الگوهای پایگاه داده توزیع شده

مدیر، کسی را که، کارها را بین افرادی توزیع می‌کند که قادر به انجام آن نیستند، سرزنش می‌کند.

Franz Schubert

سرعت ارتباطات شگفت‌انگیز است. بنابراین این مسئله درست است که سرعت می‌توانند توزیع اطلاعات نادرست را چندین برابر کند.

Edward R.Murrow

مدیران برنامه‌های کاربردی وب به‌طور سنتی، به‌هنگام تقاضای برنامه‌ای بیش از ظرفیت پایگاه‌داده، دو راه بیشتر ندارند: مقیاس‌گذاری با افزایش قدرت سرورهای شخصی، یا مقیاس‌گذاری با اضافه کردن سرورهای بیشتر. برای بسیاری از پایگاه‌داده‌های رابطه‌ای، مقیاس‌گذاری با افزایش قدرت سرورهای شخصی گزینه عملی‌تری بود. اخیرا پایگاه‌داده‌های رابطه‌ای، گزینه‌ی خوشه‌بندی را فراهم نمی‌کردند، درحالی‌که پردازنده و حافظه عرضه شده توسط یک سرور به‌طور مداوم و به‌صورت تصاعدی و بنا به قانون Moore در حال افزایش بود. در نتیجه، مقیاس‌گذاری با اضافه کردن سرورهای بیشتر نه‌تنها عملی نیست بلکه لازم هم نیست.

بااین‌حال، زمانی‌که حجم‌کار پایگاه‌داده از برنامه‌های کلاینت سرور درحال اجرا به پشت دیوار آتش برنامه‌های کاربردی وب با دامنه بالقوه جهانی منتقل می‌شود، پشتیبانی از حجم‌کار و دسترسی به نیازمندی‌ها بر روی تک سرور به‌طور فزاینده‌ای دشوار می‌شود. علاوه براین، برنامه‌های کاربردی اینترنت اغلب در معرض رشد غیرقابل پیش‌بینی و عظیمی در حجم‌کار هستند: بنابراین برای یک نرم‌افزار، "go viral" امکان‌پذیر می‌شود و به‌طور ناگهانی رشد نمایی در تقاضا را تجربه می‌کند. نقطه شیرین اقتصادی برای سخت‌افزار کامپیوتر و الزامات رشد فزاینده‌ای در خوشه‌های سرور به جای تک سرور داشته است. بنابراین یک راه‌حل مقیاس‌پذیر برای پایگاه‌داده ضروری به نظر می‌رسد.

برای پرداختن به خواسته‌های برنامه‌های کاربردی در مقیاس وب، نسل جدیدی از پایگاه‌داده‌های رابطه‌ای توزیع‌شده پدید آمدند. در این فصل، معماری این سیستم پایگاه‌داده توزیع شده بحث شده است.

پایگاه‌داده‌های رابطه‌ای توزیع‌شده

سیستم‌های پایگاه‌داده برای اولین بار برای اجرا بر روی یک کامپیوتر طراحی شده بودند. در واقع، قبل از انقلاب کلاینت سرور، تمام اجزای برنامه‌های کاربردی از جمله پایگاه‌داده‌های اولیه، تماما در یک سیستم واحد اجرا می‌شدند: پردازنده مرکزی.

در این مدل متمرکز، تمام کدهای برنامه برروی سرور اجرا می‌شود و کاربران با کد برنامه از طریق پایانه‌های dump (پایانه‌ها "dump" هستند چرا که حاوی هیچ کد نرم‌افزاری نیستند) ارتباط برقرار می‌کنند. در مدل کلاینت سرور، منطق کسب‌وکار به اجرا درآمده معمولا رایانه‌های شخصی ویندوز هستند-که با یک تماس پایان، با سرور پایگاه‌داده ارتباط برقرار می‌کنند. در برنامه‌های کاربردی اینترنت، منطق کسب‌وکار در یک یا چند سرور برنامه‌های تحت وب اجرا می‌شد، درحالی‌که منطق ارائه شده بین مرورگر وب و سرور برنامه کاربردی، به اشتراک گذاشته می‌شود و هنوز هم تقریبا همیشه با یک سرور پایگاه‌داده ارتباط برقرار می‌کند.

شکل 8-1 سه معماری را که، نشان‌دهنده‌ی چگونگی ارتباط هر الگو در سرور پایگاه‌داده است، نشان می‌دهد.

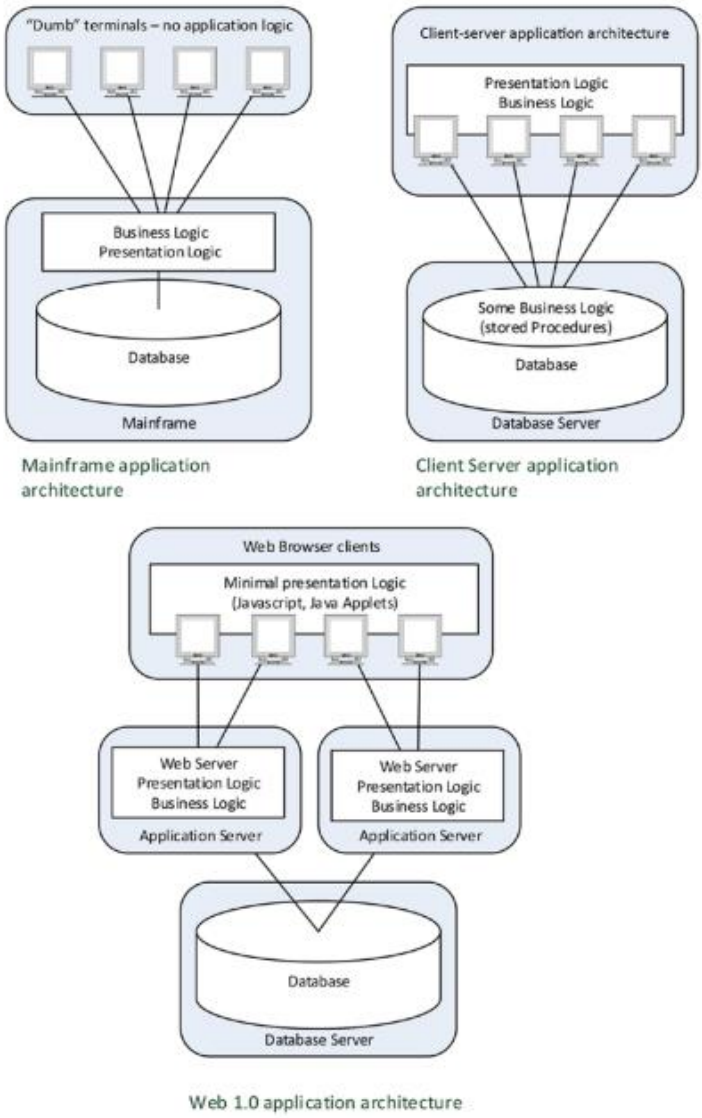


Figure 8-1. *Mainframe, client-server, and early web architectures relied on single, monolithic database servers*

تکرار

تکرار پایگاه‌داده ،در ابتدا به‌عنوان وسیله‌ای برای رسیدن به دسترسی بالا استفاده می‌شد. با استفاده از تکرار، مدیران پایگاه‌داده می‌توانند یک پایگاه‌داده آماده به کار را که می‌تواند بر روی پایگاه‌داده اولیه در صورت شکست عمل کند پیاده‌سازی می‌کنند.

تکرار پایگاه‌داده اغلب از تراکنش‌های صورت گرفته در بسیاری از پایگاه‌داده‌های رابطه‌ای برای حمایت از تراکنش‌های ACID استفاده می‌کند. در این مقاله الگوی تراکنش‌های ورودی در مورد حافظه پایگاه‌داده‌ها در فصل 7 معرفی شده

است. هنگامی که یک تراکنش در یک پایگاه‌داده ACID سازگار باشد، رکورد تراکنش بلافاصله پس از ورود به لاگ تراکنش برای حفاظت در برابر شکست نوشته می‌شود. نظارت بر فرآیند تکرار در لاگ تراکنش‌ها می‌تواند منجر به تغییرات پشتیبان‌گیری در پایگاه‌داده گردد، در نتیجه یک کپی ایجاد می‌کند.

شکل 8-2 روش تکثیر مبتنی بر ورود به سیستم را نشان می‌دهد. تراکنش‌های پایگاه‌داده در یک فایل پایگاه‌داده‌ی ناهمزمان "تنبل" نوشته می‌شود (1)، اما تراکنش پایگاه‌داده بلافاصله وارد حالت commit می‌شود (2). روند تکرار بر ورود تراکنش‌ها و اعمال معاملات همانند نوشتن فقط خواندنی در پایگاه‌داده نظارت می‌کند (3). تکرار معمولا ناهمزمان است، اما در برخی از پایگاه‌داده‌ها commit می‌تواند تا تکرار تراکنش به تعویق بیافتد.

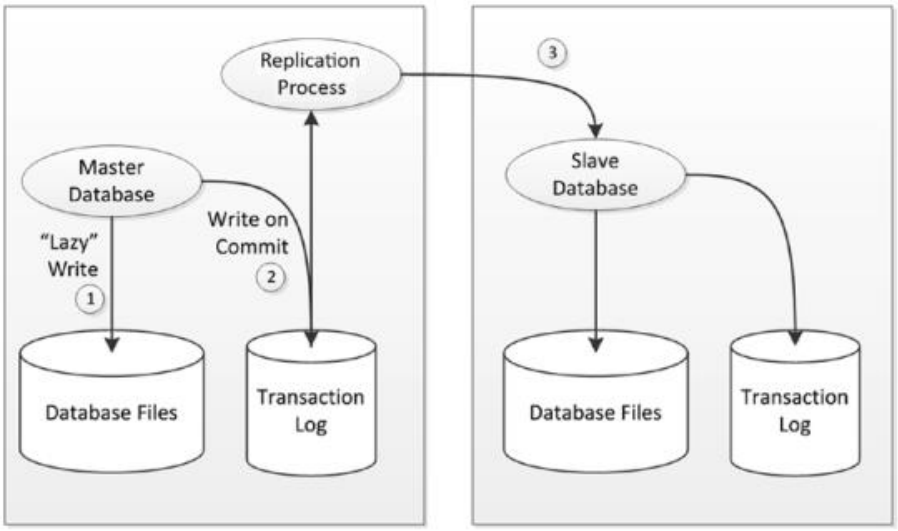


Figure 8-2. *Log-based replication*

همانطور که در فصل 3 دیدیم، تکرار، اولین گام به سوی پایگاه‌داده‌ی توزیع‌شده در سرورهای متعدد است. با استفاده از تکرار، حجم‌کار خواندن را می‌توان برای توزیع مقیاس‌گذاری کرد، اگر چه تراکنش‌های پایگاه‌داده هنوز هم باید به نسخه اصلی اعمال شود.

دیسک به‌اشتراک گذاشته شده و به اشتراک گذاشته نشده

الگوی تکرار برای توزیع حجم‌کار پایگاه‌داده برای توزیع فعالیت خواندن توسط سرورهای متعدد به خوبی عمل می‌کند، اما بارهای نوشتن تراکنش‌ها توزیع نمی‌شود، که هنوز هم باید به‌طور انحصاری به سرور اصلی ارسال شوند.

تکرار، برای توزیع حجم‌کار انبار داده‌ها محدودیت دارد. حجم‌کار OLTP معمولا شامل تعداد زیادی درخواست کوتاه مدت است. بااین‌حال، در یک محیط انبار داده‌ها، حجم‌کار معمولا تعداد کوچکتری از اطلاعات فشرده شده است. در سرورهای پایگاه‌داده، پرسش عظیم توسط فرایندهای متعدد یا موضوعات انجام می‌گیرد، که هر کدام می‌تواند هسته پردازنده جداگانه‌ای را با کانال‌های ورودی و خروجی متعدد استفاده کند.

موازی‌سازی یک پرس‌وجو در چندین سرور پایگاه داده مختلف، نیاز به یک رویکرد جدید دارد. فروشندگان انبار داده یک راه‌حل برای این مشکل با اجرای یک پایگاه‌داده مشترک ارائه کرده‌اند. همانند مفاهیم بسیاری در جهان رابطه، ایده به اشتراک گذاشتن هیچ چیزی مهمترین آنها بود که در سال 1980 توسط Michael Stonebraker برنامه‌ریزی شد. سرور پایگاه‌داده ممکن است به‌صورت زیر طبقه‌بندی شود:

• به اشتراک گذاشتن هر چیزی: در این مورد، هر فرایند پایگاه‌داده، حافظه، CPU، و منابع دیسک یکسانی را به اشتراک می‌گذارد. به اشتراک‌گذاری حافظه نشان می‌دهد که هر فرایند در سرور یکسانی است و از این رو معماری پایگاه‌داده تک گره است.

• دیسک به اشتراک گذاشته شده: در این مورد، فرآیندهای پایگاه‌داده ممکن است بر روی گره‌های جداگانه در خوشه وجود داشته باشند و به پردازنده و حافظه سرور که بر روی آنها قرار دارند دسترسی یابند. بااین‌حال، هر فرآیند دارای دسترسی برابری به دستگاه‌های دیسک است، که در سراسر گره‌های خوشه به اشتراک گذاشته شده‌اند.

• به اشتراک گذاشتن هیچ چیزی: در این مورد، هر گره در خوشه نه تنها به حافظه خود و CPU دسترسی دارد بلکه به دستگاه‌های دیسک اختصاص داده شده و زیر مجموعه پایگاه‌داده خود نیز دسترسی دارد. ما چند نمونه از معماری اشتراک هیچ چیزی را در کتاب حاضر بیان کرده‌ایم، از جمله طراحی MySQL اشتراکی در فصل 3 و طرح پارتیشن‌بندی VoltDB در فصل 7.

مدل اشتراک هیچ چیزی، پایه و اساس سیستم‌های چند پایگاه‌داده‌ای خوشه‌ای، مانند Teradata است. که یک مدل جذاب برای وظایف سنگین انبار داده‌ها فراهم می‌کند که می‌تواند به راحتی در گره‌های متعدد براساس دسترسی به داده‌های آنها موازی‌سازی شود. برای یک سیستم که آرزوی به حداکثر رساندن حجم‌کار براساس خواندن را دارد،

پیاده‌سازی آن به‌طور قابل توجهی آسان است. پیاده‌سازی پایگاه‌داده‌ها با مدل اشتراک هر چیزی، اغلب خود را به‌عنوان پردازنده‌های موازی پایگاه‌داده (MPP) معرفی می‌کنند. شکل 8-3 مدل اشتراک هیچ چیزی را نشان می‌دهد.

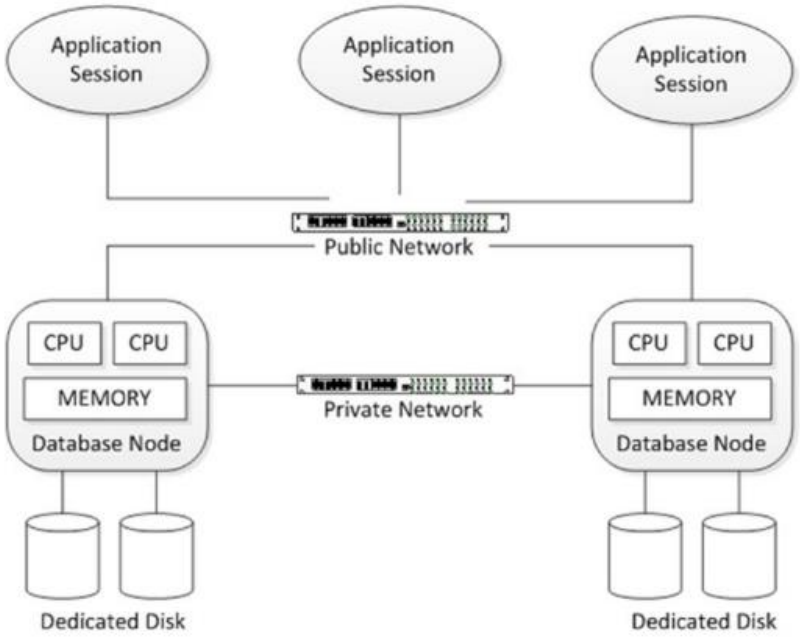


Figure 8-3. *Shared-nothing database architecture*

معماری مشترک هیچ چیزی، تمایل به شکستن در حالات کاربردی دارد، چرا که نیاز به هماهنگ‌سازی تراکنش‌ها دارد که ممکن است داده‌ها بر روی گره‌های متعدد را ملاقات کند. از آنجا که تراکنش‌های ACID، "همه یا هیچ چیز،" هستند، بنابراین هماهنگ‌سازی برای تمام گره‌ها در اجرای تراکنش ضروری است. این هماهنگی، که به‌عنوان commit دو فازی شناخته شده است، برای پیاده‌سازی مشکل است و ممکن است نتایج "in doubt" در تراکنش‌ها ایجاد کند و موجب تضعیف عملکرد تراکنش‌ها گردد.

اشکال دیگر معماری اشتراک هر چیزی این است که بدون پارتیشن‌بندی دقیق، حجم‌کار خوشه نامتعادل می‌شود. حفظ پارتیشن‌بندی درست یک فعالیت عملیاتی است. زمانی که گرهی از خوشه اضافه یا حذف می‌شود، ایجاد توازن مورد نیاز است.

معماری به اشتراک‌گذاری دیسک، از لحاظ نظری مقیاس‌پذیری بیشتری را منجر می‌شود و حذف آن نیاز به انجام عملیات توازن دارد. این سرویس همچنین یک راه‌حل در دسترس مقرون به صرفه‌تر ارائه می‌کند، زیرا هیچ گرهی مسئولیت منحصر به فردی برای مجموعه‌ای خاص از داده‌ها ندارد. در اشتراک هیچ چیزی، نتایج خرابی گره در بخشی

از پایگاه‌داده در دسترس نیست، درحالی‌که در اشتراک دیسک، گره باقی‌مانده قادر به از سرگیری مسئولیت گره شکست خورده است.

چالش معماری اشتراک دیسک، ضرورت در هماهنگی داده‌های cache شده در سراسر گره است. بدون کش در حافظه، عملکرد تمام عملیات بنا به سرعت دیسک کاهش می‌یابد. اما برای حفظ تداوم داده‌ها در تمام گره‌ها، هر گره نیاز به حفظ کش سازگار دارد. نگهداری از این کَش یک فشار بر روی شبکه و بین گره‌ها قرار می‌دهد که پیاده‌سازی موفق آن را دچار مشکل می‌کند.

تا به امروز، تنها بازمانده تجاری و موفق RDBMS اشتراک دیسک در پایگاه‌داده‌ی خوشه‌ای نرم‌افزار اوراکل (RAC) است. RAC پایه و اساس پایگاه‌داده اوراکل است. شکل 8-4 مدل اشتراک دیسک را نشان می‌دهد.

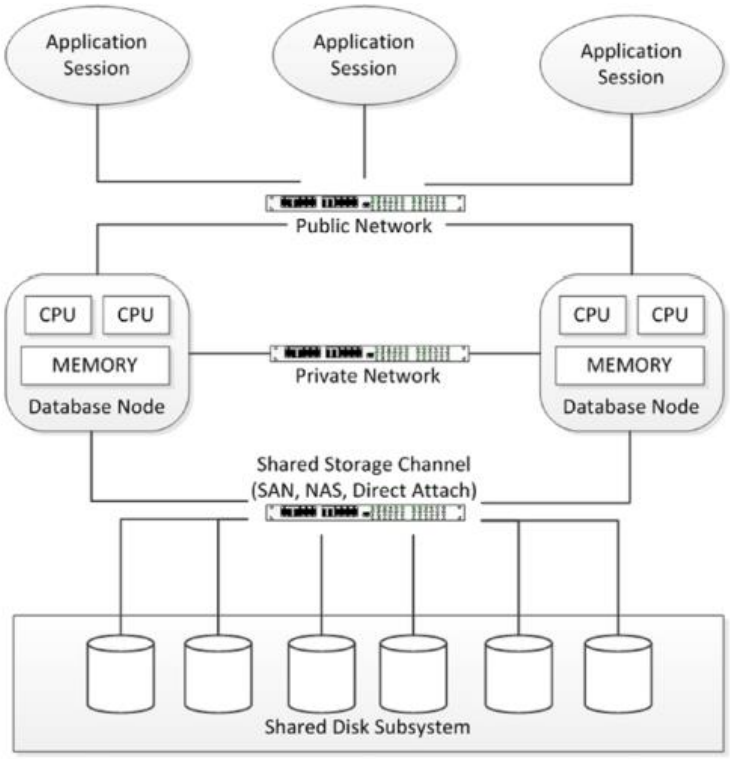


Figure 8-4. *Shared-disk database architecture*

پایگاه‌داده‌ی توزیع شده‌ی غیررابطه‌ای

حفظ تمامیت تراکنش ACID در سراسر گره‌های متعدد در یک پایگاه‌داده‌ی رابطه‌ای توزیع شده، چالش قابل توجهی دارد. بااین‌حال، در سیستم‌های پایگاه‌داده‌ی غیررابطه‌ای، از ACID پشتیبانی نمی‌کنند. برای پایگاه‌داده‌ی توزیع شده غیررابطه‌ای، ملاحظات زیر قابل‌توجه است:

• در دسترس بودن ثبات و پایداری: همانطور که در فصل 3 دیدیم، قضیه CAP استدلال می‌کند که یک هدف پایگاه‌داده توزیع شده فراتر از یک مقیاس محلی در شبکه است که باید بین دسترس بودن و ثبات در شبکه یکی را انتخاب کند. یک پایگاه‌داده ACID سازگار موظف است ثبات را بر روی دیگر عوامل برقرار کند. با این حال، یک پایگاه‌داده غیررابطه‌ای بدون انطباق با محدودیت ACID می‌تواند تعادل متفاوتی ایجاد کند.

• اقتصاد سخت‌افزار: زمانی که یک سیستم به هزاران و یا صدها هزار گره متصل می‌شود تفاوت‌های کوچک در هزینه سرورهای شخصی به سرعت افزایش می‌یابد. بنابراین، یک معماری پایگاه‌داده با استفاده از بهترین نسبت قیمت / عملکرد موجود مقرون به صرفه خواهد بود. علاوه‌براین، ممکن است لازم باشد تا با اختلافات بین تنظیمات سرور کنار بیاییم، به‌طوری‌که سخت افزار جدید را می‌توان به خوشه پایگاه‌داده بدون نیاز به تمام گره‌های موجود برای آخرین مشخصات سخت‌افزاری به روز شده اضافه کنیم.

• انعطاف‌پذیری: در یک خوشه عظیم پایگاه‌داده، گره‌ها از زمانی به زمان دیگر دچار شکست می‌شوند. به هنگام شکست، می‌توان از دست دادن داده‌ها، وقفه دردسترسی، یا شاید حتی شکست در سطح تراکنش را تحمل کرد.

سه گروه معماری پایگاه‌داده توزیع شده توسط نسل بعدی پایگاه‌داده‌ها وجود دارد. سه مدل عبارتند از:

• تغییرات در معماری اشتراکی سنتی، که در آن داده‌ها در سراسر گره براساس ارزش "کلید اشتراکی" تقسیم‌بندی می‌شوند.

• تنوع در مدل هادوپ HDFS / HBase، که در آن یک "استاد دانای مطلق" تعیین می‌کند که داده‌ها باید براساس بار و عوامل دیگر در خوشه قرار گیرند.

• مدل هش سازگار آمازون Dynamo، که در آن داده‌ها در سراسر گره‌های خوشه براساس هش ریاضی قابل پیش‌بینی در یک مقدار کلید توزیع می‌شوند.

تکرار ممکن است در درون هر یک از این معماری‌ها به‌منظور اطمیناندر مورد از دست ندادن داده در صورت بروز شکست در سرور به‌صورت ذاتی وجود دارد، اگرچه استراتژی تکثیر متفاوت باشد. به نمونه‌هایی از این روش‌ها در باقی فصل نگاهی می‌اندازیم. در این مقاله از MongoDB به‌عنوان مثالی از یک معماری اشتراکی، HBase به‌عنوان نمونه‌ای از یک استاد دانای مطلق و Cassandra به‌عنوان مثالی از Dynamo استفاده می‌کنیم.

MongoDB shardingو تکرار

MongoDB از اشتراک‌گذاری برای ارائه قابلیت مقیاس‌پذیری و تکرار برای دسترسی بالا پشتیبانی می‌کند. با اینکه می‌توانند به‌طور مستقل از دیگر اجرا شوند، ما هر دو آنها یک سناریو تولید می‌کنند.

sharding

نمایشی سطح بالا از معماری MongoDB اشتراکی در شکل 8-5 نشان داده شده است. هر قسمت توسط یک پایگاه‌داده MongoDB متمایز اجرا می‌شود، که در بسیاری از جهات گسترده‌تر از سرور است (1). پایگاه‌داده‌ی MongoDB جداگانه-سرور پیکربندی (2)، شامل آَبرداده است که می‌تواند برای تعیین چگونگی توزیع داده‌ها در سراسر ذرات استفاده شود. روند روتر (3) مسئول مسیریابی درخواست‌ها به سرور مناسب است.

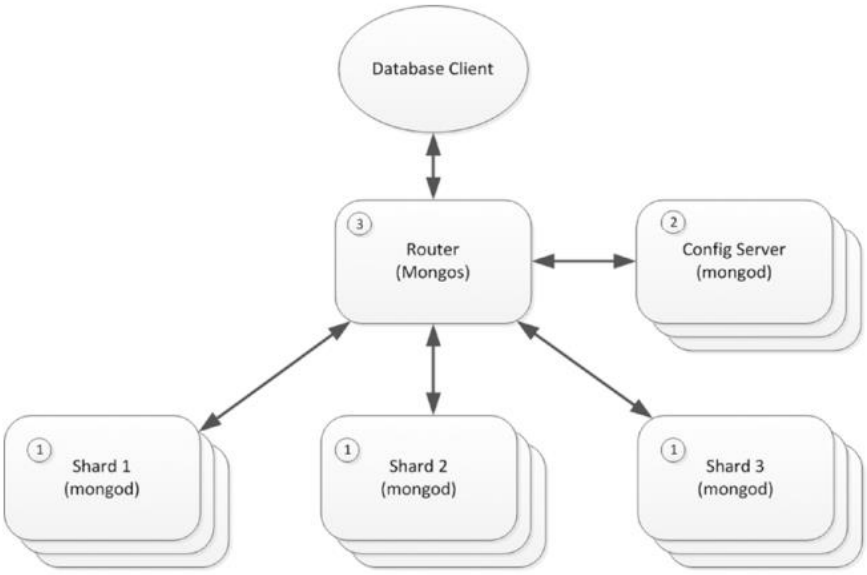


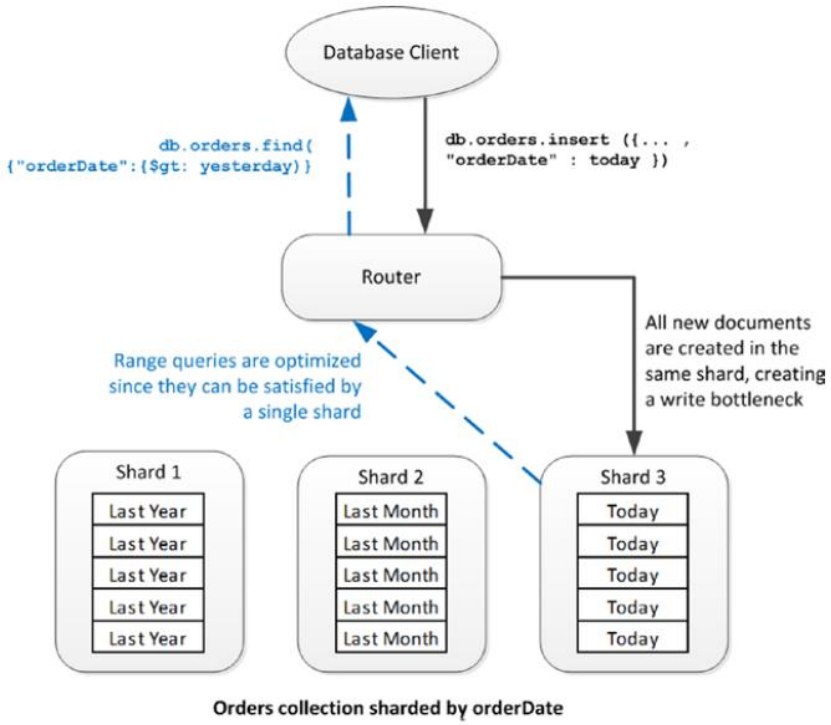
Figure 8-5. *MongoDB sharding architecture*

ممکن است در MongoDB، مجموعه‌ای برای ذخیره اسناد JSON استفاده شود که معمولا برخی از ویژگی‌های مشترک را دارند. برای اجتماع یک مجموعه، یک کلید shard انتخاب می‌شود که یک یا چند ویژگی نمایه برای تعیین توزیع اسناد در سراسر ذرات استفاده شده است. ساختار درختی شاخص MongoDB شامل اطلاعات لازم برای توزیع کلید به‌طورمساوی در سراسر ذرات است.

مکانیزم sharding

توزیع داده‌ها در سراسر ذرات می‌تواند براساس محدوده و یا براساس هش باشد. در پارتیشن‌بندی مبتنی بر محدوده، هر shard به یک محدوده خاص از مقادیر کلیدی shard اختصاص داده می‌شود. MongoDB درباره توزیع مقادیر کلیدی در شاخص برای اطمینان از هر shard اختصاص داده شده به همان تعداد کلید مشورت می‌گیرد. در sharding مبتنی بر هش، کلیدها براساس یک تابع هش به کلید shard توزیع شده اعمال می‌شوند.

هر یک از روش‌ها مزایا و معایبی دارد. شکل 8-6 عملکرد ذاتی درsharding محدوده و هش را برای درج و نمایش پرس‌وجوها نشان می‌دهد.



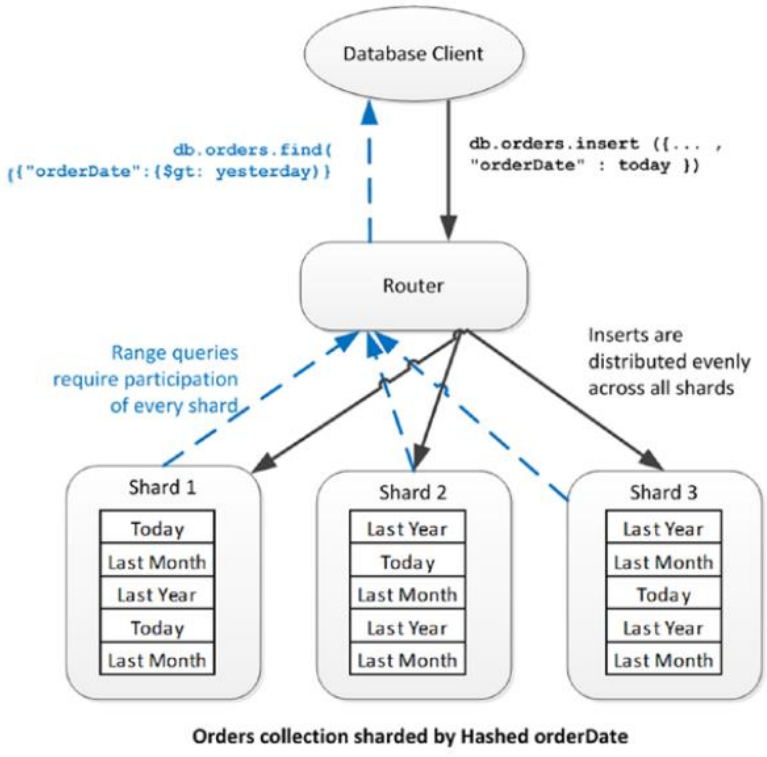


Figure 8-6. *Comparison of range and hash sharding in MongoDB*

پارتیشن‌بندی مبتنی بر محدوده اجرای کارآمدتر پرس‌وجوها را اجازه می‌دهد، چرا که این پرس‌وجو اغلب می‌تواند با دسترسی به shard حل‌وفصل شود. sharding مبتنی بر هش مستلزم پرس‌وجوهای وسیعی است که می‌تواند به

تمام shard ها دسترسی داشته باشد. از سوی دیگر، sharding مبتنی بر هش به احتمال بیشتر برای توزیع اسناد "hot" (سفارشات پر نشده و یا پست‌های اخیر) به‌طور مساوی در خوشه است، در نتیجه توازن بار موثر خواهد بود.
بااین‌حال، هنگامی‌که پارتیشن‌بندی اعمال می‌شود و کلید shard به‌طور مداوم افزایش می‌یابد، بار تمایل به جمع تنها یکی از shard ها دارد، در نتیجه موجب عدم تعادل خوشه می‌شود. با پارتیشن‌بندی مبتنی بر هش، اسناد جدید به‌طور مساوی در تمام اعضای خوشه توزیع می‌شود. علاوه‌براین، اگرچه MongoDB برای توزیع مساوی کلید shard در سراسر خوشه تلاش می‌کند، ممکن است نقاط خاصی وجود داشته باشد که محدوده کلید shard موجب عدم تعادل بار گردد. sharding مبتنی بر هش احتمال دارد به‌طور مساوی توزیع بار را انجام دهد.
sharding برچسب آگاه به مدیر MongoDB این اجازه را می‌دهد که توزیع اسناد به shrad را تنظیم کند. با مرتبط کردن shard با برچسب، و ارتباط طیف وسیعی از کلیدها در یک مجموعه با همان برچسب، می‌توانید shard را تعیین کند. این مسئله می‌تواند برای آرشیو اطلاعات، ذخیره‌سازی کندتر یا ارتباط مستقیم داده‌های خاص به یک مرکز داده‌ی خاص یا جغرافیا استفاده شود.

تعادل خوشه

هنگامی که sharding مبتنی بر هش اجرا می‌شود، تعدادی از اسناد در هر shard تمایل به حفظ تعادل در اغلب موارد دارند. بااینحال، در یک سناریوی sharding مبتنی بر محدوده، عدم تعادل shard ها بسیار آسان است، به‌خصوص اگر کلید shrad، مانند افزایش خودکار کلید اولیهID در یک مقدار به‌طور مداوم افزایش یابد.
به‌همین دلیل، MongoDB به‌صورت دوره‌ای، تعادل shrdها در سراسر خوشه را ارزیابی می‌کند و عملیات توازن، در صورت نیاز انجام می‌گیرد. واحد توازن shard chunk است. shard از تکه‌های معمول MB64 تشکیل شده است- که حاوی مقادیر به‌هم پیوسته از کلید shard (یا از کلیدهای هش shard) است. اگر یک تکه به خوشه اضافه و یا از خوشه حذف شود و یا اگر تعادل‌سنج تشخیص دهد که یک تکه نامتعادل شده است، می‌تواند تکه‌ها را جابه‌جا کند. اگر تکه بیش از حد رشد کند، تقسیم می‌شود.

تکرار

Sharding تقریبا همیشه با تکرار ترکیب می‌شود تا از دردسترس بودن و مقیاس‌پذیری در یک تولید MongoDB اطمینان حاصل کند.

در MongoDB، اطلاعات می‌تواند در برخی ماشین‌ها توسط مجموعه تکرار، تکرار شوند. مجموعه تکرار شامل یک گره اولیه همراه با دو یا چند گره ثانویه است. گره اصلی تمام درخواست‌های نوشتن را می‌پذیرد، که غیرهمزمان با گره ثانویه تکثیر می‌شوند.

گره اولیه توسط انتخابات با مشارکت همه گره‌های موجود تعیین می‌شود. برای اینکه گره اولیه واجد شرایط باشد، باید قادر به برقراری ارتباط با بیش از نیمی از مجموعه‌ی تکرار باشد. این مسئله تضمین می‌کند که اگر یک شبکه یک مجموعه تکرار را به دو بخش تقسیم کند، تنها یکی از پارتیشن‌ها برای ایجاد گره اولیه تلاش خواهد کرد.

گره موفق اولیه براساس تعداد گره‌های در تماس با آن انتخاب می‌شود، بنابراین ممکن است توسط مدیر سیستم اختصاص داده شود. تنظیم اولویت 0 برای یک نمونه مانع از انتخاب آن به‌عنوان گره اولیه می‌شود.

اطلاعات اولیه ذخیره شده در مورد تغییرات سند در یک مجموعه در پایگاه‌داده محلی، oplog نامیده می‌شود.گره اولیه به طور مداوم در حال تلاش برای اعمال تغییرات به گره‌های ثانویه است.

کاربران مجموعه تکرار اغلب از طریق پیام‌های ضربان قلب ارتباط برقرار می‌کنند. اگر گره اولیه یافت شود قادر به دریافت پیام‌های ضربان قلب از بیش از نیمی از گره‌های ثانویه خواهد بود، سپس وضعیت اولیه خود را از دست داده و یک انتخابات جدید انجام خواهد شد. شکل 8-7 مجموعه تکرار سه عضوی و چگونگی تغییر گره اولیه توسط پارتیشن‌بندی شبکه را نشان می‌دهد.

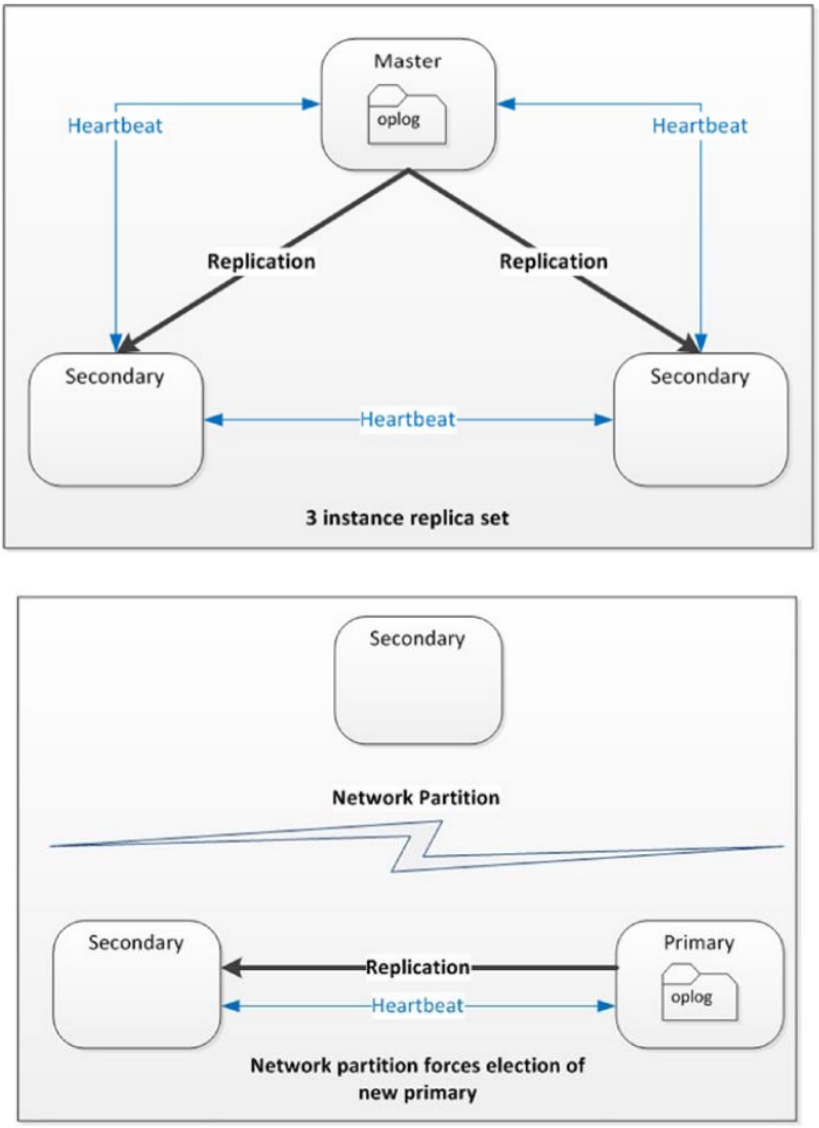


Figure 8-7. *MongoDB replica set and primary failover*

داوران، سرورهای ویژه‌ای هستند که می‌توانند در انتخابات اولیه رای دهند، اما داده را نگه نمی‌دارند. برای پایگاه داده‌های بزرگ، این داوران می‌توانند مانع از ضرورت ایجاد سرورهای اضافی شوند درغیراین‌صورت سرورهای غیرضروری برای اطمینان از به حد نصاب رسیدن دسترسی ایجاد می‌شوند.

نگرانی نوشتن و اولویت خواندن

نرم‌افزار MonoDB برخی از کنترل‌ها را بر عملیات خواندن و نوشتن اعمال می‌کند، که ثبات و در دسترس بودن را ارائه می‌کند.

• تنظیمات نگرانی در نوشتن، تعیین می‌کند که چه زمانی MongoDB عملیات نوشتن را تکمیل می‌کند. به‌طورپیش فرض، عملیات نوشتن یک بار و وقتی که گره اولیه دستور اصلاح را دریافت کرد کامل می‌شود. این به این معنی است که گره اولیه باید به هنگام از دست دادن داده با شکست مواجه شود. برای اطمینان از اینکه عملیات نوشتن فراتر از گره اولیه صورت گرفته است، مشتری می‌تواند یک تماس مسدود کردن ارسال کند، که تا ارسال توسط همه گره‌های ثانویه صبر می‌کند.

• اولویت خواندن به‌طوری تعیین می‌شود که در آن مشتری درخواست خواندن را می‌فرستد. به‌صورت پیش‌فرض، تمام درخواست‌های خواندن به گره اولیه ارسال می‌شود. بااین‌حال، کلاینت می‌تواند درخواست کند که اگر گره اصلی دردسترس نباشد، درخواست‌های خواندن به گره ثانویه و یا به هر سرور "نزدیکتری" فرستاده شوند. مورد آخر، زمان تاخیر کم و سازگاری بیشتری دارد.

تنظیمات پیش‌فرض برای اولویت خواندن و نوشتن، در MongoDB به‌صورت یک سیستم به‌شدت سازگار در نظر گرفته می‌شود: همه نسخه‌ی مشابهی از یک سند را مشاهده خواهند کرد. اجازه‌ی خواندن از گره ثانویه نتایج خوبی به همراه داشته است، مگر اینکه نگرانی در نوشتن طوری پیکربندی شود تا رسیدن به گره ثانویه بلاک گردد.

ثبات MongoDB در فصل بعد با جزئیات بیشتری بیان شده است.

HBase

HBase را می‌توان به‌صورت "پایگاه‌داده Hadoop" و به " BigTable منبع باز" تصور کرد. بنابراین می‌توانیم HBase را به‌عنوان یک مکانیسم برای دسترسی تصادفی به پایگاه‌داده بر روی سیستم فایل در HDFS هادوپ توصیف کرد یا می‌توان HBase را به‌عنوان یک پیاده‌سازی منبع باز از پایگاه‌داده BigTable گوگل تصور کرد که برای استفاده از HDFS جهت ذخیره‌سازی داده‌ها است. هر دو توصیف دقیق هستند: اگر چه از لحاظ نظری HBase را می‌توان بر روی هر فایل سیستم توزیع شده‌ای، یا حتی یک فایل توزیع‌نشده پیاده‌سازی کرد، معمولا بر روی هادوپ HDFS

پیاده‌سازی می‌شود و بسیاری از مفروضات معماری HBase منعکس‌کننده این موضوع است. ازسوی دیگر، HBase، دسترسی تصادفی در زمان واقعی به پایگاه‌داده را پیاده‌سازی می‌کند که اساسا از قابلیت‌های پایه هادوپ می‌باشد.

در بحث پیش رو، بر روی معماری HBase که معمولا بیشتر با آن مواجه می‌شویم تمرکز خواهیم کرد: همان‌طور که در HDFS پیاده‌سازی شده است. پیاده‌سازی HBase بر روی HDFS یک ترتیب ترکیبی، ترکیبی از اشتراک هیچ چیز و الگوهای خوشه‌بندی اشتراکی دیسک، ایجاد می‌کند. از یک طرف، هر گره HBase می‌تواند به هر عنصر از داده‌ها در پایگاه‌داده دسترسی داشته باشید چرا که همه داده‌ها از طریق HDFS قابل دسترسی هستند. از سوی دیگر، همکاری سرورهای HBase با HDFS DataNodes حالت عادی دارد، بدین معنی که هر گره برای پاسخگویی به زیرمجموعه منحصربه فرد از داده‌های ذخیره شده در دیسک محلی گرایش دارد.

در هر صورت، HDFS تضمین قابلیت اطمینان برای داده‌ها روی دیسک را فراهم می‌کند: معماری HBase برای نوشتن معکوس یا مدیریت شکست دیسک لازم نیست، چرا که این‌ها به‌صورت خودکار توسط سیستم HDFS قابل اعمال هستند.

در این مقاله، HDFS و معماری هادوپ در فصل 2 معرفی شده است؛ اگر نیاز به یادآوری دارید لطفا به آن فصل مراجعه کنید. HDFS یک سیستم فایل توزیع شده با استفاده از دیسک پیاده‌سازی می‌شود که به‌طور مستقیم به سرورهای DataNodes متصل است که یک خوشه از Hadoop را تشکیل می‌دهند. HDFS به‌طور خودکار افزونگی داده‌ها را مدیریت می‌کند: به‌طور پیش‌فرض، داده‌ها در سراسر سه DataNodes تکرار شده‌اند که یکی از آنها (در صورت امکان) بر روی یک سرور جداگانه واقع شده است.

جداول، مناطق، و RegionServers

HBase یک ستون گسترده براساس خصوصیات BigTable گوگل برای ذخیره‌سازی پیاده‌سازی می‌کند. این مدل برای داده‌ها در فصل 2 بیان شده است و در مورد آن در فصل 10 بیشتر بحث خواهد شد. در بحث حاضر، می‌توانیم

جدول HBase را مانند مجموعه داده در نظر بگیریم که بر روی دیسک‌ها و توسط تعدادی متغیر از فایل‌های HDFS به نام Hfiles پیاده‌سازی شده است.

تمام ردیف‌ها در جدول HBase توسط یک ردیف کلید منحصربه فرد مشخص شده‌اند. جدول با اندازه کوچک به پارتیشن‌های متعدد افقی تقسیم می‌شود. هر منطقه شامل یک محدوده به‌هم پیوسته و مرتب از مقادیر کلیدی است. این مورد شبیه‌سازی از طرح sharding مبتنی بر MongoDB است که قبلا در این فصل بیان شد.

دسترسی به خواندن یا نوشتن به یک منطقه توسط یک RegionServer کنترل می‌شود. هر RegionServer به‌طور معمول بر روی میزبان اختصاص داده شده اجرا می‌شود و به‌طورمعمول با هادوپ DataNode محلی شده است.

معمولا بیش از یک منطقه در هر RegionServer خواهد بود. هنگامی که مناطق رشد می‌کنند، به مناطق متعدد براساس سیاست‌های تنظیم تقسیم می‌شوند. مناطق ممکن است به صورت دستی نیز تقسیم شوند. این موضوع در ادامه فصل بیشتر بحث خواهد شد.

نصب و راه‌اندازی HBase شامل یک سرویس هادوپ است که در سراسر گره‌های متعدد پیاده‌سازی شده است. Hbase ممکن است این گروه را همراه بقیه خوشه‌های هادوپ و برای استفاده از خدمات اختصاص داده شده به اشتراک بگذارید.

هنگامی که یک مشتری HBase آرزو خواندن و نوشتن یک مقدار کلیدی خاص را دارد، از Zookeeper آدرس RegionServer را برای کنترل HBase پرس‌وجو می‌کند. این کاتالوگ شامل جداول -ROOT- و .META. است، که RegionServers مسئول در محدوده کلید خاص را شناسایی می‌کند. سپس مشتری با RegionServer ارتباط برقرار می‌کند و درخواست خواندن و نوشتن مقدار کلید را ارسال می‌کند.

سرور اصلی HBase انواع وظایف را انجام می‌دهد. به‌طور خاص، تعادل میان مناطق RegionServers را کنترل می‌کند. اگر یک RegionServer اضافه یا حذف شده باشد، سرور اصلی مناطق آن را استقرار مجدد در RegionServers سازماندهی خواهد کرد.

شکل 8-8 برخی از عناصر این معماری را نشان می‌دهد. یک سرویس‌گیرنده HBase با Zookeeper برای تعیین محل فهرست جداول HBase مشورت می‌کند (1)، کدام یک می‌تواند برای تعیین محل مناسب RegionServer استفاده شود (2) پس از آن مشتری درخواست خواندن و یا تغییر یک ارزش کلیدی مناسب را به RegionServer ارسال می‌کند. (3) RegionServer به برای دسترسی به فایل‌های مناسب دیسک که در HDFS واقع شده‌اند عمل خواندن و نوشتن را شروع می‌کند (4).

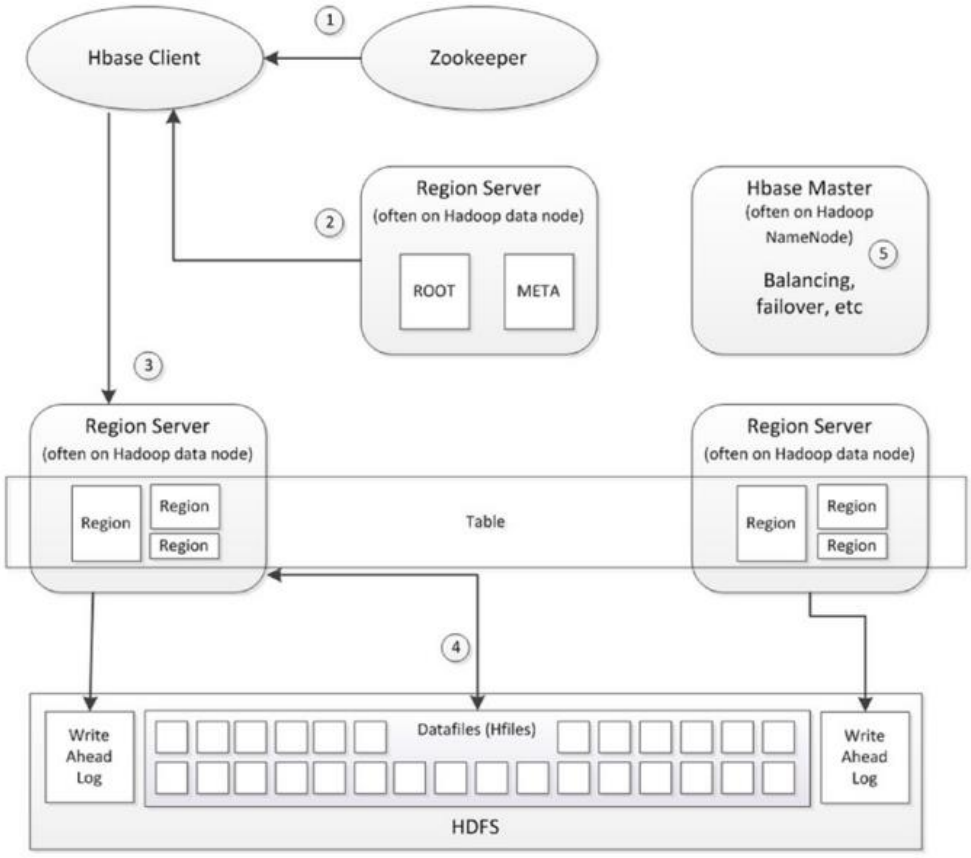


Figure 8-8. *HBase architecture*

ذخیره و محلیت داده

RegionServer شامل یک بلوک کش است که می‌تواند بسیاری از بارها برای خواندن از حافظه را برآورده کند و MemStore قبل از اینکه به دیسک برسد در حافظه شروع به نوشتن می‌کند. بااین حال، برای اطمینان از پایداری عملیات نوشتن، هر RegionServer یک لاگ ورود اختصاص داده شده برای نوشتن (WAL) دارد، که تمام عملیات

نوشتن را به HDFS اختصاص می‌دهد. این معماری یک پیاده‌سازی از الگوی درخت ادغام براساس ساختار ورود به سیستم (LSM) است که به‌طورکامل در فصل 10 شرح داده شده است.

RegionServer می‌تواند به‌عنوان یک مشتری HDFS عمومی برای برقراری ارتباط با HDFS NameNode و انجام عملیات خواندن و نوشتن در فایل عمل کند. در معمول‌ترین سناریو تولید، هر RegionServer در Hadoop NameNode واقع شده است و به‌عنوان نتیجه، اطلاعات در منطقه با همکاری RegionServer و برای ارائه محل خوب برای داده قرار می‌گیرند. محلیت داده با عملیات توازن و غلبه بر خرابیهای RegionServer انجام می‌گیرد، اما تراکم برای ادغام فایل‌های دیسک HDFS در فصل 10 بحث خواهد شد که محل داده‌ها را برمی‌گرداند.

Hadoop و HBase از یک حالت شناخته شده به‌عنوان اتصال کوتاه پشتیبانی می‌کنند، که در آن RegionServer می‌تواند به‌طور مستقیم از دیسک محلی بخواند. البته، تنها زمانی این مسئله امکان‌پذیر است که داده‌ها بر روی DataNode ذخیره شده باشند و همچنین میزبان RegionServer باشد.

سه سطح از محلیت داده‌ها در شکل 8-9 نشان داده شده است. در پیکربندی اول، RegionServer و DataNode بر روی سرورهای مختلف قرار گرفته‌اند و تمام خواندن‌ها و نوشتن‌ها باید در سراسر شبکه انجام گیرد. در پیکربندی دوم، RegionServer و DataNode با هم همکاری دارند و عبور از DataNode را می‌نویسند، اما آنها توسط دیسک محلی ارضا شده‌اند. در سناریو سوم، خواندن از طریق اتصال کوتاه پیکربندی می‌شود و RegionServer می‌تواند به‌طور مستقیم از دیسک محلی بخواند.

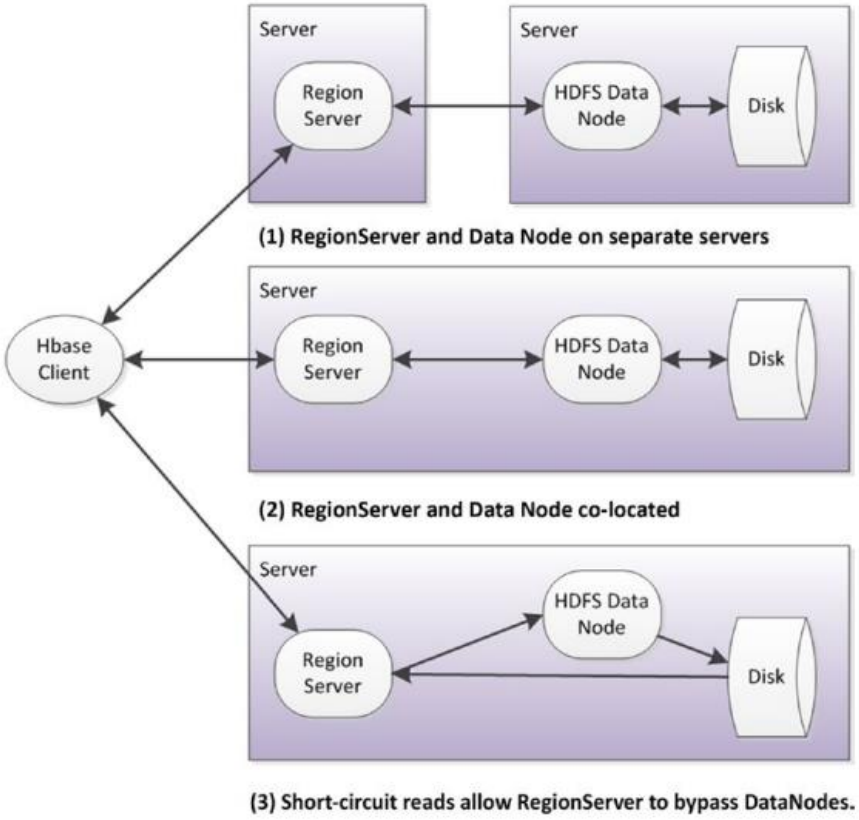


Figure 8-9. *Data locality in HBase*

مرتب‌سازی Rowkey

طرح پارتیشن‌بندی HBase مستلزم مناطقی شامل، محدوده به هم پیوسته از rowkeys است. این پارتیشن‌بندی مبتنی بر محدوده زمانی‌که شامل برخی از اشکال یکنواخت افزایش مقدار، مانند زمان و یا افزایش شمارنده باشد، rowkey تاثیر قابل توجهی بر عملکرد آن دارد. در این رویداد، تمام عملیات نوشتن به یک منطقه خاص اعمال می‌شود و از این رو به یک RegionServer منحصربه‌فرد نیاز دارد و می‌تواند یک گلوگاه براساس توان عملیاتی نوشتن ایجاد کند.

HBase هیچ مکانیسم داخلی برای کاهش این مسئله ارائه نمی‌دهد. این مسئله به طراح نرم‌افزار برای ساخت یک کلید نیاز دارد که به صورت تصادفی، یک نمونه هش از زمان است. در محدوده زمانی HBase در پایگاه‌داده OpenTSDB، برچسب زمان توسط یک متغیر که تعداد زیادی مقادیر دارد تعیین می‌شود. داده‌ها برای یک متریک

در یک RegionServer معینی قرار دارند، اما داده‌ها در یک زمان خاص در تمام RegionServers توزیع شده هستند.

انشعاباتRegionServer، حفظ تعادل و شکست

همان‌گونه که مناطق رشد می‌کنند، توسط RegionServer نیز تقسیم می‌شوند. مناطق جدید توسط RegionServer اصلی کنترل می‌شوند، اما در جریان عملیات‌های متعادل‌کننده بار، واجد شرایط جابجایی هستند. نتایج سیاست‌های پیش‌فرض منطقه با اولین تقسیم 128 مگابایتی به تدریج تقسیم می‌شوند، درحالی‌که دهمین منطقه در حدود GB10 خواهد بود. بااین‌حال، ممکن است مناطق به‌صورت دستی تقسیم شوند یا سیاست تقسیم با کدهای سفارشی نادیده گرفته شود.

یکی از مسئولیت‌های مهم گره HBase اصلی، تعادل مناطق در سراسر RegionServers است. گره اصلی تعادل مناطق را به‌صورت دوره‌ای در تمام RegionServers ارزیابی می‌کند و باید عدم تعادل تشخیص داده شود، تا به سرور دیگری مهاجرت کند. این مسئله توازن "نرم" نامیده می‌شود که منطقه داده در محل اصلی خود بر روی دیسک HDFS باقی خواهد ماند، اما مسئولیت مدیریت اطلاعات به یک RegionServer متفاوت سپرده می‌شود.

همانطور که قبلا اشاره شد، ایجاد توازن، منجر به از دست دادن محلیت داده‌ها می‌شود: زمانی‌که RegionServer مسئولیت یک منطقه جدید را بدست می‌آورد، احتمالا منطقه در یک گره از راه دور واقع می‌شود.

تکرار منطقه

در نسخه‌های قبلی HBase، شکست RegionServer نیاز به یک عدم موفقیت برای RegionServer جدید دارد. از آنجا که RegionServers داده‌ها را در یک منطقه ذخیره نمی‌کند (داده‌ها در HDFS)، شکست فاجعه‌بار نخواهد بود. گره اصلی شکست و تخصیص مناطق مربوط به RegionServers را با یک روش مشابه به عملیات حفظ تعادل شناسایی می‌کند. بااین‌حال، برخی از وقفه‌ها در خدمات اتفاق می‌افتد.

تکرار منطقه باعث می‌شود تا تکرار مناطق در چند RegionServers ذخیره شود. RegionServer باید شکست بخورد تا این کپی به سرویس درخواست مشتری استفاده شود.

RegionServer اصلی به‌عنوان کپی اصلی از منطقه عمل می‌کند. کپی فقط خواندنی منطقه در صورت امکان، در RegionServers دیگر توزیع می‌شود، که "به‌دنبال" RegionServer اولیه است. نوشتن در این کپی به‌صورت ناهمزمان در RegionServer اولیه اتفاق می‌افتد، بنابراین داده‌های موجود در کپی همیشه به روز نخواهد بود. در فصل بعدی چگونگی پیکربندی تکرار HBase، ثبات و دردسترس بودن را تحت تاثیر قرار می‌دهد.

HBase از یک مرکز تکثیر پشتیبانی می‌کند که می‌تواند برای پایگاه‌داده کپی HBase مورد استفاده قرار گیرد. این مسئله معمولا برای کپی یک پایگاه‌داده HBase در مرکز داده‌ی دیگری به کار می‌رود.

Cassandra

در فصل 3، پایگاه‌داده Daynamo آمازون و مفهوم هش سازگار معرفی شده است. تعدادی از سیستم‌های منبع باز، مدل Daynamo را اجرا کرده‌اند. در این فصل، پیاده‌سازی Cassandra را در نظر می‌گیریم.

شایعات بی‌اساس

در HBase و MongoDB، مفهوم گره اصلی برای نظارت، هماهنگی فعالیت‌های دیگر گره‌ها و ثبت وضعیت فعلی خوشه پایگاه‌داده بیان شده است. در cassandra و دیگر پایگاه‌داده‌های Dynamo، هیچ گره اصلی تخصصی وجود ندارد. هر گره برابر است و هر گره قادر به انجام هر یک از فعالیت‌های مورد نیاز برای عملیات خوشه است.

گره‌ها در cassandra مسئولیت‌های تخصصی کوتاه مدت دارند. به‌عنوان‌مثال، هنگامی که یک مشتری یک عملیات انجام می‌دهد، یک گره به‌عنوان هماهنگ‌کننده برای عملیات اختصاص داده می‌شود. زمانی‌که یک عضو جدید به خوشه اضافه می‌شود، یک گره به‌عنوان گره کاندید به‌دنبال گره جدید برای دریافت اطلاعات است. بااین‌حال، مسئولیت کوتاه‌مدت می‌تواند توسط هر گره در خوشه انجام شود.

یکی از مزایای استفاده از گره اصلی این است که می‌تواند از یک نسخه متعارف پیکربندی خوشه و حالت پشتیبانی کند. در صورت عدم وجود چنین گره‌ای، cassandra نیاز دارد که همه اعضای خوشه با وضعیت فعلی پیکربندی خوشه به روز نگه داشته شود. این مسئله با استفاده از پروتکل شایعات قابل دسترسی است. هر عضو از کلاستر اطلاعات مربوط به وضعیت آن را منتقل می‌کند و هر حالت گره‌های دیگر را به سه گره دیگر در خوشه ارسال می‌کند. به این ترتیب، وضعیت خوشه به‌طور مداوم در سراسر اعضای خوشه به‌روز می‌شود.

پروتکل شایعات: زمانی که مردم شایعات بی‌اساس می‌گویند، به‌طورکلی تمایل به دیگر شایعات بی‌اساس مردم نیز دارند! به همین دلیل، در cassandra، گره‌ها در مورد حالت خود و دیگر گره‌ها شایعات بی‌اساس بیان می‌کنند.

پیکربندی خوشه در فضای کلید سیستم بیان شده است، که دردسترس تمام کاربران است. فضای کلید در یک پایگاه‌داده با فضای کلید سیستم ارتباطی مشابه است و شامل جداولی در مورد آبرداده و پیکربندی خوشه است. این معماری هر نقطه‌ی شکست در خوشه را از بین می‌برد. اگرچه پایگاه‌داده‌های توزیع شده با گره‌های اصلی، استراتژی برای عدم موفقیت سریع دارند، شکست یک گره اصلی معمولا موجب کاهش موقت دسترسی، مانند لحظه‌ی رسیدن به حالت فقط خواندن می‌شود.

یکی از موضوعات اصلی در شایعات بی‌اساس در خوشه cassandra دردسترس بودن گره است. مکانیزم سنتی برای تشخیص خرابی گره ارسال علایم حیاتی بین گره‌ها است. بااین حال، در سیستم توزیع شده، علایم حیاتی ممکن است به دلیل مسائل مربوط به شبکه به جای شکست گره واقعی، از دست بروند. به‌همین دلیل، تشخیص شکست cassandra احتمال بیشتری دارد: اگر دوست دارید، گره‌ها در خوشه به‌طور فزاینده‌ای در مورد سایر گره‌ها "نگران" خواهند بود.

هش سازگار

cassandra و پایگاه‌داده‌های مبتنی بر Daynamo داده‌ها را در سراسر خوشه با استفاده از هش کردن سازگار توزیع می‌کند. rowkey (شبیه به کلید اصلی در یک RDBMS) درهم است. هر گره در طیف وسیعی از مقادیر هش قرار

دارد و گرهی که محدوده‌ی خاصی برای یک مقدار کلیدی دارد در مورد محل اولیه داده‌ها مسئول است. به‌طور پیش‌فرض طرح پارتیشن‌بندی cassandra، مقادیر هش بین -263 تا 263-1 دارد. از این رو، اگر چهار گره در خوشه وجود داشته باشد و بخواهیم تعداد مساوی از رشته‌های هش را به هر گره اختصاص دهیم، محدوده هش برای هر گره تقریبا به شرح زیر است:

گره	هش پایین	هش بالا
گره A	-2^{63}	$-2^{63}/2$
گره B	$-2^{63}/2$	0
گره C	0	$2^{63}/2$
گره D	$2^{63}/2$	2^{63}

معمولا خوشه به‌عنوان یک حلقه مصورسازی می‌شود: دور حلقه نشان‌دهنده‌ی تمام مقادیر هش ممکن است و محل گره در حلقه نشان‌دهنده‌ی منطقه‌ی تحت مسئولیت آن است. شکل 8-10 هش ساده سازگار را نشان می‌دهد: مقدار یک rowkey درهم است که موضع خود را در "حلقه" تعیین می‌کند. گره‌ها در خوشه مسئولیت محدوده مقادیر در درون حلقه و در نتیجه مالکیت مقادیر rowkey خاص را برعهده دارند.

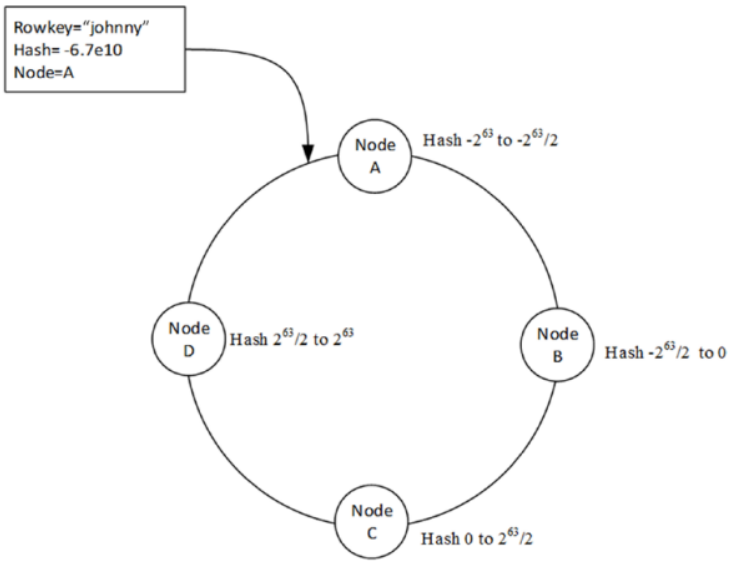


Figure 8-10. *Consistent hashing*

چهار گره خوشه در شکل 8-10 به‌خوبی متعادل شده‌اند، زیرا هر گره مسئول هش محدوده مقادیر مشابه است. اما عدم تعادل خوشه با اضافه کردن گره در معرض خطر قرار می‌گیرد. اگر تعداد گره‌ها در خوشه را دو برابر کنیم، می‌توانیم

گره‌های جدید در نقاط حلقه بین گره‌های موجود اختصاص دهیم تا خوشه متعادل باقی بماند. بااین‌حال، دو برابر شدن خوشه معمولا غیرعملی است: اما رشد تدریجی خوشه مقرون به صرفه است.

نسخه‌های اولیه از cassandra دو انتخاب به هنگام اضافه کردن یک گره جدید دارند. بنابراین می‌توانیم تمام مقادیر در محدوده هش را مجددا نگاشت کنیم یا گره جدید در محدوده موجود نگاشت کنیم. در مورد اول یک خوشه متعادل، پس از فرایند ایجاد توازن به دست می‌آوریم. در مورد دوم خوشه نامتعادل می‌شود؛ زیرا هر گره مسئول منطقه خود در حلقه است و اضافه کردن یک گره جدید بدون تغییر محدوده‌ی گره‌های دیگر، اساسا یک منطقه را به دو نیمه تجزیه می‌کند. شکل 8-11 چگونگی اضافه کردن یک گره به خوشه را نشان می‌دهد که می‌تواند توزیع محدوده کلید هش را با عدم تعادل روبه‌رو کند.

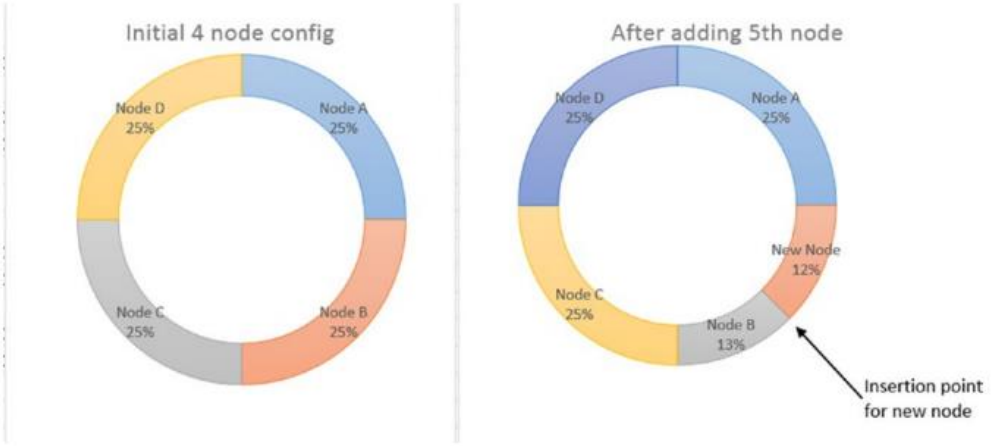


Figure 8-11. *Adding a node to a Cassandra cluster (without virtual nodes)*

گره مجازی، که در Cassandra پیاده‌سازی شده است، Riak، و بسیاری از سیستم‌های دیگر براساس Dynamo، راه‌حلی برای این موضوع ارائه می‌دهند. هنگام استفاده از گره‌های مجازی، محدوده هش برای تعداد زیادی گره-256 گره مجازی در هر گره فیزیکی، محاسبه می‌شود و به‌طور معمول این گره مجازی به گره‌های فیزیکی اختصاص می‌یابد. وقتی که یک گره جدید اضافه می‌شود، گره مجازی خاص را می‌توان به گره جدید مجددا تخصیص داد و در نتیجه یک پیکربندی متعادل با حداقل سربار به دست آورد. شکل 8-12 ارتباط بین گره‌های مجازی و گره‌های فیزیکی را نشان می‌دهد.

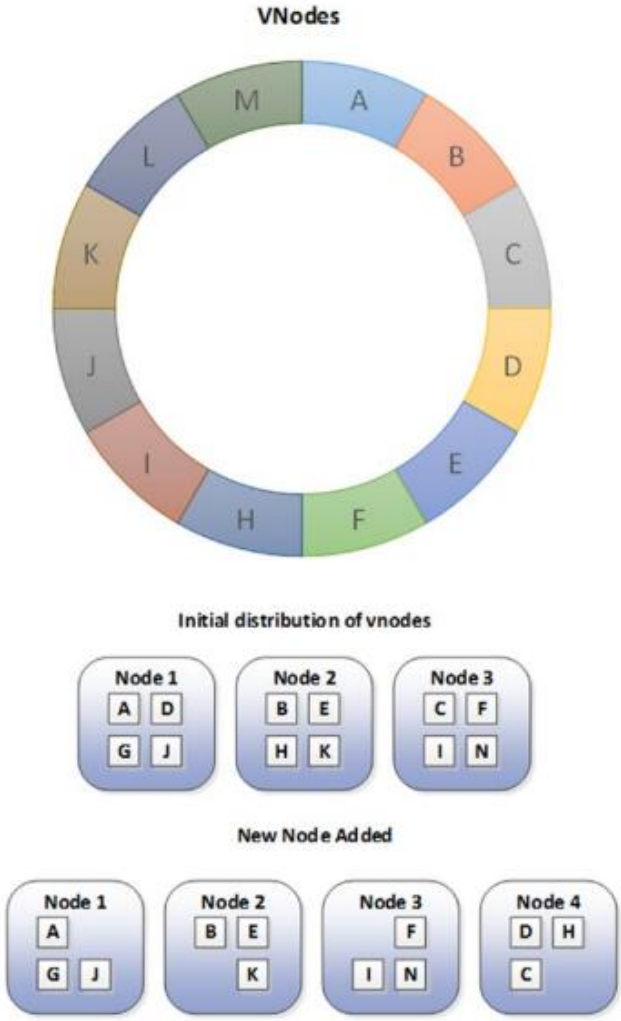


Figure 8-12. *Using virtual nodes to partition data among physical nodes*

گره مجازی برخی مزایای دیگر نیز دارد. به‌عنوان مثال، ایجاد تعادل در خوشه ساخته شده نسبت به سیستم‌های ناهمگن آسان است، شما می‌توانید گره‌های مجازی بیشتری به دستگاه‌های قوی و جدید و گره‌های مجازی کمتری به ماشین‌های قدیمی‌تر تخصیص دهید. همچنین، اگر یک گره بمیرد می‌توان آن را دوباره از تعداد زیادی ماشین فیزیکی ساخت، در نتیجه به‌اشتراک‌گذاری سربار در سراسر خوشه بهبود ایجاد می‌کند.

پارتیشن‌بندی جهت محافظت از سفارش‌ها

پارتیشن‌بندی Cassandra چگونگی توزیع کلید در سراسر گره‌ها را تعیین می‌کند. پارتیشن‌بندی پیش‌فرض از هش سازگار استفاده می‌کند، همانطور که در بخش قبلی توضیح داده شد. Cassandra نیز از پارتیشن‌بندی برای توزیع

داده‌ها در سراسر گره‌های خوشه به عنوان محدوده واقعی (به‌عنوان مثال، هش نشده) rowkeys پشتیبانی می‌کند. برای مثال، اگر مقدار کلید یک برچسب زمان باشد و پارتیشن‌بندی پیاده‌سازی شده باشد، تمام سطرهای جدید تمایل به ایجاد یک گره در خوشه دارند.

در نسخه‌های اولیه Cassandra، پارتیشن‌بندی درخواست سفارش ممکن است برای بهینه‌سازی محدوده نمایش داده شده بسیار خوب عمل کند؛ بااین حال، پس از معرفی شاخص ثانویه، پارتیشن‌بندی درخواست سفارش در درجه اول برای حفظ مستندات سازگاری و سپس استفاده از آن در برنامه‌های جدید است.

تکرارها

تاکنون، مشاهده کرده‌ایم که چگونه Cassandra، کپی اولیه از یک داده را به گره اختصاص می‌دهد. الگوریتم هش سازگار نیز که محل کپی داده ذخیره شده را تعیین می‌کند.

گره، مسئول محدوده‌ی هش برای یک مقدار خاص rowkey است که هماهنگ‌کننده گره نامیده می‌شود. هماهنگ‌کننده مسئول تضمین تعداد مورد نیاز نسخه المثنی از داده‌ها است. تعداد گره‌ها باید به‌عنوان عامل تکرار نوشته شود و "N" در نماد NWR، برای اولین بار در فصل 3 بیان شده است.

به‌طور پیش‌فرض، هماهنگ‌کننده نسخه‌هایی از داده‌ها را به N-1 گره بعدی در حلقه ارسال می‌کند. بنابراین اگر عامل تکرار 3 باشد، هماهنگ‌کننده کپی داده را به دو گره بعدی در حلقه ارسال خواهد کرد. در این سناریو، هر گره، به شکل تکرار داده‌ها از دو گره قبلی در حلقه و تکرار به دو گره بعدی در حلقه خواهد بود. این روش ساده به‌عنوان استراتژی تکرار ساده استفاده می‌شود.

Cassandra اجازه می‌دهد تا طرح‌های پیچیده و بسیار دردسترس را پیکربندی کنید. استراتژی تکرار توپولوژی شبکه تضمین می‌کند که تکرارها در گره‌های موجود در سرور دیگر نوشته شده‌اند، یا به‌صورت اختیاری به گره‌ها در مرکز در دسترس دیگری انتقال می‌یابند.

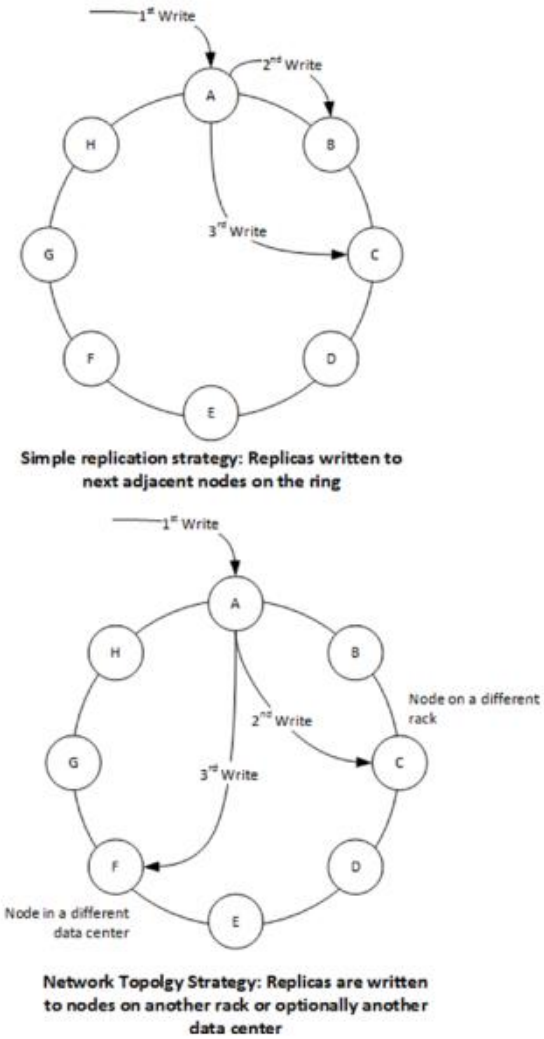


Figure 8-13. *Replication strategies in Cassandra*

Snitches

Cassandra از snitches برای کمک به بهینه‌سازی عملیات خواندن و نوشتن استفاده می‌کند. انواع snitchesها ممکن است پیکربندی شده باشند.

• simpleSnitch تنها یک لیست از گره‌ها در حلقه را برمی‌گرداند. که برای استراتژی تکرار ساده که در بخش قبلی بحث شده کافی است.

• RackInferringSnitch از آدرس‌های IP میزبان برای پی بردن به مرکز رَک و محل مرکز داده‌ها استفاده می‌کند. این مورد از استراتژی تکرار شبکه آگاه پشتیبانی می‌کند.

• PropertyFileSnitch از داده‌ی فایل پیکربندی به جای آدرس IP برای تعیین مرکز داده و توپولوژی رَک استفاده می‌کند.

علاوه براین، تمام snitchesها بر زمان تاخیر خواندن در یک درخواست نظارت می‌کنند و از این مورد برای ساخت یک مدل آماری که بتواند درخواست را به گره بهتر هدایت کند استفاده می‌کنند.

snitchesهای تخصصی برای درک توپولوژی شبکه در داخل سیستم‌عامل‌های مختلف ابر، مانند EC2 آمازون وجود دارند.

خلاصه

در این فصل الگوهای پایگاه‌داده‌ی توزیع شده برای پایگاه‌داده‌های رابطه‌ای سنتی و برای چند سیستم غیررابطه‌ای را بررسی کرده‌ایم. معماری پایگاه‌داده‌ی رابطه‌ای در عصر توسعه‌ی سرورهای پایگاه‌داده‌ی یکپارچه و بسیاری از پایگاه‌داده‌های رابطه‌ای هنوز هم به عنوان یک نمونه تک اجرا می‌شود. بااین‌حال، خوشه‌بندی sharding امری عادی در انبار موازی داده‌ها است، و اوراکل دارای یک موفقیت تجاری در خوشه RDBMS است.

جزئیات سه سیستم پایگاه‌داده‌ی توزیع شده غیررابطه‌ای بررسی شدند. MongoDB از ترکیب sharding و تکثیر برای پردازش توزیع شده استفاده می‌کند. HBase از سیستم فایل توزیع شده در سیستم فایل توزیع شده Hadoop همراه با یک استراتژی وسیع پارتیشن‌بندی برای رسیدن به یک راه‌حل بسیار مقیاس‌پذیر استفاده می‌کند. Cassandra از طرح‌های هش سازگار در سیستم Dynamo آمازون برای ایجاد یک راه‌حل خوشه‌بندی متقارن در سرورهای مورد نیاز استفاده می‌کند.

در یک پایگاه‌داده‌ی توزیع شده، نسخه‌های متعددی از داده‌ها به‌طور معمول در سراسر خوشه وجود دارند. که در فصل بعدی، چگونگی مدیریت انسجام داده‌ها در این پایگاه‌داده را در چنین سیستم توزیع شده‌ای خواهیم دید.