کم دارد را از پرسش‌های موسمی جدا کرد و بر روی پردازنده‌ای دیگر اجرا کرد چون پرسش‌های موسمی قابل پیش‌بینی نیستند و تاثیر بدی بر کارایی کل سیستم می‌گذارند.

۴-۴-۱ تمرکز تاثیر و معماری‌های چند لایه

در طراحی سیستم‌های لایه‌بندی شده قانونی وجود دارد که می‌گوید هر چیزی را در جایی قرار بده که به آن‌جا تعلق دارد. در سیستم‌های دولایه ما از این قانون استفاده می‌کنیم. به این صورت که اطلاعاتی را که بین مشتریان ثابت است را بر روی کارگزار قرار می‌دهیم و قسمت‌هایی را که به هر مشتری و به صورت جداگانه نعلق دارد را بر روی خود مشتری قرار می‌دهیم. به طور مثال عملیات نمایش که به سرعت پاسخ بالا نیاز دارد و بیش‌تر به مشتری بازمی‌گردد بر روی خود مشتری قرار می‌گیرد در حالی که عملیات با پردازش بالا بر روی اطلاعات مشترک که متعلق به همه است بر روی کارگزار قرار می‌گیرد.

برخی از عملیات هستند که مانند سایه‌ها به همه‌جا و هیچ جا تعلق دارند. عملیاتی مثل رمزنگاری، بهینه سازی پرسش‌ها، تحلیل آماری و ... مواردی هستند که با توجه به اطلاعاتی که بر روی آن عمل می‌کنند می‌توانند در جاهای مختلف سیستم پخش شده باشند.

سیستم‌های سه لایه نیز در راستای همان قانون ذکر شده هستند. گاهی اوقات سیستم آن‌قدر بزرگ می‌شود که برخی از مشتریان به صورت دسته‌ای نیازهای خاصی پیدا می‌کنند و نیازها نه آن‌قدر مشترک است که به کارگزار تعلق پیدا کند و نه آن‌قدر خصوصی است که به مشتریان اختصاص پیدا کند. برای این موارد معمولا یک لایه‌ی میانی قرار می‌دهند که همین مشترکات دسته‌ای را رفع و رجوع می‌کند. عملا قسمت‌های مخصوص کاربران نزد خود آن‌ها، قسمت‌هایی که بین چندین کاربر مشترک است نزد لایه‌ی

میانی مخصوص همان گروه از کاربران و در نهایت قسمت‌هایی که برای همه‌ی کاربران مشترک است بر روی کارگزار اصلی قرار می‌گیرد.

همان‌طور که مشاهده می‌شود معماری لایه‌ای هم نمونه‌ای از به‌کار گیری اصل تمرکز است.

۴-۴-۲ طراحی بالا به پایین یا پایین به بالا

در تعامل با طراحی هنگامی که طراحی نرم‌افزار از بالا به پایین صورت می‌گیرد، اجزا و مولفه‌های سیستم کم‌تر با یکدیگر هماهنگی دارند و در نتیجه استفاده‌ی دوباره از اجزا به سختی صورت می‌گیرد. اما در طراحی پایین به بالا روند برعکس است. در این نوع طراحی ما ابتدا برای بالا بردن سطح انتزاع اقدام به ساختن اجزای کلی‌تر می‌کنیم. این اجزا قطعا در سطوح بعد استفاده خواهند شد.

در مقابل در صورتی که طراحی از پایین به بالا صورت بگیرد از آن‌جایی که هر کدام از مولفه‌ها از نحوه‌ی استفاده و بار کاری مربوطه مطلع نیست و کلی نوشته می‌شوند، احتمال دارد با بارهای کاری مختلف که از آن استفاده می‌کنند ناهماهنگی وجود داشته باشد. یعنی یک مولفه به حدی کلی و بزرگ است که برای سرویس دادن به یک بارکاری فقط به یک زیرمجموعه‌ی آن نیاز است؛ اما از آن زیرمجموعه هم نمی‌توان استفاده کرد چون بزرگ و کلی بودن مولفه برای ما هزینه‌ی اضافی دارد.

این صفحه مخصوصا خالی رها شده است

بخش ۵ اصل به اشتراک گذاری

تا بدین‌جا بررسی سه اصل اول مهندسی کارایی نرم‌افزار بدین‌نکته تکیه داشت که چگونه می‌توان کارایی برنامه‌های خاص کاربردی و یا مولفه‌های آنها را بهبود بخشید. اما در این بخش موضوع مورد بررسی از دید برنامه‌ای به دید اجتماع برنامه‌ها و یا به بیان بهتر دید سیستمی تغییر پیدا می‌کند.

یک ماتریس برنامه کاربردی می‌تواند یک سیستم را به عنوان مجموعه‌ای از بارهای کاری برنامه‌های کاربردی، منابع سیستم و ارتباط این دو باهم توصیف کند. در این ماتریس که در جدول ۳ قالب کلی آن نمایش داده شده است، هر سطر بیان‌کننده یک برنامه کاربردی و یا گروهی از برنامه‌هاست که از منظر بررسی کاملا شبیه به هم می‌باشند. هر ستون نیز یا یک منبع فیزیکی را معرفی می‌کند(مانند یک پردازنده، دیسک، ...) و یا یک منبع منطقی مانند یک شیء، یک جدول و یا یک پایگاه داده.

	منبع ۱	منبع ۲	منبع ۳	منبع ۴	منبع ۵
برنامه ۱					
برنامه ۲					
برنامه ۳					

جدول ۳: ماتریس برنامه کاربردی-منبع

اطلاعاتی که در یک ماتریس ذخیره می‌شود توصیف می‌کند که چگونه مجموعه برنامه‌ها (سطرهای جدول) از مجموعه منابع(ستون‌ها) استفاده می‌کنند. در حقیقت هر سلول جدول در محل تقاطع سطر و ستون، استفاده یک برنامه خاص از یک منبع خاص را نشان می‌دهد.

اطلاعات آمده در ماتریس کلید کارایی سیستم هستند. بخش‌های قبلی که در مورد اصل بارکاری و اصل کارایی صحبت می‌کردند بر روی سطرهای این جدول تمرکز داشتند. اصل تمرکز نیز هدف تطبیق منابع و برنامه‌های کاربردی را داشت. در این بخش توجه اصلی به ستون‌های ماتریس(منابع) است و به خصوص منابعی که به‌وسیله بسیاری از برنامه‌ها به اشتراک گذارده شده‌اند.

۵-۱ مفاهیم

اصل به اشتراک‌گذاری منابع کوتاه و ساده است: **منابع را بدون ایجادکردن گلوگاه به اشتراک بگذارید.** کانی اسمیت حتی این جمله را با جمله‌ای کوتاه‌تر جایگزین می‌کند بدین شکل که: منابع را هر زمان که ممکن است به اشتراک بگذارید. او سپس این راهنمایی را نیز به جمله خود اضافه می‌کند که هنگامی که استفاده انحصاری از یک منبع مورد نیاز است مجموع زمان استفاده و زمان‌بندی را بهینه کنید. این قانون ساده کلید فهم کارایی منابع به اشتراک گذارده شده است.

در ادامه چند تعریف را که در این بخش حائز اهمیت است مورد بررسی قرار می‌دهیم:

- **زمان پردازش:** هنگامی که یک منبع محاسباتی میان تعدادی پردازه به اشتراک گذارده شده است، میانگین زمانی که هر پردازه صرف استفاده از این منبع می‌کند به عنوان زمان پردازش شناخته می‌شود.
- **زمان انتظار:** این زمان که در بسیاری از مواقع به عنوان زمان در صف بودن معرفی می‌شود، همان‌گونه که از نامش برمی‌آید، زمانی است که یک پردازه صرف می‌کند تا نوبت استفاده از یک منبع به او برسد.
- **زمان سرویس:** مجموع زمان پردازش و زمان انتظار به عنوان زمان سرویس یک منبع شناخته می‌شود.(برای هر پردازه)

حال اگر ماتریس برنامه کاربردی-منبع در نظر گرفته شود، اگر این ماتریس، هر منبع را در ستون‌های خود شامل شود آنگاه هر برنامه کاربردی که درسیستم وجود دارد در ماتریس آمده است. حتی برای کامل بودن، زمان پاسخ کاربر در برنامه‌های محاوره‌ای نیز به عنوان یک منبع شناخته می‌شود و یک ستون ماتریس را دربرمی‌گیرد. در واقع کاربر به عنوان یک دستگاه ورودی-خروجی کند در نظر گرفته می‌شود، دستگاهی که به نسبت سایر دستگاه‌های ورودی-خروجی، بسیار کند عمل می‌کند.

برای جمع بندی در این قسمت، راهنمایی‌هایی نیز ارایه می‌شود که توجه مارا به دو حقیقت بدیهی ولی مهم جلب می‌کند:

۱. هر زمان که منبع محدودی به اشتراک گذارده می‌شود، این موضوع غیرقابل اجتناب است که پردازه‌ها باید زمانی را صرف انتظار کنند زیرا که ممکن است سایر پردازه‌ها در حال استفاده از آن منبع باشند.

۲. آن موضوعی که در کارایی اهمیت دارد، مجموع زمان سرویس است- زمانی که پردازه صرف استفاده از منبع می‌کند به اضافه زمانی که صف انتظار برای استفاده می‌نماید.

۵-۲ به اشتراک گذاری منابع

فرض کنید که دو برنامه کاربردی به نامهای DataMiner و DataReporter مورد بررسی قرار می‌گیرند به هدف اینکه کارایی آنها بهبود پیدا کند. پس از بررسی این برنامه‌ها، با بررسی جدول ۴ مشخص می‌شود که هریک از آنها زمان پاسخ ۲۱ ثانیه را دارا می‌باشد.

	برنامه DataReporter	برنامه DataMiner
I/O	۱۰۰۰ عمل مورد نیاز	۱۰ عمل مورد نیاز
	۲۰ میلی ثانیه برای هر I/O	۲۰۰ میلی ثانیه برای هر I/O
	زمان پاسخ ۲۰ ثانیه	زمان پاسخ ۲ ثانیه
CPU	۱ میلیون دستورالعمل	۱ میلیارد دستورالعمل
	پردازنده MIPS ۵۰، ۹۹٪ مشغول	پردازنده MIPS ۵۰
	زمان پاسخ ۲ ثانیه	زمان پاسخ ۲۰ ثانیه

جدول ۴: بررسی دو برنامه با عملیات متفاوت

- در برنامه DataMiner مقداری از زمان صرف I/O می‌گردد(که احتمالا بیشتر آن، زمان انتظار است زیرا ms ۲۰۰ ، زمان بسیار زیادی برای یک عمل I/O دیسک می‌باشد.) و حدود ۲۰ ثانیه نیز صرف پردازش ۱ میلیارد دستورالعمل برروی یک پردازنده انحصاریMIPS ۵۰ می‌گردد.

- در برنامه DataReporter، هزار درخواست I/O در ۲۰ ثانیه پردازش می‌شوند- هریک حدود ۲۰ ms . ۲ ثانیه نیز صرف پردازش ۱ میلیون دستورالعمل برروی یک پردازنده ۵۰ MIPS می‌شود که ۹۹ درصد اشغال است و مانند یک پردازنده ۰,۵ MIPS عمل می‌کند.

حال فرض کنید که سعی می‌شود هریک از این برنامه‌های کاربردی تنظیم شوند. با استفاده از هشت اطلاع آماری که در دست است(فرکانس‌ها و زمان‌های پاسخ برای CPU و I/O دیسک برای دو برنامه کاربردی) می‌توان نتایج بیشتری راجع به این برنامه‌ها به‌دست آورد و اینکه چگونه می‌توان آنها را تنظیم کرد.

- **برنامه DataMiner**: هنگام تنظیم کردن DataMiner دقت نظر به پردازش ۱ میلیارد دستورالعمل معطوف می‌شود. به طور مشخص مشکل در اینجا مقدار بکارگیری CPU است و اولین سوالی که پرسیده می‌شود این است که چرا برنامه از این مقدار زیاد دستورالعمل استفاده می‌کند. پس از آن سعی می‌شود تا حد ممکن تعداد دستورالعمل‌های مورد استفاده کاهش یابد، اقدامی که احتمالا سودمند نخواهد بود.
- **برنامه DataReporter**: با بررسی این برنامه به نظر می‌رسد که بیشتر زمان پاسخ از استفاده از دیسک حاصل شده است. برای بالا بردن سرعت این برنامه اولین گام بررسی الگوی دسترسی این برنامه به پایگاه داده است، تا فهمیده شود آیا می‌توان این برنامه را به دوباره به گونه‌ای طراحی کرد که به طرز کاراتری از مقدار کمتری I/O استفاده کند؛ اتفاقی که شاید امکان آن موجود نباشد.

نتیجه گیری: مشاهده اول از کل ماجرا این است که اقدام به تنظیم کردن DataMiner اثر چندانی بر DataReporter نخواهد داشت و بالعکس. دوم اینکه تمامی دستگاه‌ها یکسان ساخته نشده‌اند و از آنجا که CPU بسیار سریعتر از دیسکها است و از آنجایی که توسط تمامی پردازه‌ها به اشتراک گذارده شده است، یک بار کاری که به ۲۰ ثانیه از زمان پردازنده احتیاج داشته باشد قابل مقایسه با بار کاری نیست که به ۲۰ ثانیه I/O احتیاج داشته باشد.

در مورد مثالی که در بالا طرح شد DataMiner یک مثال از نوع برنامه‌هایی است که به عنوان خوک شناخته می‌شوند-تمامی منابع پردازشی را می‌بلعند و اجازه به اشتراک گذارده شدن این منابع با دیگران را نمی‌دهند.- برای جلوگیری از اینکه این برنامه‌ها محیطهای اشتراکی را به محیطهایی غیر قابل استفاده برای دیگران تبدیل کنند، راه حل مناسب این است که این برنامه‌ها به

منابع پردازشی دیگری انتقال داده شوند. به عنوان مثال در مورد بالا شاید راه حل مناسب انتقال DataMiner به یک پردازنده اختصاصی دیگر باشد.

قوانین تنظیم برنامه‌ها:

۱. تمامی زمان مصرفی در محاسبات از مجموع زمانی که برای انتظار منابع محاسباتی و زمانی که برای استفاده از منابع صرف می‌شود، به دست می‌آید.

۲. برای تنظیم کارایی یک برنامه کاربردی باید ابتدا فهمید که این برنامه زمان خود را در کجا مصرف می‌کند.

قالب کاری برای رفع گلوگاه‌ها:

برای بررسی یک برنامه کاربردی خاص به شکل زیر عمل می‌شود:

۱. سطری که این برنامه در آن قرار گرفته است، بررسی می‌شود و مشخص می‌شود که آیا مجموع زمان پاسخ قابل قبول است یا خیر.

۲. سلولهای این سطر تک به تک مورد بررسی قرار می‌گیرد، هر مقداری که بیش از اندازه بزرگ باشد نشان می‌دهد که این برنامه مقدار زمان زیادی را صرف استفاده از یک منبع خاص می‌کند و در نتیجه می‌توان تمرکز را در کاهش استفاده از آن منبع قرار داد.

۳. اگر استفاده برنامه از منبع مشکل خاصی نداشته باشد، مشکل باید از اشغال بودن بیش از حد منبع ناشی شود.–برنامه زمان بیش از حدی را صرف انتظار برای استفاده از منبع می‌کند– با بررسی کردن به این شکل می‌توان منابعی را شناسایی کرد که بیش از حد مورد استفاده قرار گرفته‌اند.

۴. با بررسی ستونهایی که منابعی که بیش از حد مورد استفاده قرار می‌گیرند در آنها قرار دارند، می‌توان فهمید که کدامیک از برنامه‌های کاربردی این استفاده بیش از حد از منبع را باعث شده‌اند. با شناسایی این برنامه‌ها می‌توان بر روی این موضوع تمرکز کرد که استفاده آنها را از این منابع تا حد امکان کاهش داد.

۵. برای کاهش دادن زمان استفاده از منابع می‌توان:

a. تعداد دفعاتی را که به منبع درخواست داده می‌شود کاهش داد.

b. اندازه درخواست را کاهش داد.

c. سرعت پردازش درخواست را بالا برد.

d. بعضی از درخواستها را به وسیله منابع دیگر پردازش کرد.

۵-۳ کارگزاران به اشتراک گذارده شده

در یک سیستم مشتری-کارفرما، باید برنامه کاربردی به گونه‌ای طراحی شود که زمان دراختیار داشتن کندترین منبع مشترک کمینه شود. منابع کندتر بحرانی‌تر هستند برای آنکه معمولا گلوگاه سیستم می‌شوند. دو راه حل مناسبی که در اینگونه سیستم‌های کارگزار-مشتری می‌تواند مورد توجه قرار گیرد، یکی جداکردن کارهای از کلاس مختلف و دیگری استفاده از موازی سازی یا کارگزاران خوشه‌ای می‌باشد.

۵-۳-۱ جدا کردن کارهای از کلاسهای مختلف

در یک محیط کارگزار-مشتری، یک راه برای کاهش بار بر یک کارگزار به اشتراک گذارده شده، جدا کردن کارهایی است که پارامترهای اصلی پاسخگویی در آنها متفاوت می‌باشد. مثالهایی برای اینگونه جداسازیها به شرح زیر می‌باشد:

- جدا کردن کارهای دسته‌ای که زمان طولانی می‌برند از کارهایی که قالب تراکنشی حساس به زمان را دارا می‌باشند.
- ایجاد Data Warehouse ها برای جداسازی کارهای تحلیل داده‌ای از کاربردهای برنامه‌های عملیاتی پایگاه داده‌ای.
- جداسازی برنامه‌های با کاربرد Data Mining از سایر برنامه‌های کمتر پردازش بر و اختصاص پردازنده‌های اختصاصی به آنها.

۵-۳-۲ موازی سازی یا کارگزاران خوشه‌ای

مفهوم و فواید کارگزاران خوشه‌ای

خوشه‌ای کردن کارگزاران به مجموعه‌ای از عملیات گفته می‌شود که به دسته‌ای از کارگزاران این امکان را می‌دهد تا از دید کاربر به شکل یک کارگزار دیده شوند و در عین حال بار موجود را میان خود تقسیم کنند. این کار گروهی به سیستم کارگزاری امکان می‌دهد تا مجموعه‌ای از قابلیت‌ها را در اختیار داشته باشد که در سیستم‌های دیگر کارگزاری نمی‌توان به دست آورد.

به منظور درک فواید خوشه‌ای کردن کارگزاران، در ادامه خواسته‌های ما از سیستم‌های کارگزاری بیان و سپس یکی از سیستمهای متداول کارگزاری با نام کارگزار انفرادی با کارگزاران خوشه‌ای مقایسه می‌گردد.

قابلیتهای مورد نیاز سیستم کارگزار

هر سیستم کارگزاری باید دارای قابلیتهای مشخصی باشد تا بتوان بر قدرت کاری آن ارزش‌گذاری انجام داد. مجموعه‌ای از قابلیتها که بر روی آن توافق جهانی وجود دارد به شرح زیر می‌باشند:

۱. **مقیاس پذیر بودن**[1]

مقصود از مقیاس پذیر بودن این است که سیستم در صورت زیاد شدن بار موجود، توانایی پاسخگویی به این بار اضافه شده را داشته باشد و در واقع بتواند به تعداد زیادی کاربر به طور همزمان خدمت ارایه کند.

۲. **در دسترس بودن بالا**[2]

در دسترس بودن بالا همان‌گونه که از نامش برمی‌آید به این مفهوم است که سیستم کارگزار تا حد بسیار زیادی قابل اطمینان بوده و امکان از کارافتادن بسیار پایینی داشته باشد.

۳. **مدیریت ساده**[3]

[1] Scalability

[2] High Availability

[3] Easy Management

با توجه به این که ممکن است سیستم کارگزار، سیستم بزرگی باشد این قابلیت، اطمینان می‌دهد که مدیریت سیستم همچنان ساده می‌ماند.

۴. به صرفه بودن از نظر اقتصادی[1]

این قابلیت تضمین می‌کند که تهیه و گسترش این سیستم هزینه اقتصادی بسیار بالایی را بر مسوولین سیستم تحمیل نکند.

کارگزار انفرادی در مقایسه با کارگزاران خوشه‌ای

یکی از سیستمهای متداول کارگزار، کارگزار انفرادی می‌باشد. کارگزار انفرادی معمولا دستگاهی است دارای امکانات سخت افزاری بسیار قویتر از دستگاه‌های شخصی، دارای قدرت پردازش بسیار بالا–معمولا با استفاده از چند پردازنده– ، حافظه حجیم و ...، که به عنوان یک کارگزار قدرتمند از آن استفاده می‌شود. در مقابل، کارگزاران خوشه‌ای غالبا دستگاه‌های شخصی هستند که البته سعی می‌شود با سرعت پردازنده مناسب و حافظه کافی انتخاب شوند. با اینکه کارگزاران انفرادی دستگاه‌های بسیار قدرتمندی می‌باشند اما امروزه در جهان، در ساختن بزرگترین سیستم‌های کارگزاری از آنها استفاده نمی‌شود و استفاده از کارگزاران خوشه‌ای به آنها ترجیح داده می‌شود که به عنوان نمونه می‌توان به سایتهای بزرگی از قبیل yahoo، google و amazon اشاره کرد.

در ادامه سعی خواهیم کرد با استفاده از معیارهایی که در قسمت قبل ارایه شد، این دو نوع سیستم کارگزاری را با یکدیگر مقایسه کنیم تا دلایل گرایش به کارگزاران خوشه‌ای بیشتر روشن شود[1].

- **مقیاس پذیر بودن**

به طور معمول هر کارگزار انفرادی دارای قابلیت تحمل و پاسخگویی به مقدار حداکثری بار می‌باشد که به سختی قابل افزایش است و با افزایش این بار دیگر پاسخی از کارگزار دریافت نخواهد شد.

در سیستم‌های کارگزاران خوشه‌ای برای بالا بردن پاسخگویی به بار بیشتر، می‌توان به تعداد دستگاه‌های موجود در گروه کارگزاران اضافه کرد.

- **در دسترس بودن بالا**

[1] Cost-effectiveness

اگر کارگزار انفرادی ما به هر دلیل از کار بیفتد، دیگر نقطه‌ای در شبکه برای پاسخگویی به درخواستهای کاربران وجود نخواهد داشت و تا به کارافتادن دوباره این دستگاه، خروجی سیستم کارگزاری ما برابر صفر می‌باشد.

در سیستمی که با مجموعه‌ای از کارگزاران ساخته شود، با از کارافتادن یک دستگاه، دستگاه دیگری در سیستم، وظیفه دستگاه از کارافتاده را برعهده خواهد گرفت. بدین ترتیب از کارافتادن یک دستگاه باعث عدم پاسخگویی سیستم کارگزاری به درخواستها نخواهد شد.

- **مدیریت ساده**

معمولا مدیریت یک سیستم کارگزار انفرادی تا حدی ساده تر از مدیریت مجموعه‌ای از کارگزاران می‌باشد اما مجموعه عملیات به‌روز کردن سیستم کارگزار انفرادی، غالبا پیچیده تر از به‌روز کردن دستگاه‌های موجود در مجموعه کارگزاران می‌باشد.

- **به صرفه بودن از نظر اقتصادی**

معمولا هزینه‌ای که صرف خریدن، نگهداری و به روز کردن یک دستگاه انفرادی می‌شود بسیار قابل توجه تر از هزینه ای است که انجام این عملیات برای تمام دستگاه‌های موجود در یک مجموعه کارگزار در پی‌دارد. در عمل می‌توان گفت که به راه انداختن یک مجموعه کارگزار از نظر اقتصادی بسیار به‌صرفه تر از راه اندازی سیستم کارگزاری با استفاده از یک دستگاه کارگزار انفرادی می‌باشد.

ساختار سیستم کارگزاران گروهی

کارگزاران گروهی پیاده سازی شده در مکانهای مختلف، اگرچه در بسیاری از جزییات پیاده سازی با یکدیگر تفاوت دارند اما معمولا در ساختار اصلی با یکدیگر مشترکند. این ساختار مشترک به ساختار سه لایه معروف است و همان طور که در شکل ۳ مشخص می‌باشد، از سه عنصر اساسی با نامهای کارگزاران حقیقی، توزیع کننده بار و محل ذخیره سازی تشکیل شده است.

۱. کارگزاران حقیقی

کارگزاران حقیقی گروهی از کارگزارها هستند که خدمات اصلی سیستم کارگزاری از قبیل HTTP ، FTP و ... را ارایه می‌کنند.

۲. توزیع کننده بار

این عضو از مجموعه، دستگاهی است که توسط کاربران دیده می‌شود و کاربران در حقیقت درخواستهایشان را برای این دستگاه می‌فرستند. این عضو، درخواستها را دریافت کرده و بر حسب تشخیص خود میان کارگزاران حقیقی این مجموعه توزیع می‌کند. به این ترتیب بار ورودی میان مجموعه‌ای از کارگزاران به وسیله توزیع کننده بار تقسیم می‌شود.

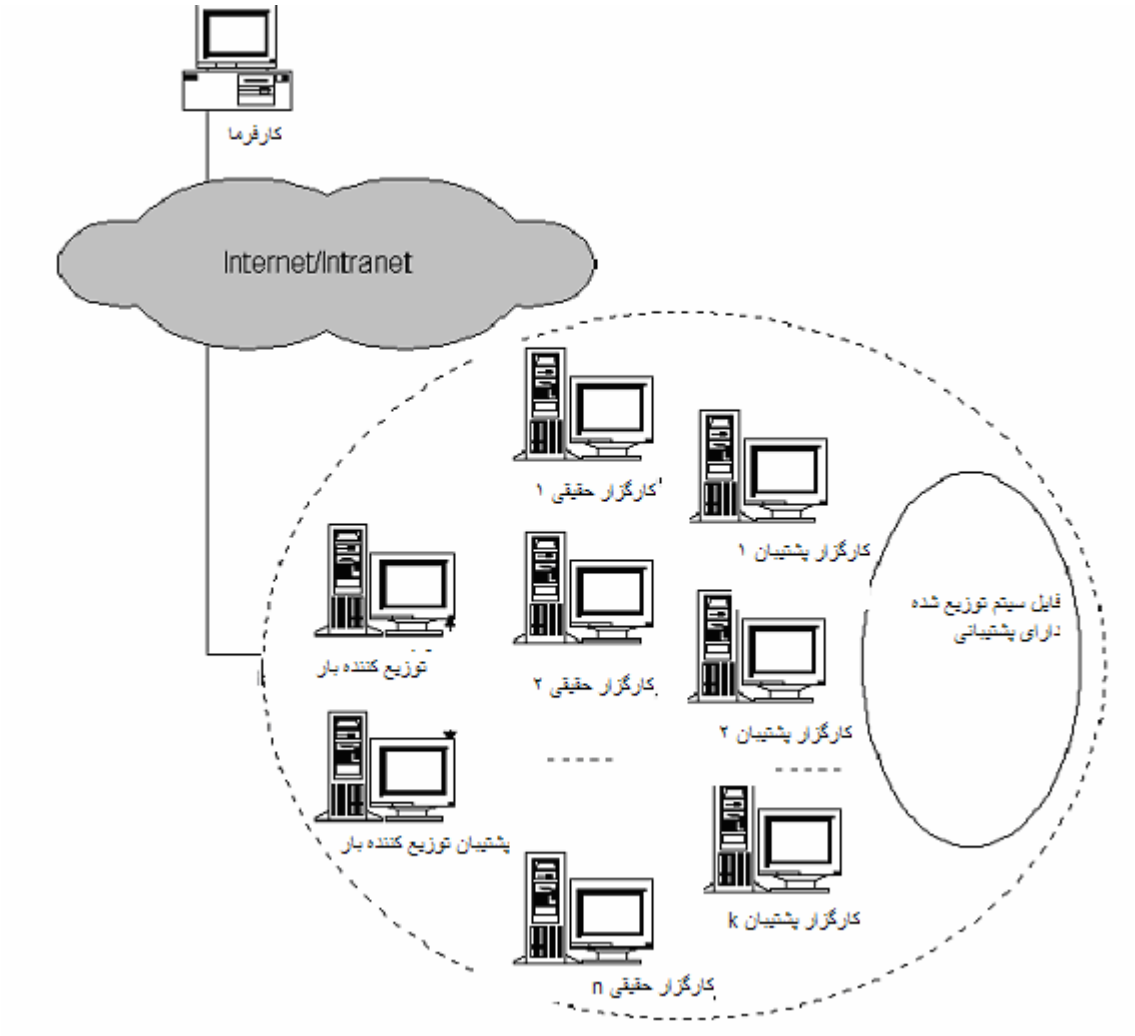
دو مورد بسیار مهمی که در مورد توزیع کننده بار وجود دارد یکی روش کاری توزیع کننده بار در توزیع بار است و دیگری الگوریتمی است که توزیع کننده بار به وسیله آن تشخیص می‌دهد که کدامیک از کارگزارها را برای پاسخگویی به درخواست آمده انتخاب کند.

برای الگوریتم انتخاب، روشهای زیادی می‌توانند استفاده شوند که ساده ترین آنها انتخاب چرخشی بین کارگزاران است. از جمله دیگر الگوریتمها انتخاب وزندار بین کارگزاران، انتخاب کارگزاری که کمترین اتصال در حال حاضر با او برقرار است، انتخاب وزندار کارگزاری که کمترین اتصال در حال حاضر با او برقرار است و ... می‌باشد که انتخاب بین استفاده از این روشها بر اساس شرایط صورت می‌پذیرد.

در مورد روش‌های کاری توزیع کننده بار به دلیل اهمیت موضوع، در قسمت بعد به طور کاملتر بحث خواهد شد.

۳. محل ذخیره سازی

محل ذخیره‌سازی، مکان مشترکی برای ذخیره‌سازی داده‌های مورد نیاز کارگزاران است تا همه آنها بتوانند به داده‌های یکسانی دسترسی داشته باشند و خدمات یکسانی ارایه کنند.



شکل ۳: ساختار سه لایه سیستم کارگزار

در این ساختار هر زمان که یکی از کارگزاران حقیقی از کار بیفتد، در صورتی که دستگاه پشتیبان غیر فعال در سیستم موجود باشد، این دستگاه پشتیبان جای دستگاه از کارافتاده را در سیستم می‌گیرد و در این صورت تغییری در کار توزیع کننده صورت نمی‌گیرد. در صورتی که دستگاه پشتیبان غیرفعال در سیستم موجود نباشد، در صورت از کارافتادن یک کارگزار حقیقی، توزیع کننده بار، دیگر درخواستی را به دستگاه از کار افتاده نمی‌فرستد و بدین ترتیب وظایف دستگاه از کارافتاده بین سایر کارگزاران تقسیم می‌شود.

از جمله دیگر مواردی که باید در این ساختار در نظر گرفته شود این است که باید یک دستگاه پشتیبان برای توزیع کننده بار و یک دستگاه پشتیبان برای محل ذخیره‌سازی وجود داشته باشد که در صورت از

کارافتادن هریک از این دو دستگاه، دستگاه پشتیبان جایگزین آن شود. پشتیبان محل ذخیره سازی باید تمامی داده‌هایی را که محل ذخیره‌سازی در اختیار دارد در اختیار داشته باشد و در حقیقت یک سیستم آینه[۱] برای محل ذخیره سازی باشد. البته برای محل ذخیره سازی می‌توان از روشهای دیگری نیز استفاده کرد.

بدین ترتیب در این ساختار برای از کارافتادن تمامی دستگاه‌ها راه‌حلهایی پیش‌بینی شده است و احتمال پاسخ نگرفتن از این سیستم تا حد ممکن پایین آمده است.

روش کار توزیع کننده بار

عمل توزیع بار در لایه‌های مختلف TCP/IP از جمله در لایه IP و لایه کاربرد[۲] قابل انجام است. روشهایی که در لایه IP توزیع بار را انجام می‌دهند از جهت کارایی و سادگی به روشهای انجام شونده در لایه کاربرد غلبه دارند. در این قسمت تنها بعضی از روشهای مختلف توزیع بار در لایه IP بررسی خواهد شد.

- **NAT**[3]

با توجه به کوتاهی اندازه IP در IPV4 و همچنین به دلیل پاره‌ای از مسایل امنیتی، بسیاری از شبکه ها در داخل خود از نشانی‌های IP استفاده می‌کنند که خصوصی بوده و در اینترنت قابل استفاده نمی‌باشند[۴]. نیاز به ترجمه این آدرس‌های خصوصی به آدرسهای قابل استفاده در اینترنت زمانی مشخص می‌شود که این دستگاه‌ها قصد اتصال به اینترنت را داشته باشند. به این عمل تغییر سرآیند IP بسته‌ها و تعویض نشانی‌های IP، NAT گفته می‌شود.

در ساختار توزیع باری که توسط روش NAT انجام می‌شود، توزیع کننده بار، مسیریاب[۵] شبکه ای است که کارگزاران حقیقی در این شبکه قرار دارند. هنگامی که یک کارفرما[۱] اتصال جدیدی با سیستم کارگزار

---

[1] Mirror

[2] Application Layer

[3] Network Address Translation

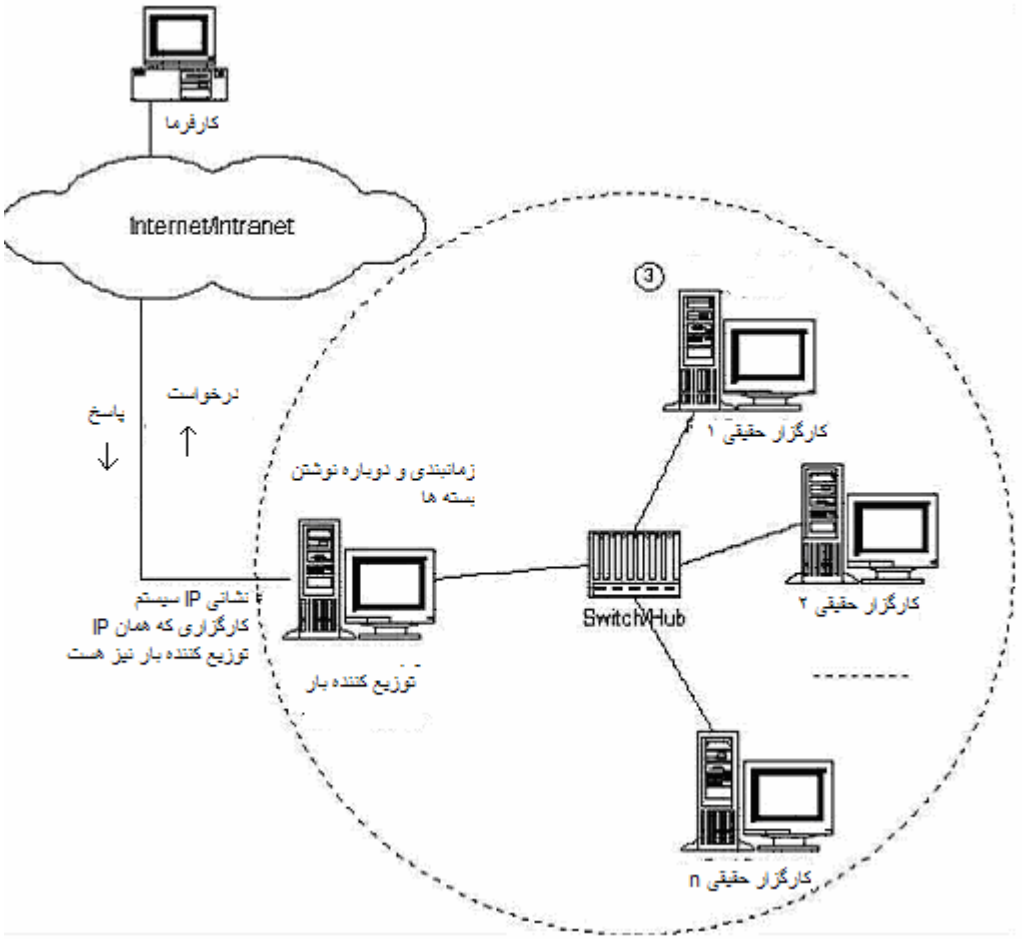
[4] IP های خصوصی در این محدوده‌ها قرار دارند: 10.0.0.0 به عنوان یک محدوده از کلاس A ، 172.16.0.0 تا 172.31.0.0 به عنوان شانزده محدوده از کلاس B و 192.168.0.0 تا 192.168.255.0 به عنوان دویست و پنجاه و شش محدوده از کلاس C

[5] Router

برقرار می‌کند–کاربر از وجود چند دستگاه کارگزار بی‌خبر است و درخواست خود را به توزیع کننده بار که تصور می‌کند کارگزار است می‌دهد–، درخواست به دست توزیع کننده بار می‌رسد. توزیع کننده بار توسط یک روش مشخص انتخاب می‌کند که کدام‌یک از کارگزارهای حقیقی باید به این درخواست پاسخ دهد. پس از آن IP و Port فرستنده بسته و آدرس کارگزار حقیقی را در جدولی که در اختیار دارد اضافه می‌کند و بسته را به کارگزار حقیقی انتخاب شده می‌فرستد. از این پس اگر بسته‌های دیگر این اتصال از جانب کارفرما فرستاده شوند، توزیع کننده بار از اطلاعات موجود در جدول تشخیص می‌دهد که باید بسته را به دست کدام کارگزار حقیقی بدهد.

از طرف دیگر هر زمانی که بسته‌ای از جانب کارگزار حقیقی به سمت کارفرما فرستاده شود، به دست توزیع کننده بار که مسیریاب شبکه نیز می‌باشد می‌رسد. مسیریاب شبکه عملیات NAT را بر روی بسته انجام می‌دهد و IP فرستنده بسته را از IP کارگزار حقیقی، به آدرس خود که همان IP است که کارفرما می‌شناسد، تغییر می‌دهد. هنگامی که یک اتصال بسته می‌شود نیز اطلاعات درج شده در جدول که مربوط به این اتصال است حذف می‌شود. ساختار این روش در شکل ۴ آمده است.

[1] Client



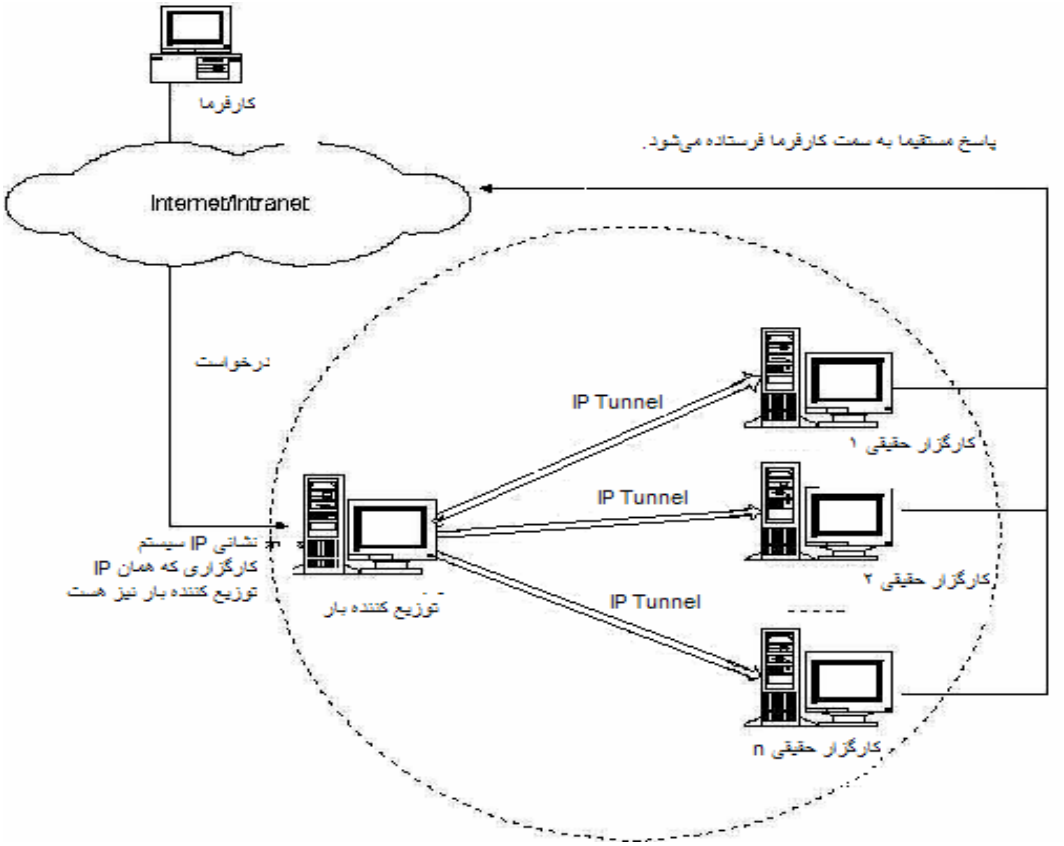
شکل ۴: توزیع کننده بار به وسیله NAT

- **IP Tunneling**

IP Tunneling یا IPIP Encapsulation ، روشی برای قرار دادن یک بسته IP در داخل یک بسته IP دیگر می‌باشد. به وسیله IP Tunneling می‌توان بسته‌ای را داخل بسته دیگر قرار داد–در حقیقت به سر یک بسته کامل، سرآیند[1] IP متصل می‌شود– تا گیرنده بسته بیرونی آن را به دست صاحب آدرس بسته داخلی که شاید باز هم خودش باشد، برساند. یکی از مهمترین استفاده‌هایی که از IP Tunneling در شبکه می‌شود، استفاده از این روش در ساختن VPN ها می‌باشد[3].

[1] Header

در این روش کارگزاران اصلی ممکن است با توزیع کننده در یک شبکه محلی قرار نداشته باشند و از طریق اینترنت یا شبکه های دیگر توانایی ارتباط با توزیع کننده بار را داشته باشند. کارگزاران همچنین باید دارای قابلیت استفاده از IP Tunneling بوده و به Tunneling Device آنها که یک دستگاه مجازی است، IP توزیع کننده بار که IP سیستم کارگزار است، نسبت داده شود.



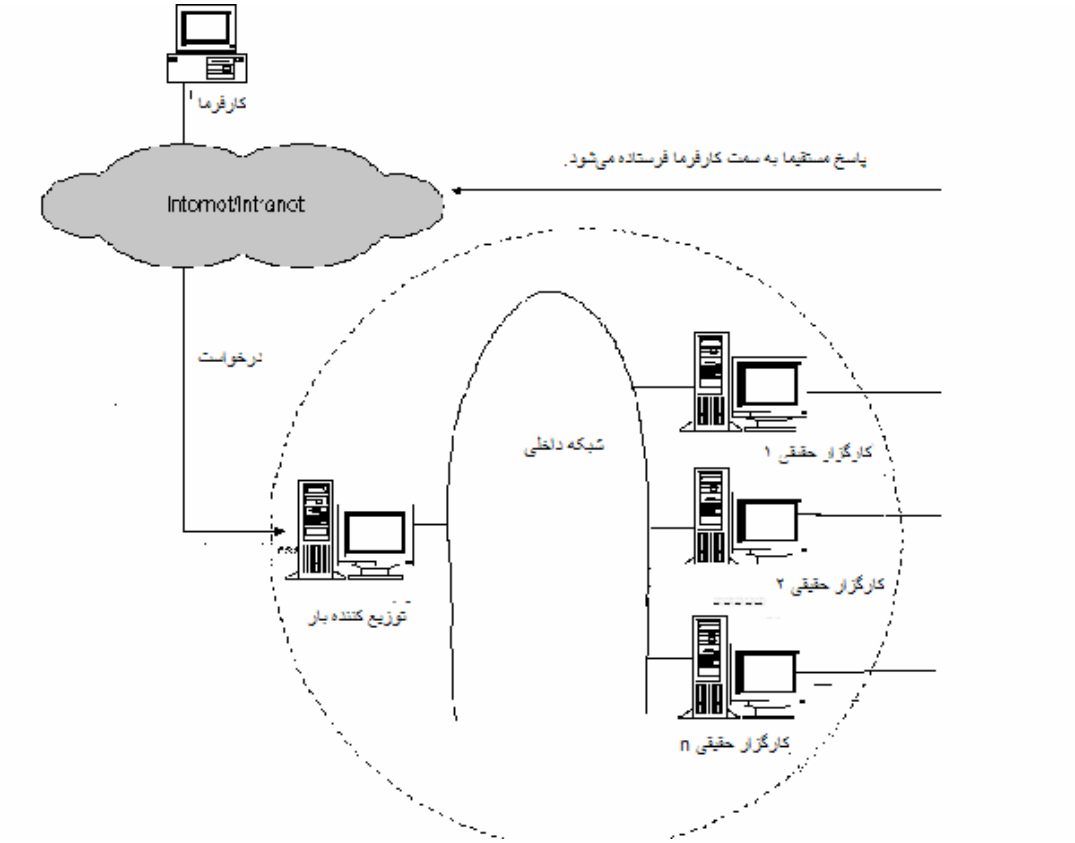
شکل ۵: توزیع بار به وسیله IP Tunneling

نحوه استفاده از این روش بدین ترتیب است که کارفرما بسته‌های خود را به دست توزیع کننده بار که از دید آنها همان کارگزار است، می‌رسانند. توزیع کننده بار با استفاده از روشی یکی از کارگزاران حقیقی را برای پاسخگویی به این کارفرما انتخاب می‌کند. پس از آن بسته دریافتی را در داخل سرآیند IP دیگری که به نشانی IP کارگزار اصلی ساخته شده است قرار می‌دهد و این بسته را به سمت کارگزار اصلی می‌فرستد. کارگزار اصلی پس از دریافت بسته و کنار زدن سرآیند IP اول و دریافت بسته دوم، مشاهده می‌کند که نشانی این بسته منطبق با IP نسبت داده شده به Tunneling Device خودش است و در نتیجه تشخیص

می‌دهد که بسته برای او آمده است و بسته را پردازش می‌کند و بسته یا بسته‌های شامل پاسخ درخواست را مستقیما به کارفرما می‌فرستد. ساختار این روش در شکل ۵ آمده است.

- **Direct Routing**

در این روش یکی از کارتهای شبکه توزیع کننده بار و کارگزاران حقیقی در یک شبکه داخلی قرار داده می‌شوند و از طریق HUB و یا Switch توانایی برقراری ارتباط با یکدیگر را خواهند داشت. IP توزیع کننده بار و IP که برای alias دستگاه مجازی loopback [1] در کارگزارهای حقیقی قرار داده می‌شود یکسان و برابر IP است که می‌خواهیم کاربران از بیرون، سیستم کارگزاری را با آن بشناسند.



شکل ۶: توزیع بار به وسیله Direct Routing

[1] می‌توان این IP را برای NIC(Network Interface Card) دیگری غیر از loopback هم قرار داد، فقط لازم است که به این device به گونه ای تنظیم شده باشد که جواب درخواستهای ARP را ندهد.

هر بسته‌ای که از جانب کارفرمایی فرستاده شود، به دست مسیریاب شبکه داخلی که توزیع کننده بار و کارگزاران حقیقی در آن قرار دارند، می‌رسد. مسیریاب برای فرستادن بسته به دست صاحب آن، باید آدرس سخت افزاری صاحب بسته را بر روی بسته قرار داده و بسته را بر روی HUB و یا Switch بفرستد. چون کارگزارهای حقیقی این IP را بر روی loopback alias خود قرار داده‌اند و این دستگاه مجازی جواب درخواستهای ARP را نمی‌دهد–و در حقیقت اصلا دارای نشانی سخت افزاری نمی‌باشد– و از طرفی توزیع کننده بار نیز دارای این IP است و جواب درخواستهای ARP را هم می‌دهد، پس مسیریاب شبکه نشانی سخت افزاری توزیع کننده بار را بر روی بسته قرار می‌دهد و بسته را بر روی HUB و یا Switch می‌فرستد و در عمل بسته تنها به دست توزیع کننده بار می‌رسد.

توزیع کننده بار پس از دریافت بسته تشخیص می‌دهد که بسته را به دست کدامیک از کارگزارهای حقیقی بدهد. پس از انتخاب کارگزار حقیقی، آدرس سخت افزاری کارت شبکه آن کارگزار را بر روی بسته قرار می‌دهد و بسته را بر روی شبکه می‌فرستد. کارگزار مورد نظر بسته را دریافت می‌کند، آن را پردازش می‌کند و جواب کارفرما را نیز از طریق مسیریاب شبکه داخلی به دست او می‌رساند. ساختار این روش در شکل ۶ آمده است.

در جدول ۵–۲ مقایسه ای میان روشهای مختلف کار توزیع کننده بار صورت گرفته است.

	NAT	TUN	DR
ملزومات کارگزار	هر نوع کارگزار	امکان داشتن Tunneling	داشتن device که به ARPپاسخ ندهد.
شبکه کارگزار	خصوصی	LAN/WAN	LAN
تعداد کارگزارها	۱۰ تا ۲۰	به تعداد دلخواه	تعداد زیاد(معمولا کمتر از ۱۰۰)
مسیریاب شبکه	توزیع کننده بار	مسیریاب شبکه	مسیریاب شبکه

جدول ۵–۲

زمانبندی اتصال

در این قسمت چهار روش برای انتخاب کارگزار مورد نظر برای هر اتصال جدید مطرح می‌شود که توزیع کننده بار می‌تواند بر اساس یکی از این روشها عمل کند. این روشها در ادامه به تفصیل شرح داده می‌شوند.

- **زمانبندی چرخشی**

در این الگوریتم اتصالات بر اساس روش چرخشی به کارگزاران مختلف ارسال می‌گردد. در این روش بدون توجه به تعداد اتصالات هر کارگزار و یا زمان پاسخگویی هر کارگزار، با تمامی کارگزاران به شیوه یکسان برخورد می‌شود.

- **زمانبندی چرخشی وزن‌دار**

در این روش مشابه روش زمانبندی چرخشی عمل می‌شود با این تفاوت که در این روش با کارگزاران مختلف بر اساس وزنی که به آنها نسبت داده می‌شود برخورد می‌شود. این وزن در حقیقت قدرت پردازش کارگزار را در مقابل سایر کارگزاران نشان می‌دهد. تعداد بسته‌های دریافتی هر کارگزار در واقع در نسبت مستقیم با وزن نسبت داده شده به آن کارگزار قرار دارد و هر کارگزار دارای وزن بیشتری باشد بسته‌های بیشتری در مقابل کارگزاران با وزن کمتر دریافت می‌کند. لازم به ذکر است که روش زمانبندی چرخشی حالت خاصی از این روش است که وزن تمامی کارگزاران ۱ در نظر گرفته شده باشد.

- **زمانبندی با کمترین اتصال**

در این روش بر اساس تعداد اتصالاتی بسته‌نشده‌ای که با هر کارگزار موجود در مجموعه برقرار شده است عمل توزیع صورت می‌گیرد. به این شکل که بسته دریافتی از اتصال جدید به کارگزاری که کمترین اتصالات بسته‌نشده را دارا می‌باشد سپرده می‌شود.

- **زمانبندی با کمترین اتصال وزن‌دار**

این روش مشابه روش زمانبندی با کمترین اتصال است با این تفاوت که در این روش بر اساس قدرت پردازش کارگزاران، وزنی نیز به هر کارگزار نسبت داده می‌شود. عمل توزیع در این روش بر اساس ترکیب کمترین اتصال با وزن هر کارگزار صورت می‌گیرد.

۵-۴ شبکه‌های به اشتراک گذارده شده

هر منبع به اشتراک گذارده شده‌ای یک منبع بالقوه برای تاخیرهای زمان پاسخ برنامه‌های کاربردی می‌باشد. هنگامی که دستگاه‌های اشتراکی در یک محیط مشتری-کارگزار در نظر گرفته می‌شوند، شبکه ارتباطی شاید نخستین دستگاهی باشد که به ذهن می‌رسد. شبکه شلوغ از آنجایی که معمولا کندترین و مشکل-سازترین دستگاه اشتراکی در یک سیستم توزیع شده می‌باشد، می تواند تاثیر زیادی بر زمان پاسخ برنامه-های کاربردی داشته باشد.

پارامترهای زیادی می‌توانند در کارایی واقعی شبکه تاثیر بگذارند. به طور خلاصه این پارامترها عبارتند از:

- **طراحی برنامه:** چه زمانی، کی و چگونه برنامه کاربردی از شبکه استفاده می‌کند؟
- **رسانه برنامه کاربردی:** چه چیزی انتقال می‌یابد؟ به عنوان مثال: متن یا صدا یا تصویر و یا فیلم.
- **نرم افزار شبکه:** پروتکل‌های شبکه‌ای که مورد استفاده قرار می‌گیرند.
- **رسانه انتقال:** مسیر فیزیکی که مورد استفاده قرار می‌گیرد. به عنوان مثال: twisted pair یا کابل Coaxial یا فیبر نوری و یا انتقال بی‌سیم.
- **دستگاه‌های شبکه‌ای و توپولوژی:** معماری قالب شبکه و ارتباط مابین دستگاه‌های شبکه که شبکه مورد نظر را تشکیل می‌دهند.

۵-۴-۱ تاخیرهای شبکه

شبکه‌های ارتباطی بعد جدیدی را به طراحی برنامه‌های کاربردی معرفی کردند. هر شبکه از آنجایی که یک منبع به اشتراک گذارده شده می‌باشد، می‌تواند منبعی برای تاخیرهای تصادفی، خطای برنامه‌ای غیر قابل انتظار، از دست دادن داده و یا حتی از کار افتادن کل سیستم باشد. در این ارتباط برخی موضوعات و مفاهیم که درک بهتری از شبکه به دست می‌دهد، در ادامه خواهد آمد.

- **پهنای باند:**

پهنای باند شبکه نرخ ممکن جریان داده میان دو نقطه از شبکه می‌باشد. این پارامتر تابعی از رسانه انتقال و پروتکل مورد استفاده می‌باشد. متاسفانه هیچ شبکه‌ای داده‌های برنامه‌ای را در بیشترین سرعت ممکن برای خود انتقال نمی‌دهد.

اگر اطلاعات در مورد شبکه‌ها ناقص باشد، ممکن است تصور شود که می‌توان زمان ارسال در شبکه را به سادگی با تقسیم کردن تعداد بایتهای برنامه بر پهنای باند به دست آورد. البته این درست است که داده، هنگامی که منتقل می‌بود با این نرخ مشخص منتقل می‌شود اما، زمانهایی نیز وجود دارد که داده‌ای منتقل نمی‌شود و همچنین تمامی داده‌های که منتقل می‌شوند نیز داده‌های مفید نیستند بلکه مقداری از آنها سربار پروتکل‌های شبکه‌ای می‌باشد.

- **محدودیت‌های پروتکل**

پروتکل‌های انتقال، نرخ واقعی را که داده میان دو دستگاه متصل در شبکه منتقل می‌شود، کنترل می‌کنند. به خاطر محدودیتی که بر روی اندازه بسته‌های فرستاده شونده و زمانهایی که می‌توانند فرستاده شوند وجود دارد، هیچ دو دستگاهی نمی‌توانند که با نرخ واقعی پهنای باند خود ارتباط برقرار کنند و در نتیجه کل شبکه نمی‌تواند به نرخ انتقال داده‌ای که قابلیت آن را دارد برسد.

- **تبدیل پروتکل‌ها و کپسوله‌سازی**

هنگامی که شبکه‌هایی که به طور جداگانه توسعه داده شده‌اند به یکدیگر متصل می‌شوند، پروتکل‌های متفاوت شبکه باید با یکدیگر ارتباط برقرار کنند. به عنوان مثال در یک شبکه تجاری ممکن است داده‌ها با استفاده از TCP/IP یا SNA یا IPX/SPX یا DSLW و یا NetBIOS منتقل شوند. بریج‌ها و مسیریاب‌ها باید یا بسته‌های یک پروتکل را به بسته‌های پروتکل دیگر تبدیل کنند و یا بسته‌های یک پروتکل را به عنوان داده یک پروتکل دیگر در نظر گرفته و عمل کپسوله‌سازی را انجام دهند. تبدیل یک پروتکل به پروتکل دیگر زمان بر است و کپسوله‌سازی نیز سربار را افزایش می‌دهد.

- **تاخیر دستگاه**

مهم نیست که چه کاری انجام می‌شود که نرخ خروجی برنامه بالا برود، در هر حال انتقال داده، تاخیر را نیز به همراه دارد. مقداری که می‌توان سرعت انتقال داده را بالا برد محدود است و محدودیت آن به محدودیت سرعت نور بازمی‌گردد. این موضوع زمان می‌برد که داده را به وسیله یک اتصال ماهواره‌ای

منتقل کرد، به عنوان مثال ۱۰۰ ms برای فرستادن داده در کل ایالات متحده آمریکا. حتی اگر بخواهیم به شکل کاملتری بررسی کنیم، هر شبکه تجاری از قسمتهای بسیاری تشکیل شده است که به وسیله هابها، بریجها، مسیریابها و سوییچها متصل شده‌اندو هریک از این دستگاه‌ها نیز تاخیری را دارا می‌باشند که به طور معمول بیش از زمان انتقال است. در نتیجه یک زمان حداقل نیاز است تا پیغام ها از قسمتهای مختلف گذشته به نقاط دیگر برسد. این پدیده یک کران پایین برای زمان سرویس شبکه تولید می‌کند.

- **کارگزاران اشباع شده**

اگر یک دستگاه در شبکه (به عنوان مثال، یک هاب، مسیریاب یا کارگزار) واقعا تحت فشار زیادی قرار گرفته باشد، بسته‌های بسیاری را که باید سرویس دهد از دست می‌دهد. اگر مقدار باراضافه اعمال شده، بسیار زیاد نباشد، سخت‌افزار و نرم‌افزار شبکه معمولا می‌توانند این مشکل را به صورت پنهان از دید برنامه حل کنند. هرچند که این کار شاید چندان ارزان تمام نشود و به ازای بسته‌های از دست رفته ترافیک پیغام-های خطا و دوباره فرستاده شدن بسته‌ها ایجاد شود.

- **اداره کردن خطاها**

مانند تصادفاتی که در بزرگراه‌ها رخ می‌دهد، خطاهای دستگاه‌ها در یک شبکه باعث مشکلات جدی می-گردند. از کارافتادن یک خط ارتباطی باعث مسیریابی داده‌ها از یک مسیر دیگر و بالارفتن داده های انبوه شده در قسمتی دیگر از شبکه می‌شود. دستگاه‌های دارای مشکل می‌توانند داده‌هایی را گم کرده و از دست بدهند و یا شبکه را با فرستادن بسته‌های نادرست و زاید اشباع کنند. معمولترین دلیل ایجاد خطاها در شبکه‌ها عبارتند از:

- خطاهای زمانی که توسط دستگاه‌های شبکه که به نادرستی پیکر بندی شده‌اند ایجاد می‌شود.
- خطاهایی که به خاطر ساختاربندی شبکه بروز می‌کنند و علت آنها تجاوز از بیشینه فاصله مجاز می‌باشد.
- پارازیت‌هایی که توسط اتصالات فیزیکی آسیب دیده و یا سایر مشکلات سیم بندی ایجاد می-شوند.
- ترافیک بیش از حدی که توسط استفاده نامناسب از پروتکل‌های مونیتورینگ دستگاه‌ها ایجاد می-گردد.

۵-۵ فشرده سازی داده

در گذشته اصلی‌ترین هدف فشرده سازی ذخیره فضای دیسک‌ها بود. این یک موضوع بسیار مهم برای کاربران PC ها بود که دیسکهای سخت و دیسکتهای آن فضای کافی را دارا نبود.

از نظر تاریخی از فشرده سازی بجز در شبکه‌های عمومی تلفن برای بالا بردن سرعت انتقال داده ، در هیچ مورد دیگری برای کمتر کردن زمان استفاده نشده بود. در حقیقت زمانی که پردازنده‌ها کند بودند و منابع پردازشی کمیاب، مقدار پردازشی که برای فشرده‌سازی داده و از حالت فشرده درآوردن آن لازم بود یک نگرانی مهم در سرراه فشرده سازی بود.

امروزه استفاده از فشرده سازی داده تغییر کرده است. هرساله هزینه ذخیره‌سازی داده‌ها کاهش می‌یابد و در مقابل سرعت پردازنده‌ها نیز افزایش می‌یابد. حجم داده‌های مورد استفاده توسط برنامه‌ها نیز افزایش یافته است. با درنظر گرفتن تمامی این شرایط هنوز هم در بسیاری از سازمانها، خری پردازنده‌های اضافی برای فشرده‌سازی داده در مقابل خرید حافظه پایدار پرسودتر است.

اما اگر از جهت کارایی به قضیه نگاه شود، حتی موارد مهم‌تری نیز یافت می‌شود. از آنجایی که مشخص است، پیشرفتهای اخیر تکنولوژی باعث رشد کامپیوترها و سریعتر شدن آنها گشته است. اما متاسفانه سرعت دستگاه‌های ذخیره‌ساز و شبکه‌ها-به طور خاص WAN – همگام با افزایش ظرفیت نگهداری داده و سرعت پردازش افزایش نیافته است.

در دنیای تجاری محاسبات توزیع شده که غالبا نیاز است حجم زیادی از داده‌ها از میان خطوط انتقالی WAN ها که به نسبت کند می‌باشد عبور کند، فشرده‌سازی داده به شدت دارای جذابیت شده است البته نه از بعد ذخیره فضا بلکه به خاطربالابردن کارایی.

هنگامی که ارزش فشرده‌سازی داده مورد بررسی قرار می‌گیرد، بعضی موارد که باید در نظر گرفته شوند به شرح زیر می‌باشند:

- **سرعت منبع:**

هرچقدر که رسانه کندتر باشد، مزایای کارایی بیشتری از به کاربردن فشرده‌سازی با استفاده از پردازنده‌ها که سریع می‌باشند، به دست می‌آید. به عنوان مثال فشرده‌سازی اطلاعات برای ذخیره شدن بر روی دیسک

و یا فرستاده‌شدن بر روی شبکه، در این زمان با توجه به نسبت سرعت پردازنده‌ها و سرعت دیسک و شبکه بسیار به صرفه است.

- **مقدار استفاده از منبع:**

هنگامی که یک منبع به اشتراک گذارده شده، به شدت مورد استفاده قرار می‌گیرد، کمتر کردن مقدار استفاده از آن حتی به مقدار کم می‌تواند تاخیر آن را به شدت و به شکل بسیار موثری کاهش دهد. به عنوان مثال اگر یک شبکه به شدت تحت ارسال بار باشد، فشرده‌سازی پیغام‌ها به یک–سوم و یا یک–چهارم طول آنها می‌تواند مقدار قابل توجهی درگیری بر سر استفاده از یک منبع را کاهش داده و در نتیجه کاهش قابل توجهی در زمان انتظار تولید کند.

- **واحد اندازه کار:**

همانگونه که در اصل کارایی مورد بحث قرار گرفته است، هر زمانی که نیاز است عملیات محاسباتی صورت گیرد، همواره یک هزینه برپاسازی وجود دارد. برای بلاک‌های کوچک داده ، سربار به کاربردن روشهای فشرده‌سازی، شاید ارزش ذخیره فضا یا زمان انتقال را نداشته باشد اما برای بلاک‌های بزرگ‌تر، فشرده‌سازی همواره پهنای باند شبکه‌را ذخیره می‌کند و معمولا زمان پاسخ را نیز کاهش می‌دهد.

- **روش فشرده‌سازی:**

روش‌های فشرده‌سازی به دو دسته Lossless و Lossy تقسیم می‌شوند. در دسته اول هیچ داده‌ای از بین نمی‌رود و بیت به بیت داده حفظ می‌شود. در روش دوم که عملا به تصاویر، صوت و ویدیو اعمال می-شوند، بسته به انتخاب فشرده‌ساز مقداری از داده از دست می‌رود. در روشهای نوع دوم یک سبک–سنگین کردن میان کارایی و کیفیت داده مورد انتقال وجود دارد. یک مثال مناسب برای روشهای نوع دوم JPEG می‌باشد که به طور گسترده‌ای برای انتقال تصاویر گرافیکی در اینترنت استفاده می‌شود و با پایین‌تر آوردن کیفیت تصویر، فشرده‌سازی را انجام می‌دهد.

- **نوع داده:**

بعضی از انواع داده به نسبت انواع دیگر به طور ساده‌تری قابل فشرده سازی می‌باشند:

  - داده‌های متنی به طور کارایی فشرده می‌شوند که معمولا در حدود ۵۰ درصد می‌باشد.

- برنامه‌های اجرایی در حدود ۸۰ یا ۹۰ درصد اندازه خود فشرده می‌شوند.
- گرافیکها در حدود ۷۵ تا ۸۵ درصد از اندازه خود فشرده می‌شوند.

۵-۶ پایگاه داده‌های مشترک

یک علت اصلی برای استفاده کردن از DBMS ، به اشتراک گذاردن داده میان تعدادی از کاربران است. به اشتراک گذاردن شامل حفظ صحت داده‌ها با در نظر گرفتن کاربران همزمانی است که عملیات خواندن و نوشتن را بر روی داده‌ها انجام می‌دهند. به منظور اینکه از دستیابی برنامه‌های همزمان به اطلاعات نادرست جلوگیری شود. DBMS ها از مکانیزمهای مختلفی استفاده می‌کنند تا برنامه‌های مختلفی را که از پایگاه داده استفاده می‌کنند به صورت جدا از هم نگاه دارند.یکی از متداولترین مکانیزمها، مکانیزم مبتنی بر Lock می‌باشد. هنگامی که تمامی برنامه‌هایی که احتیاج به به‌روز سازی داده‌ای دارند، باید ابتدا Lock انحصاری را بر روی داده بگیرند، تمامی به‌روزسازی‌ها به صورت پشت سر هم انجام می‌گیرند. این نحوه انجام عملیات، امکان اینکه دو برنامه همزمان که داده‌ای را به‌روز می‌سازند خروجی نادرستی ایجاد کنند را از میان می‌برد.

حتی با وجود یک پردازه به‌روزساز، همچنان عملیات Lock برای حفظ صحت داده‌های به اشتراک گذارده شده ضروری به نظر می‌رسد. یک برنامه نمی‌تواند به صورت امنی یک صفحه از داده‌ها را بخواند زمانی که یک پردازه دیگر در همان زمان در حال اضافه کردن یک سطر به آن صفحه می‌باشد. برای حل این مشکل بیشتر DBMS ها از Lock اشتراکی هنگام خواندن و از Lock انحصاری هنگام به‌روزرسانی و یا نوشتن استفاده می‌کنند.

هزینه کارایی استفاده از Lock، کاهش در دسترس بودن داده و در نتیجه افزایش زمان پاسخ می‌باشد.

۵-۶-۱ راه‌کارهای Lock کردن در DBMS

یک تفاوت مهم میان Lock ها در DBMS و سایر منابع محاسباتی، این موضوع است که راه‌کار واقعی Lock کردن در DBMS به طور مستقیم تحت کنترل قرار ندارد و به وسیله خود DBMS انتخاب می‌شود و برپایه پارامترهای زیر قرار دارد:

- اندازه Lock مشخص شده(سطر، صفحه و جدول و ...)

- سطح جداسازی مشخص شده برای برنامه(Dirty Read، Cursor Stability، Repeatable Read و ...)
- نوع پردازش(SELECT، UPDATE و ...)
- تعداد آیتم‌های لازم برای Lock شدن در یک درخواست خاص
- مسیر دسترسی انتخاب شده توسط بهینه‌ساز DBMS برای پردازش درخواست(بررسی فضای جدول، بررسی فضای اندیس‌ها، جستجو برای اندیس و ...)

از آنجایی که تنها یک تاثیر غیر مستقیم برروی آنچه واقعا اتفاق می‌افتد، وجود دارد، این موضوع به طور خاص بسیار مهم است که برنامه‌ها و طراحان پایگاه داده متوجه شوند که چگونه DBMS خاص آنها عملیات Lock کردن را پیاده سازی کرده است و چه گزینه‌های طراحی در دسترس می‌باشند.

۵-۶-۲ حالتهای Lock

از قبل به این موضوع اشاره شد که DBMS از Lock اشتراکی هنگام خواندن و Lock انحصاری هنگام به-روزرسانی استفاده می‌کنند.این دو مورد به عنوان حالتهای Lock شناخته می‌شوند. Lock های اشتراکی با Lock های اشتراکی دیگر سازگار می‌باشند-بدین معنی که دو پردازه به صورت همزمان می‌توانند Lock اشتراکی را بر روی یک داده نگاه دارند- و همچنین Lock های اشتراکی و Lockهای انحصاری با یکدیگر ناسازگار می‌باشند. حالت دیگر Lock که متداول می‌باشد، Lock به‌روز رسانی می‌باشد که با Lock های اشتراکی سازگار و با بقیه Lock ها ناسازگار می‌باشد. DBMS همچنین از انواع دیگری از Lock های به نام Intent Lock ها استفاده می‌نماید.Intent Lock ها بر روی جدول و یا فضای جدولی در نظر گرفته می‌شوند. Intent Lock ها شامل سه دسته Intent Lockهای اشتراکی، Intent Exclusive Lock و Intent Exclusive Lock اشتراکی می‌باشند.

جدول ۵-۲ یک ماتریس سازگاری می‌باشد که نحوه تراکنش Lock های مختلف را با یکدیگر مشخص می‌کند. هر سلول جدول نشان می‌دهد که آیا یک Lock گرفته می‌شود یا خیر، در صورتی‌که Lock در حالت سطر درخواست داده شود هنگامی که پردازه دیگری، Lock را در حالت ستون قبلا گرفته باشد.

Lock Mode Requested	None	IS	IX	S	SIX	U	X
I S	y es	y es	y es	y es	y es	y es	y es
IX	y es	y es	y es	y es	y es	n o	n o
S	y es	y es	y es	no	no	no	no
S IX	y es	y es	n o	y es	n o	y es	n o
U	y es	y es	n o	n o	n o	n o	n o
X	yes	n o	n o	n o	n o	n o	n o

جدول ۵-۲

۵-۶-۳ اندازه Lock

DBMS های رابطه‌ای چهار سطح مختلف اندازه Lock را پشتیبانی می‌کنند:

- **Lock های سطری**: این Lock ها در مقابل استفاده همزمان از یک سطر، DBMS را محافظت می‌نمایند.
- **Lock های صفحه‌ای**: این Lock ها تمامی سطرهای موجود در یک صفحه را در مقابل استفاده همزمان حفاظت می‌کنند. اندازه صفحه بسته به DBMS و محیط سیستم عامل معمولا 2K یا 4K می‌باشد.
- **Lock های جدولی**: این Lock ها تمامی یک جدول را در مقابل استفاده همزمان حفاظت می-کنند.
- **Lockهای پایگاه داده**: این Lock ها تمامی جدول‌ها و داده‌ساختارهای یک پایگاه‌داده را در مقابل استفاده همزمان مراقبت می‌کنند.

۵-۶-۴ بن بستها

هنگامی که دو یا چند برنامه Lock هایی را در پردازش خواندن و به‌روز سازی داده‌های یکسان در دست می‌گیرند، بن‌بستها می‌توانند رخ دهند. رخ دادن بن‌بست مانند یکی از بازیهای کودکانه می‌باشد که در آن

یکی از بچه‌ها چوب بیس‌بال را در دست داشته و برای بازی به توپ احتیاج دارد و کودک دیگر توپ را در اختیار داشته و به چوب بیس‌بال احتیاج دارد. هریک به چیزی احتیاج دارد که در دست دیگری است و هیچ اتفاقی رخ نمی‌دهد مگر اینکه یکی از آنها موضوع مورد علاقه دیگری را رها کند. این اتفاق مشابه اتفاقی است که در DBMS رخ می‌دهد. یک پردازه به داده‌ای احتیاج دارد که توسط پردازه دیگری Lock شده است و پردازه دوم که این داده را Lock کرده است به داده‌ای احتیاج دارد که توسط پردازه اول Lock شده است.این اتفاق میان چند پردازه نیز به صورت حلقه می‌تواند رخ دهد.

یک مکانیزم اجرایی مناسب برای بیرون بردن سیستم‌ها از حالت بن‌بست، تشخیص حضور سیستم در حالت بن‌بست می‌باشد به این شکل که در بازه‌های زمانی مشخص وجود بن‌بست در سیستم بررسی می‌شود. با بررسی کردن فرکانس رخ دادن بن‌بستها در یک کارگزار پایگاه داده‌ای و تنظیم کردن زمان میان بازه‌های تشخیص بن‌بست با توجه به این زمان، می‌توان به صورت مناسب عملیات تشخیص بن‌بست را انجام داد.

بهترین روش برای جلوگیری از بن‌بست‌های وابسته به داده واقعی، حل مشکل در منبع مشکل می‌باشد. بررسی کردن تمامی بن‌بستها تا دریافتن یک الگو در آنها و سپس دوباره بررسی کردن استفاده پردازه‌های موجود از جدولهای اشتراکی راه حل مناسبی است که برای جلوگیری از بن‌بست می‌تواند مورد استفاده قرار گیرد. به وسیله این بررسی معمولا می‌توان راهی برای کاهش دادن بن‌بستها با طراحی دوباره منطق برنامه‌های کاربردی یافت.

۵-۶-۵ ارتقا Lock

موضوع مهمی که در مورد انتخاب اندازه Lock ها وجود دارد، هزینه نگهداری اطلاعات مربوط به تمامی Lock هایی است که یک برنامه در اختیار دارد. تعداد بایتهایی که برای نگهداری اطلاعات مربوط به هر Lock نیاز است میان DBMS های مختلف بین ۵۰ تا ۲۵۰ بایت می‌باشد. حال به عنوان مثال اگر یک برنامه از Lock های سطری استفاده کند و DBMS از ۲۰۰ بایت به ازای هر Lock استفاده نماید، در ازای هر ۵۰۰۰ سطری که برنامه Lock کند، 1MB حافظه احتیاج می‌باشد. اگر برنامه دارای تعداد زیادی کاربر باشد، ذخیره‌سازی Lock های این برنامه اثر نامناسب زیادی در کارایی استفاده از حافظه اشتراکی خواهد داشت. به خاطر این جنبه استفاده از Lock، بسیاری از DBMS ها به صورت خودکار درخواست Lock را به سطح بالاتری ارتقا می‌دهند تا تعداد موارد نگهداری Lock کاهش پیدا کند. به عنوان مثال در صورت

Lock کردن سطرهای زیادی از یک جدول توسط یک برنامه، DBMS کل جدول را به صورت ضمنی برای برنامه Lock می‌کند.

۵-۶-۶ سطوح جداسازی برنامه‌ها

در کارگزاران پایگاه داده، تولیدکنندگان DBMS قابلیت‌های جداسازی برنامه‌ها را از یکدیگر قرار می‌دهند تا صحت اطلاعات موجود در پایگاه داده در استفاده‌های همزمان از آن تامین گردد.

به عنوان یک تعریف مناسب‌تر می‌توان گفت که کارگزاران DBMS اجازه سطوح مختلفی را برای جداسازی برنامه‌ها از یکدیگر می‌دهند که می‌توان برای هر برنامه کاربردی این سطح جداسازی را مشخص نمود. سطح جداسازی در حقیقت درجه اجباری بودن صحت اطلاعات در برنامه‌هایی که از پایگاه داده استفاده می‌کنند را کنترل می‌کند. موارد زیر این سطوح جداسازی را ازضعیف به قوی بیان می‌نمایند:

- **Dirty Read**: Lock ها در نظر گرفته نمی‌شوند. برنامه‌ها ممکن است داده‌هایی را بخوانند که به‌روزرسانی شده اند اما هنوز در پایگاه داده Commit نشده‌اند(Dirty Data) و ممکن است به دلایل مختلف از جمله از کارافتادن برنامه به‌روزرسان Commit نشوند.
- **Commited Read**: برنامه‌ها Dirty data ها را نمی‌خوانند و نمی‌توانند Dirty Data های تولید شده توسط برنامه‌های دیگر را بازنویسی کنند.
- **Cursor Stability**: سطری که توسط یک برنامه خوانده شده است تا زمانی که مورد استفاده این برنامه قرار دارد تغییر نخواهد کرد. به عنوان مثال در صورت استفاده از Lock صفحه‌ای، یک صفحه‌ی Lock شده، تا زمانی که صفحه‌ی دیگری مورد استفاده برنامه قرار بگیرد Lock خواهد ماند.
- **Repeatable Read**: این سطح جداسازی باعث می‌شود که تمامی داده‌هایی که خوانده می‌شوند تا زمانی که برنامه Lock کننده Commit یا Abort کند، Lock باقی بمانند.

از دیدگاه ارزیابی کارایی، هرچقدر سطح جداسازی قویتر باشد، پتانسیل رقابت بر سر Lock و تاخیرهای زمان پاسخ بالاتر خواند رفت و در نتیجه برنامه‌های کاربردی باید به صورتی طراحی شوند که از ضعیفترین سطح جداسازی سازگار با درست کارکردن برنامه استفاده نمایند.

۵-۶-۷ مدت زمان نگهداری Lock

در اصل به‌اشتراک گذاری این موضوع مورد بررسی قرار گرفت که هنگامی که دسترسی انحصاری مورد نیاز است، مجموع زمان انتظار و استفاده از منبع مشترک را باید کمینه کرد.زمان استفاده از منبع مشترک در پایگاه‌داده های مشترک مبتنی بر مکانیزم Lock، مدت زمان نگهداری Lock می‌باشد.دو زمینه اصلی وجود دارد که طی آن طراحی برنامه کاربردی می‌تواند به کمینه کردن مدت زمان Lock کمک کند:

- استفاده از گزینه‌های مدت زمان نگهداری Lock در DBMS
- طراحی منطق استفاده برنامه‌های کاربردی از داده‌ها

این صفحه مخصوصا خالی رها شده است

بخش ۶ اصل موازی سازی[1]

در این بخش اصل موازی سازی از اصول مهندسی کارایی نرم‌افزار مورد بررسی قرار می‌گیرد. به طور سنتی هنگامی که برای یک منبع مشترک تقاضای زیادی وجود دارد، از روش تقسیم و حل[2] استفاده می‌گردد. منظور از موازی سازی انجام چند کار به طور همزمان می‌باشد.

از موازی سازی به منظور انجام کار بیشتر در زمان یکسان، و یا انجام کار مشخص در زمان کمتر استفاده می‌گردد. نکته اساسی در استفاده از این اصل، آن است که موازی‌سازی مناسب‌تر برای کارهایی است که اشتراک کمی دارند؛ یعنی اگر کارهای موازی شونده به طور ذاتی اشتراک زیادی با هم داشته باشند، استفاده از این اصل ممکن است حتی به کاهش کارایی منجر گردد.

در واقع با استفاده از موازی سازی دو مشکل عمده وجود دارد:

- **هزینه سربار همگامی**: هزینه اضافه‌ای که برای موازی‌کردن جریان کار (تقسیم کار به چند رشته موازی و سپس پیوستن نتایج) انجام می‌شود.
- **تداخل بین جریان‌های موازی**: جریان‌های موازی ممکن است به داده‌های و نتایج میانی یکدیگر نیاز داشته باشند که این امر موازی سازی را مشکل می‌سازد.

۶-۱ دشواری‌های موازی سازی

در موازی سازی یک فرآیند کاری، در انجام دو دسته فعالیت دشواری عمده وجود دارد:

- **کم کردن سربار شکستن کار و پیوستن نتایج**

  این هزینه سربار شامل موارد زیر می‌باشد:
  - هزینه شکستن کار
  - عوض کردن فعالیت جاری
  - کم شدن اثر نهان سازی
  - تداخل در استفاده از منبع مشترک
  - هزینه‌های همگام سازی

[1] Parallelism Principle

[2] Devide and Conquer

- **تعادل در پخش بار**

تعادل در پخش بار بدین معناست که اگرمثلا ۴ پردازنده داریم، از همه آن‌ها استفاده نماییم. عوامل تاثیر گذار در این امر عبارتند از:

- تغییر در بار کاری
- بزرگ یا کوچک بودن پردازه‌ها
- تقسیم داده‌ها
- پشتیبانی نرم‌افزاری
- پشتیبانی سخت افزاری

به عنوان مثال تغییر در بار کاری بدین معناست که مثلا اگر نسبت استفاده از ورودی/خروجی به استفاده از پردازنده تغییر کرد، فرآیند با هم به طور متناسبی موازی شده باقی بماند.

قانون امدال[1]

بیشینه افزایش سرعتی که با بهینه کردن یک مولفه از سیستم حاصل می‌شود، به درصدی از زمان که آن سیستم در استفاده از آن مولفه می‌گذارد بستگی دارد.

به عنوان مثال اگر یک سیستم ۱۰٪ از پردازنده و ۹۰٪ از دیسک استفاده می‌نماید، با بهینه کردن استفاده از پردازنده حداکثر می‌توان زمان اجرای سیستم را ۱۰٪ کاهش داد.

در زمینه موازی سازی، قانون امدال به طور عمده‌ای کاربرد دارد. استفاده از این قانون، در موازی سازی (در جهت افزایش کارایی) قسمت‌های زیر از سیستم تلاش می‌شود:

- مولفه بیشتر استفاده شونده
- کند ترین مولفه
- مشغول‌ترین مولفه
- قسمتی از برنامه که بیشترین زمان در آن گذاشته می‌شود.

[1] Amdahl Law

به عنوان مثال در سیستم‌های کارگزار–مشتری، معمولا مشکل کارایی به دلیل ارتباطات کند است و نه کند بودن مولفه‌ها. به همین دلیل تلاش برای بهینه کردن کارایی (که یکی از ابزارهای آن استفاده کردن از اصل موازی سازی می‌باشد) باید در آن قسمت بیشتر باشد.

۶–۲ سطوح مختلف موازی سازی

موازی سازی را می‌توان در سطوح مختلفی انجام داد. هر چه سطح موازی سازی بالاتر باشد، به لایه کسب و کار نزدیک‌تر است و عملیات مورد نیاز آشکارتر است. در عوض هر چه سطح موازی سازی پایین‌تر باشد، به سخت افزار نزدیک‌تر است و عملیات مورد نیاز نهان‌تر می‌باشد. این سطوح از بالا به پایین در زیر بررسی مختصر شده‌اند:

۱) موازی سازی بار کاری

به معنای جدا سازی بار کاری برنامه و پردازش توزیع شده آن می‌باشد. در این سطح تلاش در موازی سازی بار کاری یک سیستم، بدون توجه به ماهیت آن سیستم می‌باشد. یک راه‌کار برای این کار، پردازش توزیع شده بار می‌باشد. پردازش توزیع شده هر چند به موازی سازی کمک می‌نماید، مشکلات زیر را نیز دارد:

- تاخیر شبکه و تاثیر در زمان پاسخ
- تقسیم داده‌ها و دسترسی به داده‌های مشترک
- تکرار داده‌ها و نحوه به روز آوری آن‌ها

۲) موازی سازی جریان کاری

به معنای آن است که چگونه بارکاری به تراکنش‌ها تبدیل شده است. این سطح در واقع موازی سازی فعالیت‌ها در لایه کسب و کار می‌باشد. عملیات اصلی در این سطح موارد زیر می‌باشد:

- بازبینی فعالیت‌های کسب و کار
- طراحی جریان‌های کاری (تقسیم جریان کاری به فازهای مجزایی که ممکن است بتوانند به صورت موازی پردازش شوند)

۳) موازی سازی برنامه

به معنای یافتن سطح بهینه همروندی برنامه‌ها می‌باشد. در انجام این کار تعامل با قله‌های بار کاری پخش بار بین پردازنده‌ها از اهمیت بالایی برخوردار است.

به عنوان مثالی از این سطح موازی سازی می‌توان به استفاده از میان افزار در معماری کارگزار–مشتری اشاره نمود. این میان افزار می‌توان فراخوانی از راه دور رویه یا میان افزار مبتنی بر پیغام باشد.

۴) موازی سازی پردازه

که نمونه دستی آن شکستن پردازه به پردازه‌های کوچک‌تر (fork) و همگام سازی نتایج (join) می-باشد.

۵) موازی سازی اداره داده‌ها

به عنوان مثال با استفاده از بهینه‌ساز DBMS، عباراتی که می‌توانند به طور موازی اجرا شوند به طور خودکار تعیین می‌گردند.

۶) موازی سازی دستگاه ورودی و خروجی

استفاده از تکنولوژی RAID و استفاده از پایگاه داده موازی نمونه‌هایی از فعالیت‌های این سطح موازی سازی هستند.

۷) موازی سازی پردازنده

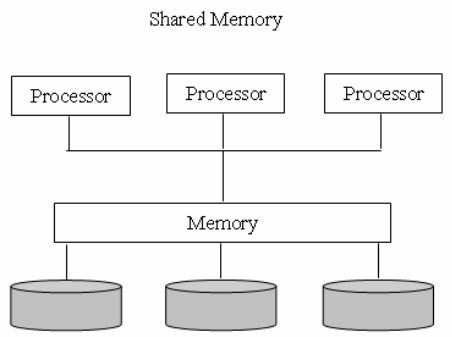
استفاده از کامپایلر موازی ساز به منظور موازی سازی توالی دستورات برنامه نمونه‌ای از آن می‌باشد.

معماری‌های چند پردازنده

در زمینه موازی سازی در سطوح نزدیک به سخت‌افزار کارهای زیادی انجام گرفته است که در این بخش و بخش بعد، به دو نمونه از آن‌ها اشاره می‌گردد. سه معماری چند پردازنده متداول وجود دارند:

- **مدل حافظه مشترک**

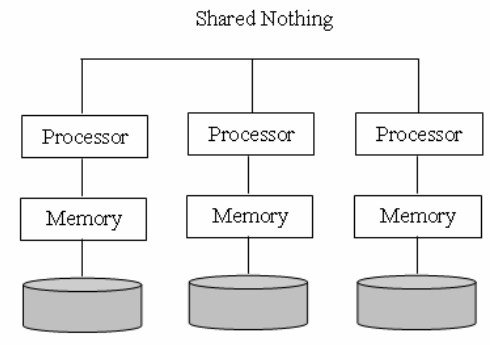
گسترش مدل کامپیوتر تک پردازنده می‌باشد که برای بار کاری با اشتراک زیاد مناسب است. گلوگاه این مدل محل اتصال (باس) می‌باشد که همین موضوع توسعه پذیری کم این مدل (سقف ۳۲ پردازنده) را باعث می‌گردد. شمای کلی این مدل در شکل ۷ نشان داده شده است.



شکل ۷: معماری مدل حافظه مشترک

- **مدل بدون اشتراک**

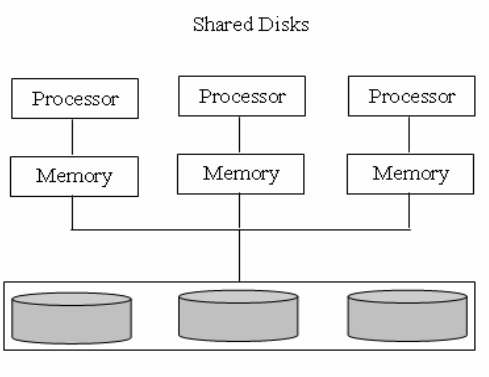
با استفاده از این مدل، موازی سازی در سطوح بالا به سادگی صورت می‌گیرد. نیاز به تعادل در پخش بار مهم‌ترین نکته در مورد این مدل می‌باشد. شمای کلی این مدل در شکل ۸ نشان داده شده است.



شکل ۸: معماری مدل بدون اشتراک

- **مدل دیسک مشترک**

این مدل که ایده اصلی آن یکسان سازی حافظه‌ها می‌باشد برای داده‌های مشترک فقط خواندنی مناسب می‌باشد. شمای کلی این مدل در شکل ۹ نشان داده شده است.



شکل ۹: معماری مدل دیسک مشترک

تقسیم داده‌ها در دیسک‌ها

برای این کار نیز روش‌های مختلفی وجود دارد که چند نمونه از آن‌ها موارد زیر می‌باشد:

- **تقسیم بر اساس schema** (یک جدول روی یک دیسک، یک جدول روی دیسک دیگر)
- **تقسیم بر اساس hash** (استفاده از یک الگوریتم hashing برای یافتن دیسک مورد نظر)
- **تقسیم بر اساس range** (مثلا E بر روی یک دیسک، F-J بر روی دیسک دیگر)
- **تقسیم بر اساس round-robin** (سطر اول در دیسک ۱، سطر دوم در دیسک ۲، ...)

در انتهای بحث اصل موازی سازی باز هم تاکید می‌کنیم که از این اصل تنها در زمانی باید استفاده کرد که در ساختار طبیعی ماهیت توازی وجود داشته باشد. باز هم تاکید می‌کنیم که هزینه سربار همگام سازی، کشنده‌ی موازی سازی می‌باشد.

این صفحه مخصوصا خالی رها شده است

بخش ۷ اصل سبک و سنگین کردن[1]

در این بخش، آخرین اصل از اصول مهندسی کارایی نرم‌افزار مورد بررسی قرار می‌گیرد. تا کنون دیدیم که طراحی نرم‌افزار اهداف مختلفی دارد. این اهداف شامل اهداف اولیه (مانند درستی عملکرد، قابلیت اطمینان بالا، جامعیت داده‌ها، قابلیت حمل، قابلیت استفاده مجدد، سهولت نگهداری) و کارایی بالا (که راه‌کارهای متفاوتی دارد که به برخی از آن‌ها در بخش‌های قبل اشاره گردید) می‌باشد.

همان طور که اشاره گردید اصل اول طراحی نرم‌افزار سادگی می‌باشد و نگاه ما به کارایی، به عنوان یکی از اهداف مهندسی نرم افزار می‌باشد. با توجه به این که راه‌کارهای بررسی شده در بخش‌های قبل ممکن است در جهت خلاف هم قرار بگیرند، دانستن اهمیت هر یک از آن‌ها در جهت انتخاب راه‌کار نهایی طراحی اهمیت بالایی دارد. در این بخش، به ایده اصلی اولویت بندی میان بخش‌های مختلف اشاره می‌گردد و پس از آن، به گروه‌بندی به عنوان یک مثال عملی از سبک و سنگین گردن میان اصول مختلف مهندسی کارایی نرم‌افزار پرداخته خواهد شد.

۷-۱ اولویت بندی

به معنای سبک و سنگین کردن میان بارهای کاری و منابع محاسباتی می‌باشد. در واقع طراح نرم‌افزار باید همیشه طوری از اصول کارایی استفاده نماید که راه‌هایی برای انجام موارد زیر بدست آورد:

- **جانشین‌کردن دستگاه‌های کند با سریع**
- **منحرف کردن کار از دستگاه‌های با بار زیاد**
- **جانشین‌کردن ورودی/خروجی دیسک با پردازنده و** حافظه (مثلا استفاده از حافظه نهان به دنبال رسیدن به این مورد می‌باشد)
- **جانشین‌کردن منابع شبکه با منابع** محلی (مثلا اصل تمرکز به دنبال رسیدن به این مورد می‌باشد)
- **گلوگاه کردن پردازنده** (مثلا فشرده‌سازی داده‌ها به دنبال رسیدن به این مورد می‌باشد)

[1] Tradoff Principle

۷-۲ مثال: گروه بندی و کارایی

گروه‌بندی در جاهای مختلف استفاده می‌گردد. مثال‌هایی از گروه‌بندی موارد زیر می‌باشد:

- گروه‌بندی رویه‌های مرتبط در قالب پیمانه
- گروه‌بندی داده‌ها و رویه‌ها مرتبط در قالب شیء
- گروه‌بندی ستون‌های مرتبط در قالب جدول
- گروه‌بندی عبارات SQL مرتبط در رویه ذخیره شده
- گروه‌بندی فیلدهای مرتبط داده در قالب فرم و گزارش
- گروه‌بندی داده‌های ارسالی در قالب پیغام
- گروه‌بندی داده‌های ذخیره شده در قالب بلوک‌های پایگاه داده

یک مورد اساسی در گروه بندی، اندازه گروه می‌باشد. اندازه گروه در طراحی برنامه، طراحی پایگاه داده و برنامه روی پایگاه داده، طراحی واسط کاربر، ارتباطات داده‌ای و زیر سیستم ورودی/خروجی حائز اهمیت می‌باشد. مثلا در طراحی واسط کاربر، اگر همه اطلاعات یک مشتری را در بالای همه صفحات نشان دهیم، واسط کاربر استانداردی خواهیم داشت، اما در عین حال ممکن است در اکثر صفحات تنها نیاز به نام و تلفن مشتری باشد.

حال برای تعیین اندازه گروه در یک کاربرد خاص، باید مزایا و معایب آن را با توجه به اصول مختلف بررسی کنیم تا در نهایت اندازه گروه مناسب برای آن کاربرد را پیدا کنیم. یک جدول بررسی نمونه در زیر نشان داده شده است.

Group Size and the SPE Design Principles

	Workload	Efficiency	Locality	Sharing	Parallelism
Large group advantages	Can match a large workload	Increased efficiency and throughput	May increase spatial locality		
Large group disadvantages	More chance of wasted work	Less responsive	May reduce functional cohesion	May increase contention	May inhibit parallelism
Small group advantages	Can match a small workload	More responsive	May increase cohesion	More flexibility,better for sharing	More opportunities for parallelism
Small group disadvantages	May not minimize costs	May reduce throughput	May increase coupling	Minor increase in deadlocks	

جدول ۵: بررسی اصول مهندسی کارایی نرم‌افزار در گروه بندی مولفه‌ها

این صفحه مخصوصا خالی رها شده است

بخش ۸ نتیجه گیری

در این گزارش، اصول مختلف مهندسی کارایی نرم‌افزار بررسی گردید. اصول بار کاری، کارایی، و محلی بودن در سطح جزیی‌تر، و اصول به اشتراک‌گذاری و موازی‌سازی در سطح کلان‌تر مورد استفاده قرار می-گیرند.

همان طور که اشاره شد، وابستگی میزان استفاده از این اصول و اولویت‌های میان آن‌ها به هدف طراحی بستگی دارد. به عنوان مثال نهان سازی، مناسب برای کم کردن زمان پاسخ است، هر چند استفاده از حافظه را افزایش می‌دهد.

در انتها بار دیگر تاکید می‌کنیم که مباحث مطرح شده در اینجا، مجموعه تکنیک‌هایی برای کمک به فکر کردن در مورد تصمیمات طراحی هستند و نباید به عنوان یک نسخه طراحی به آن‌ها نگاه کرد.

این صفحه مخصوصا خالی رها شده است

بخش ۹ مراجع

[1] C. Loosley, and F. Douglas, **High Performance Client/Server**. John Willy and Sons, 1998.

[2] C. Lung, A. Jalnapurkar, and A. El-Rayess, **Performance-Oriented Software Architecture Engineering: an Experience Report**. Proc. of the 1st International Workshop on Software and Performance, Santa Fe, Oct. 1998.

[3] V. Cortellessa, A. Di Marco, and P. Inverardi, **Comparing performance models from a software designer perspectivem,** Dipartimento di Informatica, Universit`a di L'Aquila, Italy.

[4] W. Zhang, **Linux Virtual Server for Scalable Network Services**, National Laboratory for Parallel & Distributed Processing, http://www.linuxvirtualserver.org/ols/lvs.pdf.