کلان داده

اصول و بهترین اقدامات

سیستم‌های داده‌ای بلادرنگ مقیاس‌پذیر

مؤلفین:

مارز، ناتان

با جیمز وارن

مترجمین:

دکتر جواد وحیدی– دانشگاه علم و صنعت ایران

دکتر رمضان عباس‌نژادورزی

مهندس زهرا علیجان‌نژادبایی

فن‌آوری نوین

سرشناسه	:	مارز، ناتان Marz, Nathan
عنوان و نام پدیدآور	:	کلان داده: اصول و بهترین اقدامات سیستم‌های داده‌ای بلادرنگ مقیاس‌پذیر / مولفین ناتان مارز، جیمز وارن؛ مترجمین جواد وحیدی، رمضان عباس‌نژادورزی، زهرا علیجان‌نژادبایی.
مشخصات نشر	:	بابل: فناوری نوین، ۱۳۹۹.
مشخصات ظاهری	:	۳۴۹ ص.: مصور(بخشی رنگی)، جدول(بخشی رنگی)، نمودار(بخشی رنگی).
شابک	:	978-622-7393-20-0
وضعیت فهرست نویسی	:	فیپا
یادداشت	:	عنوان اصلی: Big data : principles and best practices of scalable real-time data systems.
یادداشت	:	ترجمه دیگری از کتاب حاضر با عنوان "داده‌های کلان : اصول و بهترین تجارب سیستم‌های داده‌ای بلادرنگ مقیاس‌پذیر" توسط ارسطو فیپا دریافت کرده است.
عنوان دیگر	:	اصول و بهترین اقدامات سیستم‌های داده‌ای بلادرنگ مقیاس‌پذیر.
عنوان دیگر	:	داده‌های کلان : اصول و بهترین تجارب سیستم‌های داده‌ای بلادرنگ مقیاس‌پذیر.
موضوع	:	داده‌های کلان
موضوع	:	Big data
موضوع	:	پایگاه‌های اطلاعاتی -- مدیریت
موضوع	:	Database management
موضوع	:	پایگاه‌های اطلاعاتی -- طراحی
موضوع	:	Database design
موضوع	:	داده‌کاوی
موضوع	:	Data mining
شناسه افزوده	:	وارن، جیمز
شناسه افزوده	:	Warren, James
شناسه افزوده	:	وحیدی، جواد، ۱۳۴۸ -، مترجم
شناسه افزوده	:	Vahidi, Javad
شناسه افزوده	:	عباس‌نژاد ورزی، رمضان، ۱۳۴۸ -، مترجم
شناسه افزوده	:	علیجان‌نژاد بایی، زهرا، ۱۳۶۶-، مترجم
رده بندی کنگره	:	QA۷۶/۹
رده بندی دیویی	:	۶۵۸/۴۰۳۸
شماره کتابشناسی ملی	:	۷۳۹۱۵۹۶
وضعیت رکورد	:	فیپا

فن‌آوری نوین بابل، کد پستی ۴۷۱۶۷-۷۳۴۴۸ تلفن: ۳۲۲۵۶۶۸۷-۰۱۱

کلان داده اصول و بهترین اقدامات سیستم‌های داده‌ای بلادرنگ مقیاس‌پذیر

ترجمه: جواد وحیدی، رمضان عباس نژادورزی، زهرا علیجان نژادبایی.

نوبت چاپ: چاپ اول

سال چاپ: تابستان ۹۹

شمارگان: ۲۰۰

قیمت: ۸۷۰۰۰ تومان

نام چاپخانه و صحافی: دفتر فنی سورنا

حروف‌چینی و تایپ: فناوری نوین

شابک: 978-622-7393-20-0

نشانی ناشر: بابل، چهارراه نواب، کاظم‌بیگی، جنب مسجد منصور کاظم‌بیگی، طبقه اول

طراح جلد: کانون آگهی و تبلیغات آبان (احمد فرجی)

تهران، خ اردیبهشت، نبش وحید نظری، پلاک ۱۴۲ تلفکس: ۶۶۴۰۰۲۲۰-۶۶۴۰۰۱۴۴

فهرست مطالب

پیش‌گفتار

وقتی اولین بار وارد دنیای کلان داده‌ها شدم،این مانند غرب وحشی در توسعه نرم‌افزار احساس می‌شود. بسیاری از آن‌ها در حال رها کردن پایگاه داده‌های رابطه‌ای و راحتی آن برای پایگاه داده‌های NoSQL با مدل‌های داده بسیار محدود بودند که برای مقیاس هزاران دستگاه طراحی‌شده‌اند. تعداد دیتابیس‌های NoSQL، بین بسیاری از آن‌ها که تنها تفاوت‌های جزئی وجود دارد، بسیار زیاد است. پروژه جدیدی به نام Hadoop شروع به ایجاد امواج کرد و نوید توانایی انجام تجزیه‌وتحلیل‌های عمیق بر روی مقادیر عظیمی از داده‌ها را فراهم کرد. درک نحوه استفاده از این ابزارهای جدید باعث حیرت شد.

در آن زمان، من در تلاش بودم تا مشکلات مقیاس‌پذیری را که با آن روبرو هستیم در شرکتی که در آن کار می‌کردم، برطرف کنم. معماری به‌مراتب پیچیده بود - شبکه‌ای از پایگاه داده‌های رابطه‌ای خرد، صف‌ها، کارگران، استادان و بردگان. انحراف راه خود در پایگاه‌های داده پیداکرده بود،و کد ویژه‌ای در برنامه، برای رسیدگی به انحراف وجود داشته است. سخت‌کوشان همیشه عقب بودند. تصمیم گرفتم تا فناوری‌های جایگزین کلان داده را بررسی کنم تا ببینم آیا طراحی بهتری برای معماری داده‌های وجود دارد یا خیر.

یک تجربه از حرفه مهندسی نرم‌افزار اولیه، دیدگاهم را در مورد نحوه معماری سیستم‌ها عمیقاً شکل داده است. همکارم چند هفته برای جمع‌آوری داده‌ها از اینترنت بر روی یک پرونده سیستم مشترک صرف کرد. منتظر بود تا داده‌های کافی را جمع کند تا بتواند آنالیز را بر روی آن‌ها انجام دهد. یک روز درحالی‌که برخی از کارهای روزمره را انجام می‌دادم، به‌طور تصادفی تمام داده‌های همکارم را حذف کردم و وی را هفته‌ها بر پروژه‌اش قرار دادم.

می‌دانستم که اشتباه بزرگی کرده‌ام، اما به‌عنوان یک مهندس نرم‌افزار جدید نمی‌دانستم عواقب آنچه خواهد بود. آیا قرار بود اخراج شوم به خاطر اینکه این‌قدر بی‌دقت هستم؟ یک ایمیل برای تیم ارسال کردم که با عذرخواهی عمیقاً معذرت‌خواهی کردم - و با کمال تعجب از همه، آن‌ها بسیار دلسوز بودند. هرگز فراموش نخواهم کرد وقتی یک همکار به میزم آمد، به پشتم تکیه کرد و گفت: "تبریک می‌گویم. شما اکنون یک مهندس نرم‌افزار حرفه‌ای هستید. "

در اظهار شوخی او یک حقیقت ناگفته در توسعه نرم‌افزار وجود دارد: نمی‌دانیم چگونه یک نرم‌افزار کامل بسازیم. اشکالات می‌توانند و به تولید تبدیل می‌شوند. اگر برنامه می‌تواند در بانک اطلاعاتی بنویسد، یک اشکال نیز می‌تواند در بانک اطلاعاتی بنویسد. وقتی قصد داشتم معماری داده را دوباره طراحی کنم، این تجربه عمیقاً روی من تأثیر گذاشت. می‌دانستم که معماری جدید نه‌تنها مقیاس‌پذیر، قابل‌تحمل در برابر خرابی دستگاه است و برای استدلال در مورد اشتباهات انسانی نیز آسان است.

تجربیاتم در زمینه معماری مجدد در مورد آن سیستم، مسیری را به سمتم سوق داد و باعث شد هر آنچه را که فکر می‌کردم در مورد بانک‌های اطلاعاتی و مدیریت داده‌ها صحیح است، زیر سؤال ببرم. به یک معماری مبتنی بر داده‌های غیرقابل تغییر و محاسبات دسته‌ای دست‌یافتم و از اینکه سیستم جدید چقدر از یک محاسبه افزایشی ساده‌تر است، متحیر شدم. همه‌چیز آسان‌تر شد، ازجمله عملیات، تکامل سیستم برای پشتیبانی از

ویژگی‌ها جدید، بهبودی از اشتباهات انسانی و انجام بهینه‌سازی عملکرد. این رویکرد چنان عمومی بود که به نظر می‌رسید می‌تواند برای هر سیستم داده‌ای مورداستفاده قرار گیرد.

چیزی من را گیج کرد وقتی به بقیه صنعت نگاه کردم، دیدم که به‌سختی کسی از تکنیک‌های مشابه استفاده می‌کند. در عوض، مقادیر پیچیده‌ای از پیچیدگی را در استفاده از معماری‌های مبتنی بر خوشه‌های عظیم از پایگاه داده‌های تدریجی به‌روز شده، شامل شد. بنابراین بسیاری از پیچیدگی‌ها موجود در آن معماری‌ها با رویکردی که توسعه داده بودم کاملاً از آن جلوگیری شده یا کاملاً نرم شدند.

در طی چند سال آینده، رویکردی را گسترش و آن را به آنچه در معماری لامبدا لقب دادم، رسمیت بخشیدم. هنگام کار روی یک استارتاپ به نام BackType، تیم ۵ نفره ما یک محصول تحلیلی در رسانه‌های اجتماعی ایجاد کرده است که مجموعه متنوعی از آنالیزها را در زمان بلادرنگ روی بیش از ۱۰۰ ترابایت داده ارائه می‌دهد. تیم کوچک ما همچنین با استقرار، بهره‌برداری و نظارت بر سیستم بر روی خوشه صدها دستگاه را مدیریت کردند. وقتی محصول خود را به مردم نشان دادیم، متحیر شدند که ما تیمی فقط ۵ نفره هستیم. آن‌ها اغلب سؤال می‌کردند "چگونه می‌توانند چنین تعداد کمی از مردم، چنین کنند؟" پاسخ من ساده بود: "این کاری نیست که ما انجام می‌دهیم بلکه کاری را که انجام نمی‌دهیم." با استفاده از معماری لامبدا، از پیچیدگی‌هایی که معماری‌های سنتی را به هم می‌رساند، دوری کردیم. با اجتناب از این پیچیدگی‌ها، ما به طرز چشمگیری بیشتر تولید کردیم.

جنبش کلان داده‌ها فقط پیچیدگی‌هایی را که در معماری داده‌ها برای ده‌ها سال وجود داشته است، بزرگ کرده است. هر معماری که اساساً بر پایه بانک‌های اطلاعاتی بزرگی است و به‌صورت تدریجی به‌روز می‌شود، از این پیچیدگی‌ها رنج می‌برد و باعث ایجاد اشکالات، عملیات سنگین و ایجاد محصول می‌شود. اگرچه پایگاه داده‌های SQL و NoSQL اغلب به‌عنوان مخالف یا به‌عنوان دوتایی یکدیگر مصور می‌شوند، در یک سطح اساسی واقعاً یکسان هستند. آن‌ها با پیچیدگی‌ها اجتناب‌ناپذیر آن، همین معماری را تشویق می‌کنند. پیچیدگی یک جانور شرور است، صرف‌نظر از اینکه آن را تصدیق کنید یا نه، گاز می‌گیرد.

این کتاب نتیجه تمایل به گسترش دانش معماری لامبدا و چگونگی جلوگیری از پیچیدگی‌ها معماری سنتی است. این کتابی است که آرزو می‌کردم وقتی‌که با کلان داده کار کردم، شروع کنم. امیدوارم که با این کتاب به‌عنوان یک سفر رفتار کنید - سفری برای به چالش کشیدن آنچه که فکر می‌کردید در مورد سیستم‌های داده می‌دانید، و پی می‌برید که کار با کلان داده چقدر می‌تواند ظریف، ساده و سرگرم‌کننده باشد.

NATHAN MARZ

سپاس‌گزاری

این کتاب بدون کمک و پشتیبانی افراد زیادی در سراسر جهان امکان‌پذیر نبود. من باید از والدینم شروع کنم، که از سنین جوانی عشق به یادگیری و کاوش در دنیای اطرافم را به من القا کردند. آن‌ها همیشه مرا در تمام مراحل شغلی تشویق می‌کردند.

به همین ترتیب، برادرم یورا از جوانی علایق فکری من را تشویق کرد. من هنوز به یاد دارم وقتی در مدرسه ابتدایی بودم به من جبر آموخت. او اولین کسی بود که من را برای اولین بار به برنامه‌نویسی معرفی آشنا کرد - او به من ویژوال‌بیسیک را آموخت که در دبیرستان می‌خواند. آن درس‌ها اشتیاق به برنامه‌نویسی را ایجاد کردند که منجر به حرفه من شد.

از مایکل مونتانو و کریستوفر گلدا، بنیان‌گذاران BackType بسیار سپاسگزارم. از لحظه‌ای که آن‌ها مرا به‌عنوان اولین کارمند خود به خدمت گرفتند، به من آزادی فوق‌العاده‌ای برای تصمیم‌گیری داده شد. این آزادی برای من کاملاً اساسی بود که بتوانم از معماری لامبدا به کاوش و بهره‌برداری بپردازم. آن‌ها هرگز ارزش منبع آزاد و زیر سؤال نمی‌برد و به من اجازه داد تا تکنولوژی خود را به‌طور آزادانه منبع باز کنم. درگیری عمیق با منبع آزاد یکی از امتیازات مهم زندگی من بوده است.

بسیاری از اساتید از زمان من به‌عنوان دانشجو شایسته در استنفورد تشکر ویژه کردند. تیم روفگاردن بهترین معلمی است که من تاکنون داشته‌ام – او توانایی من را برای تجزیه‌وتحلیل دقیق، ساختارشکنی و حل مشکلات دشوار بهبود بخشید. برگزاری هرچه بیشتر کلاس با او یکی از بهترین تصمیمات زندگی من بود. همچنین از مونیکا لام تشکر می‌کنم که از ظرافت ورود به سیستم اطلاعاتی درونم قدردانی کرده است. مهروموم‌ها بعد با «ورود به سیستم» و MapReduce پیوستم تا اولین پروژه مهم منبع باز، Cascalog را تولید کنم.

کریس ونزل اولین کسی بود که به من نشان داد که پردازش داده‌ها در مقیاس می‌تواند بسیار زیبا و کارآمد باشد. کتابخانه‌ی Cascading او شیوه نگاه به پردازش کلان داده را تغییر داد.

هیچ‌یک از کارهای من بدون پیشگامان حوزه کلان داده امکان‌پذیر نبود. تشکر ویژه از جفری دین و سانجی گیماوات برای مقاله اصلی MapReduce، جوزپه دی کاناندیا، دنیز هاستورون، مادان جامپانی، گوناواردان کاکولپاتی، آویناش لاکشمن، الکس پیلچین، سوامیناتان سیواسوبرمانیان، پیتر ووششال و ورنر وگلز برای مقاله اصلی دینامو و مایکل کافارلا و داگ برش برای بنیان‌گذاری پروژه Apache Hadoop.

ریچ هیکی‌یکی از بزرگ‌ترین الهامات من در طول کار برنامه‌نویسی بوده است.Clojure بهترین زبانی است که تاکنون استفاده کرده‌ام و با آموختن آن به یک برنامه‌نویس بهتر تبدیل‌شده‌ام. من از عملی بودن آن و تمرکز روی سادگی قدردانی می‌کنم. فلسفه غنی در مورد وضعیت و پیچیدگی برنامه‌نویسی بر من تأثیر گذاشته است.

وقتی نوشتن این کتاب را شروع کردم، تقریباً نویسنده‌ای نبودم که الآن هستم. رنه گریگویر، یکی از سردبیران توسعه‌ای من در منینگ، سزاوار تشکر ویژه برای کمک به من در مقام نویسندگی است. او اهمیت استفاده از مثال‌هایی را برای هدایت به مفاهیم کلی به من گوشزد کرد، و او بسیاری از چراغ‌ها را برای من تنظیم کرد که چگونه می‌توان نوشتارهای فنی را مؤثر ساخت. مهارت‌هایی که او به من آموخت، نه‌تنها در نوشتن کتاب‌های فنی بلکه در وبلاگ نویسی، صحبت کردن، و به‌طورکلی ارتباط، برقرار است. برای به دست آوردن یک مهارت مهم زندگی، برای همیشه سپاسگزارم.

این کتاب بدون تلاش همکار من جیمز وارن تقریباً با این کیفیت نمی‌توانست باشد. او کار جذابی انجام می‌داد، مفاهیم نظری را جذب می‌کرد و راه‌های بهتری برای ارائه مطالب پیدا می‌کرد. بخش عمده‌ای از وضوح کتاب از مهارت‌های ارتباطی عالی او ناشی می‌شود.

ناشر من، منینگ، خوشحال شدم که با شما همکاری کردم. آن‌ها با من صبور بودند و می‌فهمیدند که پیدا کردن راه درست برای نوشتن در چنین موضوعی بزرگی، به زمان نیاز دارد. آن‌ها در طول کل فرآیند پشتیبان و کمک‌کننده بودند، و همیشه منابع لازم برای موفقیت را به من می‌دادند. با تشکر از مرجان بیس و مایکل استیفن برای همه پشتیبانی و از کلیه کارمندان دیگر برای کمک و راهنمایی‌های آن‌ها در این راه.

سعی می‌کنم تا حد امکان در مورد نوشتن از مطالعه سایر نویسندگان یاد بگیرم. برادفورد کراس، کلیتون کریستنسن، پل گراهام، کارل ساگان و درک سیورز که به‌ویژه، تأثیرگذار بوده‌اند.

سرانجام، نمی‌توانم به صدها نفری که در حال نوشتن، به بررسی، اظهارنظر و بازخورد در مورد کتاب می‌پرداختند، تشکر کنم. این بازخورد باعث شد تا چندین بار تجدیدنظر، بازنویسی و بازسازی مجدد انجام شود تا زمانی که روش‌های ارائه مطالب را به‌طور مؤثر پیدا کنیم. با تشکر ویژه از آرون کلکورد، آرون کرو، الکس هولمز، آرون جیکوب، آصف جان، آئون سینا، بیل گراهام، چارلز برفی، دیوید بکویث، درریک برنز، داگلاس دانکن، هوگو گارزا، جیسون کورکوکس، جاناتان استرازی، کارل کانتز، کوین مارتین، لئو Polovets، مارک فیشر، Massimo ایلاریو، مایکل Fogus، مایکل G. نول، پاتریک دنیس، پدرو Ferrera Bertran، فیلیپ Janert، Rodrigo Abreu، رودی Bonefas، سام ریچی، سیوا Kalagarla، سورن Macbeth، تیموتی Chklovski، ولید فرید، و ژنهوا گوو.

NATHAN MARZ

وقتی هرکسی را که به‌نوعی در این کتاب مشارکت داشته‌اند، متعجب می‌کنم. متأسفانه، نمی‌توانم یک لیست جامع ارائه دهم، اما این قدردانی من را کاهش نمی‌دهد. بااین‌وجود، افرادی هستند که می‌خواهم صریحاً تشکر خود را ابراز کنم:

- همسر من، ون-ینگ فنگ — به خاطر عشق، تشویق و حمایت شما، نه‌تنها برای این کتاب بلکه برای هر کاری که با هم انجام می‌دهیم.
- والدینم، جیمز و گرتا وارن — به خاطر ایمان بی‌پایان شما به من و فداکاری‌هایی که کردید و برایم هر فرصتی را فراهم کردم.
- خواهرم، جولیا وارن-اولانچ — برای اینکه نمونه‌ای درخشان را در نظر گرفتید تا بتوانم قدم‌های شما را دنبال کنم.
- اساتید و مربیانم، الن توبی و سو گلر - به خاطر تمایل شما برای پاسخ به هر سؤال من و نشان دادن شادی در به اشتراک‌گذاری دانش، نه‌فقط کسب آن.
- چاک لام — برای گفتن "سلام، آیا شما چیزی را به نام" Hadoop "شنیدید؟ مهروموم‌ها پیش برای من.
- دوستان و همکاران من در !RockYou، Storm۸ و Bina — به خاطر تجربیاتی که با هم به اشتراک گذاشتیم و فرصتی برای استفاده از تئوری‌ها.
- مرجان بیس، مایکل استیفن، جنیفر استوت، رنه گریگوئر، و کل کارمندان تحریریه و نشر منینگ - برای راهنمایی و صبر شما در دیدن این کتاب به اتمام رسیده‌اند.
- داوران و خوانندگان اولیه این کتاب — برای نظرات و انتقادات شما که ما را به روشن کردن سخنان ما سوق داده است. نتیجه نهایی برای آن بسیار بهتر است.
- سرانجام، می‌خواهم بیشترین قدردانی خود را به ناتان برای دعوت از من برای حضور در این سفر تقدیم کنم. قبل از پیوستن به این سرمایه‌گذاری،تحسین‌کننده کار شما بودم، و همکاری با شما فقط باعث احترام من به ایده‌ها و فلسفه شما شده است. این یک افتخار و یک امتیاز بوده است.

جیمز وارن

درباره این کتاب

سرویس‌هایی مانند شبکه‌های اجتماعی، تجزیه‌وتحلیل وب و تجارت الکترونیکی هوشمند اغلب باید داده‌ها را در مقیاس خیلی بزرگ برای یک بانک اطلاعاتی سنتی مدیریت کنند. پیچیدگی با مقیاس و تقاضا افزایش می‌یابد، استفاده از کلان داده به‌سادگی دو برابر کردن RDBMS یا استفاده از فناوری جدید و مرسوم، نیست. خوشبختانه، مقیاس‌پذیری و سادگی دوطرفه نیستند – فقط باید رویکرد متفاوتی اتخاذ کنید. سیستم‌های کلان داده از بسیاری از ماشین‌آلات استفاده می‌کنند که به‌موازات ذخیره و پردازش داده‌ها کار می‌کنند، که چالش‌های اساسی ناآشنا را برای اکثر توسعه‌دهندگان معرفی می‌کند.

کلان داده می‌آموزد که این سیستم‌ها را با استفاده از معماری که از سخت‌افزارهای گروه‌بندی‌شده به همراه ابزارهای جدیدی که به‌طور خاص برای گرفتن و تحلیل داده‌های وب طراحی‌شده‌اند، بسازید. این یک رویکرد مقیاس‌پذیر و آسان برای درک سیستم‌های کلان داده را که توسط یک تیم کوچک ساخته و اجرا می‌شود، توصیف می‌کند. این کتاب به دنبال یک مثال واقع‌گرایانه، خوانندگان را از طریق تئوری سیستم‌های کلان داده و نحوه اجرای آن‌ها در عمل راهنمایی می‌کند.

کلان داده‌ها نیازی به‌مواجهه قبلی با تجزیه‌وتحلیل داده‌های در مقیاس بزرگ یا ابزارهای NoSQL ندارند. هدف از کتاب، آموزش این است که چگونه در مورد سیستم‌های داده فکر کنید و چگونه مشکلات دشوار را به راه‌حل‌های ساده حل کنید. از اصول اولیه شروع می‌کنیم و از آن خصوصیات لازم برای هر مؤلفه یک معماری استنباط می‌شود.

نقشه راه

مروری بر ۱۸ فصل این کتاب به شرح زیر است.

فصل ۱ اصول سیستم‌های داده را معرفی می‌کند و مروری بر معماری لامبدا می‌کند: یک رویکرد کلی برای ساختن هر سیستم داده. فصل‌های ۲ تا ۱۷ در تمام بخش‌های معماری لامبدا پرداخته است، که فصول مختلف بین تئوری و تصویرگری متناوب هستند. اجازه ندهید که اسامی شما را فریب دهند، اما همه فصل‌ها بسیار مثال‌زدنی هستند.

فصول ۲ تا ۹ بر لایه دسته‌ای معماری لامبدا تمرکز دارد. در اینجا در مورد مدل‌سازی مجموعه داده‌های اصلی، با استفاده از پردازش دسته‌ای برای ایجاد نماهای دلخواه از داده‌ها، و مبادلات بین پردازش‌های افزایشی و دسته‌ای، اطلاعات کسب خواهید کرد.

فصول ۱۰ و ۱۱ بر لایه سرویس تمرکز می‌کنند، که امکان تأخیر کم به نماهای تولیدشده توسط لایه دسته‌ای را فراهم می‌کند. در اینجا با بانک اطلاعاتی تخصصی که فقط به‌صورت عمده به‌صورت انبوه ارسال می‌شوند، آشنا می‌شوید. خواهید فهمید که این بانک‌های اطلاعاتی به‌طور چشمگیری ساده‌تر از بانک‌های اطلاعاتی سنتی هستند و به آن‌ها ویژگی‌ها عملکردی، عملیاتی و استحکامی عالی می‌بخشد.

فصول ۱۲ تا ۱۷ بر لایه سرعت تمرکز می‌کنند، که تأخیر زیاد لایه دسته را برای ارائه نتایج به‌روز برای همه پرس‌وجوها جبران می‌کند. در اینجا با بانک‌های اطلاعاتی NoSQL، پردازش جریان و مدیریت پیچیدگی‌ها محاسبات افزایشی آشنا می‌شوید.

در فصل ۱۸ از دانش جدید یافته شده استفاده می‌شود تا یک بار دیگر معماری لامبدا را مرور کرده و شکاف‌های باقی‌مانده را پر کنید. در مورد پردازش دسته‌ای افزایشی، انواع معماری اصلی لامبدا و چگونگی استفاده بیشتر از منابع را بیاموزید.

بارگیری‌ها و کنوانسیون‌های کد

منبع کتاب را می‌توانید در https://github.com/Big-Data-Manning پیدا کنید. کد منبع را برای مثال در حال اجرا SuperWebAnalytics.com ارائه کرده‌ایم.

بخش عمده‌ای از کد منبع در لیست‌های شماره‌دار نشان داده‌شده است. این لیست‌ها برای ارائه بخش‌های کامل کد است. برخی از لیست‌ها برای کمک به برجسته یا توضیح بخش‌های خاصی از کد حاشیه‌نویسی شدند. در سایر نقاط در طول متن، در صورت لزوم از قطعات کد استفاده می‌شود. برای مشخص کردن کد برای جاوا از نوع پیک استفاده می‌شود.

در هر دو فهرست و قطعات، از یک فونت کد ضخیم استفاده می‌کنیم تا به شناسایی قسمت‌های اصلی کد که در متن توضیح داده می‌شود کمک کند.

نویسنده آنلاین

خرید کلان داده شامل دسترسی رایگان به یک انجمن وب خصوصی است که توسط انتشارات Manning اداره می‌شود و در آنجا می‌توانید در مورد کتاب اظهارنظر کنید، سؤالات فنی بپرسید و از نویسندگان و سایر کاربران کمک بگیرید. برای دسترسی به انجمن و عضویت در آن، مرورگر وب خود را به آدرس www.manning.com/BigData نشان دهید. این صفحه نویسنده آنلاین (AO) اطلاعاتی در مورد نحوه دستیابی به انجمن پس از ثبت‌نام، انواع کمک در دسترس و قوانین مربوط به این انجمن در اختیار قرار می‌دهد.

تعهد منینگ برای خوانندگان، فراهم آوردن محلی است که می‌توانند گفتگوی معناداری بین خوانندگان فردی و بین خوانندگان و نویسندگان انجام دهند. این تعهد به مشارکت خاصی از طرف نویسندگان نیست، که سهم وی در انجمن AO داوطلبانه (و بدون پرداخت) باقی می‌ماند. پیشنهاد می‌کنیم از نویسندگان سؤالاتی چالش‌برانگیز بپرسید، مبادا علاقه‌مندی آن‌ها از بین برود!

تا زمان چاپ این کتاب، انجمن AO و بایگانی مباحث قبلی از وب‌سایت ناشر قابل‌دسترسی خواهد بود.

فصل ۱ یک روش جدید برای کلان داده‌ها

این فصل مباحث زیر را پوشش می‌دهد:

- مشکلات معمول هنگام مقیاس‌گذاری یک پایگاه داده سنتی با آن روبه‌رو می‌شوند
- چرا NoSQL راهکار نیست
- تفکر در مورد سیستم‌های کلان داده از اولین اصول
- چشم‌انداز ابزارهای کلان داده‌ها
- معرفی SuperWebAnalytics.com

در دهه گذشته، میزان داده‌های ایجادشده بسیار بالا رفته است. بیش از ۳۰،۰۰۰ گیگابایت داده در هر ثانیه تولید می‌شود و سرعت ایجاد داده‌ها فقط در حال تسریع است. اطلاعاتی که با آن‌ها سروکار داریم متنوع است. کاربران محتوایی مانند توییت پست‌های وبلاگ، تعامل در شبکه‌های اجتماعی و عکس‌ها ایجاد می‌کنند. سرورها پیغام‌هایی را در مورد آنچه انجام می‌دهند وارد می‌کنند. دانشمندان اندازه‌گیری‌های دقیقی از جهان پیرامون ما ایجاد می‌کنند. اینترنت، منبع نهایی داده‌ها، تقریباً غیرقابل‌درک است.

این رشد حیرت‌انگیز در داده‌ها، تجارت‌ها را به‌شدت تحت تأثیر قرار داده است. سیستم‌های پایگاه داده سنتی، مانند پایگاه داده‌های رابطه‌ای، به حد مجازی رسیده‌اند. در تعداد فزاینده‌ای از موارد، این سیستم‌ها تحت‌فشار **"کلان داده‌ها "** قرار می‌گیرند. سیستم‌های سنتی و تکنیک‌های مدیریت داده‌هایشان، نتوانسته‌اند به مقیاس بزرگ داده تبدیل شوند.

برای مقابله با چالش‌های کلان داده‌ها، نسل جدیدی از فناوری‌ها پدید آمده است. بسیاری از این فناوری‌های جدید تحت عنوان NoSQL گروه‌بندی‌شده‌اند. از بعضی جهات، این فن‌آوری‌های جدید پیچیده‌تر از بانک‌های اطلاعاتی سنتی هستند و بهروش‌های دیگر ساده‌تر هستند. این سیستم‌ها می‌توانند مجموعه‌ای از داده‌های گسترده‌تری را ترسیم کنند، اما استفاده از این فن‌آوری‌ها به‌طور مؤثری نیاز به مجموعه‌ای از تکنیک‌های اساسی دارد. آن‌ها راه‌حل‌های یک اندازه مناسب نیستند. بسیاری از این سیستم‌های داده بزرگ توسط گوگل پیشگام شدند، شامل سیستم‌های فایل توزیع‌شده، کاهش نقشه چهارچوب محاسبات و خدمات توزیع قفل می‌شود. یکی دیگر از پیشگامان قابل‌توجه در فضا، آمازون بود که یک فروشگاه مهم و کلیدی توزیع‌شده با ارزش، به نام دینامو ایجاد کرد. جامعه منبع باز در سال‌های بعد به هادوپ، ایچ بیس، مانگو دی بی، کاساندرا، ربیت ام کیو و پروژه‌های بی‌شماری دیگر پاسخ داد.

این کتاب در مورد پیچیدگی به همان اندازه که درباره مقیاس‌پذیری است، صحبت می‌کند. برای پاسخگویی به چالش‌های کلان داده‌ها، به سیستم‌های داده از پایه و اساس فکر خواهیم کرد. خواهید فهمید که برخی از ابتدایی‌ترین روش‌های مدیریت داده‌ها در سیستم‌های سنتی مانند سیستم‌های مدیریت پایگاه داده رابطه‌ای (RDBMS) برای سیستم‌های کلان داده بسیار پیچیده هستند. رویکرد ساده‌تر و جایگزین، الگوی جدید برای داده‌های بزرگ است که کشف خواهید کرد. این رویکرد را معماری لامبدا لقب داده‌ایم.

در این فصل اول، به بررسی **"مشکل کلان داده"** می‌پردازید و چرا به الگوی جدیدی برای کلان داده‌ها نیاز دارید. خطرات برخی از تکنیک‌های سنتی برای مقیاس‌گذاری و کشف برخی از نقص‌های عمیق در روش سنتی ساخت سیستم‌های داده را مشاهده خواهید کرد. با شروع از اصول اولیه سیستم‌های داده، روش دیگری را برای ساختن سیستم‌های داده ایجاد می‌کنیم که از پیچیدگی تکنیک‌های سنتی جلوگیری می‌کند. نگاهی به این موضوع می‌اندازید که چگونه روندهای اخیر فناوری، استفاده از انواع جدید سیستم را ترغیب می‌کند، و در آخر به یک مثال از سیستم کلان داده که می‌خواهیم از طریق این کتاب بسازیم نگاهی بیندازید تا مفاهیم کلیدی را نشان دهید.

۱–۱. ساختار این کتاب

باید در مورد این کتاب به‌عنوان یک کتاب تئوری درجه اول، و با تمرکز روی چگونگی دستیابی به ساختن یک راه‌حل برای هر مشکل کلان داده فکر کنید. اصولی که می‌آموزید صرف‌نظر از ابزارهای موجود در چشم‌انداز فعلی صحیح است، و می‌توانید با استفاده از این اصول، با دقت بیش‌تری ابزارهای مناسب برای برنامه خود را انتخاب کنید. این کتاب بررسی بانک اطلاعاتی، محاسبات و سایر فناوری‌های مرتبط نیست. اگرچه یاد خواهید گرفت که چگونه بسیاری از این ابزارها را در طول این کتاب، مانند هادوپ، کاساندرا، استورم و ثریف استفاده کنید، هدف این کتاب یادگیری این ابزارها به‌خودی‌خود نیست. در عوض، این ابزار وسیله‌ای برای یادگیری اصول اساسی معماری سیستم‌های داده قوی و مقیاس‌پذیر است. انجام یک مقایسه و تقابل بین ابزارها، باعث عدالت نمی‌شود، زیرا این امر فقط از یادگیری اصول اساسی منحرف می‌شود. به‌عبارت‌دیگر، می‌خواهید نحوه ماهی‌گیری را یاد بگیرید، نه‌فقط نحوه استفاده از یک میله ماهی‌گیری خاص.

در همین راستا، کتاب را در فصل‌های تئوری و تصویرسازی قرار داده‌ایم. می‌توانید فقط فصل‌های تئوری را بخوانید و درک کاملی از نحوه ساخت سیستم‌های داده بزرگ داشته باشید، اما فکر می‌کنیم روند نقشه‌برداری این تئوری بر روی ابزارهای خاص در فصل‌های تصویرگری، درک غنی‌تر و ظریف‌تری از مطالب را برای شما رقم خواهد زد. با این اسم‌ها فریب نخورید – فصل‌های نظریه بسیار موردبررسی قرار می‌گیرند. مثال فراگیر در کتاب — SuperWebAnalytics.com در هر دو بخش تئوری و تصویرسازی استفاده می‌شود. در فصل‌های تئوری الگوریتم‌ها، طرح‌های شاخص و معماری برای SuperWebAnalytics.com را مشاهده خواهید کرد.

۲–۱. مقیاس‌گذاری با یک بانک اطلاعاتی سنتی

بیایید با شروع از جایی که بسیاری از برنامه‌نویسان شروع به کاوش در مورد داده‌های بزرگ می‌کنیم: ضربه زدن به محدوده فن‌آوری‌های سنتی پایگاه داده.

فرض کنید رئیس شما از شما می‌خواهد یک نرم‌افزار ساده آنالیز وب بسازید. برنامه باید تعداد بازدیدهای صفحه را برای هر URL که مشتری مایل به پیگیری آن است ردیابی کند. صفحه وب مشتری هر بار که نمای صفحه

دریافت می‌کند، سرور وب برنامه را با آدرس اینترنتی خود درمی‌آورد. علاوه بر این، برنامه باید بتواند در هر نقطه به شما بگوید که ۱۰۰ URL برتر ازنظر تعداد بازدید صفحه کدام‌یک هستند.

با یک طرح روابط سنتی برای بازدیدهای صفحه شروع می‌کنید که چیزی شبیه شکل ۱-۱ است. انتهای کار شما شامل RDBMS با جدول آن طرح و یک وب سرور است. هر زمان که شخصی صفحه‌ای را که توسط برنامه شما ردیابی می‌شود بارگیری می‌کند، صفحه وب سرور وب خود را با نمای صفحه متصل می‌کند، و سرور وب شما سطر مربوطه را در پایگاه داده افزایش می‌دهد.

بیایید ببینیم با تکامل برنامه، چه مشکلی ایجاد می‌شود. همان‌طور که می‌خواهید ببینید، هم با مقیاس‌پذیری و هم پیچیدگی با مشکل روبه‌رو خواهید شد.

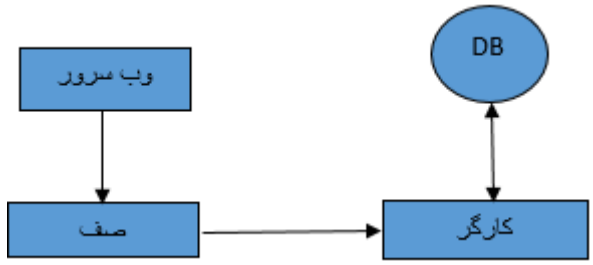
نام ستون	نوع
شناسه	Integer
شناسه کاربر	Integer
url	Varchar(۲۵۵)
بازدید صفحه	Bigint

شکل ۱-۱. طرح رابطه‌ای برای کاربرد ساده آنالیز

۱-۲-۱. مقیاس‌گذاری با یک صف

محصول تجزیه‌وتحلیل وب موفقیت بزرگی است و ترافیک برنامه مانند آتش‌سوزی در حال رشد است. شرکت شما مهمانی بزرگی را برپا می‌کند، اما در اواسط جشن شروع به دریافت ایمیل‌های زیادی از سیستم نظارت خود می‌کنید. همه آن‌ها یکسان می‌گویند: **"خطای وقفه در هنگام ورود به پایگاه داده"**. به ورودی‌های سیستم مربوط نگاه می‌کنید و مشکل آشکار است. بانک اطلاعاتی نمی‌تواند بار را حفظ کند، بنابراین درخواست‌هایی را برای توسعه صفحه‌های نمایش است، را به پایان می‌رسانید. برای رفع مشکل باید کاری انجام دهید، و باید کاری سریع را انجام دهید. می‌دانید که فقط انجام یک واحد افزودنی در یک زمان به پایگاه داده بی‌فایده است. اگر تعداد درخواست‌های بسیاری را در یک درخواست واحد انجام دهید، می‌تواند کارآمدتر باشد. بنابراین می‌توانید با استفاده از بخش پشت‌صحنه خود مجدداً معماری کنید تا این امر امکان‌پذیر شود. به‌جای اینکه سرور وب مستقیماً به دیتابیس برخورد کند، یک صف بین سرور وب و پایگاه داده قرار می‌دهید. که هر وقت نمای صفحه جدید دریافت می‌کنید، آن رویداد به‌صف اضافه می‌شود. سپس یک فرآیند کارگر ایجاد می‌کنید که ۱۰۰ وقایع را در یک زمان خاموش صف می‌خواند و آن‌ها را در یک به‌روزرسانی پایگاه داده واحد قرار می‌دهد. این در شکل ۲-۱ نشان داده‌شده است. این طرح به‌خوبی برای حل مسائل مربوط به زمان کار می‌کند. حتی این امتیاز اضافی را هم

دارد که اگر بانک اطلاعاتی بار دیگر بارگیری شود، صف به‌جای زمان‌بندی به سرور وب و از دست دادن داده‌های بالقوه، بزرگ‌تر می‌شود.



شکل ۲-۱. به‌روزرسانی دسته‌ای با صف و کارگر.

۲-۲-۱. مقیاس‌گذاری با خرد کردن بانک اطلاعات

متأسفانه، افزودن صف و انجام به‌روزرسانی‌های دسته‌ای، تنها یک مشکل گروهی برای مشکل مقیاس‌گذاری بود. برنامه شما همچنان بیش‌تر و بیش‌تر محبوب می‌شود و دوباره پایگاه داده بارگیری می‌شود. کارگر شما نمی‌تواند از نوشتن مطالب خودداری کند، بنابراین سعی می‌کنید کارگران بیش‌تری را اضافه کنید تا به‌روزرسانی‌ها را موازی کنید. متأسفانه این کمکی نمی‌کند. دیتابیس به‌وضوح تنگنا شده است. برخی از جستجوهای گوگل را برای چگونگی مقیاس بندی یک رابطه سنگین نوشتن بانک اطلاعاتی انجام می‌دهید. دریافتید که بهترین روش استفاده از چندین سرور پایگاه داده و پخش جدول در سرورهای مختلف است. هر سرور یک زیرمجموعه از داده‌ها برای جدول خواهد داشت. این به پارتیشن‌بندی یا سایه زنی افقی معروف است. این روش بار نوشتن را در چندین ماشین پخش می‌کند. روش سایه زنی که استفاده می‌کنید این است که با گرفتن hash از کلید تعدیل‌شده توسط تعداد محافظ‌ها، قسمت مربوط به هر کلید را انتخاب کنید. نقشه‌برداری از کلیدها برای محافظت از محافظ با استفاده از یک عملکرد هش باعث می‌شود که کلیدها به‌طور یکنواخت در قسمت‌های مختلف توزیع شوند. یک اسکریپت را برای نقشه‌برداری در تمام سطرها در نمونه پایگاه داده واحد خود می‌نویسید و داده‌ها را به چهار بخش تقسیم می‌کنید. مدت‌زمان زیادی طول می‌کشد تا اجرا شود، بنابراین می‌توانید کارگر را که صفحات صفحه را افزایش می‌دهد متوقف کنید تا به پایان برسد. در غیر این صورت سودتان را در طول انتقال از دست می‌دهید. سرانجام، همه کد برنامه باید بدانند که چگونه می‌توانید کلید هر قسمت را پیدا کنید. بنابراین یک کتابخانه را در اطراف کد مدیریت داده‌های خود می‌پیچید که تعداد محافظ‌های یک پرونده پیکربندی را می‌خواند، و تمام کد برنامه خود را مجدداً تنظیم می‌کنید. برای به دست آوردن ۱۰۰ آدرس اینترنتی برتر از هر بخش، باید پرس‌وجو URL-۱۰۰ خود را اصلاح کنید و آن‌ها را برای ۱۰۰ URL برتر جهانی ادغام کنید.

هرچه این برنامه بیش‌تر و بیش‌تر محبوب می‌شود، برای ادامه کار با بار نوشتن، مجبور هستید که پایگاه داده را در قسمت‌های بیش‌تری تغییر دهید. هر بار سخت‌تر می‌شود زیرا کارهای بسیار بیش‌تری برای هماهنگی وجود

دارد. فقط نمی‌توانید یک اسکریپت را برای انجام اشتراک مجدد استفاده کنید، زیرا خیلی کند است. باید همه اشتراک‌گذاری مجدد را به‌صورت موازی انجام دهید و بسیاری از اسکریپت‌های فعال کارگران را به‌طور هم‌زمان مدیریت کنید فراموش می‌کنید که کد برنامه را با تعداد جدید محافظ‌ها به‌روز کنید، و باعث می‌شود بسیاری از این نوشتن‌ها در قسمت‌های نادرست نوشته شوند. بنابراین باید یک اسکریپت را یک بار به‌صورت دستی بنویسید تا داده‌ها را طی کرده و هرچه را که نابجا قرارگرفته بود، جابجا کنید.

۳-۲-۱.آغاز مسائل مربوط به تحمل خطا

سرانجام آن‌قدر بند دارید که به یک اتفاق نادر تبدیل می‌شود که باعث خرابی دیسک روی یکی از ماشین‌های بانک اطلاعاتی شود. در هنگام پایین آمدن آن دستگاه، آن بخش از داده‌ها در دسترس نیست. برای حل این مسئله چند کار انجام می‌دهید:

سیستم صف کارگر / کارگر خود را به‌روز می‌کنید و می‌توانید بر روی صفحات غیرقابل‌دسترسی در صف جداگانه **"در انتظار"** جداگانه قرار دهید که می‌خواهید هر پنج دقیقه یک‌بار از آن استفاده کنید.

از قابلیت‌های تکرار بانک اطلاعاتی برای اضافه کردن کارگر به هر قسمت استفاده می‌کنید، بنابراین در صورت پایین آمدن کارفرما، از نسخه پشتیبان تهیه خواهید کرد. به کارگر نمی‌نویسید، اما حداقل مشتریان هنوز هم می‌توانند آمار موجود در برنامه را مشاهده کنند.

با خودتان فکر می‌کنید، **"در روزهای نخست وقت خود را صرف ساختن ویژگی‌ها جدید برای مشتریان کردید. اکنون به نظر می‌رسد که تمام‌وقت خود را صرف مشکلات خواندن و نوشتن داده‌ها می‌کنم. "**

۴-۲-۱. مسائل مربوط به انحراف

هنگام کار بر روی صف / کد کارگر، به‌طور تصادفی یک اشکال را تولید می‌کنید که تعداد بازدیدهای صفحه را به‌جای یکی از هر URL افزایش می‌دهد. تا ۲۴ ساعت بعد متوجه آن نمی‌شوید، اما تا آن زمان خسارت واردشده است. تهیه نسخه پشتیبان هفتگی شما کمک نمی‌کند زیرا هیچ راهی برای فهمیدن اطلاعات خراب نیست. بعدازاین همه کار در تلاش برای تبدیل‌شدن به سیستم مقیاس‌پذیر، و تحمل خرابی دستگاه، سیستم هیچ‌گونه مقاومتی در برابر خطای انسانی ندارد. و اگر یک تضمین در نرم‌افزار وجود دارد، این اشکالات به‌ناچار آن را به تولید می‌رساند، مهم نیست که چقدر سخت برای جلوگیری از آن تلاش می‌کنید.

۵-۲-۱.چه اشتباهی رخداده است؟

با تکامل برنامه تحلیلی ساده وب، این سیستم همچنان پیچیده‌تر می‌شود: صف‌ها، محافظ‌ها، ماکت‌ها، اسکریپت‌های جدید و غیره. توسعه برنامه‌های کاربردی روی داده‌ها به چیزی بیش از دانستن شمای پایگاه داده نیاز دارد. کدها باید بدانند که چگونه با بخش‌های مناسب صحبت کنید، و اگر اشتباهی انجام دهید، هیچ‌چیزی مانع از خواندن یا نوشتن قسمت‌های اشتباه نمی‌شود. یک مشکل این است که بانک اطلاعاتی شما از ماهیت توزیع‌شده

خودآگاهی ندارد، بنابراین نمی‌تواند در مقابله با بخش‌ها، تکثیر و نمایش داده‌های توزیع‌شده کمک کند. تمام پیچیدگی‌ها، هم در زمینه کار با دیتابیس و هم در ایجاد کد برنامه سوق می‌دهد. اما بدترین مشکل این است که سیستم به دلیل خطاهای انسانی مهندسی نشده است. در حقیقت کاملاً برعکس: این سیستم پیچیده‌تر و پیچیده‌تر می‌شود و باعث می‌شود که اشتباه بیش‌ازپیش انجام شود. اشتباه در نرم‌افزار اجتناب‌ناپذیر است، و اگر برای آن مهندسی نکنید، ممکن است در حال نوشتن اسکریپت‌هایی باشید که به‌طور تصادفی داده‌ها را خراب کنند. پشتیبان گیری کافی نیست؛ سیستم باید با دقت اندیشیده شود تا بتواند خسارت‌هایی را که یک خطای انسانی می‌تواند ایجاد کند محدود کند. تحمل خطای انسان اختیاری نیست. این ضروری است، به‌ویژه هنگامی‌که کلان داده‌ها پیچیدگی‌ها بسیاری را برای ایجاد برنامه‌های کاربردی اضافه می‌کنند.

۶-۲-۱. چگونه تکنیک‌های کلان داده کمک خواهد کرد؟

تکنیک‌های کلان داده‌ای که می‌خواهید یاد بگیرید مسائل مقیاس‌پذیری و پیچیدگی را به روشی چشمگیری برطرف می‌کنند. اول‌ازهمه، پایگاه داده‌ها و سیستم‌های محاسباتی که برای کلان داده‌ها استفاده می‌کنید از ماهیت توزیع‌شده آن‌ها آگاه هستند. بنابراین، مواردی مانند کوچک کردن و تکثیر برای شما انجام می‌شود. هرگز در شرایطی قرار نخواهید گرفت که به‌صورت تصادفی از بخش صحیح پرس‌وجو کنید، زیرا این منطق در پایگاه داده داخلی می‌شود. وقتی مقیاس بندی می‌شود، گره‌ها را اضافه می‌کنید، و سیستم‌ها به‌طور خودکار روی گره‌های جدید دوباره تعادل ایجاد می‌کنند.

یکی دیگر از تکنیک‌های اصلی که در مورد آن یاد خواهید گرفت، تغییر داده‌ها غیرقابل تغییر است.

۳-۱. NoSQL یک پاناسرا نیست

دهه گذشته شاهد نوآوری عظیمی در سیستم‌های داده مقیاس‌پذیر بودیم. این‌ها شامل سیستم‌های محاسباتی در مقیاس بزرگ مانند Hadoop و پایگاه‌های داده مانند Cassandra و Riak است. این سیستم‌ها می‌توانند مقادیر بسیار زیادی از داده‌ها را، با وجود تبادلات جدی اداره کنند.

به‌عنوان‌مثال Hadoop می‌تواند محاسبات دسته‌ای بزرگ را در مقادیر بسیار زیاد داده‌ها موازی سازد، اما محاسبات دارای تأخیر بالایی هستند. از Hadoop برای هر چیزی که نیاز به نتایج با تأخیر کم دارید استفاده نکنید. پایگاه داده‌های NoSQL مانند Cassandra با ارائه یک مدل داده بسیار محدودتر از آنچه قبلاً با چیزی مانند SQL استفاده می‌کرده‌اید، به مقیاس‌پذیری خود دست پیدا می‌کنند. فشردن برنامه در این مدل‌های داده محدود می‌تواند بسیار پیچیده باشد.

این ابزارها به‌خودی‌خود یک نوع نوش دارو نیستند. اما هنگامی‌که به‌طور هوشمند در رابطه با یکدیگر استفاده می‌شود، می‌توانید برای مشکلات داده‌های دلخواه با تحمل خطای انسانی و حداقل پیچیدگی، سیستم‌های مقیاس‌پذیر تولید کنید. این معماری لامبدا است که در طول کتاب خواهید آموخت.

۴-۱. اصول اولیه

برای فهمیدن چگونگی درست ساختن سیستم‌های داده، باید به اصول اول برگردید. در اساسی‌ترین سطح، یک سیستم داده چه‌کاری انجام می‌دهد؟

بیایید با یک تعریف شهودی شروع کنیم: یک سیستم داده به سؤالات مبتنی بر اطلاعاتی که در گذشته تا به امروز به‌دست‌آمده پاسخ می‌دهد. بنابراین یک پروفایل شبکه اجتماعی به سؤالاتی از قبیل "نام این شخص چیست؟" و "چه تعداد از دوستان این شخص وجود دارد؟" یک صفحه وب حساب بانکی به سؤالاتی از قبیل "مانده فعلی من چیست؟" و "اخیراً چه حساب‌هایی روی حساب من اتفاق افتاده است؟" پاسخ می‌دهد.

سیستم‌های داده فقط اطلاعات را به خاطر نمی‌آورند و مجدداً مورداستفاده قرار می‌گیرند. آن‌ها بیت‌ها و تکه‌های مختلف را با هم ترکیب می‌کنند تا پاسخ‌های خود را تولید کنند. به‌عنوان‌مثال، موجودی حساب بانکی در تلفیق اطلاعات مربوط به کلیه معاملات موجود در حساب است. یکی دیگر از مشاهدات مهم این است که همه اطلاعات مربوط به یکدیگر برابر نیستند. برخی اطلاعات از دیگر اطلاعات به دست می‌آید. مانده‌حساب بانکی از تاریخچه معاملات انجام می‌شود. تعداد دوستان از لیست دوستان گرفته می‌شود و لیست دوستان از همه زمان‌هایی که کاربر دوستان خود را به پروفایل خود اضافه و از پروفایل خود حذف می‌کند مشتق شده است.

هنگامی‌که اطلاعات را از جایی که به دست می‌آید، ردیابی می‌کنید، درنهایت به اطلاعاتی منتهی می‌شوید که از چیز دیگری به دست نمی‌آیند. این اطلاعات خام شما است: اطلاعاتی که در اختیار دارید به دلیل وجودشان به‌سادگی صادق هستند. بیایید سوابق این اطلاعات را فراخوانی کنیم.

ممکن است برداشت دیگری از معنای کلمه داده داشته باشید. داده‌ها اغلب با کلمه اطلاعات به‌صورت متقابل استفاده می‌شوند. اما برای بقیه این کتاب، هنگامی‌که از کلمه داده استفاده می‌کنیم، به اطلاعات ویژه‌ای که از آن همه‌چیز مشتق شده است، اشاره می‌کنیم. اگر یک سیستم داده با نگاه کردن به داده‌های گذشته به سؤالات پاسخ دهد، آنگاه عمومی‌ترین سیستم داده با استفاده از کل داده‌ها به سؤالات پاسخ می‌دهد. بنابراین عمومی‌ترین تعریفی که می‌توانیم برای یک سیستم داده تعریف کنیم، موارد زیر است:

```
query = function(all data)
```

هر کاری که تابه‌حال بتوانید با داده انجام دهید تصور می‌شود به‌عنوان تابعی که تمام داده‌ها را به‌عنوان ورودی در نظر می‌گیرد بیان شود. این معادله را به خاطر بسپار، زیرا این اصلی‌ترین چیزهایی است که یاد خواهید گرفت. بارها و بارها به این معادله خواهیم پرداخت. معماری لامبدا یک رویکرد کلی برای اجرای یک عملکرد دلخواه در یک مجموعه داده دلخواه ارائه می‌دهد و این عملکرد را با تأخیر کم نتایج خود بازمی‌گرداند. این بدان معنا نیست که همیشه در هنگام اجرای سیستم داده از همان فناوری‌های دقیق استفاده خواهید کرد. فن‌آوری‌های خاصی که

استفاده می‌کنید بسته به نیاز ممکن است تغییر کند. اما معماری لامبدا یک رویکرد مداوم در انتخاب آن فناوری‌ها و اتصال آن‌ها به یکدیگر برای برآورده کردن نیازها تعریف می‌کند.

بیایید اکنون درباره خصوصیاتی که یک سیستم داده باید از خود نشان دهد بحث کنیم.

۵–۱. ویژگی‌ها موردنظر یک سیستم کلان داده

خواصی که باید در سیستم‌های کلان داده به دنبال آن باشید، به‌اندازه پیچیدگی در مورد مقیاس‌پذیری است. نه‌تنها یک سیستم کلان داده باید عملکرد خوبی داشته باشد و هم ازنظر منابع کارآمد باشد، بلکه باید به‌راحتی نیز در مورد آن، دلایل منطقی داشت. بیایید درباره هر خاصیت یکی‌یکی پیش برویم.

۵–۱–۱. استحکام و تحمل خطا

ساختن سیستم‌هایی که **"کار درست را انجام می‌دهند"** در مواجهه با چالش‌های سیستم‌های توزیع‌شده دشوار است. سیستم‌ها باوجوداینکه به‌طور تصادفی پایین می‌روند، باید به‌درستی در، معانی پیچیده ثبات در بانک‌های اطلاعاتی توزیع‌شده، داده‌های تکراری، هم‌زمانی و موارد دیگر رفتار کنند. این چالش‌ها حتی استدلال در مورد کاری که یک سیستم انجام می‌دهد را دشوار می‌کند. بخشی از ساختن یک سیستم کلان داده قوی، جلوگیری از این پیچیدگی‌ها است تا بتوانید به‌راحتی در مورد سیستم استدلال کنید.

همان‌طور که قبلاً موردبحث قرار گرفت، ضروری است که سیستم‌ها تحمل خطای انسانی را داشته باشند. این خاصیت غفلت مکرر از سیستم است که نمی‌خواهیم از آن چشم‌پوشی کنیم. در یک سیستم تولید، اجتناب‌ناپذیر است که شخصی مرتباً مرتکب اشتباهی شود، ازجمله با استفاده از کد نادرست که ارزش‌ها را در یک پایگاه داده خراب می‌کند. اگر تغییرناپذیری و اعتبار مجدد را به هسته یک سیستم کلان داده تبدیل کنید، این سیستم با ارائه یک مکانیسم روشن و ساده برای بازیابی به‌صورت ذاتی در برابر خطاهای انسانی مقاومت خواهد کرد. این در طول فصل‌های ۲ تا ۷ توضیح داده‌شده است.

۵–۱–۲. خواندن و به‌روزرسانی بازمان تأخیر کم

اکثریت قریب به‌اتفاق برنامه‌ها نیاز دارند که از تأخیر بسیار کم، به‌طورمعمول بین چند میلی‌ثانیه تا چند صد میلی‌ثانیه استفاده کنند. از طرف دیگر، نیازهای تأخیر به‌روزرسانی بین برنامه‌ها بسیار متفاوت است. برخی از برنامه‌ها برای انتشار سریع نیاز به، به‌روزرسانی دارند، اما در سایر برنامه‌ها تأخیر چندساعته خوب است. صرف‌نظر از آن، باید وقتی در سیستم‌های کلان داده خود به آن‌ها نیاز دارید به‌روزرسانی‌ها بازمان کوتاه را به دست آورید. مهم‌تر از همه، باید بدون به خطر انداختن استحکام سیستم قادر به دستیابی به خواندن و به‌روزرسانی بازمان کوتاه باشید. بحث درباره چگونگی دستیابی به‌روزرسانی‌های با تأخیر کم در مورد لایه سرعت را، از فصل ۱۲ شروع می‌کنید.

۳-۵-۱. مقیاس‌پذیری

مقیاس‌پذیری توانایی حفظ عملکرد در مقابل افزایش داده یا بار، با افزودن منابع به سیستم است. معماری لامبدا ازنظر افقی در تمام لایه‌های پشته سیستم مقیاس‌پذیر است: مقیاس‌گذاری با اضافه کردن ماشین‌های بیش‌تر انجام می‌شود.

۴-۵-۱. تعمیم

یک سیستم کلی می‌تواند طیف گسترده‌ای از برنامه‌ها را پشتیبانی کند. درواقع، اگر این برنامه به طیف گسترده‌ای از برنامه‌ها تعمیم ندهد، این کتاب مفید نخواهد بود! ازآنجاکه معماری لامبدا مبتنی بر توابع همه داده‌ها است، به کلیه برنامه‌های کاربردی اعم از سیستم‌های مدیریت مالی، تجزیه‌وتحلیل رسانه‌های اجتماعی، برنامه‌های علمی، شبکه‌های اجتماعی یا هر چیز دیگر تعمیم می‌یابد.

۵-۵-۱. قابلیت توسعه

نمی‌خواهید هر بار اضافه کردن ویژگی مرتبط، چرخ را دوباره اختراع کنید یا تغییراتی در نحوه عملکرد سیستم ایجاد کند. سیستم‌های توسعه‌پذیر امکان عملکرد را با حداقل هزینه توسعه اضافه می‌کنند. اغلب اوقات ویژگی جدید یا تغییر ویژگی قبلی، نیاز به انتقال داده‌های قدیمی به یک قالب جدید دارد. بخشی از ساختن سیستم قابل توسعه، انجام کارهای کوچکی را آسان می‌کند. قابلیت انجام مهاجرت‌های بزرگ، با سرعت و به‌آسانی رویکردی اصلی است که یاد خواهید گرفت.

۶-۵-۱. پرس‌وجوهای Ad hoc

قابلیت انجام نمایش داده موقت بر روی‌داده‌ها بسیار مهم است. تقریباً هر مجموعه کلان داده دارای مقادیر غیرقابل‌پیش‌بینی درون آن است. توانایی استخراج یک مجموعه داده به‌طور دلخواه فرصت‌هایی را برای بهینه‌سازی تجارت و برنامه‌های جدید فراهم می‌کند. درنهایت، نمی‌توانید چیزهای جالبی را با داده‌های خود کشف کنید، مگر این‌که بتوانید سؤالات موردنظر خود را از آن بپرسید. زمانی که در فصل‌های ۶ و ۷ در مورد پردازش دسته‌ای صحبت می‌کنیم، چگونگی نمایش داده‌های موقت را یاد خواهید گرفت.

۷-۵-۱. حداقل نگه‌داری

تعمیر و نگه‌داری مالیات بر توسعه‌دهندگان است. تعمیر و نگه‌داری کار موردنیاز برای بدن اشکال نگه‌داشتن سیستم است. این شامل پیش‌بینی زمان اضافه کردن ماشین به مقیاس، نگه‌داشتن فرآیند و کارکردن و اشکال‌زدایی هرگونه اشتباه در تولید است.

بخش مهمی از به حداقل رساندن تعمیر و نگهداری، انتخاب مؤلفه‌هایی است که تا حد امکان پیچیدگی اجرای کمی دارند. می‌خواهید به مؤلفه‌هایی متکی باشید که مکانیسم‌های ساده‌ای در زیر آن‌ها وجود دارد. به‌ویژه، بانک‌های اطلاعاتی توزیع‌شده تمایل بسیار پیچیده‌ای به داخلی دارند. هرچه یک سیستم پیچیده‌تر باشد، احتمالاً

پیش آمدن مشکل بیش‌تر خواهد آمد، و نکات بیش‌تری را برای درک اشکال و تنظیم آن باید در مورد سیستم بدانید.

با تکیه بر الگوریتم‌های ساده و مؤلفه‌های ساده، با پیچیدگی پیاده‌سازی مبارزه می‌کنید. ترفندی که در معماری لامبدا به‌کاررفته است این است که پیچیدگی را از اجزای اصلی و قطعات سیستم خارج می‌کند و خروجی‌های آن‌ها پس از چند ساعت قابل جدا شدن می‌باشند. پیچیده‌ترین مؤلفه‌های استفاده‌شده، مانند خواندن / نوشتن پایگاه داده‌های توزیع‌شده، در این لایه قرار دارند که خروجی‌ها درنهایت دور ریخته می‌شوند. هنگام بحث در مورد لایه سرعت در فصل ۱۲، به‌طور عمیق در مورد این‌روش صحبت خواهیم کرد.

۸–۵–۱. اشکال‌زدایی

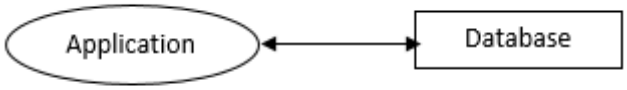
یک سیستم کلان داده باید اطلاعات لازم را برای اشکال‌زدایی سیستم در هنگام اشتباه انجام دهد. نکته اصلی این است که بتوانیم برای هر مقدار در سیستم ردیابی کنیم، دقیقاً چه چیزی باعث شده است که آن ارزش را داشته باشد.

" اشکال‌زدایی" در معماری لامبدا از طریق ماهیت عملکردی لایه دسته‌ای انجام می‌شود و ترجیح می‌دهد در صورت امکان از الگوریتم‌های بازپرداخت اعتبار استفاده کند.

دستیابی به همه این خصوصیات در یک سیستم ممکن است یک چالش هولناک به نظر برسد. اما با شروع از اصول اول، همان‌طور که معماری لامبدا انجام می‌دهد، این خصوصیات به‌طور طبیعی از طراحی سیستم حاصل می‌شوند. قبل از غرق شدن در معماری لامبدا، اجازه دهید نگاهی به معماری‌های سنتی‌تر – که با اتکا به محاسبات افزایشی مشخص می‌شود – بپردازیم و به چه دلیل آن‌ها قادر به برآورده کردن بسیاری از این ویژگی‌ها نیستند.

۶–۱. مشکلات معماری کاملاً افزایشی

در بالاترین سطح، معماری‌های سنتی شبیه شکل۳–۱هستند. آنچه این معماری‌ها را توصیف می‌کند، استفاده از بانک‌های اطلاعاتی خواندن و نوشتن و حفظ وضعیت در این بانک‌های اطلاعاتی به‌تدریج با مشاهده داده‌های جدید است. به‌عنوان‌مثال، یک روش افزایشی برای شمارش تعداد بازدیدهای صفحه می‌تواند پردازش نمای صفحه جدید با اضافه کردن یکی از پیشخوان‌ها برای URL آن باشد. این خصوصیات معماری بسیار اساسی‌تر از فقط رابطه در مقابل غیر رابطه است--- در حقیقت، اکثریت قریب به‌اتفاق استقرار بانک اطلاعاتی رابطه‌ای و غیر مرتبط به‌عنوان معماری کاملاً افزایشی انجام می‌شوند. این امر برای چندین دهه صادق بوده است. شایان‌ذکر است که معماری‌های کاملاً افزایشی آن‌قدر گسترده است که بسیاری از مردم درک نمی‌کنند که نمی‌توان از معماری متفاوت با مشکلاتشان جلوگیری کرد. این‌ها نمونه‌های خوبی از پیچیدگی‌ها آشنا هستند – پیچیدگی که بسیار مرسوم است، حتی فکر نمی‌کنید راهی برای جلوگیری از آن‌ها پیدا کنید.



شکل ۳-۱. معماری کاملاً افزایشی.

مشکلات معماری‌های کاملاً افزایشی قابل‌توجه است. با جستجوی پیچیدگی‌ها کلی معماری کاملاً افزایشی، اکتشاف خود را در مورد این موضوع آغاز خواهیم کرد. سپس برای یک مشکل به دو راه‌حل متضاد خواهیم پرداخت: یکی با استفاده از بهترین راه‌حل کاملاً افزایشی ممکن، و دیگری با استفاده از معماری لامبدا. خواهید دید که نسخه کاملاً افزایشی از هر لحاظ به طرز چشمگیری بدتر است.

۱-۶-۱. پیچیدگی عملیاتی

بسیاری از پیچیدگی‌ها ذاتی در معماری‌های کاملاً افزایشی وجود دارد که در ایجاد زیرساخت‌های تولید مشکل ایجاد می‌کند. در اینجا به یک موردتوجه خواهیم کرد: نیاز به خواندن / نوشتن پایگاه داده برای انجام تراکم آنلاین و کارهایی که باید به‌صورت عملیاتی انجام دهید تا امور به‌طور روان اجرا شود.

در یک دیتابیس خواندن / نوشتن، به‌عنوان یک فهرست دیسک به‌صورت افزایشی به آن اضافه و اصلاح می‌شود، بخش‌هایی از این شاخص استفاده‌نشده می‌مانند. این قسمت‌های بلااستفاده فضا را به خود می‌گیرند و درنهایت نیاز به پس گرفتن مجدد برای جلوگیری از پر شدن دیسک هستند. پس گرفتن مجدد فضا به‌محض استفاده از آن، بسیار گران است، بنابراین گاهی اوقات در فرآیندی بنام فشرده‌سازی، فضا به‌صورت عمده بازپرداخت می‌شود.

تراکم یک عمل فشرده است. سرور تقاضای قابل‌ملاحظه‌ای بالاتر را بر روی CPU و دیسک‌ها در هنگام تراکم قرار می‌دهد، که عملکرد آن دستگاه را در طی این مدت، زمان را به‌طور چشمگیری کاهش می‌دهد. بانک اطلاعاتی مانند HBase و Cassandra به دلیل نیاز به پیکربندی و مدیریت دقیق برای جلوگیری از بروز مشکلات یا قفل شدن سرور در حین فشرده‌سازی کاملاً شناخته‌شده‌اند. افت عملکرد در هنگام تراکم یک پیچیدگی است حتی می‌تواند باعث شکست آبشاری شود--- اگر خیلی از ماشین‌آلات به‌طور هم‌زمان جمع‌وجور شوند، بار پشتیبانی آن‌ها توسط سایر دستگاه‌های موجود در خوشه تحمل می‌شود. این به‌طور بالقوه می‌تواند بقیه خوشه‌ها را تحت‌فشار قرار دهد و باعث خرابی کامل شود. بارها دیده‌ایم که این حالت خرابی اتفاق می‌افتد.

برای مدیریت صحیح تراکم، باید فشرده‌سازی‌ها را بر روی هر گره برنامه‌ریزی کنید تا گره‌های خیلی زیاد به‌یک‌باره تحت تأثیر قرار نگیرند. باید از چگونگی فشرده‌سازی - و همچنین واریانس - آگاه باشید تا از داشتن گره‌های بیش‌تر تحت‌فشار تراکم از آنچه در نظر گرفته‌شده بود، خودداری کنید. باید مطمئن شوید که ظرفیت اندازه کافی دیسک روی گره‌های خود دارید تا بین فشرده‌سازی‌ها دوام داشته باشید. علاوه بر این، باید اطمینان

حاصل کنید که از ظرفیت کافی در خوشه خود برخوردار هستید تا هنگام از بین رفتن منابع در هنگام تراکم، دچار بار اضافی نشود.

همه این‌ها را می‌توان توسط کارکنان عملیاتی صالح مدیریت کرد، اما این ادعای ما است که بهترین راه مقابله با هر نوع پیچیدگی، خلاص شدن از شر این پیچیدگی است. هرچه سیستم حالت خرابی کم‌تری داشته باشد احتمال وقوع خرابی غیرمنتظره کم‌تر است. پرداختن به فشرده‌سازی آنلاین پیچیدگی ذاتی برای معماری‌های کاملاً افزایشی است، اما در معماری لامبدا پایگاه داده‌های اولیه نیازی به فشرده‌سازی آنلاین ندارند.

۲-۶-۱. پیچیدگی شدید دستیابی به قوام نهایی

یکی دیگر از نتایج پیچیدگی معماری‌های افزایشی وقتی است که سعی می‌کنید سیستم‌ها را بسیار در دسترس قرار دهید. یک سیستم کاملاً در دسترس امکان جستجو و به‌روزرسانی‌ها را حتی در صورت خرابی دستگاه یا بخشی از شبکه فراهم می‌کند. به نظر می‌رسد که دستیابی به دسترسی بالا به‌طور مستقیم با ویژگی مهم دیگری به نام سازگاری رقابت می‌کند. یک سیستم استوار نتایج را با در نظر گرفتن تمام نوشته‌های قبلی به دست می‌آورد. یک قضیه به نام CAP نشان داده است که حضور هم‌زمان دستیابی زیاد و استحکام در یک سیستم، در حضور پارتیشن‌های شبکه غیرممکن است. بنابراین یک سیستم همیشه در دسترس در بعضی مواقع نتایج قطعی را در حین پارتیشن شبکه بازمی‌گرداند.

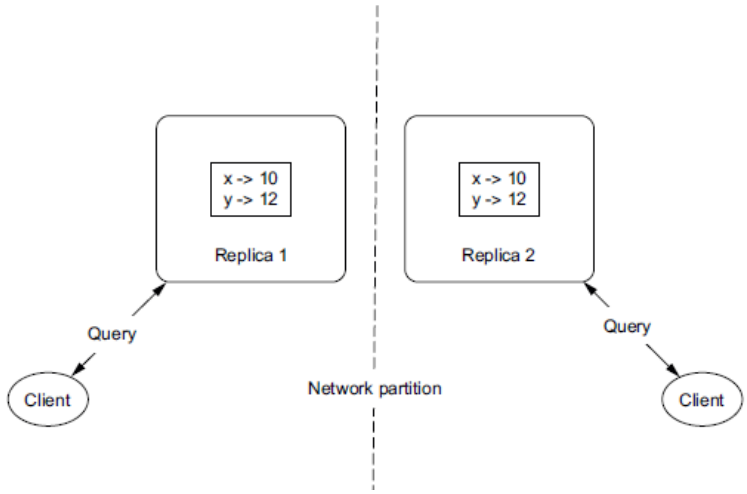
قضیه CAP در فصل ۱۲ به‌طور کامل موردبحث قرارگرفته است. در اینجا می‌خواهیم به این موضوع بپردازیم که چگونه ناتوانی در دستیابی به سازگاری کامل و در دسترس بودن زیاد در همه زمان‌ها بر توانایی شما در ساخت سیستم‌ها تأثیر دارد. نشان می‌دهد که اگر نیازهای شغلی خواستار دسترسی زیاد و استحکام بیش‌تر است، پیچیدگی زیادی وجود دارد که باید با آن مقابله کنید.

به‌منظور پایان دادن به یک سیستم همیشه در دسترس، پس از اتمام پارتیشن شبکه (معروف به ثبات نهایی)، کمک زیادی از برنامه شما نیاز دارد. به‌عنوان‌مثال، موارد اصلی استفاده از شمارش در یک پایگاه داده را در نظر بگیرید. راه واضح در مورد این امر ذخیره یک شماره در دیتابیس و افزودن هر شماره در هر رویدادی است که نیاز به افزایش تعداد دارد. شاید تعجب کنید که اگر می‌خواستید از این رویکرد استفاده کنید، در هنگام پارتیشن‌های شبکه از دست دادن داده‌های گسترده رنج خواهید برد.

دلیل این امر این است که نحوه توزیع پایگاه داده با دستیابی به ماکت‌های متعدد از همه اطلاعات ذخیره‌شده، به دسترسی بالایی می‌رسد. هنگامی‌که بسیاری از کپی‌های همان اطلاعات را حفظ می‌کنید، حتی اگر یک دستگاه پایین بیاید یا شبکه تقسیم شود، همان‌طور که در شکل ۴-۱ نشان داده‌شده است، آن اطلاعات هنوز هم موجود هستند. در حین پارتیشن شبکه، سیستمی که بسیار در دسترس باشد، مشتری را به‌روز می‌کند تا هرگونه کپی برای

آن‌ها قابل‌دستیابی باشد. این باعث می‌شود که ماکت‌ها از هم جدا شوند و مجموعه‌های مختلفی از به‌روزرسانی را دریافت کنند. فقط وقتی پارتیشن از بین می‌رود، ماکت‌ها را می‌توان در یک مقدار مشترک ادغام کرد.

فرض کنید با شروع یک پارتیشن شبکه، دو ماکت با تعداد ۱۰ داشته باشید. فرض کنید که ماکت اول دو مرحله و ماکت دوم یک مرحله افزایش می‌یابد. وقتی زمان فرارسیدن ادغام این ماکت‌ها با مقادیر ۲ و ۱۱ فرا رسید، ارزش ادغام‌شده باید چه باشد؟ اگرچه جواب صحیح ۱۳ است، اما نمی‌توان با نگاه کردن به شماره‌های ۱۲ و ۱۱، فهمید. می‌توانستند از ۱۱ تغییر کنند (در این صورت جواب ۱۲ خواهد شد)، یا می‌توانستند از ۰ (یا در این حالت پاسخ ۲۳ می‌شد) تغییر پیدا کنند.



شکل ۴-۱. استفاده از تکثیر برای افزایش در دسترس بودن.

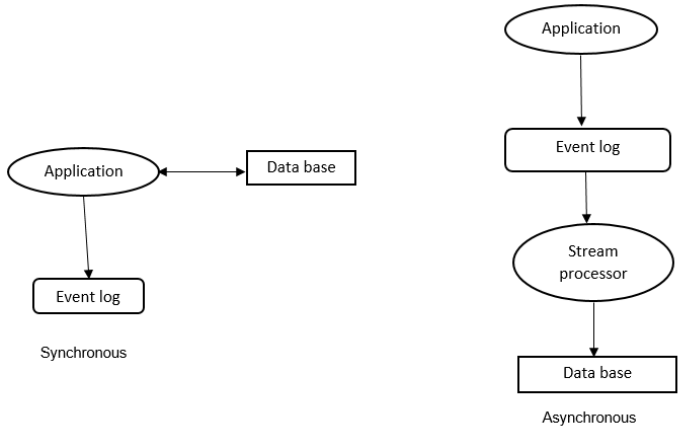
برای انجام درست شمارش در دسترس، کافی است فقط یک تعداد را ذخیره کنید. به یک ساختار داده نیاز دارید که در هنگام واگرایی مقادیر قابل ادغام باشد، و باید کدی را اجرا کنید که پس از پایان بخش‌ها، مقادیر را ترمیم کند. این یک مقدار پیچیدگی شگفت‌انگیز است که فقط برای حفظ یک تعداد ساده باید با آن سروکار داشته باشید. به‌طورکلی، رسیدگی به سازگاری نهایی در سیستم‌های افزایشی و همیشه در دسترس غیرضروری است و مستعد خطا است. این پیچیدگی به‌صورت سیستم‌های کاملاً افزاینده و کاملاً در دسترس است. بعداً خواهید دید که چگونه معماری لامبدا خود را به روشی متفاوت ساختار می‌دهد که بار دستیابی به سیستم‌های همیشه در دسترس و درنهایت سازگار را بسیار کاهش می‌دهد.

۳-۶-۱. عدم تحمل خطای انسانی

آخرین مشکل معماری‌های کاملاً افزایشی که می‌خواهیم به آن اشاره کنیم عدم تحمل خطای انسان طبیعی آن‌ها است. یک سیستم افزایشی دائماً حالتی را که در پایگاه داده نگهداری می‌کند، اصلاح می‌کند، به این معنی

که یک اشتباه همچنین می‌تواند حالت موجود در پایگاه داده را تغییر دهد. ازآنجاکه اشتباهات اجتناب‌ناپذیر است، پایگاه داده در یک معماری کاملاً افزایشی تضمین می‌دهد که خراب شود.

توجه به این نکته حائز اهمیت است که این یکی از معدود پیچیدگی‌ها معماری کاملاً افزایشی است که بدون تجدیدنظر کامل در مورد معماری قابل‌حل است. دو معماری که در شکل ۵-۱ نشان داده‌شده است را در نظر بگیرید: یکی معماری هم‌زمان، که در آن برنامه مستقیماً به پایگاه داده می‌دهد و دیگری معماری ناهم‌زمان، که در آن وقایع قبل از به‌روزرسانی بانک اطلاعاتی در پس‌زمینه، به یک صف می‌روند. در هر دو حالت، هر رویدادی به‌طور دائم به یک پایگاه داده وقایع وارد می‌شود. با نگه‌داشتن هر رویدادی، اگر یک اشتباه انسانی باعث خرابی دیتابیس شود، می‌توانید به انبار رویدادها برگردید و وضعیت مناسب پایگاه داده را بازسازی کنید. ازآنجاکه انبار رویدادها تغییرناپذیر است و به‌طور مداوم در حال رشد است، می‌توان چک‌های زائد مانند مجوزها را نیز در آن قرار داد تا خطای بسیار ناچیز برای اشتباه در انبار رویدادها بسیار کم باشد. این تکنیک همچنین اساس کار در معماری لامبدا است و در فصل‌های ۲ و ۳ به‌طور کامل موردبحث قرار می‌گیرد.



شکل ۵-۱. اضافه کردن ورود به سیستم به‌طور کامل به‌صورت افزایشی.

اگرچه معماری‌های کاملاً افزایشی با ورود به سیستم می‌توانند بر روی تحمل خطای انسان در معماری‌های کاملاً افزایشی بدون ورود به سیستم غلبه کنند، اما ورود به سیستم هیچ کاری را برای رفع سایر پیچیدگی‌ها موردبحث انجام نمی‌دهد. و همان‌طور که در بخش بعدی مشاهده خواهید کرد، هر معماری که صرفاً مبتنی بر محاسبات کاملاً افزایشی است، ازجمله آن‌هایی که دارای ورود به سیستم هستند، برای حل بسیاری از مشکلات تلاش خواهند کرد.

۴-۶-۱. راه‌حل کاملاً افزایشی در مقابل راه‌حل معماری لامبدا

یکی از نمونه نمایش داده شد که در سراسر کتاب پیاده‌سازی شده است تضاد بسیار خوبی بین معماری‌های کاملاً افزایشی و لامبدا است. هیچ مشکلی درباره این سؤال وجود ندارد - درواقع، این مبتنی بر مشکلات دنیای

واقعی است که چندین بار در حرفه خود با آن‌ها روبرو شده‌ایم. پرس‌وجو مربوط به تجزیه‌وتحلیل صفحه است و بر روی دو نوع داده وارد می‌شود:

بازدید از صفحه، که حاوی شناسه کاربر، URL و نشانگر زمانی است.

معادل، که شامل دو شناسه کاربر است. یک معادل نشان می‌دهد که دو شناسه کاربری به یک شخص مراجعه می‌کنند. به‌عنوان‌مثال، ممکن است بین ایمیل sally@gmail.com و نام کاربری سالی اشتراکی داشته باشید. اگر sally@gmail.com همچنین نام کاربری sally۲ را ثبت می‌کند، باید بین sally@gmail.com و sally۲ یک اشتراک داشته باشید. با انتقال‌پذیری، می‌دانید که نام‌های کاربری سالی و سالی ۲ به همان شخص اشاره دارند. هدف پرس‌وجو محاسبه تعداد بازدیدکنندگان منحصربه‌فرد به URL در مدت‌زمان معینی است.

نمایش داده شد باید با تمام داده‌ها به‌روزرسانی شود و با حداقل تأخیر (کم‌تر از ۱۰۰ میلی‌ثانیه) پاسخ دهد. در اینجا رابط نمایش داده شد:

آنچه پیاده‌سازی این پرس‌وجو را دشوار می‌کند همان معیارهای معادل است. اگر شخصی در یک بازه زمانی هم‌زمان با دو شناسه کاربر متصل از طریق سه برابر (حتی به‌صورت انتقالی) به همان آدرس اینترنتی مراجعه کند، فقط باید به‌عنوان یک بازدید حساب شود. یک اشتراک جدید که می‌تواند نتایج هر پرس‌وجو در هر بازه زمانی برای هر URL را تغییر دهد.

از نشان دادن جزئیات راه‌حل‌ها در این مرحله خودداری خواهیم کرد، زیرا مفاهیم زیادی برای درک آن‌ها باید پوشانده شوند: نمایه‌سازی، پایگاه داده‌های توزیع‌شده، پردازش دسته‌ای، Hyper Log Log و موارد دیگر. غلبه بر شما با همه این مفاهیم در این مقطع کارایی ندارد. در عوض، به ویژگی‌ها راه‌حل‌ها و تفاوت‌های چشمگیر بین آن‌ها خواهیم پرداخت. بهترین راه‌حل کاملاً افزایشی ممکن در بخش ۱۰ به‌تفصیل نشان داده‌شده است، و راه‌حل معماری لامبدا در فصل‌های ۸، ۹، ۱۴ و ۱۵ ساخته‌شده است.

دو راه‌حل را می‌توان در سه محور مقایسه کرد: دقت، تأخیر و توان. راه‌حل معماری لامبدا از هر نظر به‌طور قابل‌توجهی بهتر است. هر دو باید تقریبی عمل کنند، اما نسخه کاملاً افزایشی مجبور است از یک روش تقریبی درجه دوم با یک خطا ۳-۵ برابر بدتر استفاده می‌کند. انجام پرس‌وجوها در نسخه کاملاً افزایشی بسیار مهم است و تأثیر آن بر تأخیر و توان است. اما بارزترین تفاوت این دو رویکرد، نسخه‌های کاملاً افزایشی برای دستیابی به هر نقطه نزدیک به توان معقول، نیاز به استفاده از سخت‌افزار مخصوص دارد. ازآنجاکه نسخه کاملاً افزایشی باید بسیاری از جستجوگرهای دسترسی تصادفی را برای برطرف کردن نمایش داده‌ها انجام دهد، استفاده از درایوهای حالت جامد برای جلوگیری از گلوگاه شدن در جستجوهای دیسک، عملاً لازم است.

اینکه یک معماری لامبدا می‌تواند از هر نظر راه‌حل‌هایی با عملکرد بالاتر تولید کند، ضمن اینکه از پیچیدگی‌هایی که معماری‌های افزایشی را به‌طور کامل در پی دارد، جلوگیری می‌کند، نشان می‌دهد که چیزی

بسیار اساسی در جریان است. نکته اصلی فرار از مجموعه محاسبات کاملاً افزایشی و پذیرش تکنیک‌های مختلف است. حال ببینیم چگونه این کار را انجام دهیم.

۷–۱. معماری لامبدا

محاسبه توابع دلخواه در یک مجموعه داده دلخواه در زمان واقعی یک مشکل هولناک است. هیچ ابزار واحدی وجود ندارد که یک راه‌حل کامل ارائه دهد. در عوض، باید از ابزارها و تکنیک‌های متنوعی برای ساختن یک سیستم کامل کلان داده استفاده کنید.

ایده اصلی معماری لامبدا ساخت سیستم‌های کلان داده به‌عنوان یک سری لایه‌ها، همان‌طور که در شکل ۶-۱ نشان داده‌شده است، می‌باشد. هر لایه زیرمجموعه‌ای از خواص را برآورده می‌کند و بر عملکرد ارائه‌شده توسط لایه‌های زیر آن ساخته می‌شود. کل کتاب را صرف یادگیری نحوه طراحی، پیاده‌سازی و استقرار هر لایه می‌کنید، اما ایده‌های سطح بالا در مورد چگونگی قرار گرفتن کل سیستم با یکدیگر، قابل‌درک هستند.

همه‌چیز از معادله پرس‌وجو = تابع (کلیه داده‌ها) شروع می‌شود. در حالت ایده آل، برای به دست آوردن نتایج، توابع را در حالت پرواز انجام می‌دهید. متأسفانه، حتی اگر این امکان‌پذیر بود، این کار منابع عظیم و هزینه بسیار زیادی را نیاز دارد. تصور کنید که هر بار که می‌خواهید به سؤال از مکان فعلی شخصی پاسخ دهید، می‌توانید یک مجموعه داده پتابایت بخوانید.

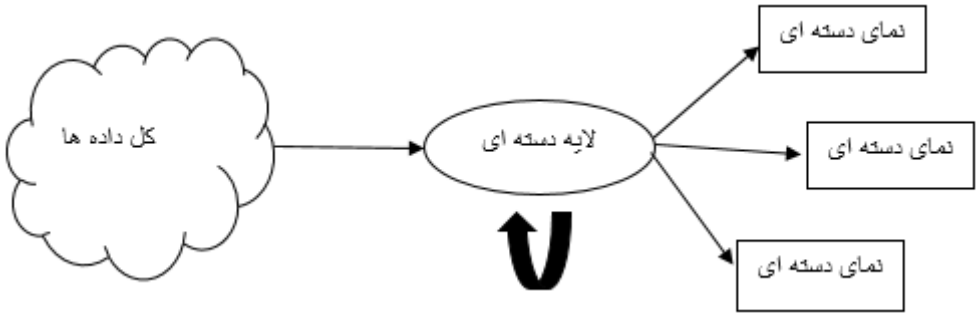
شکل ۶-۱. معماری لامبدا.

بدیهی‌ترین روش جایگزین، پیش‌پردازش عملکرد پرس‌وجو است. بیایید عملکرد جستجوی پیش‌نمایش را نمایش دسته‌ای بنامیم. به‌جای محاسبه پرس‌وجو در حالت پرواز، نتایج را از نمای پیش‌نمایش می‌خوانید. نمای پیش‌نمایش شده ایندکس می‌شود که می‌توان با خواندن تصادفی به آن دسترسی پیدا کرد. این سیستم مانند این است:

```
batch view = function(all data)
   query = function(batch view)
```

در این سیستم، برای به دست آوردن نمای دسته‌ای، عملکردی را روی تمام داده‌ها اجرا می‌کنید. سپس، وقتی می‌خواهید مقدار یک پرس‌وجو را بدانید، عملکردی را در آن نمای دسته‌ای اجرا می‌کنید. نمای دسته‌ای این امکان را فراهم می‌آورد که مقادیر موردنیاز خود را خیلی سریع و بدون نیاز به اسکن کردن همه‌چیزهای موجود در آن، دریافت کنید.

ازآنجاکه این بحث تا حدی انتزاعی است، بگذارید آن را با یک مثال بیان کنیم. فرض کنید در حال ساختن یک برنامه تجزیه‌وتحلیل وب هستید (دوباره)، و می‌خواهید در هر محدوده روز تعداد بازدیدهای صفحه را برای یک URL جستجو کنید. اگر در حال محاسبه پرس‌وجو به‌عنوان تابعی از تمام داده‌ها بودید، مجموعه داده را برای مشاهده صفحه در آن URL در آن بازه زمانی اسکن می‌کنید، و تعداد آن نتایج را برمی‌گردانید. در عوض، رویکرد نمای دسته‌ای، عملکردی را در کلیه صفحات اجرا می‌کند تا یک فهرست را از کلید [url، day] به شمارش تعداد بازدیدهای صفحه برای آن URL برای آن روز، بکشاند. سپس، برای حل پرس‌وجو، تمام مقادیر را از آن نمایه برای تمام روزها در آن بازه زمانی بازیابی می‌کنید، و برای به دست آوردن نتیجه، شمارش‌ها را جمع می‌کنید. این روش در شکل ۷-۱ نشان داده‌شده است.



شکل ۷-۱. معماری لایه دسته‌ای.

مشخص است که، چیزی که تاکنون توضیح داده‌شده است از این روش وجود ندارد. ایجاد نمای دسته‌ای به‌وضوح عملیاتی با تأخیر بالا خواهد بود، زیرا عملکردی را روی تمام داده‌ها اجرا می‌کند. تا زمان اتمام، داده‌های جدید زیادی جمع‌آوری می‌شوند که در نمای دسته مشاهده نمی‌شوند، و بسیاری از سؤالات ساعت‌ها به‌روز نمی‌شوند. اما بیایید فعلاً این مسئله را نادیده بگیریم، زیرا قادر خواهیم بود آن را برطرف کنیم. بیایید وانمود کنیم که درست نیست که نمایش داده تا چند ساعت دیگر به‌روزرسانی شده نباشد و با پیاده‌سازی یک عملکرد روی مجموعه داده کامل، به بررسی این ایده از پیش‌پردازش نمای دسته‌ای بپردازید.