پایگاه داده های موازی

در این بخش, بحث ما در مورد الگوریتمهای اصلی و پایه ای برای سیستمهای پایگاه داده موازی است که بر اساس مدل داده رابطه ای هستند. مخصوصا, توجه ما بر روی مکان داده هایی است که بر روی چندین دیسک قرار گرفته اند و هدف, ارزیابی موازی عملیات رابطه ای است. هر دو مورد ذکر شده وسیله ای هستند برای دستیابی به موفقیت در پایگاه داده های موازی.

۱-۱۸ مقدمه

در دو دهه قبل, سیستمهای پایگاه داده موازی تقریبا رو به فراموشی بود حتی توسط کسانیکه روزی طرفدار پر و پا قرص آن بودند. اما امروزه آنها با عرضه همین سیستمهای پایگاه داده ای توانسته اند موفقیت را از آن خود کنند.

چندین دلیل برای این موفقیت:

- رشد تراکنشهای مدیریت شده نیازمند استفاده از کامپیوتر بود. علاوه بر با این رشد **WWW** تعداد بسیار زیادی سایت با میلیونها بیننده بوجود آمد, داده های جمع آوری شده حاصل حجم بسیار زیادی از داده ها را برای شرکتها بوجود آورد.
- سازمانها از این داده های رو به رشد برای برنامه ریزی فعالیتهایشان و همچنین قیمت گذاری آن استفاده میکنند. داده هایی از قبیل: مردم چه چیزهایی خریده اند؟ روی چه پیوندهایی کلیک کرده اند؟ چه موقعی تماس گرفته اند؟ و ... . پرس و جو ها برای اهدافی که ((پرس و جوی تصمیم یار)) نامیده میشوند, استفاده میشوند. همچنین داده های مورد نیاز برای هر پرس و جو ممکن است از بین چندین ترا بایت داده انتخاب شوند. سیستمهای تک پردازنده قادر نیستند عملیات مورد نظر را بر روی حجم عظیمی از داده ها با نرخ مطلوبی اجرا کنند.
- Set-oriented های پرس و جوهای پایگاه داده بصورت طبیعی و ذاتا میتوانند بصورت موازی اجرا شوند. تعدادی از سیستم های تجاری و پژوهشی نشان داده اند که قدرت و مقیاس پذیری پردازشهای موازی را دارا هستند.
- از زمانیکه میکرو پروسسورها ارزان شده اند, ماشینهای موازی نیز رایج و نسبتا مقرون به صرفه شده اند.

- پردازنده های شخصی نیز توسط معماری چند هسته ای به ماشین های موازی بدل شده اند.

ص ۷۹۷

همانطور که در فصل ۱۷ بحث شد, موازی سازی برای فراهم کردن و بالا بردن speed up بکار برده میشود, در جاهاییکه پرس و جوها باید سریعتر انجام شوند چراکه منابع زیاد هستند مانند: پردازنده ها و دیسکها. موازی سازی همچنین بمنظور فراهم کردن scale up استفاده میشود. در جاهاییکه افزایش بار کاری وجود دارد, با افزایش درجه موازی سازی, از افزایش زمان پاسخ جلوگیری میشود.

در فصل ۱۷ معماری های مختلفی برای سیستم های پایگاه داده ای بیان شد:

معماری حافظه اشتراکی , دیسک اشتراکی , هیچ چیز مشترک و معماری سلسله مراتبی.

واضح است که در معماری حافظه اشتراکی, همه پردازنده ها یک حافظه و چندین دیسک را به اشتراک میگذارند. در معماری دیسک اشتراکی, هر پردازنده حافظه اختصاصی دارد اما دیسکها مشترک هستند. در معماری هیچ چیز مشترک پردازنده ها نه دیسکی به اشتراک می گذارند و نه حافظه ای. در معماری سلسله مراتبی تعدادی نود داریم که هیچگونه دیسک یا حافظه ای را به اشتراک نمیگذارند, اما هر نود بصورت داخلی یک معماری دیسک اشتراکی یا حافظه اشتراکی دارد.

۱۸-۲ موازی سازی I/O

در ساده ترین شکل, موازی سازی I/O به کاهش زمان مورد نیاز برای بازیابی روابط از دیسک, توسط تقسیم بندی آنها بر روی چندین دیسک, اشاره دارد. متداول ترین حالت تقسیم بندی داده ها در محیطهای پایگاه داده ی موازی, تقسیم بندی افقی است. در تقسیم بندی افقی هر تاپل یک رابطه در میان تعداد زیادی دیسک تقسیم میشود. بنابراین هر تاپل روی یک دیسک قرار میگیرد بجای اینکه در میان دیسکها پخش شود. در اینجا چندین استراتژی تقسیم بندی ارائه شده است:

۱-۲-۱۸ تکنیکهای تقسیم بندی

در این قسمت ما ۳ استراتژی و تکنیک تقسیم بندی برای داده ها بیان می کنیم. فرض می شود که n دیسک وجود دارد(D_0 , D_1, . . . ,D_{n-1}) که داده ها در سراسر این دیسکها تقسیم میشوند.

- *روش Round-Rabin:* در این روش رابطه ورودی اسکن میشود و بترتیبی که داده ها اسکن میشوند, تاپل i ام در دیسک D_i mod n جای میگیرد. این روش, توزیع متوازن تاپل ها را در

سراسر دیسکها تضمین می کند, بطوریکه همه دیسکها دارای تعداد مساوی از تاپل ها خواهند بود.

- *روش Hash-partitioning:* این روش بر اساس یک یا چند ویژگی از رابطه مورد نظر تقسیم بندی را انجام می دهد. به این صورت که یک تابع Hash انتخاب می شود که در محدوده ی (0,1, . . . ,n-1) قرار دارد. برای هر تاپل از رابطه اصلی عمل Hash بر روی ویژگی مورد نظر برای تقسیم بندی(ویژگی تقسیم کننده) انجام می شود. آنگاه تابع عددی مانند i را تولید خواهد کرد و تاپل مذکور در D_i قرار خواهد گرفت.
- *روش Range-partitioning* : این روش تاپلها را توسط اختصاص دادن هر محدوده از صفات به یک دیسک تقسیم بندی می کند. این استراتژی یک قسمت از یک صفت را مثلا A را انتخاب می کند و از یک بردار تقسیم کننده$[v_0 , v_1 , . . . ,v_{n-1}]$ استفاده می کند. اگر i<j آنگاه $V_i<V_j$.

اکنون داده ها به روش زیر تقسیم بندی می شوند:

یک تاپل t را در نظر بگیرید بطوریکه t[A]=x

اگر $x<v_0$ باشد, آنگاه تاپل t در D_0 قرار خواهد گرفت.

اگر $x\geq v_{n-2}$ باشد, آنگاه تاپل t در D_{n-1} قرار خواهد گرفت.

و اگر $v_i\leq x<v_{i+1}$ باشد, آنگاه تاپل t در D_{i+1} قرار میگیرد.

ص۷۹۸

بعنوان مثال اگر ۳ دیسک با شماره های ۰و۱و۲ داشته باشیم با بردار تقسیم کننده ای مانند v[5,40] , آنگاه تاپل هایی با مقادیر بین ۰و۵ در D_0 , تاپل هایی با مقادیری بین ۵ تا ۴۰ در دیسک D_1 و تاپل های با مقادیر بیشتر از ۴۰در D_2 قرار خواهند گرفت.

۲-۲-۱۸ مقایسه تکنیک های تقسیم بندی

هنگامیکه یک رابطه میان چند دیسک تقسیم میشود, میتوان آن را با استفاده از همه دیسک ها بصورت موازی بازیابی کرد. بطور مشابه وقتی یک رابطه تقسیم می شود میتوان آنرا بصورت موازی در چند دیسک نوشت. بنابراین نرخ انتقال برای انجام عمل خواندن یا نوشتن کل یک رابطه توسط موازی سازی I/O ,نسبت به حالتی که از موازی سازی استفاده نشده است, بسیار بالاتر خواهد بود. با این حال خواندن یا اسکن یک رابطه, تنها یک نوع از دسترسی به داده هاست. دسترسی به داده ها میتواند بصورت زیر طبقه بندی شود:

۱. اسکن کل یک رابطه
۲. تعیین محل یک تاپل انجمنی(بعنوان مثال: employee name:''Campbell'') این پرس و جو یک نوع پرس و جوی نقطه ایست (point query). این عمل یک تاپل دارای یک value مشخص را برای یک صفت (attribute) خاص جستجو می کند.
۳. تعیین محل تاپل هایی با یک ویژگی مشخص در یک محدوده خاص. (بعنوان مثال: 1000<salary<20000) این نوع از پرس و جو ها پرس و جوی محدوده ای(range query) نامیده می شود.

روشهای متفاوت تقسیم بندی از این نوع دسترسی ها, با سطوح مختلف از بهره وری , پشتیبانی می کنند.

- Round-Rabin : این روش برای وقتی مناسب است که پرس و جو بر روی کل یک رابطه بصورت ترتیبی انجام شود. این روش برای پرس و جوهای نقطه ای و محدوده ای مناسب نیست و مشکلات زیادی بوجود می آورد, زیرا برای انجام هر پرس و جو باید کلیه ی دیسکها مورد بررسی قرار گیرند.
- Hash-partitioning: این روش بهترین و مناسب ترین روش برای پرس و جو های نقطه ای مبتنی بر صفات تقسیم کننده است. به عنوان مثال اگر یک رابطه بر اساس صفت خاصی مانند-telephone'' number'' تقسیم بندی شده باشد آنگاه می توان به پرس و جوهایی مانند: یافتن رکوردی از کارمندان با شماره تلفنی خاص مثلا"telephone-number=555-3333'' ؛را توسط فرستادن 555-3333 به تابع Hash دیسک حاوی تاپل مورد نظر را براحتی بدست آورد. رفتن مستقیم به دیسک مورد نظر بجای جستجوی همه ی دیسک ها در هزینه راه اندازی یک پرس و جو صرفه جویی می کند. همچنین چون بر روی بقیه دیسکها پردازشی انجام نمی شود, آنها برای پاسخ گویی به پرس و جو های دیگری که ممکن است بصورت همزمان درخواست شوند آزاد هستند.

این روش همچنین برای اسکن ترتیبی کل یک رابطه نیز مناسب است. اگر تابع Hashیک تابع تصادفی خوب انتخاب شده باشد و تقسیم بندی صفات نیز بر روی یک ویژگی کلیدی از رابطه انجام شده باشد, آنگاه تعداد تاپل های اختصاص یافته به هر دیسک, با انحراف کمی, تقریبا با هم برابر است.(توزیع متوازن داده ها در کل دیسک ها). از این رو زمان صرف شده برای اسکن کل یک رابطه, حدود 1/n زمان مورد نیاز برای اسکن کل یک رابطه روی یک سیستم تک دیسکی است.

با این حال, این روش برای پرس و جو های نقطه ای, که بر روی صفاتی غیر از صفات تقسیم کننده ایجاد شده اند, مناسب نیست و خوب جواب نمی دهد. این تکنیک برای پرس و جوهای محدوده ای نیز خوب عمل نمی کند, زیرا تابع Hash معمولا داده هایی را در یک محدوده هستند به یک دیسک واحد

اختصاص نمی دهد و در واقع آنها را پراکنده می سازد. بنابراین برای پاسخ به یک پرس و جوی محدوده ای باید همه ی دیسکها اسکن شوند.

ص۷۹۹

- Range-partitioning: این روش برای پرس و جو های نقطه ای و محدوده ای که بر روی صفت تقسیم کننده ایجاد شده اند بسیار مناسب است و خوب عمل می کند. برای پرس و جوهای نقطه ای میتوان, توسط بردار تقسیم کننده, مکان داده های مورد نظر را یافت. برای پرس و جوهای محدوده ای میتوان, توسط بردار تقسیم کننده, دیسکهایی را که محدوده تاپل های مورد نظر در آنجا ذخیره شده اند یافت. در هر دو مورد جستجو ما را دقیقا به همان دیسکی راهنمایی می کند که تاپل های مورد نظر در آنجا قرار دارند. یکی از مزایای استفاده از این روش اینست که اگر تنها تعداد کمی تاپل در محدوده پرس و جو وجود داشته باشد, آنگاه معمولا بجای اینکه به همه ی دیسک ها مراجعه شود تنها یک دیسک جستجو می شود. از آنجا که دیگر دیسک ها میتوانند پاسخ گوی پرس و جو های دیگری که ممکن است بصورت همزمان درخواست شوند باشند, این روش دارای زمان پاسخ خوب و مطلوبی است. از طرف دیگر اگر تعداد زیادی تاپل در محدوده پرس و جو باشد (مثلا وقتی که پرس و جوی محدوده ای شامل طیف وسیعی از رابطه باشد) آنگاه مجبوریم تعداد زیادی تاپل را از میان تعداد کمی دیسک بازیابی کنیم که در اینصورت در قسمت I/O دیسک ها دچار تنگنا خواهند شد. این حالت پردازش در یک یا چند دیسک چولگی(skew)ایجاد می کند. در این موارد از پرس و جوها روش های Round-Rabin و Hash-partitioning ,بدلیل اینکه کل دیسک ها را درگیر می کنند, نسبت به روش Range-partitioning زمان پاسخ بهتری خواهند داشت.

انتخاب روش تقسیم بندی بر روی دیگر عملگرهای رابطه ای نیز مانند join ,که میتوان در بخش ۵-۱۸ دید, تاثیر دارد. بنابراین انتخاب تکنیک تقسیم بندی همچنین به نوع عملگرهایی که باید بر روی رابطه اجرا شوند بستگی دارد. عموما روشهای Range-partitioning و Hash-partitioning به روش Round-Rabin ترجیح داده می شوند.

در یک سیستم با تعداد زیادی دیسک میتوان با این روش دیسکهایی را که قرار است رابطه را در خود جای دهند انتخاب کرد: اگر یک رابطه حاوی تعداد کمی تاپل باشد بگونه ای که کل آن بر روی یک بلوک جا شود, آنگاه بهتر است کل رابطه را به یک دیسک اختصاص دهیم. رابطه های بزرگ ترجیحا باید در کل دیسک های موجود

پخش شوند. اگر یک رابطه حاوی m بلاک از دیسک باشد و n دیسک در سیستم موجود باشد, آنگاه تعداد دیسکهای مورد استفاده برای جای گیری رابطه برابر با min(m,n) خواهد بود.

۱۸-۲-۳ کنترل چولگی(skew)

هنگامی که یک رابطه تقسیم می شود(توسط روشی غیر از Round-Rabin) ممکن است در توزیع تاپل ها مشکل چولگی پیش بیاید. یعنی ممکن است تعداد بسیار زیادی از تاپل ها بر روی یک دیسک قرار گیرند در حالی که دیگر دیسکها حاوی تعداد کمی تاپل باشند(توزیع نامتوازن داده ها). مشکلات چولگی را می توان به ۲ دسته زیر تقسیم کرد :

- Attribute-value skew
- Partition skew

ص ۸۰۰

Attribute-value skew به این واقعیت اشاره دارد که برخی از value های صفات تقسیم کننده, ممکن است در بسیاری از تاپل ها یکسان باشند. بنابر این همه تاپل ها با value برابر, در تقسیم بندی ایجاد چولگی خواهند کرد.

Partition skew به این واقعیت اشاره دارد که ممکن است حتی زمانیکه صفت تقسیم کننده هیچگونه چولگی ایجاد نکرده, مشکل عدم تعادل بار بروز کند.

Attribute-value skew می تواند در تقسیم بندی چولگی ایحاد کند صرف نظر از اینکه روش Range-partitioning یا Hash-partitioning استفاده شده باشد.

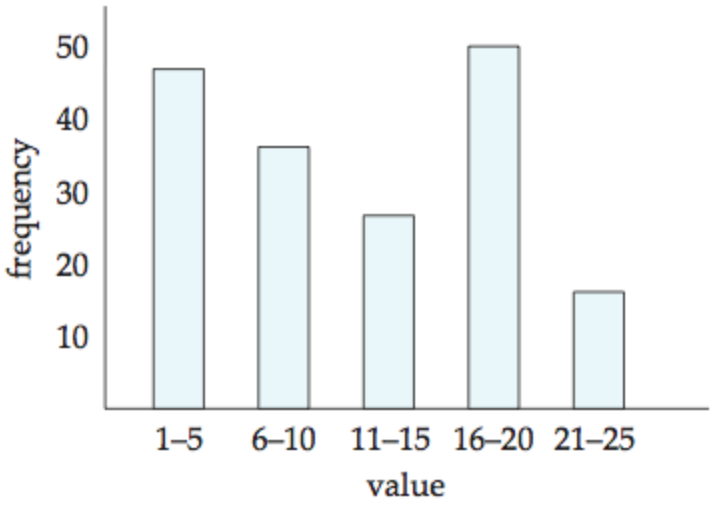
اگر بردار تقسیم کننده با دقت انتخاب نشود, ممکن است روش Range-partitioning باعث ایجاد Partition skew شود. اگر در روش Hash-partitioning تابع Hash بخوبی انتخاب شود احتمال بروز Partition skew بسیار کم است.

در بخش ۱-۳-۱۷ اشاره شد حتی یک چولگی کوچک میتواند کاهش چشمگیری در کارایی ایجاد کند. چولگی با افزایش درجات موازی سازی افزایش می یابد. بعنوان مثال اگر یک رابطه با ۱۰۰۰ تاپل به ۱۰ قسمت تقسیم شود و قسمتهای حاصل دارای چولگی باشد, آنگاه ممکن است چند دیسک حاوی کمتر از ۱۰۰ تاپل باشد و چند دیسک دیگر بیش از ۱۰۰ تاپل داشته باشند. اگر تنها یک دیسک حاوی ۲۰۰ تاپل باشد, آنگاه speed up دسترسی به دیسک بصورت موازی به ۵ کاهش می یابد(در صورتیکه انتظار داریم ۱۰ باشد). اگر همین رابطه به ۱۰۰ بخش

تقسیم شود, بطور میانگین هر بخش باید ۱۰ تاپل را در خود جای دهد. حتی اگر یک قسمت دارای ۴۰ تاپل باشد(این امر محتمل است چراکه تعداد بخشها زیاد است), آنگاه speed up مورد نظر که باید ۱۰۰ باشد به ۲۵ کاهش خواهد یافت. بنابراین میبینیم که موازی سازی به دلیل مشکل چولگی,speed up از دست خواهد داد.

یک بردار تقسیم کننده بالانس شده در روش Range-partitioning میتواند در حین مرتب سازی ساخته شود. به این صورت که ابتدا رابطه بر اساس صفت تقسیم کننده مورد نظر مرتب می شود. رابطه برای مرتب شدن اسکن خواهد شد. پس خواندن هر 1/n از رابطه ,value تاپل بعدی به بردار تقسیم کننده اضافه خواهد شد. در اینجا n نشان دهنده تعداد تقسیم بندی هایی است که باید انجام شود. در این روش نیز ممکن است بسیاری تاپل با value یکسان بر روی صفت تقسیم کننده وجود داشته باشد و این باعث ایجاد چولگی خواهد شد. نقطه ضعف اصلی این روش سربار اضافی I/O در حین انجام مرتب سازی اولیه است. سربار I/O برای ساخت و ایجاد بردار تقسیم کننده بالانس شده می تواند توسط ساخت و ذخیره سازی یک جدول فرکانس کاهش یابد, یا برای این کاهش از هیستوگرام value های یک صفت برای هر صفت از رابطه استفاده کرد. شکل ۱–۱۸ مثالی از یک هیستوگرام برای یک صفت با value ای از نوع integer که عددی بین ۰ تا ۲۵ را اختیار می کند, نمایش می دهد. یک هیستوگرام فضای کوچکی را اشغال می کند, بنابراین هیستوگرام های چند صفت مختلف می توانند در کاتالوگ سیستم ذخیره شوند. با توجه به هیستوگرام صفات تقسیم کننده , ساختن یک بردار تقسیم کننده بالانس شده آسان است. اگر هیستوگرام ذخیره نشده باشد, می توان با نمونه گیری از رابطه, تنها با انتخاب زیر مجموعه ای از تاپل های یک بلاک دیسک رابطه بصورت تصادفی, آن را بدست آورد.

روش دیگر برای کاهش اثر چولگی, استفاده از پردازنده مجازی است.در این روش بجای اینکه تعداد واقعی پردازنده ها را در نظر بگیریم وانمود می کنیم که تعداد زیادی پردازنده موجود است. هر کدام از روش های تقسیم بندی و تکنیک های ارزیابی پرس و جو را که بعدا در همین بخش بیان می شود, می توان استفاده کرد. اما بجای اینکه کارها و تاپل ها را به پردازنده های واقعی بدهیم آنها را به پردازنده های مجازی اختصاص می دهیم. پردازنده های مجازی نیز به نوبه خود توسط روش Round-Rabin به پردازنده های واقعی map می شوند. ایده اینست که حتی اگر در یک محدوده بدلیل چولگی تعداد تاپل بیشتری وجود داشت,این تاپل های مازاد در سراسر قسمت های پردازنده های مجازی پخش شوند.



شکل ۱–۱۸ نمونه ای از هیستوگرام

Map کردن پردازنده های مجازی به واقعی توسط روش Round-Rabin یک کار اضافی را در میان پردازنده های واقعی توزیع می کند, بطوریکه یک پردازنده مجبور نیست تمام بار را به تنهایی تحمل کند.

۱۸–۳ Inter query parallelism

در موازی سازی Inter query, پرس و جوهای مختلف یا اجرای تراکنش ها بصورت موازی با یکدیگر انجام می شود. توان عملیاتی تراکنش ها می تواند توسط این شکل از موازی سازی افزایش یابد.

با این حال زمان پاسخ اجرای تراکنش ها بصورت جداگانه و غیر موازی نسبت به زمانی که بصورت موازی انجام می شود, بیشتر نیست. بنابراین, استفاده اولیه از موازی سازی Inter query بمنظور پشتیبانی از مقیاس پذیری تعداد زیادی تراکنش در هر ثانیه است.

موازی سازی Inter query آسان ترین شکل موازی سازی برای پشتیبانی از یک سیستم پایگاه داده بویژه در سیستم هایی با حافظه اشتراکی است. سیستم های پایگاه داده طراحی شده برای سیستم های تک پردازنده می تواند, با تغییر کوچکی یا حتی بدون تغییر, در سیستم هایی با معماری حافظه مشترک موازی استفاده شود. حتی سیستم های ترتیبی نیز از پردازش همزمان پشتیبانی می کنند. تراکنش ها می توانند به شیوه ای همزمان ,در یک حالت اشتراکی بر روی یک ماشین عملیاتی ترتیبی, بصورت موازی در یک معماری حافظه اشتراکی انجام شوند.

پشتیبانی از موازی سازی Inter query در سیستم هایی با معماری دیسک اشتراکی یا هیچ چیز اشتراکی بسیار پیچیده تر است. پردازنده ها باید تعداد زیادی عمل را مانند قفل گذاری, ورود به سیستم, بودن در حالت هماهنگ کننده و ارسال پیام به یکدیگر انجام دهند. یک سیستم پایگاه داده موازی شده همچنین باید این اطمینان را بدهد که دو پردازنده نتوانند بصورت همزمان بر روی یک داده یکسان کار کنند یا آن را بروز رسانی کنند. علاوه بر این, هنگامیکه یک پردازنده به یک داده دسترسی پیدا می کند یا آنرا بروزرسانی می کند, سیستم پایگاه داده باید اطمینان داشته باشد که داده مورد استفاده توسط پردازنده آخرین نسخه خوانده شده از بافر است. حصول اطمینان از آخرین نسخه داده بعنوان cache coherency شناخته می شود.

ص۸۰۲

پروتکل های متفاوتی برای تضمین cache coherency در دسترس هستند.اغلب این پروتکل ها با پروتکل های کنترل همزمانی انسجام می یابد و در نتیجه سربار کاهش می یابد. یکی از پروتکل هایی که برای سیستم هایی با دیسک مشترک مورد استفاده قرار می گیرد, بصورت زیر است:

۱. قبل از دسترسی برای خواندن یا نوشتن یک صفحه, تراکنش صفحه را توسط قفل اشتراکی یا انحصاری قفل گذاری می کند. بلافاصله پس از برست آوردن قفل یک صفحه, تراکنش آخرین نسخه صفحه را از دیسک اشتراکی می خواند.
۲. قبل از اینکه تراکنش قفل انحصاری را رها کند, تغییرات حاصل را به دیسک مشترک می فرستد و سپس قفل را رها می کند.

پروتکل مذکور این اطمینان را می دهد که وقتی یک تراکنش قفل انحصاری یا اشتراکی روی یک صفحه می گذارد, از یک کپی صحیح و معتبر صفحه استفاده می شود. پروتکل های پیچیده تری نیز وجود دارند که از خواندن و نوشتن مکرر پروتکل فوق دوری می کنند. مانند پروتکل هایی که در هنگام رها کردن قفل انحصاری بر روی صفحات دیسک چیزی نمی نویسند. موقعی که یک قفل اشتراکی یا انحصاری گذاشته می شود, اگر تعداد زیادی از نسخه های یک صفحه در بافر چند پردازنده باشد, صفحه مورد نظر بجای اینکه از دیسک انتقال داده شود از بافرهای مذکور در اختیار تراکنش قرار می گیرد. پروتکل ها باید طوری طراحی شوند که بتوانند درخواست های همزمان را پاسخگو باشند. پروتکل دیسک اشتراکی می تواند با طرح زیر به معماری هیچ چیز اشتراکی تعمیم یابد:هر صفحه یک پردازنده خانگی P_i دارد, و این صفحه روی دیسک D_i ذخیره شده است. هنگامیکه دیگر

پردازنده ها بخواهند این صفحه را بخوانند یا بر روی آن بنویسند, درخواستی را به پردازنده خانگیP_i آن صفحه می فرستند زیرا نمی توانند بصورت مستقیم با دیسک در ارتباط باشند. دیگر عملیات ها نیز در پروتکل دیسک اشتراکی به همین صورت هستند.

سیستم هایی اوراکل و اوراکل Rdb نمونه هایی از سیستم های پایگاه داده موازی با دیسک اشتراکی هستند که از پرس و جو های Inter query پشتیبانی می کنند.

۱۸-۴ Intra query parallelism

موازی سازی Intra query به اجرای یک پرس و جو به تنهایی بصورت موازی بر روی چند پردازنده و دیسک اشاره دارد. استفاده از این نوع موازی سازی برای افزایش speed up پرس و جوهایی با زمان اجرای طولانی اهمیت دارد. در این موارد موازی سازی Inter query کمکی نخواهد کرد زیرا هر پرس و جو بصورت ترتیبی اجرا می شود.

برای ارزیابی یک پرس و جو بصورت موازی, پرس و جویی را در نظر بگیرید که می خواهد رابطه را مرتب کند. فرض کنید که رابطه ی مفروض با روش Range-partitioning بر روی چندین دیسک بر اساس صفت تقسیم کننده مرتب و تقسیم بندی شده است. عملیات مرتب سازی را می توان در هر قسمت بصورت موازی انجام داد سپس قسمت های مرتب شده را برای تولید نهایی رابطه بصورت مرتب با هم ادغام کرد.

بنابراین, میتوان یک پرس و جو را با موازی اجرا کردن عملیات جداگانه با یکدیگر انجام داد. منبع دیگری از موازی سازی در ارزیابی یک پرس و جو وجود دارد: درخت عملیات برای یک پرس و جو می تواند حاوی چند عملیات متعدد باشد. میتوان ارزیابی درخت عملیات را, با ارزیابی عملیاتی که به یکدیگر وابسته نیستند, موازی کرد. علاوه بر این, در فصل ۱۲ بیان شد که خروجی یک عملیات را می توان بصورت خط لوله برای عملیات دیگر فرستاد. دو عملیات می توانند روی دو پردازنده جداگانه بصورت موازی اجرا شوند. خروجی تولید شده توسط یک پردازنده را می توان به عنوان ورودی برای دیگر پردازنده ها در نظر گرفت.

بصورت خلاصه یک پرس و جو را می توان به دو روش مختلف موازی کرد:

- Intra operation parallelism: می توان سرعت پردازش یک پرس و جو را توسط موازی سازی اجرای هر عمل(مانند مرتب سازی, select , project و join) بصورت جداگانه افزایش داد. توضیحات بیشتر در قسمت ۶-۱۸
- Inter operation parallelism: می توان سرعت پردازش یک پرس و جو را توسط موازی سازی عملیات مختلف در قالب یک عبارت پرس و جو افزایش داد. توضیحات بیشتر در قسمت ۶-۱۸

هر دو شکل موازی سازی مکمل یکدیگرند و می توانند بصورت توامان و همزمان در یک پرس و جو استفاده شوند. از آنجا که تعداد عملیات در یک پرس وجوی معمولی, در مقایسه با تعداد تاپل های پردازش شده توسط هر عملیات کم است, شکل اول موازی سازی می تواند مقیاس پذیری بیشتری را با افزایش درجه موازی سازی ارائه دهد. با این حال, در سیستم های موازی امروزی, هنگامی که تعداد پردازنده ها نسبتا کم باشد, هر دو شکل موازی سازی مهم هستند.در ادامه, موازی سازی جستجو ها بررسی میشود. فرض میشود که که پرس و جو ها فقط خواندنی باشند. انتخاب الگوریتمی برای ارزیابی موازی سازی پرس و جو به معماری ماشین بستگی دارد. در اینجا بجای اینکه برای هر معماری الگوریتم جداگانه ای ارائه شود, معماری هیچ چیز اشتراکی در نظر گرفته می شود. این معماری را می توان توسط دیگر معماری ها شبیه سازی کرد. زیرا می توان انتقال داده را توسط حافظه مشترک در معماری حافظه اشتراکی, یا از طریق دیسک مشترک در دیسک اشتراکی انجام داد. از این رو, الگوریتم هایی که در معماری هیچ چیز اشتراکی استفاده می شوند را می توان برای دیگر معماری ها نیز بکار برد.

برای سادگی الگوریتم فرض می شود که n پردازنده ($P_0, P_1, \ldots, P_{n-1}$) و n دیسک ($D_0, D_1, \ldots, D_{n-1}$) وجود داشته دارد و دیسک D_i برای پردازش از P_i استفاده می کند. در دنیای واقعی هر پردازنده ممکن است چند دیسک داشته باشد. گسترش این الگوریتم برای تخصیص چند دیسک به یک پردازنده سخت نیست, می توان به سادگی واژه D_i را بجای یک دیسک به چندین دیسک اختصاص داد(یعنی D_i متشکل از چند دیسک است). برای سادگی کار فرض می کنیم D_i شامل تنها یک دیسک است.

۱۸-۵ Intra operation parallelism

از آنجا که عملیات رابطه ای بر روی روابط حاوی مجموعه بزرگی از تاپل ها کار می کند, می توان عملیات را توسط اجرای آنها بصورت موازی در زیر مجموعه های مختلف از رابطه موازی کرد. چون تعداد تاپل ها در

یک رابطه ممکن است بزرگ باشد, درجه موازی سازی بصورت بالقوه بسیار زیاد است.بنابراین, موازی سازی Intra query در سیستم های پایگاه داده امری طبیعی است. در بخش های ۱-۵-۱۸ و ۳-۵-۱۸ موازی سازی نسخه های برخی از عملیات مشترک و معمول بیان خواهد شد. ص۸۰۴

۱۸,۵ موازات میان عملیاتی

از آنجا که عملیات رابطه ای بر روی رابطه هایی که شامل مجموعه های بزرگی از رکورد هاست کارمی کنند، ما می توانیم عملیات را با اجرای موازی آنها بر روی زیر مجموعه های مختلف از روابط موازی سازی کنیم.درحالی که تعداد رکورد ها در یک رابطه می تواند بزرگ باشد , درجه ی مشابهت به صورت بالقوه ای بزرگ است . بنابراین موازات میان عملیاتی در یک سیستم پایگاه داده طبیعی است .ما باید ورژن های موازی از بعضی عملیات رابطه ای رایج را در بخش های 18.5.1 تا 18.5.3 مطالعه کنیم.

۱۸,۵,۱ مرتب سازی موازی

فرض کنید ما می خواهیم یک رابطه ای را که بر روی n دیسک D0,D1,….,Dn-1 قرار دارد, مرتب کنیم .اگر رابطه پارتیشن بندی محدوده ای بر روی خصیصه هایی شده باشد که بایستی مرتب شوند , پس , همانطور که در بخش 18.2.2 اشاره شد , ما می توانیم هر پارتیشن را به صورت جداگانه مرتب کنیم , و میتوانیم با الحاق نمودن نتایج, رابطه ی مرتب شده ی کامل را بدست بیاوریم . در حالی که رکورد هابر روی n دیسک پارتیشن بندی شده اند , زمان مورد نیاز برای خواندن کل رابطه با دسترسی موازی کاهش می یابد .اگر رابطه به هر روش دیگری پارتیشن بندی شده باشد , ما می توانیم آن را به یکی از دو روش زیر مرتب کنیم .

۱- ما می توانیم پارتیشن محدوده ای بر روی خصیصه های مرتب سازی بگذاریم و هر پارتیشن را به صورت جداگانه مرتب کنیم .

۲- ما می توانیم یک ورژن موازی از الگوریتم مرتب سازی ادغامی خارجی استفاده کنیم .

۱۸,۵,۱,۱ مرتب سازی پارتیشن محدوده ای

مرتب سازی پارتیشن محدوده ای در دو مرحله کار می کند : اول رابطه را پارتیشن بندی محدوده ای می کند, سپس هر پارتیشن را جداگانه مرتب می کند . وقتی ما رابطه را با پارتیشن بندی محدوده ای مرتب می کنیم , ضرورتی ندارد تا رابطه را بر روی مجموعه های مشابه از پردازنده ها یا دیسک ها که رابطه بر روی آن ذخیره گردیده است مرتب کنیم .فرض کنید که ما پردازنده های p0,p1,...,pm را انتخاب کنیم , که m < n , برای ذخیره ی رابطه , دو مرحله درگیر این عمل هستند :

۱- رکورد های رابطه توزیع مجدد می شوند , با یک استراتژی پارتیشن محدوده ای , بنابراین تمامی رکورد هایی که در محدوده ی i قرار می گیرند به پردازنده ی pi فرستاده می شوند , که رابطه را به صورت موقت بر روی دیسک

Di ذخیره می کند . برای پیاده سازی پارتیشن بندی محدوده ای , هر پردازنده به صورت موازی رکورد هایش را از دیسکش می خواند و رکورد ها را به پردازنده های مقصدش ارسال می کند .هر پردازنده ی p0,p1,….,pm نیز به همین صورت رکوردهایی را که به تقسیماتش تعلق دارد دریافت میکند و به صورت محلی ذخیره میکند . این گام عملیات ورودی خروجی دیسک و سربار ارتباطات را می طلبد .

۲- هر کدام از پردازنده ها پارتیشنش بر روی رابطه را به صورت محلی مرتب می کند ,بدون تداخل با سایر پردازنده ها .هر پردازنده عمل مشابهی را انجام می دهد –به عنوان مثال , مرتب سازی– بر روی یک دیتاست متفاوت .(اجرای عمل مشابه به صورت موازی بر روی مجموعه های مختلف از داده ها , موازی سازی داده ها خوانده می شود .)

عمل ادغام نهایی جزیی است , به خاطر اینکه پارتیشن بندی محدوده ای در فاز اول آن را حتمی می کند , برای $1<=i<j<=m$, مقادیر کلیدی در پردازنده ی pi از مقادیر کلیدی در pj کمتر هستند . ما باید پارتیشن بندی محدوده ای انجام دهیم با یک بردار پارتیشن بندی خوب ,بنابراین هر پارتیشن به صورت خوشبینانه تعداد مشابهی از رکورد ها را خواهد داشت . پارتیشن بندی مجازی پردازنده هم می تواند برای کاهش عدم توازن استفاده شود .

۱۸,۵,۱,۲ مرتب سازی ادغامی خارجی به صورت موازی

مرتب سازی ادغامی خارجی به صورت موازی جایگزینی است برای پارتیشن بندی محدوده ای .فرض می شود که یک رابطه میان دیسک های D0,D1,….,DN-1 پارتیشن بندی شده است (ربطی ندارد رابطه چگونه پارتیشن بندی شده است). مرتب سازی ادغامی خارجی به صورت موازی به صورت زیر عمل می کند :

۱- هر پردازنده ی Pi به صورت محلی داده ها را بر روی دیسک Di مرتب می کند .

۲- سپس سیستم اجراهای ذخیره شده بر روی هر پردازنده را ادغام می کند تا خروجی ذخیره شده ی نهایی را به دست آورد .

ادغام نمودن اجراهای ذخیره شده در مرحله ی ۲ می تواند به ترتیب عملیات زیر موازی سازی گردد:

۱- سیستم پارتیشن های ذخیره شده بر روی هر پردازنده ی Pi را پارتیشن بندی محدوده ای می کند (همه با یک بردار پارتیشن مشابه) از میان پردازنده های P0,P1,…,Pm-1 . سیستم رکورد ها را به ترتیب ذخیره شدن ارسال می کند , بنابراین هر پردازنده رکورد ها را در مسیر ذخیره شده دریافت می کند .

۲- هر پردازنده ی Pi یک ادغام را بر روی جریان های دریافتی انجام می دهد , تا یک اجرای ذخیره ی تک به دست آورد.

۳- سیستم اجرهای ذخیره شده بر روی پردازنده های P0,P1…Pm-1 را الحاق می کند تا نتیجه ی نهایی را به دست آورد .

هم چنان که توضیح داده شد , این توالی از نتایج اقدامات در یک فرم جالب از اجرای اریبی , زمانی که ابتدا هر پردازنده تمام بلاک ها از پارتیشن 0 به P0 را ارسال می کند , سپس هر پردازنده تمام بلاک ها از پارتیشن ۱ به P1 را ارسال می کند, وبه همین ترتیب .بنابراین , در حالی که ارسال به صورت موازی اتفاق می افتد , دریافت رکورد ها به صورت متوالی صورت می پذیرد :ابتدا فقط P0 رکورد ها را دریافن می کند , سپس فقط P1 رکورد ها را دریافت می کند , و به همین ترتیب. برای جلوگیری از این مشکل , هر پردازنده به صورت مکرر یک بلوک از داده ها را به هر پارتیشن

ارسال می کند . به عبارت دیگر , هر پردازنده اولین بلاک از هر پارتیشن را ارسال می کند , سپس دومین بلاک از هر پارتیشن را ارسال می کند , وبه همین ترتیب . نتیجه اینکه , تمام پردازنده ها داده ها را به صورت موازی دریافت می کنند.

بعضی ماشین ها , همانند ماشین ها ی خانواده ی the Teradata Purpose-Bult Platform , سخت افزار خاصی را برای اجرای ادغام استفاده می کنند . شبکه ی ارتباط داخلی The BYNET در ماشین های Teradata میتواند خروجی حاصل از چند پردازنده را برای بدست آوردن یک خروجی ذخیره شده ی تک ادغام کند .

۱۸,۵,۲ اتصال موازی

در عمل پیوند نیاز است تا سیستم رکوردها را به صورت جفت آزمایش کند تا مشخص گردد که شرایط پیوند محقق می گردد ; اگر شرایط برقرار بود, سیستم جفت رکورد را به خروجی پیوند اضافه میکند . الگوریتم های پیوند موازی تلاش دارند تا جفت رکورد ها را به گونه ای جدا نمایند تا بر روی چندین پردازنده مورد آزمون قرار گیرند .سپس هر پردازنده جزیی از پیوند را به صورت محلی محاسبه می کند .سپس , سیستم نتایج را از هر پردازنده جمع آوری می نماید تا نتیجه ی نهایی را تولید نماید .

۱۸,۵,۲,۱ پیوند پارتیشن بندی شده

برای تعداد مشخصی از پیوندها , مانند پیوند به شرط تساوی و پیوندهای طبیعی , این امکان وجود دارد تا دو رابطه ی ورودی را میان پردازنده ها پارتیشن کنیم و پیوند را به صورت محلی در هر پردازنده محاسبه کنیم . فرض می کنیم تعداد پردازنده هایی که استفاده می کنیم n تاست و رابطه هایی که باید به هم پیوند شوند r و s است . سپس پیوند پارتیشن بندی شده به این صورت عمل می کند : سیستم هر یک از رابطه های r و s را در n پارتیشن , پارتیشن بندی می کند , به صورت r0,r1,…,r n-1 و s0,s1,…,sn-1 . سیستم پارتیشن های ri و si را به پردازنده ی pi ارسال می کند , جایی که پیوندشان به صورت محلی محاسبه شده است .

تکنیک پیوند پارتیشن بندی شده فقط در صورتی به صورت صحیح کار می کند که پیوند از نوع پیوند به شرط تساوی باشد (به عنوان مثال r r.A=s.B s) و اگر ما r و s را به وسیله ی تابع پارتیشن بندی مشابه بر روی مشخصه های پیوندشان پارتیشن بندی کرده باشیم . ایده ی پارتیشن بندی دقیقا مشابه آن چیزی است که پشت سر مرحله ی پارتیشن بندی پیوند درهم وجود دارد . در یک پیوند پارتیشن بندی شده , با این وجود , دو روش متفاوت پارتیشن بندی برای r و s وجود دارد:

- پارتیشن بندی محدوده ای بر روی مشخصه ها ی پیوند
- پارتیشن بندی درهم بر روی مشخصه های پیوند

در هریک از موارد , تابع پارتیشن بندی یکسان بایستی برای هر دو رابطه استفاده شود .برای پارتیشن بندی محدوده ای , بردار پارتیشن بندی یکسان بایستی برای هر دو رابطه استفاده شود . برای پارتیشن بندی درهم , تابع درهم سازی یکسان بایستی برای هردو رابطه استفاده گردد . شکل 18.2 پارتیشن بندی در یک پیوند پارتیشن بندی شده ی موازی را نشان می دهد .

یکبار که رابطه ها پارتیشن بندی شدند , ما می توانیم هر تکنیک پیوند را به صورت محلی در هر پردازنده ی Pi برای محاسبه ی پیوند ri و si استفاده کنیم .به عنوان مثال , پیوند درهم , پیوند ادغامی , یا پیوند حلقه ی تو در تو می تواند استفاده شود . بنابراین , ما می توانیم پارتیشن بندی را برای موازی سازی هر تکنیک پیوندی استفاده کنیم .

اگر هر دو یا یکی از رابطه های r و s بر روی خصیصه های پیوند پارتیشن بندی شده باشند (با پارتیشن بندی درهم یا پارتیشن بندی محدوده ای), کاری که برای انجام پارتیشن بندی نیاز است به صورت قابل ملاحظه ای کاهش می یابد .

اگر رابطه ها پارتیشن بندی نشده باشند , یا بر اساس خصیصه هایی به غیر از خصیصه ها پیوند پارتیشن بندی شده باشند , پس نیاز می شود تا رابطه ها دومرتبه پارتیشن بندی شوند ..هر پردازنده ی Pi رکوردهایی که بر روی دیسک Di است را می خواند , برای هر رکورد t پارتیشن j را که t به آن تعلق دارد را محاسبه می کند, و رکورد t را به پردازنده ی Pj ارسال می کند .پردازنده ی Pj رکورد را بر روی دیسک Dj ذخیره می کند .

ما می توانیم الگوریتم پیوندی که به صورت محلی در هر پردازنده اجرا شده است را برای کاهش I/O بوسیله ی بافر کردن تعدادی از رکورد ها به حافظه بهینه کنیم , بجای اینکه آنها را برروی دیسک بنویسیم .ما بهینه سازی هایی نظیر این را در بخش 18.5.2.3توضیح می دهیم .

عدم توازن یک مشکل خاص را وقتی پارتیشن بندی محدوده ای استفاده می شود نشان می دهد , وقتی یک بردار پارتیشن که یکی از رابطه های join را به پارتیشن هایی با اندازه ی مساوی تقسیم می کند ممکن است دیگر رابطه ها را به پارتیشن هایی با اختلاف سایز های بزرگ تقسیم کند .بردار پارتیشن باید به گونه ای باشد که |ri|+|si| (جمع اندازه های ri وrs) به شدت برابر باشد با همه ی i=0,1,…,n-1 . با یک تابع درهم سازی خوب , پارتیشن بندی درهم به صورت قابل ملاحظه ای عدم توازن کمتری دارد , بجز زمانی که تعداد زیادی رکورد با مقادیر مشابه برای مشخصه های پیوند وجود دارد .

۱۸,۵,۲,۲پیوند قطعه و تکرار

پارتیشن بندی برای همه ی انواع پیوندها قابل قبول نیست .به عنوان مثال , به عنوان مثال زمانی که شرط پیوند نامساوی است , مانند r r.a<s.b s , امکان دارد تمامی رکوردهایی که در r هستند با تعدادی از رکوردهایی که در s هستند پیوند شوند(و برعکس).بنابراین , بنابراین ممکن است روش آسانی برای پارتیشن بندی r و s وجود نداشته باشد به صورتی که رکوردهایی که در پارتیشن ri هستند فقط با رکورد هایی پیوند شوند که در پارتیشن si هستند .

مام یتوانیم چنین پیوند هایی را با تکنیکی که *قطعه کردن و تکرار* نامیده میشود موازی سازی کنیم .در ابتدا یک مورد خاص از قطعه کردن و تکرار را در نظر می گیریم ـــــــ پیوند قطعه کردن نامساوی و تکرار ـــــــ که به صورت زیر کار می کند :

۱- سیستم یکی از رابطه ها را پارتیشن بندی می کند , می گوییم r. هرتکنیک پارتیشن بندی می تواند برروی r استفاده شود , شامل پارتیشن بندی round-robin.
۲- سیستم رابطه ی دیگر را تکرار می کند , s , در میان همه ی پردازنده ها .
۳- پردازنده ی pi سپس به صورت محلی پیوند ri با تمامی s را به صورت محلی محاسبه می کند, با استفاده از هر تکنیک پیوندی .

شمای پیوند قطه کردن نامساوی و تکرار در شکل ۱۸,۳a به تصویر کشیده شده است . اگر r هم اکنون به صورت پارتیشن بندی شده ذخیره شده باشد , دیگر نیازی نیست تا بیشتر در مرحله یک پارتیشن بندی شود .تمامی چیزی که نیاز است این است تا s میان همه ی پردازنده ها کپی شود .

مورد رایج از پیوند قطعه کردن و تکرار در شکل 18.3b نمایش داده شده است ; و به این صورت کار می کند : سیستم رابطه ی r را به n پارتیشن قسمت بندی می کند , r0,r1,…,r n-1 , و s را به m پارتیشن قسمت بندی می کند, S0,s1,…,s m-1 .همانند گذشته هر تکنیک پارتیشن بندی می تواند بر روی r و s اجرا شود .مقادیر m وn نیازی نیست که برابر باشند , اما بایستی به گونه ای انتخاب گردند که حداقل m*n پردازنده موجود باشد . پیوند تکه کردن نامساوی و تکرار به صورت ساده یک مورد خا ص از قطعه کردن و تکرار کلی است , وقتی که m=1 . قطعه کردن وتکرار اندازه ی رابطه را بر روی هرپردازنده کاهش می دهد , در مقایسه با قطعه کردن نا متناسب و تکرار .

فرض کنید پردازنده ها P0,0,P0,1,…,P0,m-1,Pn-1,m-1 باشند . پردازنده ی Pi,j پیوند ri با sj را محاسبه می کند .هر پردازنده بایستی آن رکورد هایی را در پارتیشن هایی که بر روی آن کار می کند بگیرد . برای به انجام رسانیدن این موضوع , سیستم ri را به پردازنده های Pi,0,Pi,1,…,Pi,m-1 (که یک ردیف را در شکل 18.3b تشکیل می دهد) , و Si را به پردازنده های P0,iP1,i,…,Pn-1,I (که یک ستون را در شکل 18.3b تشکیل می دهد) کپی می کند .هر تکنیک پیوندی می تواند در هر پردازنده ی Pi,j استفاده شده باشد .قطعه بندی و تکرار بر روی هر حالت پیوندی کار می کند , تا زمانی که هر رکورد در r میتواند با هر رکورد در s تست شود .بنابراین , جایی که پارتیشن بندی نمی تواند استفاده شود می تواند مورد استفاده قرار گیرد .

قطعه بندی و تکرار معمولا هزینه ی بیشتری نسبت به پارتیشن بندی دارد زمانی که هر دو رابطه به طور زیادی هم اندازه اند , زیرا حداقل یکی از رابطه ها بایستی تکرار گردد .با این حال , چنانچه یکی از رابطه ها ___به عنوان مثال , s___ کوچک باشد , ممکن است تکرار s میان همه ی پردازنده ها ارزان تر باشد , نسبت به پارتیشن بندی مجدد r و s بر روی خصیصه های پیوند . در این موارد قطعه بندی نامتقارن وتکرار ترجیح داده میشود , حتی اگر پارتیشن بندی قابل استفاده باشد.

۱۸,۵,۲,۳ پیوند موازی درهم پارتیشن بندی شده

پیوند درهم پارتیشن بندی شده ی بخش 12.5.5 می تواند موازی شود . فرض کنید ما n پردازنده داریم , P0,P1,…,Pn-1 , و دو رابطه ی r و s , به این صورت که رابطه های r و s در میان دیسک های متعدد پارتیشن بندی شده اند .یادآوری از بخش 12.5.5 که رابطه ی کوچکتر برای ایجاد نمودن رابطه انتخاب می شود .اگر اندازه ی s کوچکتر از اندازه ی r باشد , الگوریتم پیوند موازی در هم به این روش پیش می رود :

۱- یک تابع درهم سازی انتخاب کنید _____مثلا , h1___که مقادیر مشخصه های پیوند را برای هر رکورد در r وs می گیرد و رکورد را به یکی از n پردازنده نگاشت می کند . با در نظر گرفتن این که ri رکوردی از رابطه ی r را مشخص می کند که به پردازنده ی Pi نگاشت شده است , به طور مشابه, si مشخص کننده ی رکوردها یی از رابطه ی s است که به پردازنده ی pi نگاشت شده است . هر پردازنده ی pi رکوردها یی از s را که بر روی دیسکش Di هست را می خواند و هر رکورد را به پردازنده ی متناسب بر پایه ی تابع در هم سازی h1 ارسال می کند .

۲- زمانی که پردازنده ی مقصد Pi رکورد های Si را دریافت می کند , سپس آن ها را با یک تابع در هم سازی , h2
, که پردازنده برای محاسبه ی پیوند در هم به صورت محلی از آن استفاده می کند پارتیشن بندی می کند .
پارتیشن بندی در این مرحله دقیقا مشابه فاز پارتیشن بندی در الگوریتم پیوند درهم متوالی است . هر پردازنده ی Pi این مرحله را مستقل از پردازنده های دیگر اجرا می کند .

۳- زمانی که رکوردهای s توزیع شدند , سیستم رابطه ی بزرگتر r را میان n پردازنده به وسیله ی تابع درهم سازی h1 دو مرتبه توزیع می کند , به روشی مشابه روش قبل .زمانی که هر رکورد دریافت می شود , پردازنده ی مقصد آن را به وسیله ی تابع h2 دوباره پارتیشن بندی می کند , به همان صورت که رابطه ی مورد جستجو در یک الگوریتم جستجوی درهم متوالی پارتیشن بندی می شود.

۴- هر پردازنده ی Pi مراحل ایجاد و جستجوی الگوریتم پیوند درهم را بر روی پارتیشن های محلی ri و si از r و s برای تولید یک پارتیشن از نتیجه ی نهایی از پیوند درهم اجرا می کند .

پیوند درهم در هر پردازنده مستقل از دیگر پردازنده ها است , و دریافت رکوردهای ri و si شبیه خواندن آنها از دیسک است .بنابراین , هر کدام از بهینه سازی های پیوند درهم که در بخش 12 توضیح داده شد می تواند بر روی مورد موازی هم اعمال گردد . به طور خاص ما می توانیم الگوریتم پیوند درهم چند جزئی را برای گرفتن رکوردهای وارد شده در حافظه , و در نتیجه اجتناب از هزینه ی نوشتن و خواندن دوباره استفاده کنیم .

۱۸,۵,۲,۴پیوند حلقه ی موازی تو در تو

برای به تصویر کشیدن موارد استفاده ی قطعه بندی و تکرار بر پایه ی موازی سازی , موارد قابل ملاحظه جایی است که رابطه ی s بسیار کوچکتر از رابطه ی r است . فرض کنید رابطه ی r به وسیله ی پارتیشن بندی ذخیره شده است ; مشخصه ای که بر روی آن پارتیشن بندی شده است مهم نیست . همچنین فرض کنید یک شاخص بر روی یک مشخصه ی پیوند در رابطه ی r در هر پارتیشن از رابطه ی r وجود دارد .
ما از قطعه بندی نامتقارن و تکرار استفاده می کنیم , با رابطه ی S که تکرار می شود و با پارتیشن موجود از رابطه ی r .هر پردازنده ی Pj جایی که یک پارتیشن از رابطه ی s ذخیره می شود رکورد های رابطه ی s را که در Dj ذخیره شده است می خواند , و رکورد ها را در هر کدام از دیگر پردازنده های Pi تکرار می کند . در پایان این مرحله , رابطه ی s در تمامی سایت هایی کپی می شود که رکورد های رابطه ی r را ذخیره می کند .
حال , هر پردازنده ی Pi یک پیوند حلقه ی تکرار شاخص گذاری شده از رابطه ی s را با i امین پارتیشن از رابطه ی r اجرا می کند . ما می توانیم پیوند حلقه ی تکرار تودرتوی شاخص گذاری شده را با توزیع رکوردها ی رابطه ی s روی هم انجام دهیم , برای اینکه هزینه ی نوشتن رکوردهای رابطه ی s در دیسک و خواندن دوباره ی آن ها را کاهش دهیم .

به هر حال , تکرار رابطه ی s بایستی با پیوند هم گام باشد از این رو فضای کافی در حافظه ی بافرها در هر پردازنده ی Pi برای نگه داشتن رکوردهای رابطه ی s که دریافت شده ولی هنوز در پیوند استفاده نشده اند وجود دارد .

۱۸,۵,۳ دیگر عملیات رابطه ای

ارزیابی عملیات رابطه ای دیگر نیز می تواند موازی سازی شود :

- **انتخاب :** فرض کنید انتخاب $\sigma_{\theta}(r)$ باشد .در ابتدا به این مورد توجه کنید در جایی که θ به شکل ai=v , که ai یک صفت و v یک مقدار است . اگر رابطه ی r بر روی ai پارتیشن بندی شده باشد , انتخاب در یک پردازنده ی تک به پیش می رود .اگر θ به شکل $l<=ai<=u$ ـــــــ که θ یک انتخاب محدوده ای است ـــــ و رابطه بر روی ai پارتیشن بندی محدوده ای شده است , سپس انتخاب در هر پردازنده ای که پارتیشن اش با محدوده ی مشخصی از مقادیر روی هم باشد انجام می شود .در بقیه ی موارد , انتخاب به صورت موازی در همه ی پردازنده ها انجام می شود .
- **حذف تکرار :** تکراری ها می تواند به وسیله ی مرتب سازی حذف گردند ; یا اینکه تکنیک های مرتب سازی موازی , که برای حذف تکراری ها به محض اینکه در طول مرتب سازی ظاهر شوند بهینه سازی شده اند می تواند استفاده شود.ما همچنین می توانیم به وسیله ی پارتیشن بندی رکورد ها حذف تکرار را موازی سازی کنیم (به وسیله ی پارتیشن بندی محدوده ای یا درهم) و تکراری ها را به صورت محلی در هر پردازنده حذف کنیم.
  - **نگاشت :** نگاشت بدون حذف تکرار می تواند همانند رکورد هایی که از دیسک به صورت موازی خوانده می شود اجرا گردد .اگر تکراری ها بایستی حذف گردند, هر کدام از تکنیک هایی که توضیح داده شد می تواند استفاده شود .
  - **اجتماع :** یک عمل اجتماع را در نظر بگیرید . ما می توانیم عمل را به وسیله ی پارتیشن بندی رابطه بر روی مشخصه های گروهبندی موازی کنیم , و سپس مقادیر اجتماع را به صورت محلی در هر پردازنده محاسبه کنیم . یا پارتیشن بندی درهم و یا پارتیشن بندی محدوده ای می تواند استفاده شود .اگر رابطه هم اکنون بر روی مشخصه ها ی گروهبندی پارتیشن شده است , گام اول می تواند نادیده گرفته شود .

ما می توانیم هزینه ی انتقال رکورد ها در طول پارتیشن بندی را به وسیله ی محاسبه ی بخشی از مقادیر اجتماع قبل از پارتیشن بندی کاهش دهیم , حداقل برای توابع اجتماع رایج استفاده شده.یک عمل اجتماع بر روی یک رابطه ی r را در نظر بگیرید , که از یک تابع اجتماع جمع بر روی مشخصه ی B استفاده می کند , با گروه بندی بر روی مشخصه ی A . سیستم میتواند عمل را در هر پردازنده ی Pi بر روی آن r رکوردی که بر روی دیسک Di ذخیره شده است اجرا کند .این محاسبات رکورد هایی با جمع های جزیی در هر پردازنده را نتیجه می دهد ; یک رکورد در Pi برای هر مقدار از مشخصه ی A که درr رکورد ذخیره شده بر روی Di موجود است وجود دارد . سیستم نتایج اجتماع محلی بر روی مشخصه ی گروه بندی A را پارتیشن بندی می کند , و اجتماع را دوباره (بر روی رکورد هایی با جمع های جزیی) در هر پردازنده ی Pi اجرا می کند تا نتیجه ی نهایی به دست آید .

به عنوان یک نتیجه از این بهینه سازی , رکورد های کمتری نیاز است تا به پردازنده های دیگر در طول پارتیشن بندی ارسال شود .این ایده می تواند به آسانی به توابع اجتماع مینیمم و ماکزیمم گسترش داده شود .توسعه هایی به توابع اجتماع شمارش و میانگین برای شما در تمرین 18.12 جهت انجام گذاشته می شود .

موازی سازی عملیات دیگردر تمرین های متفاوت تحت پوشش قرار داده می شود .

۱۸,۵,۴ هزینه ی ارزیابی موازی عملیات

ما موازی سازی را به وسیله ی پارتیشن بندی I/O میان چندین دیسک به دست می آوریم , و پارتیشن بندی کار CPU میان پردازنده های متفاوت.اگر یک چنین تقسیم بندی بدون هیچ سرباری بدست آمده , و اگر هیچ عدم توازنی در تقسیم بندی کار وجود نداشته باشد , یک عمل موازی با استفاده از n پردازنده 1/n برابر بیشتر از عمل مشابه بر روی یک پردازنده ی تک زمان می برد . ما هم اکنون می دانیم چگونه هزینه ی یک عمل مانند پیوند یا انتخاب را تخمین بزنیم .هزینه ی زمانی یک پردازش موازی 1/n هزینه ی زمانی عمل پردازش متوالی می شود .

ما همچنین بایستی هزینه های زیر را حساب کنیم :

- **هزینه های شروع** برای آغاز کردن عمل در پردازنده های متعدد .
- **عدم توازن** در توزیع کار میان پردازنده ها , با بعضی پردازنده ها که تعداد بیشتری از رکوردها را نسبت به دیگران می گیرند .
- **درگیری برای منابع** ـــــــ به عنوان مثال حافظه , دیسک , و شبکه ی ارتباطات ـــــــ که تاخیر را نتیجه می دهد .
- **هزینه ی گردآوری** نتایج نهایی به وسیله ی انتقال نتایج جزیی از هر پردازنده .

زمانی که به وسیله ی یک عمل موازی گرفته می شود می تواند به این صورت تخمین زده شود :

$$T\text{part} + T\text{asm} + \max(T0, T1, \ldots, Tn-1)$$

که Tpart زمان پارتیشن بندی رابطه هاست , Tasm زمان گردآوری نتایج است , و Ti زمان صرف شده برای عمل در پردازنده ی Pi است .با فرض این که رابطه ها بدون هیچ عدم توازنی توزیع شده اند , تعداد رکورد هایی که به هر پردازنده ارسال شده می تواند به عنوان 1/n تعداد کل رکورد ها تخمین زده شود . با نادیده گرفتن تداخل , هزینه ی Ti برای عملیات در هر پردازنده ی Pi می تواند به وسیله ی تکنیک های فصل 12 تخمین زده شود .

تخمینی که زده می شود تخمینی خوشبینانه خواهد بود , در حالی که عدم توازن رایج است . اگر چه خورد کردن یک پرس وجوی تک به تعدادی از گام های موازی اندازه ی گام میانگین را کاهش می دهد , آن زمانی است برای محاسبه ی کندترین گام تک که زمان صرف شده برای پردازش کل پرس و جو را تعیین می کند .یک ارزیابی موازی پارتیشن بندی شده , به عنوان مثال , فقط به اندازه ی کندترین اجراهای موازی سریع است . بنابراین , هر عدم توازنی در توزیع کار میان پردازنده ها کارایی را تحت تاثیر بسیار زیادی قرار می دهد .

مشکل عدم توازن در پارتیشن بندی بسیار زیاد وابسته به مساله ی سرریز پارتیشن در پیوند های درهم متوالی است (فصل 12) . ما می توانیم سرریز تحلیل و تکنیک های اجتناب در پیوند های درهم را برای رفتار کردن با عدم توازن وقتی که پارتیشن بندی درهم استفاده می شود بکار ببریم . ما می توانیم پارتیشن بندی محدوده ای متوازن و پارتیشن بندی پردازنده ی مجازی را برای مینیمم کردن عدم توازن توسط پارتیشن بندی محدوده ای استفاده کنیم , مانند بخش 18.2.3

دو شکل از موازی سازی میان عمل وجود دارد : موازی سازی لوله ای و موازی سازی مستقل .

۱۸,۶,۱ موازی سازی خط لوله

همانطور که در فصل 12 بحث شد , خط لوله یک منبع مهم از نظر اقتصادی را برای محاسبه ی پردازش پرس وجو در پایگاه داده شکل می دهد.یادآوری می شود که , در خط لوله , رکورد های خروجی یک عمل ,A , به وسیله ی یک عمل دوم مصرف می شود ,B , حتی قبل از اینکه عمل اول کل مجموعه ی رکوردها را در خروجی اش تولید کرده باشد .مزیت اصلی اجرای خط لوله در یک ارزیابی متوالی این است که ما می توانیم یک توالی از عملیات را بدون نوشتن هیچ یک از نتایج میانی در دیسک انجام دهیم .

سیستم های موازی خط لوله ی اولیه را استفاده می کنند به همان دلیلی که سیستم های متوالی استفاده می نمایند .اگر چه , خط لوله ها یک منبع موازی سازی خوب به حساب می آیند , به دلیل مشابه خط لوله های دستورالعمل یک منبع موازی سازی در طراحی سخت افزار هستند .این امکان پذیر است که عملیات A و B را به صورت همزمان بر روی پردازنده های متفاوت اجرا کنیم , از این رو B رکورد هایی را به صورت موازی مصرف می کند که A آن ها را تولید کرده است .این شکل از موازی سازی , **موازی سازی خط لوله** خوانده می شود .

یک پیوند از چهار رابطه ی زیر را در نظر بگیرید :

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

ما می توانیم یک خط لوله را که اجازه می دهد سه پیوند به صورت موازی محاسبه گردد مستقر کنیم .فرض کنید پردازنده ی P1 اختصاص داده می شود به محاسبه ی $temp_1 \leftarrow r_1 \bowtie r_2$, و P2 اختصاص داده می شود به محاسبه ی $r_3 \bowtie temp_1$.همچنان که P1 رکورد ها را در $r_1 \bowtie r_2$ محاسبه می کند محاسبه می کند , این رکوردها را برای پردازنده ی P2 فراهم می کند . بنابراین , P2 بعضی از این رکورد ها را در $r_1 \bowtie r_2$قبل از اینکه P1 محاسبه اش را تمام کند فراهم کرده است .P2 می تواند آن سری از رکوردهایی را که برای شروع محاسبه ی $temp_1 \bowtie r_3$ فراهم است را استفاده کند , حتی قبل از اینکه $r_1 \bowtie r_2$ به طور کامل به وسیله ی P1 محاسبه شود .

همچنین , به همان صورت که P2 رکورد ها را در $(r_1 \bowtie r_2) \bowtie r_3$ محاسبه می کند , این رکورد ها را برای P3 فراهم می کند , که پیوند این رکورد ها با r4 را محاسبه می کند .

موازی سازی خط لوله با یک تعداد کوچکی از پردازنده ها مفید است , اما به خوبی مقیاس پذیر نیست.اولا , زنجیره های خط لوله عموما طول کافی برای فراهم آوردن یک درجه ی بالا از موازی سازی را بدست نمی آورند . دوم اینکه , پایپ کشی عملوندهای رابطه ای که تا زمان دستیابی به همه ی ورودی ها خروجی تولید نمی کنند امکان پذیر نیست , مانند عمل تفاضل مجموعه ها. سوم اینکه , تنها تسریع مرزی برای موارد متعددی که هزینه ی اجرای یکی از عملوند ها بسیار بیشتر از دیگران است بدست می آید .

با در نظر گرفتن همه چیز , وقتی درجه ی موازی سازی بالا است , خط لوله یک منبع موازی سازی با اهمیت کمتر نسبت به پارتیشن بندی است .دلیل اصلی برای استفاده از خط لوله این است که اجراهای پایپ لاین می تواند از نوشتن نتایج میانی در دیسک جلوگیری کند .

۱۸,۶,۲ موازی سازی مستقل

عملیات در یک عبارت پرس وجو که به عملیات دیگر وابستگی ندارد می تواند در حالت موازی اجرا شود .این شکل از موازی سازی **موازی سازی مستقل** خوانده می شود .

پیوند $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ را در نظر بگیرید .به طور واضح ,ما می توانیم $temp_1 \leftarrow r_1 \bowtie r_2$ را با $\text{temp}_2 \leftarrow r_3 \bowtie r_4$ در حالت موازی محاسبه کنیم . وقتی این دو محاسبه کامل می شوند , ما محاسبه می کنیم :

$\text{temp}_1 \bowtie \text{temp}_2$

برای بدست آوردن موازی سازی بیشتر , ما میتوانیم پایپ کنیم رکورد ها را در $temp_1$ و $temp_2$ به درون محاسبات $temp_1 \bowtie temp_2$, که خودش به وسیله ی یک پیوند خط لوله انجام می شود (بخش 12.7.2.2) .
همانند موازی سازی خط لوله , موازی سازی مستقل درجه ی بالایی از موازی سازی را فراهم نمی کند وکمتر در سیستم های بالا از نظر موازی بودن مفید است , اگر چه با درجه ی پایین تری از موازی سازی مفید است .

۱۸,۷ بهینه سازی پرس وجو

بهینه ساز های پرس و جو سهم بزرگی در موفقیت تکنولوژی رابطه ای دارند .یادآوری می شود که یک بهینه ساز پرس وجو یک پرس و جو را می گیرد و ارزان ترین نقشه ی اجرا را در میان تعداد زیادی از نقشه های اجرای ممکن که جواب مشابه می دهد پیدا می کند .
بهینه سازی کننده های پرس و جو برای ارزیابی موازی پرس و جو بسیار پیچیده تر از آنهایی هستند که برای ارزیابی متوالی پرس و جو به کار می روند .اول اینکه , مدل های هزینه پیچیدگی بیشتری دارد, تا زمانی که هزینه های پارتیشن بندی بایستی محاسبه شود , و موضوعاتی مثل عدم توازن و درگیری منبع بایستی به حساب آیند .موضوعی که بیشتر اهمیت دارد این است که چطور یک پرس و جو را موازی سازی کنیم .فرض کنید که ما به طریقی یک عبارت را انتخاب کرده ایم (از میان آن هایی که هم ارز آن پرس وجو است) که بایستی برای ارزیابی پرس و جو استفاده شود .عبارت می تواند به وسیله ی یک درخت عملگر به نمایش در بیاید , همانند بخش 12.1
برای ارزیابی درخت عملگر در یک سیستم موازی , ما بایستی تصمیمات زیر را بگیریم :

- چطور هر عمل را موازی سازی کنیم , و چه تعداد پردازنده برای آن استفاده می شود .
- چه عملیاتی در میان پردازنده های متفاوت پایپ لاین می شوند , چه عملیاتی به طور مستقل در حالت موازی اجرا می شوند , و چه عملیاتی به صورت متوالی به اجرا در می آیند , یکی بعد از دیگری .

این تصمیمات جایگزین وظیفه ی زمان بندی درخت اجرا می شود .
تعیین کردن منابع از هر نوع ـــــ مثل پردازنده ها , دیسک ها , و حافظه ـــــ که بایستی به هر عمل در درخت اختصاص داده شود جنبه ی دیگری از مساله ی بهینه سازی است . برای نمونه , ممکن است عاقلانه باشد که از حداکثر مقدار موازی سازی که در دسترس است استفاده کنیم , اما ایده ی خوب این است که بعضی عملیات را به صورت موازی اجرا نکنیم .عملیاتی که نیازمندی های محاسباتی آن به صورت مشخص کوچکتر از سربار ارتباطات است بایستی با یکی از همسایگانش خوشه بندی شود .در غیر این صورت , مزیت موازی سازی به وسیله ی سربار ارتباطات خنثی می شود .
یک نگرانی این است که خطوط لوله ی بلند خودشان را به منابع خوب بهره برداری معطوف نمی دارند .تا زمانی که عملیات درشت دانه بندی شده اند , عمل نهایی خط لوله ممکن است برای به دست آوردن ورودی ها زمان زیادی منتظر بماند, در حالی که منابع ارزشمند , نظیر حافظه را نگه می دارد .از این رو از خط لوله های بلند بایستی اجتناب گردد .

تعداد برنامه های ارزیابی موازی از آنچه که انتخاب می شود بسیار بزرگتر از تعداد برنامه های ارزیابی متوالی است .بهینه سازی پرس و جو های موازی با در نظر گرفتن تمامی جایگزین ها بسیار گران تر از بهینه سازی پرس و جوهای متوالی است .از این رو , ما معمولا رهیافت های مکاشفه ای را برای کاهش تعداد برنامه های اجرای موازی که بایستی در نظر بگیریم انتخاب می کنیم .ما دو بحث اکتشافی رایج را اینجا توضیح می دهیم .

اولین بحث اکتشافی این است که تنها برنامه های ارزیابی که هر عمل را میان تمامی پردازنده ها موازی سازی می کند مورد توجه قرار دهیم , که هیچ خط لوله ای را استفاده نمی کند .این رهیافت در سیستم های Teradata استفاده می شود .

پیدا کردن بهترین برنامه ی اجرا شبیه انجام بهینه سازی پرس وجو در یک سیستم متوالی است .تفاوت های اصلی در این است که پارتیشن بندی چگونه اجرا می شود وچه فرمول هزینه ی تخمینی استفاده می شود .

راه حل اکتشافی دوم این است که موثرترین برنامه ی ارزیابی متوالی را انتخاب کنیم , و سپس عملیات را در برنامه ارزیابی موازی سازی کنیم .دیتابیس موازی آتشفشان یک مدل موازی سازی را که مدل **معاوضه ی عملگر** خوانده می شود را عمومی کرده است .این مدل پیاده سازی های موجود عملیات را استفاده می کند , بر روی کپی های محلی داده عمل می کند, به یک عمل معاوضه که داده را بین پردازنده های متفاوت نقل مکان می دهد پیوست شده است .عملگرهای معاوضه می تواند در یک برنامه ی ارزیابی برای انتقال آن به یک برنامه ی ارزیابی موازی مطرح شوند .

در عین حال بعد دیگر بهینه سازی طراحی سازمان فیزیکی ذخیره سازی برای تسریع پرس وجو هاست.سازمان فیزیکی بهینه برای پرس وجو های متفاوت فرق می کند.مدیر پایگاه داده بایستی یک سازمان فیزیکی خوب را انتخاب کند که برای ادغام پرس وجو های مورد انتظار دیتابیس خوب باشد .بنابراین, حیطه ی بهینه سازی موازی پرس وجو پیچیده است , و هنوز یک حیطه ی تحقیقات فعال است .

۱۸,۸ طراحی سیستم های موازی

تا کنون این فصل بر روی موازی سازی ذخیره سازی داده و پردازش پرس وجو متمرکز شده است . از آنجایی که سیستم های پایگاه داده ی موازی در مقیاس بزرگ اصولا برای ذخیره سازی حجم بزرگی از داده استفاده شده اند , و برای پردازش پرس وجو های پردازش تصمیم بر روی این چنین داده هایی , این عناوین بسیار در یک سیستم پایگاه داده ی موازی مهم هستند .بارگذاری موازی داده از منابع بیرونی نیاز مهمی است , اگر ما حجم بزرگی از داده های ورودی را بایستی به کار ببریم .

یک سیستم دیتابیس بزرگ موازی همچنین بایستی این عوامل دسترسی را آدرس دهی کند :

- قابلیت بازگشت به حالت اولیه در صورت خرابی تعدادی از پردازنده ها یا دیسک ها .
- سازمان دهی بر خط داده و تغییرات مدل .

ما این موضوعات را اینجا در نظر می گیریم .

با یک تعداد بزرگی از پردازنده ها و دیسک ها , احتمال اینکه حداقل یک پردازنده یا دیسک بد عمل خواهد کرد به طور قابل ملاحظه ای بزرگتر از یک سیستم تک پردازنده با یک دیسک است .یک سیستم موازی با یک طراحی ضعیف در صورتی که هر مولفه (پردازنده یا دیسک) خراب شود از کار خواهد افتاد .با فرض اینکه احتمال شکست یک پردازنده ی

تک یا دیسک کوچک است , احتمال خرابی سیستم با تعداد پردازنده ها و دیسک ها به صورت خطی بالا می رود .اگر یک پردازنده یا دیسک در هر پنج سال یک بار خراب می شده است , یک سیستم با یک صد پردازنده در هر 18 روز یک خرابی داشته است .

از این رو , سیستم های پایگاه داده ی موازی در مقیاس بزرگ , به عنوان مثال Teradata , و IBM Informix XPS , طراحی شده اند تا حتی اگر یک دیسک یا پردازنده از کار بیفتد قادر به کار باشند .داده ها حداقل میان دو پردازنده کپی می شوند .اگر یک پردازنده از کار بیفتد , داده ای که ذخیره شده است همچنان می تواند از دیگر پردازنده ها قابل دستیابی باشد.

سیستم پیگیری پردازنده های خراب شده را ادامه می دهد و کار را میان پردازنده هایی که در حال کار هستند توزیع می کند.

درخواست ها برای داده هایی که در سایت های خراب ذخیره شده اند به صورت اتوماتیک به سایت های پشتیبان که یک کپی از داده را ذخیره می کنند هدایت می شود .اگر همه داده های یک پردازنده ی A در پردازنده ی تک B کپی شده اند , B مجبور خواهد بود همه ی درخواست ها به A را به خوبی درخواست های خودش انجام دهد, و نتیجه این خواهد شد که B تنگنا می شود .از این رو , کپی های داده ی یک پردازنده در میان دیگر پردازنده ها پارتیشن بندی می شود .

وقتی ما با حجم های بزرگی از داده سروکار داریم (در محدوده ی ترابایت) , عملیات ساده , به عنوان مثال ایجاد اندیس ها , و تغییرات مدل , به عنوان مثال اضافه نمودن یک ستون به رابطه , می تواند زمان زیادی را بگیرد ـــــ شاید ساعت ها و یا حتی روزها.از این رو , این برای سیستم پایگاه داده قابل قبول نیست که در حالی که چنین عملیاتی در حال انجام است غیر قابل دسترس باشد . بیشتر سیستم های پایگاه داده اجازه می دهند چنین عملیاتی به صورت **آنلاین** اجرا شوند , به عبارت دیگر, در حالی که سیستم در حال اجرا کردن تراکنش های دیگر می باشد .

برای نمونه , **ساختمان فهرست آنلاین** را در نظر بگیرید .یک سیستم که این خصوصیت را پشتیبانی می کند وارد نمودن , حذف , و به روز رسانی بر روی یک رابطه را حتی به صورت یک شاخص که بر روی یک رابطه ساخته می شود اجازه می دهد .عمل ساخت شاخص به این دلیل نمی تواند در حالت اشتراکی کل رابطه را قفل کند , در غیر این صورت می تواند.در عوض , پردازش پیگیری کردن را بر روی بروز رسانی هایی که اتفاق می افتد ادامه می دهد در حالتی که فعال و تغییرات را درون فهرستی که ایجاد شده جای می دهد .(اکثر سیستم های پایگاه داده امروز ساخت شاخص آنلاین را پشتیبانی می کنند , از آن جایی که این شاخص بسیار مهم است حتی برای سیستم های پایگاه داده ی غیر موازی.)

در سال های اخیر , تعدادی از شرکت ها محصولات پایگاه داده ی موازی جدید را گسترش داده اند , شامل Netezza ,DATAllegro (که به وسیله ی مایکروسافت به دست آمده) , Greenplum , و Aster Data .

هر کدام از این محصولات بر روی سیستم هایی شامل ده ها هزار نود اجرا می شوند , که هر نود بر روی یک نمونه از دیتابیس زیرین به اجرا در می آید ; هر محصولی پارتیشن بندی داده را مدیریت می کند , و نیز پردازش موازی پرس وجوها, در میان نمونه های پایگاه داده .

Netezza, Greenplum و Aster Data استفاده می کنند PostgreSQL را به عنوان پایگاه داده ی زیرین ; DATAllegro در ابتدا Ingres را به عنوان سیستم پایگاه داده ی زیرین استفاده کرد , اما متعاقبا به SQL Server که مالکیت آن با مایکروسافت است حرکت کرد . به وسیله ی ساخت بر بالای یک سیستم پایگاه داده ی

موجود , این سیستم ها قادر هستند تا به کار ببرند اهرم مخزن داده, پردازش پرس وجو, و خصیصه های مدیریت تراکنش پایگاه داده ی زیرین را, آن ها را رایگان رها کنند برای تمرکز بر روی پارتیشن بندی داده (شامل تکثیر برای تحمل خطا) , ارتباطات میان پردازنده ای سریع , پردازش موازی پرس وجو, و بهینه سازی پرس و جوی موازی .مزیت دیگر استفاده از یک دامنه ی پایگاه داده ی عمومی مانند PostgreSQL این است که هزینه ی نرم افزار به ازای هر نود خیلی پایین است ; در طرف مقابل دیتابیس های تجاری یک هزینه ی پیش پردازش قابل ملاحظه دارند .
همچنین مهم است که اشاره کنیم Netezza و DATAllegro در حقیقت مخزن داده (وسیله) می فروشند , که شامل سخت افزار و نرم افزار است , به مشتریان اجازه می دهد تا پایگاه داده های موازی با کمترین تلاش بسازند .

۱۸,۹ موازی سازی بر روی پردازنده های چند هسته ای

موازی سازی در اکثر کامپیوتر های امروزی پیش پا افتاده شده است , حتی بعضی از کوچکترین ها , به علت تمایلات کنونی در معماری کامپیوتر .به عنوان یک نتیجه , همه ی سیستم های پایگاه داده ی امروزی به صورت مجازی بر روی یک پایه ی موازی اجرا می شوند .در این بخش , ما باید به صورت مختصر دلایل این گرایش معماری و تاثیراتی که بر روی طراحی سیستم پایگاه داده و پیاده سازی آن دارد را مورد کاوش قرار دهیم .

۱۸,۹,۱ موازی سازی در تقابل با سرعت

از زمان طلوع کامپیوترها , سرعت پردازنده با یک نرخ نمایی افزایش یافته است , هر 18 تا 24 ماه دوبرابر شده است .این افزایش از یک رشد نمایی در تعداد ترانزیستورها که می تواند بر روی یک قطعه ی واحد از تراشه ی سلیکون قرار گیرد نتیجه شده است , و به صورت عام به عنوان **قانون مور** شناخته می شود , که از نام موسس اینتل Gordon Moore برگرفته شده است . از نظر تکنیکی , قانون مور یک قانون نیست , اما ترجیحا یک پیشگویی و مشاهده راجع به موضوعات تکنولوژی است . تا زمان حال , افزایش در تعداد ترانزیستورها و کاهش در اندازه شان در هر صورت منجر به سریع تر شدن پردازنده ها گردیده است.هر چند که پیشرفت های تکنولوژی به ترقی ادامه می دهد تا به شکل پیشگویی قانون مور رفتار نماید , عامل دیگری پیدا شده که رشد در سرعت پردازنده را کند می کند .پردازنده های سریع از نظر انرژی بی کفایت هستند . این مساله ای غامض در اصطلاح مصرف انرژی و هزینه است , عمر باطری برای کامپیوتر های در حال حرکت , و اتلاف گرما (تمام نیرویی که استفاده شده در نهایت تبدیل به گرما می شود) .به عنوان یک نتیجه , پردازنده های مدرن نوعا تک پردازنده نیستند اما ترجیحا شامل چندین پردازنده بر روی یک تراشه هستند . برای برقراری تمایز میان پردازنده های چند تایی on-chip و پردازنده های سنتی , اصطلاح **هسته** برای یک پردازنده ی on-chip استفاده می شود.
بنابراین ما می گوییم که یک ماشین پردازنده ی چند هسته ای دارد .

۱۸,۹,۲حافظه ی پنهان و چند نخی

هر هسته قابلیت پردازش یک جریان مستقل از دستورالعمل های ماشین را داراست .هرچند , به خاطر اینکه پردازنده ها قادر هستند داده را سریعتر از آن که می توانند از حافظه ی اصلی به آن دسترسی داشته باشند پردازش نمایند , حافظه ی اصلی می تواند تنگنایی شود که کارایی کلی را محدود می کند.به این دلیل, طراحان کامپیوتر یک یا بیشترسطح از حافظه ی **نهان** را در سیستم کامپیوتر قرار می دهند . حافظه ی نهان پر هزینه تر از حافظه ی اصلی بر روی یک پایه ی per-byte است, اما زمان دسترسی سریع تری را پیشنهاد می دهد .در طراحی های چند سطحی حافظه ی نهان , سطوح L1 , L2 , وبه همین صورت نامیده می شوند , با L1 که سریعترین حافظه ی نهان است (و از این رو پر هزینه ترین در هر بایت و کوچکترین) , L2 سریعترین بعدی , و به همین ترتیب . نتیجه یک توسعه از سلسله مراتب حافظه است که در فصل 10 بحث کردیم شامل سطوح مختلفی از حافظه ی نهان در زیر حافظه ی اصلی .

هرچند که سیستم پایگاه داده می تواند انتقال داده بین دیسک و حافظه ی اصلی را کنترل کند , سخت افزار کامپیوتر کنترل انتقال داده میان سطوح مختلف حافظه ی نهان و بین حافظه ی نهان و حافظه ی اصلی را برقرار می کند .بر خلاف این کمبود کنترل مستقیم , کارایی سیستم پایگاه داده می تواند به این وسیله که حافظه ی نهان چطور به کار گرفته شده است تحت تاثیر قرار بگیرد .اگر یک هسته نیاز داشته باشد تا به یک قلم داده که در حافظه ی نهان نیست دست یابد , بایستی از حافظه ی اصلی واکشی شود .به این خاطر که حافظه ی اصلی بسیار کندتراز پردازنده هاست , یک مقدار قابل توجه از سرعت پردازش بالقوه ممکن است در حالی که یک هسته برای داده از حافظه ی اصلی منتظر می ماند از دست برود . این انتظارها **فقدان حافظه ی نهان** نامیده می شود .

یک راه که طراحان کامپیوتر تلاش می کنند تا فشار حاصل ا ز فقدان های حافظه ی نهان را محدود کنند با استفاده از چند نخی است .یک **نخ** جریانی از اجراست که حافظه ی اصلی را با نخ های دیگر که بر روی هسته ی مشابه در حال اجرا هستند به اشتراک می گذارد .اگر نخ هم اکنون در حال اجرا بر روی یک هسته است که مشکل فقدان حافظه ی نهان دارد (یا انواع دیگر انتظار) , هسته برای اجرای نخ دیگر اقدام می کند , بدین وسیله سرعت محاسبه را در حال انتظار به هدر نمی دهد .نخ ها به عنوان منبع دیگری از موازی سازی فراسوی تعدد هسته ها مطرح شده اند .هر نسل جدید از پردازنده ها هسته ها ونخ های بیشتری را پشتیبانی می کند .پردازنده ی The Sun UltraSPARC T2 تعداد 8 هسته دارد , هر کدام از آنها 8 نخ را پشتیبانی می کند , برای تعداد کلی 64 نخ بر روی یک تراشه ی پردازنده . گرایش معماری برای افزایش کند تر در سرعت که همراه شده است با رشد در تعداد هسته ها استنباط روشنی در طراحی سیستم پایگاه داده دارد , همچنانکه به زودی خواهیم دید .

۱۸,۹,۳ وفق دادن طراحی سیستم پایگاه داده برای معماری های مدرن

آشکار است که سیستم های پایگاه داده کاربرد ایده آلی برای بهره بردن از تعداد زیادی از هسته ها و نخ ها هستند , در حالی که سیستم های پایگاه داده تعداد بزرگی از تراکنش های همروند را پشتیبانی می کنند.اگرچه, تعداد مختلفی از عوامل که استفاده ی بهینه می برند از رقابت کردن پردازنده های مدرن وجود دارد .

همچنانکه ما درجه ی همروندی بالاتری را برای بهره بردن از موازی سازی پردازنده های مدرن اجازه می دهیم , تعداد داده ی مورد نیاز که بایستی در حافظه ی نهان باشد را هم افزایش می دهیم .این می تواند منجر به فقدان های حافظه ی نهان بیشتری گردد , شاید به حدی زیاد که حتی یک هسته ی چند نخی بایستی برای داده از حافظه منتظر بماند .

تراکنش های همزمان به نوعی از کنترل همروندی نیاز دارند تا خواص ACID که در فصل 14 بحث کردیم را تضمین نمایند .وقتی تراکنش های همروند به صورت مشترک به داده دست می یابند , برخی محدودیت ها بایستی بر روی دستیابی همروند اعمال شود .این محدودیت ها , خواه بر پایه ی قفل ها , برچسب زمانی , یا معتبر سازی باشد , انتظار و یا اتلاف کار را به دلیل توقف تراکنش ها نتیجه می دهد . برای جلوگیری از انتظار بیش از حد یا زیان کار , ایده آل است که تراکنش های همروند به ندرت تداخل داشته باشند , اما تلاش برای اطمینان از این موضوع می تواند مقادیر داده ای که در حافظه ی نهان نیاز است را افزایش دهد , که منجر به فقدان های حافظه ی نهان بیشتری می گردد .
در نهایت , اجزای اصلی از یک سیستم پایگاه داده ی مشترک با تمامی تراکنش ها وجود دارد .
در یک سیستم که از قفل گذاری استفاده می کند , جدول قفل به وسیله ی همه ی تراکنش ها به اشتراک گذاشته می شود و دستیابی به آن تبدیل به یک تنگنا می گردد.مشکل های مشابه برای اشکال دیگر کنترل همروندی وجود دارد .
به طور مشابه , مدیر بافر, مدیر ثبت وقایع , و مدیر ترمیم به همه ی تراکنش ها خدمت می دهند و تنگناهای بالقوه هستند .
به خاطر داشتن یک تعداد بزرگی از تراکنش های هم روند ممکن است بهره برداری بهینه از پردازنده های مدرن صورت نگیرد , مطلوب است که روش هایی پیدا گردد که اجازه می دهد هسته های چند تایی برروی یک تراکنش تکی کار کند .به این دلیل نیاز است که پردازشگر پرس و جوی دیتابیس روش های موثری برای موازی سازی پرس وجو ها بدون درخواست های بیش از اندازه بر روی حافظه ی نهان پیدا کند .این میتواند به وسیله ی ایجاد خط لوله ی عملیات دیتابیس از پرس وجوها و به وسیله ی پیدا کردن روش هایی برای موازی سازی منحصر به فرد عملیات دیتابیس انجام گردد .
انطباق طراحی سیستم پایگاه داده و پردازش پرس وجوها با سیستم های چند هسته ای و چند نخی یک زمینه ی تحقیقات فعال باقی می گذارد .برای جزییات بیشتر اطلاعات مربوط به فهرست را ببینید .

۱۸,۱۰ خلاصه

- پایگاه داده های موازی در طی ۲۰ سال گذشته مقبولیت تجاری قابل ملاحظه ای به دست آورده اند .
- در موازی سازی I/O , رابطه ها میان دیسک های در دسترس پارتیشن بندی شده اند بنابراین سریعتر می توانند بازیابی شوند .سه تکنیک پارتیشن بندی رایج که استفاده شده عبارتند از پارتیشن بندی round-robin , پارتیشن بندی درهم , و پارتیشن بندی محدوده ای .
- عدم توازن یک مشکل اساسی است , خصوصا با افزایش درجه ی موازی سازی .بردارهای پارتیشن بندی موازی , با استفاده از گراف های آماری (histogram) , و پارتیشن بندی پردازنده ی موازی در میان تکنیک های استفاده شده برای کاهش عدم توازن می باشند .
- در موازی سازی میان جستار , ما پرس وجو های متفاوت را به صورت همزمان برای افزایش توان عملیاتی اجرا می کنیم .
- موازی سازی درون جستار تلاش می کند تا هزینه ی اجرای یک پرس و جو را کاهش دهد.دو مدل موازی سازی درون جستار وجود دارد : موازی سازی درون عمل و میان عمل .

- ما موازی سازی درون عمل را برای اجرای عملیات رابطه ای , مانند مرتب سازی ها و پیوند ها , در حالت موازی استفاده می کنیم .موازی سازی درون عمل برای عملیات رابطه ای طبیعی است , چون آنها مجموعه گرا هستند.
- دو روش ساده برای موازی سازی یک عمل دودویی مانند پیوند وجود دارد .
  - در موازی سازی پارتیشن بندی شده , روابط به چندین قسمت تقسیم می شوند , و رکورد ها در ri فقط با رکورد های si پیوند شده اند .موازی سازی پارتیشن بندی شده فقط می تواند برای پیوند های طبیعی و همسان استفاده شود .
  - در تکه کردن و تکرار , هر دو رابطه پارتیشن بندی شده و هر پارتیشن تکرار شده است .در تکه کردن و تکرار نامتقارن , یکی از روابط کپی شده در حالی که دیگری پارتیشن شده است .بر خلاف موازی سازی پارتیشن بندی شده , تکه کردن و تکرار و تکه کردن و تکرار نامتقارن می تواند با هر شرط پیوندی استفاده شود . هر دو تکنیک موازی سازی می تواند در رابطه با هر تکنیک پیوندی به کار رود .
- در موازی سازی مستقل , عملیات متفاوت که به هم وابسته نیستند به صورت موازی اجرا می شده اند .
- در موازی سازی پایپ لاین شده , پردازنده ها نتایج یک عمل را به عمل دیگر به عنوان نتایج محاسبه شده , بدون انتظار برای خاتمه ی کل عمل ارسال می کنند .
- بهینه سازی پرس وجو در پایگاه داده های موازی به طور مشخص پیچیده تر از بهینه سازی پرس وجو در پایگاه داده های سری است .
- پردازنده های چند هسته ای مدرن مشکلات تحقیقات جدید در دیتابیس های موازی را معرفی می کنند .

مرور اصطلاحات

• وابستگی حافظه ی نهان	• پرس وجو های پشتیبان تصمیم
• موازی سازی درون جستار	• موازی سازی I/O
○ موازی سازی درون عمل	• پارتیشن بندی افقی
○ موازی سازی میان عمل	• تکنیک های پارتیشن بندی
• مرتب سازی موازی	○ Round-robin
○ مرتب سازی پارتیشن بندی محدوده ای	○ پارتیشن بندی در هم

○ مرتب سازی ادغامی موازی بیرونی	○ **پارتیشن بندی محدوده ای**
• موازی سازی داده	• **خصیصه ی پارتیشن بندی**
• پیوند موازی	• **بردار پارتیشن بندی**
○ پیوند پارتیشن بندی شده	• **پرس و جوی شاخص دار**
○ پیوند تکه کردن و تکرار	• **پرس و جوی بازه ای**
○ پیوند تکه کردن و تکرار نامتقارن	• **عدم توازن**
○ پیوند موازی درهم پارتیشن بندی شده	○ **عدم توازن در اجرا**
○ پیوند موازی حلقه ی تو در تو	○ **عدم توازن مقدار مشخصه**
• انتخاب موازی	○ **عدم توازن پارتیشن**
• حذف تکثیر موازی	• **اداره ی عدم توازن**
• نگاشت موازی	○ **بردار پارتیشن بندی متوازن**
• اجتماع موازی	○ **گراف آماری**
• هزینه ی ارزیابی موازی	○ **پردازنده های مجازی**
• زمانبندی	• **موازی سازی میان جستار**
• مدل معاوضه ی عملگر	• **موازی سازی میان عمل**
• طراحی سیستم های موازی	○ **موازی سازی پایپ لاین**
• ایجاد شاخص آنلاین	○ **موازی سازی مستقل**
• پردازنده های چند هسته ای	• **بهینه سازی پرس وجو**

تمرینات

۱۸,۱ در یک انتخاب بازه ای بر روی یک خصوصیت با پارتیشن بندی محدوده ای , این احتمال وجود دارد که فقط نیاز به دسترسی به یک دیسک وجود داشته باشد.مزایا و معایب این حالت را توضیح دهید .

۱۸,۲ چه مدل موازی سازی (میان جستار , میان عمل, یا درون عمل) احتمالا بایستی برای هر کدام از وظایف زیر بیشترین اهمیت را داشته باشد ?

a .افزایش توان عملیاتی یک سیستم با تعداد زیادی از پرس و جو های کوچک
b .افزایش توان عملیاتی یک سیستم با تعداد معدودی از پرس و جو های بزرگ ,
وقتی تعداد دیسک ها و پردازنده ها زیاد است

۱۸,۳ با موازی سازی پایپ لاین , گاهی اوقات ایده ی خوبی است که چندین عملیات در یک پایپ لاین بر روی یک پردازنده ی تک اجرا شود , حتی زمانی که تعداد زیادی پردازنده در دسترس است .

a. توضیح دهید چرا.
b.آیا استدلال هایی را که شما در بخشaتوسعه داده اید در صورتی که ماشین یک معماری حافظه ی اشتراکی داشت نگه میداشتید ؟ شرح دهید چرا یا چرا نه .
c . آیا استدلال های بخش aبا موازی سازی مستقل برقرار بود ؟
(یعنی آیا مواردی وجود دارد که حتی اگر عملیات پایپ لاین نیست و تعداد زیادی پردانده در دسترس است هنوز ایده ی خوبی باشد که چندین عملیات بر روی پردازنده ی مشابه اجرا شود ؟)

۱۸,۴ فرایند پیوند با استفاده از قطعه بندی و تکرار متقارن با پارتیشن بندی محدوده ای را در نظر بگیرید. چطور می توانید ارزیابی را بهینه سازی کنید اگر شرایط پیوند به شکل $|r.A - s.B| <= K$ باشد , که K یک ثابت کوچک است ؟ اینجا , |x| مقدار قدر مطلق x را نشان می دهد . یک پیوند با چنین شرایط پیوندی یک پیوند متحد شده نامیده می شود.

۱۸,۵ یادآوری می شود که گراف های آماری برای ساخت پارتیشن بندی محدوده ای با توازن بار ساخته می شوند .

a. فرض کنید که شما یک گراف آماری دارید که مقادیر مابین 1 و 100 هستند , و در 10 محدوده پارتیشن بندی شده اند ,
1-10,11-20,….,91-100 , با فرکانس 5 و 15,5,5,20,10,10,5,5,20,5 به ترتیب باشد .یک تابع پارتیشن بندی محدوده ای با توازن بار بدهید که مقادیر را درون 5 پارتیشن ذخیره کند .

b.الگوریتمی برای محاسبه ی یک پارتیشن محدوده ای متوازن با p پارتیشن بنویسید , یک گراف آماری با فرکانس توزیع شامل n محدوده داده شده است .

۱۸,۶ جهت اجتناب از گم شدن داده ها اگر یک پردازنده با شکست مواجه شد ، سیستم های دیتا بیس موازی(مقیاس بزرگ)یک کپی اضافه از هر قلم داده ای را روی دیسک ذخیره می کنند که به یک پردازنده ی متفاوت ضمیه می شود

a. به جای نگهداری کپی اضافه از اقلام داده ای از یک پردازنده در یک پردازنده ی بک آپ مجزا، ایده ی خوب این است که کپی های اقلام داده ای یک پردازنده را در مقابل چندین پردازنده قسمت کنیم.

b. چطور قسمت بندی پردازنده های مجازی می توانند به صورت موثری قسمت بندی کپی هایی که در بالا توصیف شد را پیاده سازی کنند.

c. مزایا و بازخوردهای استفاده از حافظه ی RAID به جای ذخیره کردن یک کپی اضافه از هر اقلام داده ای چیست؟

۱۸,۷ فرض کنید که ما می خواهیم یک ارتباط بزرگ را که تقسیم شده است را ایندکس کنیم. آیا ایده ی تقسیم بندی (شامل تقسیم بندی پردازنده ی مجازی) می تواند ایندکس گذاری شود؟ در جواب سوال، ۲ مورد زیر باید در نظر گرفته شود:

a.ایندکس در کجای صفت قسمت بندی ارتباط فعال است.

b.ایندکس در کجای صفت های دیگری غیر از صفت قسمت بندی فعال است.

۱۸,۸ فرض کنید یک بردار well-balanced range-partitioning برای یک ارتباط انتخاب می شود، اما ارتباط بصورت ترتیبی آپدیت می شود، که باعث بوجودآمدن عدم توازن می شود. حتی اگر قسمت بندی پردازنده مجازی استفاده شود، یک پردازنده ی مجازی مخصوص ممکن است با تعداد زیادی از رکوردها بعد از آپدیت کم بیاورد و نیاز به قسمت بندی دوباره باشد.

a. فرض کنید یک پردازنده مجازی یک تعداد قابل توجهی از رکورد ها را دارد. در توضیح اینکه چطور قسمت بندی با جدا سازی قسمت انجام می شود، بدان وسیله تعداد پردازنده های مجازی افزایش پیدا میکنند.

b. اگر بجای تخصیص round-robin از پردازنده های مجازی استفاده شود، پارتیشن های مجازی می تواند به پردازنده ها در یک روش دلخواه اختصاص داده شود، همراه با یک جدول نگاشت که تخصیص را نگهداری و دنبال می کند. اگر یک نود مخصوص بار بیش از اندازه ای نسبت به نودهای دیگر داشت، بار می تواند بالانس و توازن شود.

c .فرض کنید هیچ آپدیتی وجود ندارد، آیا پردازش پرس و جو در حال عملیات پارتیشن بندی مجبور است تا متوقف شود یا تخصیص مجدد پردازنده های مجازی تمام می شود؟

۱۸,۹ برای هرسه تکنینک پارتیشن بندی ،راند روبین، درهم و محدوده ای، یک مثال از یک پرس و جو برای هر کدام از تکنیک پارتیشن بندی که پر سرعت ترین پاسخ را فراهم می کنید بیاورید

۱۸,۱۰ چه عامل هایی می تواند در نتیجه ی عدم توازن یک رابطه باشد که روی یکی از صفت هایش پارتیشن می شود.

a.پارتیشن بندی درهم

b.پارتیشن بندی محدوده ای

کدامیک از موراد می تواند جهت کاهش عدم توازن انجام شود.

۱۸,۱۱ یک مثال از join بیاورید که یک equi-join ساده که برای کدام پارتیشن بندی موازی می تواند استفاده شود. کدام صفت ها باید برای پارتیشن بندی استفاده شوند.؟

۱۸,۱۲ یک راه خوب برای موازی سازی هر کدام از موارد زیر توصیف کنید.

a. عملیات مختلف

b. تجمیع با عمل count

c. تجمیع با عمل count distinct

d .تجمیع با عمل avg

e.left outer join، اگر شرط join شامل تنها برابری باشد.

f. left outer join، اگر شرط join در بر گیرنده ی مقایساتی غیر از برابری باشد.

g.Full Outter Join ، اگر شرط join در بر گیرنده ی مقایساتی غیر از برابری باشد

۱۸,۱۳ توصیف مزایا و بازخوردهای موازی سازی پایپ لاین

۱۸,۱۴ فرض کنید که شما می خواهید یک بار کاری را که شامل یک تعداد زیادی از تراکنش های کوچک می باشد را با استفاده از موازی سازی shared-nothing رسیدگی کنید.

a. آیا موازی سازی پرس و جو های درون جستار چنین وضعیتی را نیاز دارند؟

b. با چنین بار کاری، چه نوعی از عدم توازن قابل توجه خواهد بود؟

c. فرض کنید اغلب تراکنش ها به یک رکورد account دسترسی دارند، که شامل صفت account_type، و یک رکورد account_type_maste مرتبط است/. که اطلاعاتی درباره ی account type فراهم می کنند. بنظر شما چگونه پارتیشن و یا انتشار دیتا سرعت تراکنش ها را بهتر میکند؟ شما ممکن است فرض کنید که ارتباط account_type_maste به ندرت آپدیت می شود.

۱۸,۱۵ مشخصه ای که یک رابطه بر اساس آن پارتیشن بندی می شود می تواند یک تاثیر مشخصی بر روی هزینه ی پرس و جو داشته باشد .

a.یک حجم کار از پرس وجو های SQL بر روی یک رابطه ی تک داده شده است , چه مشخصه هایی برای پارتیشن بندی کاندید شده اند ؟

b.چطور ممکن بود که شما از بین تکنیک های متناوب پارتیشن بندی , براساس حجم کار شروع به انتخاب کنید ؟

c.آیا امکان پذیر است که یک رابطه را بر روی بیشتر از یک خصوصیت پارتیشن بندی کنیم ؟ پاسخ خود را شرح دهید .