ویرایش دوم

مصاحبه

طراحی سیستم‌های نرم‌افزاری ۱

نویسنده **الکس ژو**

ترجمه

دانیال خسروی
جواد جعفری


بسم الله الرحمن الرحیم

مصاحبه طراحی سیستم نرم‌افزاری ۱

SYSTEM DESIGN INTERVIEW 1

تهیه و تألیف: الکس ژو

ترجمه:

دانیال خسروی

جواد جعفری

سرشناسه	:	شو، الکس Xu, Alex
عنوان و نام پدیدآور	:	مصاحبه طراحی سیستم نرم‌افزاری۱/ تهیه و تالیف الکس ژو؛ ترجمه دانیال خسروی، جواد جعفری.
مشخصات نشر	:	تهران: گنجور، ۱۴۰۳.
مشخصات ظاهری	:	۳۷۹ ص.
شابک	:	۴۰۰۰۰۰۰ ریال: 978-622-302-684-3
وضعیت فهرست نویسی	:	فیپا
یادداشت	:	عنوان اصلی: System Design Interview : An Insider's Guide, 2nd ed.
موضوع	:	طراحی سیستم System design
شناسه افزوده	:	خسروی، دانیال، ۱۳۷۰-، مترجم
شناسه افزوده	:	جعفری، جواد، ۱۳۷۴-، مترجم
رده بندی کنگره	:	۹/QA۷۶
رده بندی دیویی	:	۲۱/۰۰۴
شماره کتابشناسی ملی	:	۹۶۰۷۵۲۳
اطلاعات رکورد کتابشناسی	:	فیپا

نشر گنجور

• آدرس: تهران. خیابان انقلاب، خیابان فخر رازی، خیابان وحید نظری، پلاک ۹۲ طبقه اول •

• تلفن: ۰۹۹۰۱۹۱۰۲۰۸ - ۰۹۱۲۰۶۱۷۲۸۳ - ۰۲۱۶۶۴۹۱۰۵۶•

• سایت انتشارات: Www.ganjoorpub.ir •

● مصاحبه طراحی سیستم نرم‌افزاری ۱ ●

• تهیه و تألیف: الکس ژو •

• ترجمه: دانیال خسروی، جواد جعفری •

•طراح جلد: پویا خادمی •

• ناشر: گنجور •

• نوبت چاپ: اول- ۱۴۰۳ •

• شمارگان: ۱۰۰•

• شابک: ۳-۶۸۴-۳۰۲-۶۲۲-۹۷۸•

• قیمت: ۴۰۰۰۰۰ تومان •

تقدیم به اولین آموزگاران زندگی من.

پدر و مادرم

(دانیال)

به‌پاس محبت‌های بی‌دریغشان که هرگز فروکش نمی‌کند.

تقدیم به پدر و مادر عزیزم

(جواد)

سخن مترجم

تا جهان بود از سر آدم فراز

کس نبود از راز دانش بی‌نیاز

مردمانِ بخرد اندر هر زمان

راهِ دانش را به هر گونه زبان

گِرد کردند و گرامی داشتند

تا به سنگ اندر همی بنگاشتند

دانش اندر دل چراغ روشنست

وز همه بد، بر تن تو، جوشنست

(رودکی)

این کتاب به هدف یادگیری و نشر علم به زبان فارسی ترجمه شده است و همین‌طور ترجمه بعضی از اصطلاحات و واژه‌های فنی در علوم کامپیوتر به زبان فارسی با مشکلاتی همراه بوده که انتظار می‌رود به‌صورت تدریجی واژه‌های مناسب توسط افرادی که در این علوم اطلاعات و دانش مبادله می‌کنند، این واژگان جاافتاده و ماندگار شود.

توصیه می‌شود که کتاب به ترتیب فصل‌ها خوانده شود هر چند می‌توان هر فصل را به‌صورت جداگانه مطالعه کرد که در این حالت توصیه می‌کنیم که ابتدا فصل اول به‌خاطر وجود مفاهیم و واژگان فنی بسیار مهم آن مطالعه شود و سپس به سراغ فصل موردنظر خود رفته و آن را مطالعه کنید.

ما بسیار تلاش کردیم تا متن ترجمه شده بدون خطا و اشتباه باشد؛ اما راه‌یافتن خطا در کتاب گریزناپذیر است. مترجمین بسیار ممنون خواهند شد که این اشتباهات را توسط ایمیل به دانیال (d.kh@mail.com) یا جواد (javadsdn@outlook.com) گوشزد کنید.

همینطور از پویا خادمی (pooya.khademi01@gmail.com) برای طراحی گرافیکی جلدِ کتاب و از ابراهیم بیاگوی (ebrahim@gnu.org) برای مشاوره در امور راهبردی مباحث مهندسی نرم‌افزار مورد بحث در این کتاب و پیشنهادهای اصلاحات نگارشی، تشکر می‌کنیم. همینطور از انتشارات گنجور نهایت سپاس و قدردانی را داریم.

مقدمه

خوشحالیم که تصمیم گرفتید با ما در یادگیری مصاحبه‌های طراحی سیستم همراه شوید. سؤالات مصاحبه طراحی سیستم، دشوارترین نوع در میان تمام مصاحبه‌های فنی هستند. در این سؤالات از مصاحبه‌شونده خواسته می‌شود تا معماری یک سیستم نرم‌افزاری مانند فید خبری، جستجوی گوگل، سیستم چت و غیره را طراحی کند. این سؤالات می‌توانند دلهره‌آور باشند و هیچ الگوی خاصی برای پاسخگویی به آن‌ها وجود ندارد. معمولاً این سؤالات بسیار گسترده و مبهم هستند. همچنین فرایند پاسخگویی به آن‌ها باز بوده و هیچ پاسخ استاندارد

یا درستی وجود ندارد. شرکت‌ها به طور گسترده از مصاحبه‌های طراحی سیستم استفاده می‌کنند؛ زیرا مهارت‌های ارتباطی و حل مسئله‌ای که در این مصاحبه‌ها ارزیابی می‌شوند، مشابه مهارت‌های موردنیاز برای کار روزانه یک مهندس نرم‌افزار هستند.

در این نوع مصاحبه، فرد بر اساس نحوه‌ی تحلیل یک مشکل مبهم و حل گام‌به‌گام آن مورد ارزیابی قرار می‌گیرد. علاوه بر این، توانایی توضیح ایده، بحث با دیگران و ارزیابی و بهینه‌سازی سیستم نیز سنجیده می‌شود. سؤالات طراحی سیستم دارای پایان باز هستند. درست همانند دنیای واقعی، سیستم‌ها می‌توانند تفاوت‌ها و تغییرات زیادی داشته باشند. هدف نهایی این است که به معماری‌ای دست پیدا کنید که به اهداف طراحی سیستم برسد. مسیر بحث می‌تواند بسته به مصاحبه‌کننده متفاوت باشد. برخی از مصاحبه‌کنندگان ممکن است ترجیح دهند در سطح بالایی به معماری کلی سیستم بپردازند، درحالی‌که برخی دیگر ممکن است روی یک یا چند ناحیه خاص تمرکز کنند. به طور معمول، درک صحیح الزامات سیستم، محدودیت‌ها و گلوگاه‌ها برای هدایت مصاحبه‌کننده و مصاحبه‌شونده ضروری است. هدف این کتاب ارائه یک استراتژی قابل‌اعتماد برای مواجهه با سؤالات طراحی سیستم است. استراتژی و دانش درست، برای موفقیت در مصاحبه حیاتی هستند. این کتاب دانش جامعی در مورد ساخت یک سیستم مقیاس‌پذیر در اختیار شما قرار می‌دهد. هر چه اطلاعات بیشتری از خواندن این کتاب به دست آورید، برای حل سؤالات طراحی سیستم مجهزتر خواهید شد. این کتاب همچنین یک چارچوب گام‌به‌گام برای نحوه‌ی برخورد با سؤالات طراحی سیستم ارائه می‌دهد. این کتاب با ارائه‌ی مثال‌های متعدد، رویکرد سیستمی را با مراحل دقیق و قابل‌اجرا برای شما شرح می‌دهد. با تمرین مداوم، برای رویارویی با سؤالات مصاحبه طراحی سیستم کاملاً آماده خواهید شد.

فهرست مطالب

فصل ۱۴ طراحی یوتیوب ۳۱۳

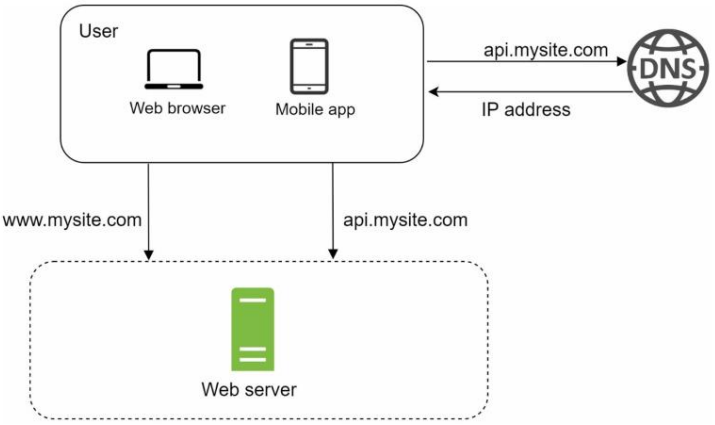
فصل ۱

مقیاس‌دهی از صفر تا میلیون‌ها کاربر

طراحی سیستمی که میلیون‌ها کاربر را پشتیبانی می‌کند یک پدیده بسیار چالش‌برانگیز است و سفری است که نیاز به اصلاح مداوم و بهبود بی‌پایان دارد. در این فصل، ما سیستمی می‌سازیم که از یک کاربر پشتیبانی می‌کند و به‌تدریج آن را برای خدمت به میلیون‌ها کاربر افزایش می‌دهیم. پس از خواندن این فصل، به تعدادی تکنیک تسلط خواهید داشت که به شما کمک می‌کند تا سؤالات مصاحبه طراحی سیستم را بررسی کنید.

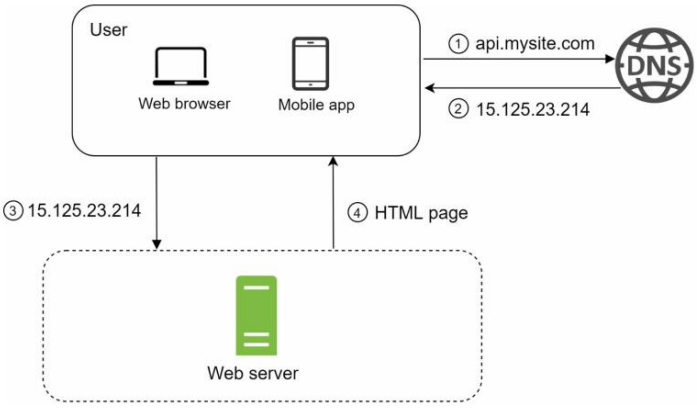
تنظیمات یک سرور

یک سفر هزار کیلومتری با یک قدم آغاز می‌شود و ساختن یک سیستم پیچیده سختی خاصی ندارد. در قدم اول با یک چیز ساده شروع می‌کنیم و همه‌چیز روی یک سرور اجرا می‌شود. شکل ۱-۱ تصویر راه‌اندازی یک سرور را نشان می‌دهد که در آن همه‌چیز روی یک سرور اجرا می‌شود: برنامه مبتنی بر وب، پایگاه‌داده، حافظه موقت/کَش و سایر موارد است.



تصویر ۱-۱

برای درک این تنظیمات، بررسی مسیر درخواست[1] و منبع ترافیک مفید است. اجازه دهید ابتدا به مسیر درخواست نگاه کنیم (شکل ۲-۱)



تصویر ۲-۱

1 request

۱. کاربران از طریق نام‌های دامنه[1] مانند api.mysite.com به وب‌سایت‌ها دسترسی دارند. معمولاً سیستم نام دامنه Domain Name System یا به‌اختصار (DNS) یک سرویس پولی است که توسط اشخاص ثالث ارائه می‌شود و توسط سرورهای ما میزبانی نمی‌شود.

۲. آدرس پروتکل اینترنت (IP) به مرورگر یا برنامه تلفن همراه بازگردانده می‌شود. در این مثال، آدرس IP برابر 15.125.23.214 برگردانده شده است.

۳. هنگامی که آدرس IP به دست آمد، درخواست‌های پروتکل انتقال ابرمتن (HTTP)[2] [۱] مستقیماً به سرور وب شما ارسال می‌شود.

۴. وب سرور صفحات HTML یا پاسخ JSON را برای پردازش بر می‌گرداند.

در مرحله بعد، اجازه دهید منابع ترافیکی را بررسی کنیم. ترافیک وب سرور شما از دو منبع می‌آید: برنامه وب و برنامه موبایل.

- **برنامه وب**: از ترکیبی از زبان‌های سمت سرور (جاوا، پایتون و گو و موارد دیگر) برای مدیریت منطق تجاری[3] و ذخیره‌سازی و سایر موارد دیگر استفاده می‌شود و از زبان‌های سمت سرویس‌گیرنده (HTML و JavaScript) برای نمایش[4] استفاده می‌کند.
- **برنامه موبایل**: پروتکل HTTP پروتکل ارتباطی بین برنامه موبایل و وب سرور است. JavaScript Object Notation یا به‌اختصار (JSON) معمولاً به دلیل سادگی فرمت آن جهت پاسخ API برای انتقال داده استفاده می‌شود. نمونه‌ای از پاسخ API در قالب JSON در زیر نشان‌داده‌شده است:

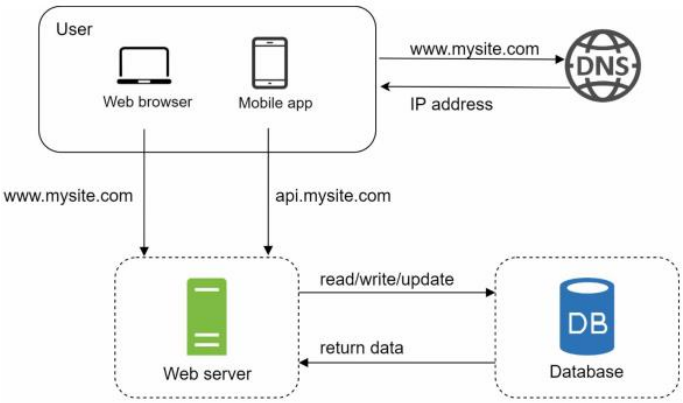
1 domain names
2 Hypertext Transfer Protocol
3 business logic
4 presentation

GET /users/12 – Retrieve user object for id = 12

```
{
    "id": 12,
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]
}
```

بررسی پایگاه‌داده

با رشد کاربرها دیگر استفاده از یک سرور به‌تنهایی کافی نیست و ما به چندین سرور نیاز داریم: یکی برای ترافیک وب/موبایل، دیگری برای پایگاه‌داده (شکل ۳-۱). جداسازی سرورهای وب/ تلفن همراه (لایه وب) و پایگاه‌داده (لایه داده) به آنها اجازه می‌دهد تا به طور مستقل مقیاس[1] شوند.



تصویر ۳-۱

1 scale

کدام پایگاه‌داده مناسب است؟

شما می‌توانید بین یک پایگاه‌داده سنتی رابطه‌ای و یک پایگاه‌داده غیررابطه‌ای یکی را انتخاب کنید. بیایید تفاوت‌های آنها را بررسی کنیم. پایگاه‌داده‌های رابطه‌ای[1] با به طور خلاصه (RDBMS)که پایگاه‌داده SQL نیز نامیده می‌شوند. محبوب‌ترین آنها پایگاه‌داده MySQL و Oracle و PostgreSQL هستند. پایگاه‌داده‌های رابطه‌ای داده‌ها را در جدول‌ها و ردیف‌ها نمایش و ذخیره می‌کنند. شما می‌توانید عملیات پیوستن یا Join را با استفاده از SQL در جدول‌های مختلف پایگاه‌داده انجام دهید.

پایگاه‌داده‌های غیررابطه‌ای نیز پایگاه‌داده NoSQL نامیده می‌شوند. محبوب‌ترین آنها CouchDB، Neo4j Cassandra، HBase، Amazon DynamoDB هستند[۲]. این پایگاه‌داده‌ها به چهار دسته تقسیم می‌شوند: ذخیره‌سازهای کلید-مقدار[2]، ذخیره‌سازهای گراف[3]، ذخیره‌سازهای ستونی[4] و ذخیره‌سازهای اسناد[5]. عملیات پیوستن/ Join به طور کلی در پایگاه‌داده‌های غیررابطه‌ای پشتیبانی نمی‌شود.

برای اکثر توسعه‌دهندگان، پایگاه‌داده‌های رابطه‌ای بهترین گزینه هستند، زیرا بیش از ۴۰ سال است که وجود دارند و از نظر سابقه تاریخی به‌خوبی کار می‌کنند. بااین‌حال، اگر پایگاه‌های داده رابطه‌ای برای موارد استفاده خاص شما مناسب نیستند، کاوش و کنجکاوی فراتر از پایگاه‌های داده رابطه‌ای ضروری است و احتمالا پایگاه‌داده‌های غیررابطه‌ای ممکن است انتخاب مناسبی باشند اگر:

- برنامه شما به تأخیر بسیار کم‌نیاز دارد.
- داده‌های شما بدون ساختار هستند یا هیچ داده رابطه‌ای ندارید.
- فقط باید داده‌ها (JSON، XML، YAML و غیره) را سریالی[6] و غیر سریالی[7] کنید.
- شما باید حجم عظیمی از داده‌ها را ذخیره کنید.

---

1 relational database management system
2 key-value
3 graph stores
4 column stores
5 document stores
6 serialize
7 deserialize

همین‌طور در این کتاب از مفهوم «**کوئری**» زدن که به معنای ارسال یا پرسیدن یک سؤال یا درخواست مشخص است، زیاد استفاده شده است. این عبارت معمولاً در زمینه‌های مختلف مانند مطالعه داده‌ها، جستجو در پایگاه‌های داده، یادگیری ماشین، وب و برنامه‌نویسی استفاده می‌شود. در کل، کوئری زدن به معنای درخواست اطلاعات یا دستورات یا پرس‌وجو از یک سیستم یا پایگاه‌داده است.

مقیاس عمودی در مقابل مقیاس افقی

مقیاس عمودی[۱] که به آن "*scale up*" نیز می‌گویند، به معنای فرایند اضافه‌کردن قدرت بیشتر (CPU، RAM و غیره) به سرورهای شما است. مقیاس افقی[۲] که به آن "*scale-out*" هم می‌گویند، به شما امکان می‌دهد با افزودن سرورهای بیشتر به منابع خود آن‌ها را مقیاس[۳] کنید. هنگامی که ترافیک کم است، مقیاس عمودی یک گزینه عالی است و سادگی مقیاس عمودی مزیت اصلی آن است که متأسفانه با محدودیت‌های جدی همراه است.

- مقیاس‌دهی عمودی محدودیت سختی دارد. اضافه‌کردن CPU و حافظه نامحدود به یک سرور غیرممکن است.
- مقیاس‌دهی عمودی گزینه‌هایی برای failover[۴] و redundancy[۵] ندارد. اگر یک سرور از کار بیفتد، وب‌سایت/برنامه به طور کامل با آن از کار می‌افتد.

---

1 Vertical scaling
2 Horizontal scaling
3 scale

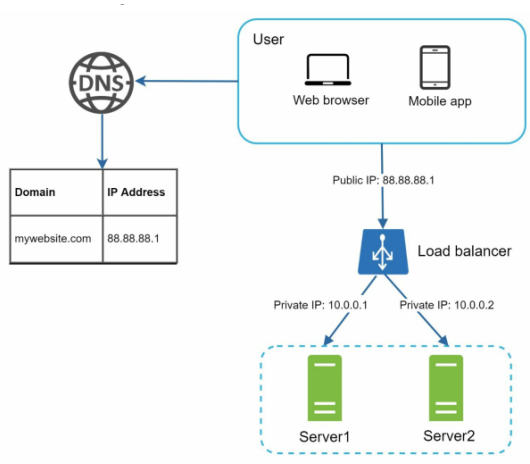
۴ Failover در علوم کامپیوتر به توانایی تغییر خودکار و یکپارچه به یک سیستم پشتیبان قابل اعتماد هنگام بروز مشکل یا اختلال اشاره دارد. وقتی یک مؤلفه یا سیستم اصلی خراب می‌شود، یک سیستم آماده به کار یا در حالت افزونگی باید Failover را محقق کرده و تأثیر منفی روی کاربران را کاهش داده یا حذف کند

۵ Redundancy در لغت به معنی افزونگی و تکرار است و در اینجا به هر نوع داده‌ای که در چند جدول مختلف نگهداری می‌شود و اطلاعات یکسانی را تکرار می‌کند، اشاره دارد. مثلاً اگر یک کاربر را با چند کد مختلف در سیستم تعریف کنیم، این افزونگی و تکرار اطلاعات را می‌توان با استفاده از کلید اصلی (Primary Key) جلوگیری کرد. به عنوان مثال، کد اقتصادی مشتری حقوقی یا کد ملی مشتری حقیقی می‌تواند از تکرار اطلاعات جلوگیری کند.

مقیاس‌دهی افقی برای کاربردهای در مقیاس بزرگ به دلیل محدودیت‌های مقیاس‌دهی عمودی مطلوب‌تر است. در طراحی قبلی، کاربران مستقیماً به وب سرور متصل می‌شوند. اگر وب سرور آفلاین باشد، کاربران نمی‌توانند به وب‌سایت دسترسی پیدا کنند. در سناریویی دیگر، اگر تعداد زیادی از کاربران به طور همزمان به وب سرور دسترسی داشته باشند و به حد مجاز بار روی وب سرور برسد در نتیجه کاربران معمولاً پاسخ آهسته‌تری را تجربه می‌کنند یا به سرور متصل نمی‌شوند. متعادل‌کننده بار[1] بهترین تکنیک برای رفع این مشکلات است.

متعادل‌کننده بار (Load balancer)

یک متعادل‌کننده یا توزیع‌کننده بار، ترافیک ورودی را بین وب سرورهایی که در یک مجموعه Load balancer تعریف شده‌اند را به طور مساوی توزیع می‌کند. شکل ۴-۱ نحوه عملکرد یک متعادل‌کننده بار را نشان می‌دهد.



تصویر ۴-۱

همان‌طور که در شکل ۴-۱ نشان‌داده‌شده است، کاربران مستقیماً به IP عمومی یا public مربوط به load balancer متصل می‌شوند. با این تنظیمات، وب سرورها دیگر به‌صورت مستقیم

1 Load balancer

توسط کاربران، غیرقابل دسترسی هستند. برای امنیت بهتر، از IPهای خصوصی یا در اصطلاح private برای ارتباط بین سرورها استفاده می‌شود. IP خصوصی که در واقع یک آدرس IP خاص است که فقط بین سرورهای یک شبکه داخلی قابل دسترسی است. بااین‌حال، از طریق اینترنت قابل دسترسی نیست. load balancer با وب سرورها از طریق IPهای خصوصی ارتباط برقرار می‌کند. در شکل ۴-۱، پس از اضافه شدن یک متعادل‌کننده بار و یک وب سرور دیگر، ما با موفقیت و بدون هیچ تلاش خاصی، مشکل را حل کردیم و سطح دسترس بودن[1] لایه وب را بهبود دادیم. جزئیات بیشتر در زیر توضیح داده شده است:

- اگر سرور ۱ آفلاین شود، تمام ترافیک به سرور ۲ هدایت می‌شود. این امر از آفلاین شدن وب‌سایت جلوگیری می‌کند. همچنین یک وب سرور سالم و جدید به منابع سرورهای خود اضافه می‌کنیم تا توزیع بار را متعادل کنیم.
- اگر ترافیک وب‌سایت به‌سرعت رشد کند و دو سرور برای مدیریت ترافیک کافی نباشد، متعادل‌کننده بار می‌تواند این مشکل را به‌خوبی حل کند. شما فقط باید سرورهای بیشتری را به منابع سروری یا در اصطلاح server pool وب اضافه کنید و در نهایت متعادل‌کننده بار به طور خودکار شروع به ارسال درخواست برای آنها می‌کند.

اکنون لایه وب، خوب به نظر می‌رسد، در مورد لایه داده اوضاع چطور است؟ طرح فعلی دارای یک پایگاه‌داده است، بنابراین از شکست[2] و افزونگی[3] پشتیبانی می‌کند. تکثیر[4] پایگاه‌داده یک تکنیک رایج برای رفع این مشکلات است. اجازه دهید نگاهی بیندازیم.

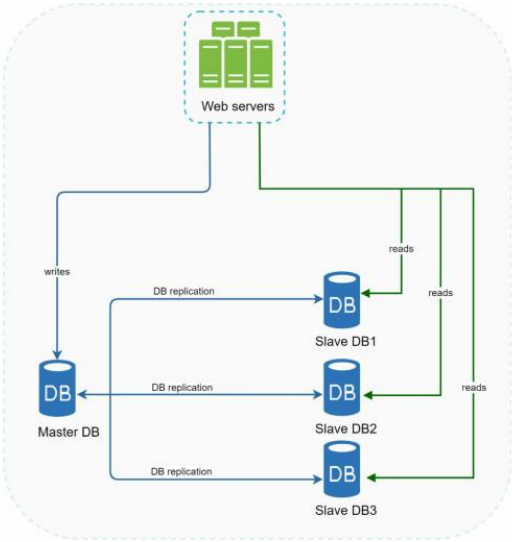
تکثیر پایگاه‌داده (Database replication)

به نقل از ویکی‌پدیا: «تکثیر یا replication پایگاه‌داده را می‌توان در بسیاری از سیستم‌های مدیریت پایگاه‌داده، معمولاً با یک رابطه master/slave بین اصلی (master) و کپی‌ها (slave) استفاده کرد» [۳].

---

1 availability
2 failover
3 redundancy
4 replication

یک پایگاه‌داده اصلی (master) معمولاً فقط از عملیات نوشتن پشتیبانی می‌کند. یک پایگاه‌داده slave کپی‌هایی از داده‌ها را از پایگاه‌داده اصلی دریافت می‌کند و فقط از عملیات خواندن پشتیبانی می‌کند. تمام دستورات تغییر داده؛ مانند درج، حذف یا به‌روزرسانی باید به پایگاه‌داده اصلی ارسال شوند. اکثر برنامه‌ها به نسبت خواندن به نوشتن بسیار بالاتری نیاز دارند؛ بنابراین، تعداد پایگاه‌داده‌های slave در یک سیستم معمولاً بیشتر از تعداد پایگاه‌های داده اصلی یا master است. شکل ۵-۱ یک پایگاه‌داده master با پایگاه‌داده‌های متعدد slave را نشان می‌دهد.



تصویر ۵-۱

مزایای تکثیر یا replication پایگاه‌داده به‌صورت زیر است:

- **عملکرد بهتر**: در مدل master-slave، تمام نوشتن‌ها و به‌روزرسانی‌ها در گره‌های اصلی (master) اتفاق می‌افتد. درحالی‌که، عملیات خواندن در سراسر گره‌های slave توزیع می‌شود. این مدل کارایی را بهبود می‌بخشد؛ زیرا امکان پردازش کوئری‌های بیشتری را به‌صورت موازی فراهم می‌کند.

- **قابلیت اطمینان**[1]: اگر یکی از سرورهای پایگاه‌داده شما در اثر بلایای طبیعی، مانند طوفان یا زلزله از بین برود، داده‌ها همچنان حفظ می‌شوند. لازم نیست نگران ازدست‌دادن داده‌ها باشید؛ زیرا داده‌ها در چندین مکان تکثیر می‌شوند.
- **دسترسی بالا**[2]: با تکثیر داده‌ها در مکان‌های مختلف، وب‌سایت شما فعال باقی می‌ماند حتی اگر پایگاه‌داده آفلاین باشد زیرا می‌توانید به داده‌های ذخیره شده در سرور پایگاه‌داده دیگری دسترسی داشته باشید.

در بخش قبل، در مورد اینکه چگونه یک متعادل‌کننده بار به بهبود دردسترس‌بودن سیستم کمک می‌کند، بحث کردیم. ما در اینجا همین سؤال را می‌پرسیم؛ اگر یکی از پایگاه‌های داده آفلاین شود چه اتفاقی می‌افتد؟ طرح معماری موردبحث در شکل ۵-۱ می‌تواند این مورد را مدیریت کند:

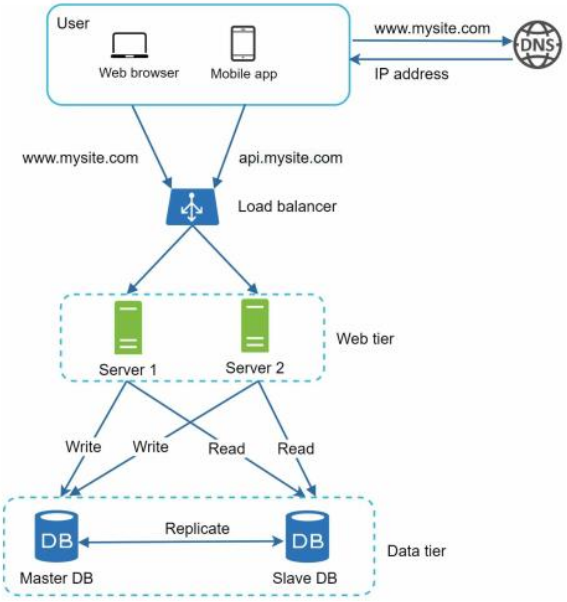
- اگر فقط یک پایگاه‌داده slave در دسترس باشد و آفلاین شود، عملیات خواندن به طور موقت به پایگاه‌داده اصلی هدایت می‌شود. به‌محض اینکه مشکل پیدا شد، یک پایگاه‌داده slave جدید جایگزین پایگاه‌داده قبلی می‌شود. درصورتی‌که چندین پایگاه‌داده slave در دسترس باشد، عملیات خواندن به دیگر پایگاه‌داده‌های slave سالم هدایت می‌شود. در نتیجه یک سرور با پایگاه‌داده جدید جایگزین سرور قبلی خواهد شد.
- اگر پایگاه‌داده اصلی (Master) آفلاین شود، یک پایگاه‌داده Slave به‌عنوان Master جدید ارتقا می‌یابد. تمام عملیات پایگاه‌داده به طور موقت در پایگاه‌داده اصلی جدید اجرا می‌شود. یک پایگاه‌داده slave جدید برای تکثیر داده‌ها بلافاصله جایگزین پایگاه‌داده قدیمی می‌شود. در سیستم‌های تولید، ترویج یک Master جدید پیچیده‌تر است، زیرا ممکن است داده‌های یک پایگاه‌داده Slave به‌روز نباشد. داده‌های ازدست‌رفته باید با اجرای اسکریپت‌های بازیابی اطلاعات به‌روز شوند. اگرچه برخی از روش‌های تکثیر دیگر مانند multi-masters و circular replication[3] می‌توانند کمک‌کننده باشد بااین‌حال این تنظیمات پیچیده‌تر هستند

1 Reliability
2 High availability

۳ تکثیر چرخشی

و بحث آنها از حوصله این کتاب خارج است. خوانندگان علاقه‌مند باید به منابع مرجع فهرست شده [۴] و [۵] مراجعه کنند.

شکل ۶-۱ طراحی سیستم را پس از افزودن متعادل‌کننده بار و تکثیر پایگاه‌داده نشان می‌دهد.



شکل ۶-۱

بیایید نگاهی به این طراحی بیندازیم:

- یک کاربر آدرس IP متعادل‌کننده بار را از DNS دریافت می‌کند.
- یک کاربر به متعادل/توزیع‌کننده بار با این آدرس IP متصل می‌شود.
- درخواست HTTP به سرور ۱ یا سرور ۲ هدایت می‌شود.
- یک وب سرور، داده‌های کاربر را از یک پایگاه‌داده slave می‌خواند.
- یک وب سرور هرگونه عملیات تغییر داده را به پایگاه‌داده اصلی هدایت می‌کند. این شامل عملیات نوشتن، به‌روزرسانی و حذف است.

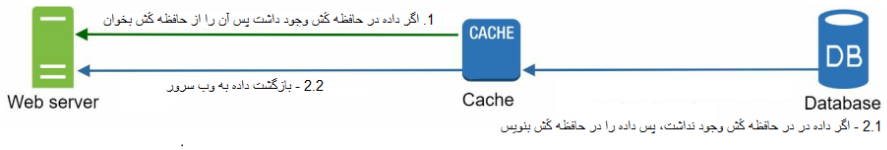
اکنون، شما درک کاملی از وب و لایه‌های داده مربوطه دارید، زمان آن رسیده است که زمان load/response را بهبود ببخشید. این را می‌توان با افزودن یک لایه کَش و جابه‌جایی محتوای استاتیک (جاوا اسکریپت/CSS/تصویر/فایل‌های ویدئو) به شبکه تحویل محتوا (CDN) انجام داد.

کَش (Cache)

کَش (Cache) یک فضای ذخیره‌سازی موقت است که نتیجه پاسخ‌های پر هزینه یا داده‌هایی که بسیار مورد ارجاع و استفاده قرار می‌گیرد را در حافظه موقت ذخیره می‌کند تا درخواست‌های بعدی سریع‌تر ارائه شوند. همان‌طور که در شکل ۶-۱ نشان‌داده‌شده است، هر بار که یک صفحه وب جدید بارگیری می‌شود، یک یا چند تماس پایگاه‌داده برای واکشی داده‌ها اجرا می‌شود. عملکرد برنامه به‌شدت با فراخوانی مکرر پایگاه‌داده تحت‌تأثیر قرار می‌گیرد. حافظه کَش می‌تواند این مشکل را کاهش دهد.

لایه کَش (Cache tier)

لایه کَش[1] یک لایه ذخیره موقت داده است که بسیار سریع‌تر از پایگاه‌داده است. از مزایای داشتن یک ردیف کَش جداگانه می‌توان به عملکرد بهتر سیستم، توانایی کاهش بار/ حجم‌کاری پایگاه‌داده و توانایی مقیاس‌دهی لایه کَش به طور مستقل اشاره کرد. شکل ۷-۱ راه‌اندازی احتمالی یک سرور کَش را نشان می‌دهد:



شکل ۷-۱

1 Cache

پس از دریافت درخواست[1]، یک وب سرور ابتدا بررسی می‌کند که آیا کَش پاسخِ موردنظر را دارد یا خیر. اگر پاسخ را داشته باشد، پس داده‌ها را به کلاینت بر می‌گرداند. در غیر این صورت، کوئری موردنظر را در پایگاه‌داده اجرا می‌کند و پاسخ آن را در حافظه کَش ذخیره می‌کند و آن را برای کلاینت ارسال می‌کند. به این استراتژی کَش که به آن کش مستقیم خواندنی[2] گفته می‌شود. سایر استراتژی‌های کَش بسته به نوع داده، اندازه و الگوهای موردنظر در دسترس هستند. منبع [۶] یک مطالعه قبلی نحوه عملکرد استراتژی‌های مختلف ذخیره‌سازی را توضیح می‌دهد. تعامل با سرورهای کش ساده است زیرا اکثر سرورهای کَش، APIهایی را برای زبان‌های برنامه‌نویسی رایج ارائه می‌دهند. قطعه کد زیر APIهای یک نرم‌افزار مربوط به کَش به نام Memcached را نشان می‌دهد:

```
SECONDS = 1
cache.set('myKey', 'hi there', 3600 * SECONDS)
cache.get('myKey')
```

توصیه‌هایی برای استفاده مناسب از کَش

در اینجا چند نکته برای استفاده از سیستم کش وجود دارد:

تصمیم بگیرید که چه زمانی از کَش استفاده کنید. به‌عنوان‌مثال؛ زمانی که داده‌ها اغلب خوانده می‌شوند؛ اما به‌ندرت تغییر می‌کنند از حافظه کَش استفاده کنید. از آنجایی که داده‌های کش در حافظه فرار[3] یا غیرماندگار (مثل حافظه RAM) ذخیره می‌شوند، یک سرور خاص کَش برای پایداری داده‌ها[4] به‌هیچ‌وجه ایده‌آل نیست. به‌عنوان‌مثال، اگر یک سرور کَش راه‌اندازی مجدد شود، تمام داده‌های موجود در حافظه از بین می‌رود؛ بنابراین، داده‌های مهم باید در ذخیره‌سازهای داده‌ی دائمی ذخیره شوند.

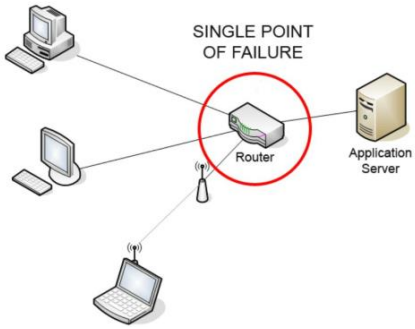
---

1 request
2 read-through cache
3 volatile
۴ در علوم رایانه، پایداری داده‌ها (persisting data) به معنای حفظ و ذخیره‌سازی داده‌ها در طول زمان است. این اصطلاح به معنای تضمین کنترل دسترسی، حفظ اصالت و دسترسی به داده‌ها در طول مدت زمان مشخص می‌شود.

- **سیاست‌های منقضی شدن**[۱]. اجرای سیاست انقضا یک روش مناسب است. زمانی که داده‌های کش منقضی می‌شوند، از حافظه کش حذف می‌شوند. وقتی خط‌مشی انقضا وجود نداشته باشد، داده‌های کش به طور دائم در حافظه ذخیره می‌شوند. توصیه می‌شود تاریخ انقضا را خیلی کوتاه نکنید، زیرا باعث می‌شود سیستم به طور مکرر داده‌ها را از پایگاه‌داده بارگیری مجدد کند. در همین حال، توصیه می‌شود تاریخ انقضا را بیش از حد طولانی نکنید؛ زیرا ممکن است داده‌ها بی‌مصرف و فرسوده شوند.
- **یکپارچگی**[۲]: این شامل همگام[۳] نگه‌داشتن ذخیره‌ساز داده[۴] و حافظه کَش است. در حالتی غیریکپارچه[۵] ممکن است حالتی اتفاق بیافتد که عملیات تغییر داده در ذخیره‌سازی داده و حافظه کش در یک تراکنش واحد نباشند. هنگام مقیاس‌دهی[۶] در چندین منطقه، حفظ یکپارچگی و سازگاری بین ذخیره‌گاه داده[۷] و حافظه کش چالش‌برانگیز است. برای جزئیات بیشتر در این مورد، به مقاله‌ای با عنوان "*Scaling Memcache at Facebook*" منتشر شده از فیس‌بوک مراجعه کنید [۷].
- **کاهش خرابی‌ها**: یک سرور کَش به‌تنهایی نشان‌دهنده یک نقطه بالقوه شکست‌پذیر[۸] که به‌اختصار (SPOF) نامیده می‌شود، است. این اصطلاح در ویکی‌پدیا به‌صورت زیر تعریف شده است: «یک نقطه شکست واحد (SPOF) بخشی از یک سیستم است که اگر خراب شود، کل آن را متوقف شده و از کار می‌افتد» [۸]. در نتیجه، چندین سرور کش در مراکز داده[۹] مختلف برای جلوگیری از وقوع SPOF توصیه می‌شود. یکی دیگر از روش‌های توصیه شده، تأمین بیش از حد حافظه موردنیاز با درصدهای معین است. پس با افزایش استفاده از حافظه، یک بافر در نظر گرفته می‌شود.

---

1 Expiration policy
2 Consistency
3 sync
4 data store
5 Inconsistency
6 Scaling
7 data store
8 single point of failure
9 data centers



شکل ۸-۱

- **خط‌مشی تخلیه**[1]: پس از پر شدن حافظه کَش، هرگونه درخواست برای افزودن موارد دیگر به حافظه کَش ممکن است باعث حذف موارد موجود شود. به این عمل تخلیه کَش می‌گویند. کمترین استفاده اخیر[2] در اختصار (LRU) محبوب‌ترین خط‌مشی تخلیه حافظه کش است. سایر خط‌مشی‌های تخلیه، مانند کمترین تناوب استفاده[3] در اختصار (LFU) یا FIFO[4] می‌توانند برای ارضای موارد استفاده مختلف در نظر رفته شوند.

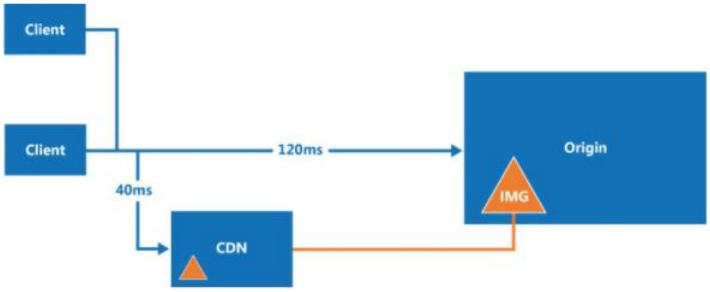
شبکه توزیع محتوی – Content delivery network (CDN)

در واقع CDN شبکه‌ای از سرورهای پراکنده جغرافیایی است که برای ارائه محتوای ثابت[5] استفاده می‌شود. سرورهای CDN محتوای ثابت مانند تصاویر، ویدئوها، CSS، فایل‌های جاوا اسکریپت و غیره را در حافظه کَش ذخیره می‌کنند. ذخیره محتوای پویا[6] یک مفهوم نسبتاً جدید و فراتر از محدوده این کتاب است. این امکان ذخیره صفحات HTML را که بر اساس مسیرهای درخواست، رشته‌های کوئری، کوکی‌ها و هدرهای درخواست هستند را فراهم می‌کند. برای اطلاعات بیشتر در این مورد به مقاله ذکر شده در مرجع [۹] مراجعه کنید. این کتاب بر نحوه استفاده از CDN برای ذخیره محتوای استاتیک تمرکز دارد. در اینجا نحوه

---

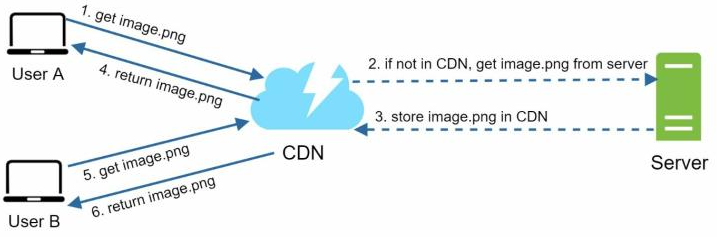
1 Eviction Policy
2 Least-recently-used
3 Least Frequently Used
4 First in First Out
5 static content
6 Dynamic content

عملکرد CDN در سطح بالا آمده است: وقتی کاربر از یک وب‌سایت بازدید می‌کند، سرور CDN نزدیک به کاربر محتوای ثابت را ارائه می‌دهد. به طور مستقیم، هرچقدر فاصله کاربران از سرورهای CDN بیشتر باشند، سرعت بارگذاری وب‌سایت کندتر می‌شود. به‌عنوان‌مثال، اگر سرورهای CDN در شهر سانفرانسیسکو باشند، کاربران در شهر لس‌آنجلس سریع‌تر از کاربران اروپا محتوا دریافت خواهند کرد. شکل ۹-۱ دارای یک مثال عالی است که نشان می‌دهد چگونه CDN زمان بارگذاری را بهبود می‌بخشد.



شکل ۹-۱

تصویر ۱۰-۱ گردش‌کار CDN را نشان می‌دهد.



شکل ۱۰-۱

۱. کاربر A سعی می‌کند با استفاده از URL تصویرِ image.png را دریافت کند. دامنه URL توسط ارائه‌دهنده CDN تهیه شده است. دو نشانی اینترنتی زیر نمونه‌هایی هستند که برای نشان‌دادن اینکه نشانی‌های اینترنتی تصویر در آمازون و CDNهای Akamai چگونه به نظر می‌رسند و استفاده می‌شوند.

- https://mysite.cloudfront.net/logo.jpg

https://mysite.akamai.com/image-manager/img/logo.jpg -

۲. اگر سرور CDN دارای image.png در حافظه کَش نباشد، سرور CDN فایل را از مبدأ درخواست می‌کند که می‌تواند یک وب سرور یا ذخیره‌سازی آنلاین مانند Amazon S3 باشد.
۳. مبدأ image.png را به سرور CDN بر می‌گرداند که شامل هدر HTTP اختیاری یا TTL[1] است که مدت‌زمان ذخیره‌سازی تصویر را توضیح می‌دهد.
۴. این CDN تصویر را کَش می‌کند و آن را به کاربر A برمی‌گرداند. تصویر تا زمانی که TTL منقضی شود در حافظه کَش می‌ماند.
۵. کاربر B درخواستی برای دریافت همان تصویر ارسال می‌کند.
۶. تا زمانی که TTL منقضی نشده باشد، تصویر از حافظه کَش بازگردانده می‌شود.

توصیه‌هایی برای استفاده مناسب از CDN

هزینه: CDN ها توسط ارائه‌دهندگان شخص ثالث اجرا می‌شوند و هزینه انتقال داده‌ها در داخل و خارج از CDN از شما دریافت می‌شود. ذخیره اطلاعاتی که به‌ندرت استفاده می‌شوند هیچ مزیت قابل‌توجهی ندارد، بنابراین باید در نظر داشته باشید که آنها را از CDN خارج کنید.

تنظیم زمان انقضای کَش: برای محتوای حساس به زمان، تنظیم زمان انقضای حافظه کَش مهم است. زمان انقضای حافظه کَش نباید خیلی طولانی باشد و نه خیلی کوتاه. اگر خیلی طولانی باشد، ممکن است محتوا دیگر تازه نباشد. اگر خیلی کوتاه باشد، می‌تواند باعث بارگیری مجدد محتوا از سرورهای مبدأ به CDN شود.

بازگشت CDN[2]: باید در نظر بگیرید که چگونه وب‌سایت/برنامه شما با شکست CDN مقابله می‌کند. اگر یک قطع موقت در CDN وجود داشته باشد، کاربرها باید بتوانند مشکل را شناسایی کرده و منابع را از مبدأ درخواست کنند.

---

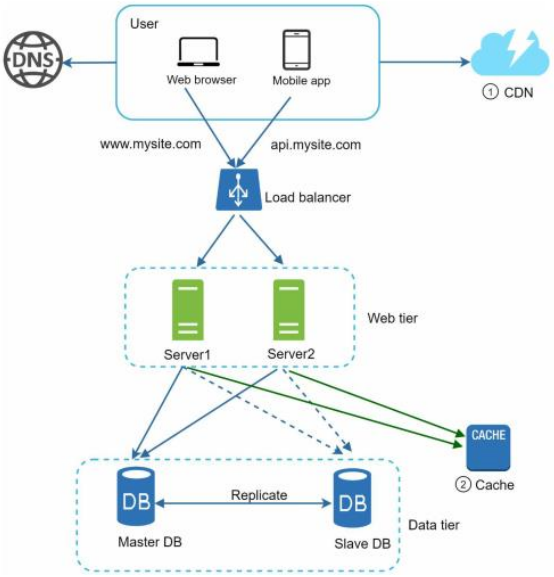
1 Time-to-Live
2 CDN fallback

نامعتبر بودن فایل‌ها: می‌توانید با انجام یکی از عملیات زیر، یک فایل را قبل از منقضی شدن از CDN حذف کنید:

با استفاده از API های ارائه شده توسط ارائه‌دهندگان CDN، موجودیت CDN را ابطال کنید.

از نسخه‌سازی شیء[1] برای ارائه نسخه دیگری از object استفاده کنید. برای نسخه یک object، می‌توانید پارامتری مانند شماره نسخه را به URL اضافه کنید. به‌عنوان‌مثال، نسخه شماره ۲ به رشته کوئری اضافه می‌شود، به‌عنوان‌مثال: image.png?v=2.

شکل ۱۱-۱ طراحی پس از افزودن CDN و حافظه کَش را نشان می‌دهد.



شکل ۱۱-۱

در نتیجه:

۱. محتوی‌های ثابت (JS، CSS، تصاویر و غیره) دیگر توسط وب سرورها ارائه نمی‌شوند. آنها برای عملکرد بهتر از CDN دریافت می‌شوند.

۲. بار روی پایگاه‌داده با ذخیره داده‌ها، کاهش می‌یابد.

---

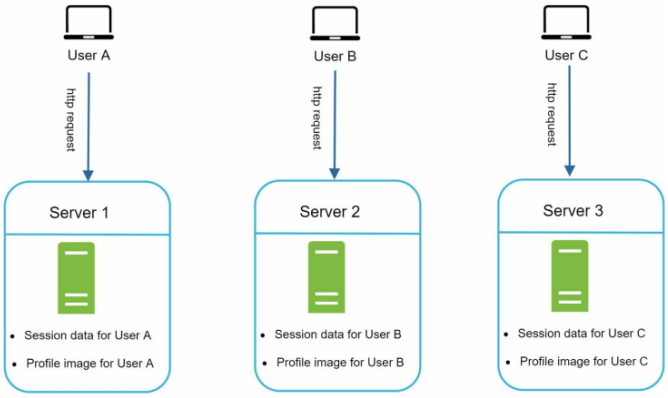
1 object versioning

لایه وب Stateless

اکنون زمان آن رسیده است که لایه وب را به‌صورت افقی مقیاس‌دهی کنید. برای این کار، باید وضعیت (به‌عنوان‌مثال داده‌های session کاربر) را به خارج از لایه وب منتقل کنیم. یک روش مناسب این است که session data[۱] را در ذخیره‌سازی دائمی داده‌ها مانند پایگاه‌داده رابطه‌ای یا NoSQL ذخیره کنید. هر وب سرور در هر خوشه[۲] می‌تواند به داده‌های وضعیت[۳] از پایگاه‌های داده دسترسی داشته باشد. به این لایه وب stateless می‌گویند.

معماری Stateful

یک سرور stateful و سرور stateless دارای تفاوت‌های کلیدی هستند. یک سرور stateful داده‌های کاربر (وضعیت/state) را از یک درخواست به درخواست دیگر به‌خاطر می‌آورد. سرور stateless هیچ اطلاعات وضعیتی یا state را نگه‌داری نمی‌کند. شکل ۱۲-۱ نمونه‌ای از معماری stateful را نشان می‌دهد.



شکل ۱۲-۱

---

۱ داده‌های جلسه (session data) در فناوری اطلاعات به معنای ذخیره و مدیریت اطلاعات در طول یک جلسه کاری می‌باشند. در این مفهوم، داده‌های مرتبط با یک کاربر در طول یک جلسه کاری (مثلاً ورود به یک وب‌سایت) ذخیره می‌شوند تا بتوانند در صفحات مختلف استفاده شوند. این داده‌ها به صورت موقت در حافظه سرور نگهداری می‌شوند و برای هر کاربر جداگانه ذخیره می‌شوند. به عبارت دیگر، داده‌های جلسه مشخصات کاربری مانند نام کاربری، رنگ مورد علاقه و ... را در طول جلسه کاری نگه می‌دارند. این داده‌ها تا زمانی که کاربر مرورگر خود را ببندد، باقی می‌مانند. اگر نیاز به ذخیره داده‌ها به صورت دائمی دارید، می‌توانید از پایگاه داده استفاده کنید.

۲ Cluster - رایانش خوشه‌ای یک نوع از پردازش موازی و توزیع شده است. در این رویکرد، مجموعه‌ای از رایانه‌های مستقل به عنوان گره‌ها با یکدیگر در ارتباط تنگاتنگ قرار دارند و به عنوان یک منبع یکپارچه عمل می‌کنند.

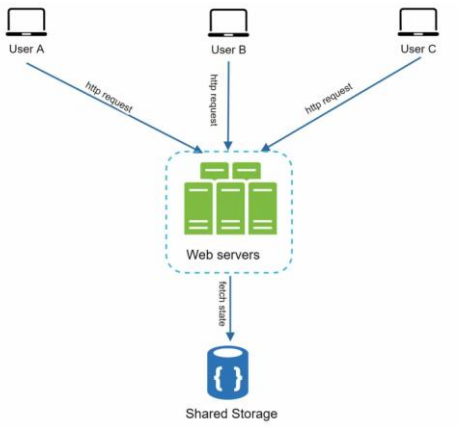
۳ state data

در شکل ۱۲-۱، داده‌های session کاربر A به همراه تصویر پروفایل آن در سرور ۱ ذخیره می‌شود. برای احراز هویت کاربر A، درخواست‌های HTTP باید به سرور ۱ هدایت شوند. اگر درخواستی به سرورهای دیگر مانند سرور ۲ ارسال شود، احراز هویت به دلیل آدرس‌دهی اشتباه، با شکست مواجه می‌شود. سرور۲ حاوی داده‌های session کاربر A نیست. به طور مشابه، تمام درخواست‌های HTTP از کاربر B باید به سرور ۲ هدایت شوند. تمام درخواست‌های کاربر C باید به سرور ۳ ارسال شود.

مسئله این است که هر درخواست از یک کلاینت باید به همان سرور هدایت شود. این کار را می‌توان با sticky sessions[1] در اکثر توزیع‌کننده‌های بار[2] انجام داد [۱۰]. بااین‌حال، این راه‌حل، سربار زیادی را اضافه می‌کند. افزودن یا حذف سرورها با این روش بسیار دشوارتر است. همچنین رسیدگی به خرابی‌های سرور نیز چالش‌برانگیز است.

معماری Stateless

شکل ۱۳-۱ معماری stateless را نشان می‌دهد.



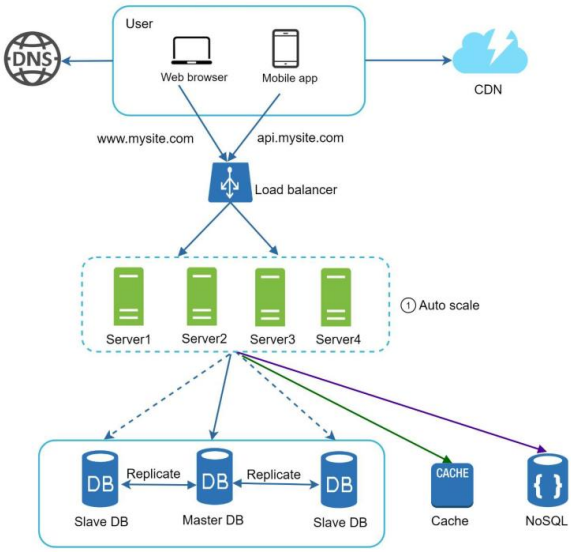
شکل ۱۳-۱

---

۱ استیکی سشن‌ها همچنین به عنوان پایداری جلسه شناخته می‌شوند، روشی است که به وسیله‌ی آن توزیع کننده بار یا load balancer به منظور شناسایی درخواست‌هایی که از یک کلاینت می‌آیند و همیشه این درخواست‌ها را به یک سرور خاص ارسال می‌کند، فراهم می‌شود.

2 load balancer

در معماری stateless، درخواست‌های HTTP از سوی کاربران می‌تواند به هر سرور وب ارسال شود که داده‌های وضعیت/state را از یک ذخیره‌گاه داده مشترک واکشی می‌کند. داده‌های حالت یا state data در یک ذخیره‌گاه داده مشترک ذخیره می‌شوند و خارج از سرورهای وب نگهداری می‌شوند. یک سیستم stateless ساده‌تر، قوی‌تر و مقیاس‌پذیرتر است. شکل ۱۴-۱ طراحی به‌روز شده را با یک لایه وب stateless نشان می‌دهد.



شکل ۱۴-۱

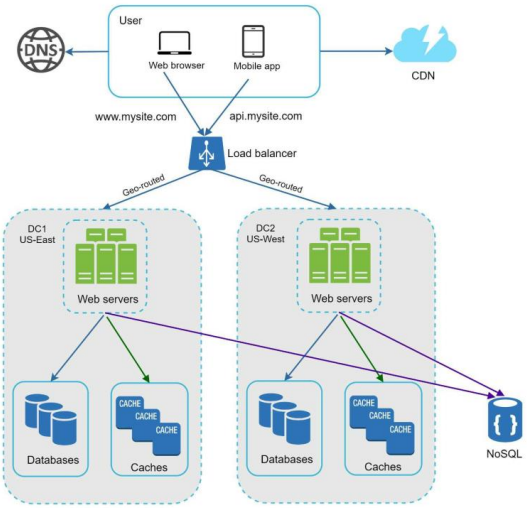
در شکل ۱۴-۱، داده‌های session را به خارج از لایه وب منتقل کرده و در ذخیره‌سازی داده‌های پایدار[1] ذخیره می‌کنیم. ذخیره‌سازی داده‌های مشترک می‌تواند یک پایگاه‌داده رابطه‌ای، Memcached/Redis، NoSQL و سایر موارد باشد. ذخیره‌گاه داده‌های NoSQL به این دلیل انتخاب می‌شود که مقیاس‌پذیری آن آسان است. مقیاس‌دهی خودکار یا Autoscaling به معنای افزودن یا حذف خودکار وب سرورها بر اساس بار ترافیکی است. پس از حذف داده‌های حالت از سرورهای وب، مقیاس‌دهی خودکارِ لایه وب به‌راحتی با افزودن یا حذف سرورها بر اساس بار ترافیکی آن به دست می‌آید.

1 persistent data store

وب‌سایت شما به‌سرعت رشد می‌کند و تعداد قابل‌توجهی از کاربران را در سطح بین‌المللی جذب می‌کند. برای بهبود دردسترس‌بودن[1] و ارائه تجربه کاربری بهتر در مناطق جغرافیایی وسیع‌تر، پشتیبانی از مراکز داده[2] متعدد بسیار مهم است.

مراکز داده (data centers)

شکل ۱۵-۱ یک نمونه راه‌اندازی با دو مرکز داده را نشان می‌دهد. در عملکرد عادی، کاربران با geoDNS مسیریابی می‌شوند که به نام مسیریابی جغرافیایی نیز شناخته می‌شود و به نزدیک‌ترین مرکز داده با ترافیک تقسیم بر x٪ در ایالات متحده-شرق و $(۱۰۰ - x)$٪ در ایالات متحده-غرب متصل می‌شوند. geoDNS یک سرویس DNS است که به نام دامنه اجازه می‌دهد تا بر اساس موقعیت مکانی کاربر با توجه به آدرس‌های IP آن، متصل شود.

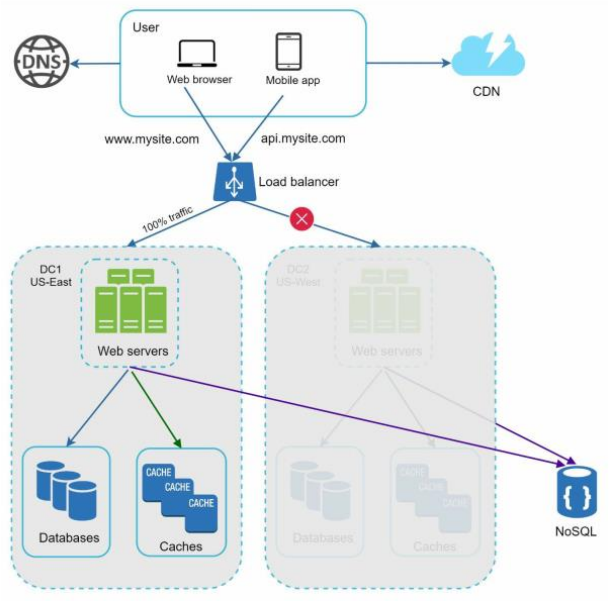


شکل ۱۵-۱

در صورت قطعی و خرابی قابل‌توجه مرکز داده، باید تمام ترافیک را به یک مرکز داده سالم هدایت شود. در شکل ۱۶-۱، مرکز داده ۲ (ایالات متحده-غرب) آفلاین است و ۱۰۰٪ ترافیک به مرکز داده ۱ (ایالات متحده- شرق) هدایت می‌شود.

---

1 availability
2 data centers



شکل ۱۶-۱

برای دستیابی به راه‌اندازی مرکز داده چندگانه[1] باید چندین چالش فنی حل شود:

- **تغییر مسیر ترافیک**[2]: ابزارهای مؤثری برای هدایت ترافیک به مرکز داده صحیح موردنیاز است. از GeoDNS می‌توان برای هدایت ترافیک به نزدیک‌ترین مرکز داده باتوجه‌به مکانی که کاربر در آن قرار دارد استفاده کرد.
- **همگام‌سازی داده‌ها**[3]: کاربران از مناطق مختلف می‌توانند از پایگاه‌های داده یا حافظه کَش محلی مختلف استفاده کنند. در موارد مواجه‌شدن با شکست، احتمال دارد که ترافیک به یک مرکز داده‌ای که در آن داده‌ها در دسترس نیستند هدایت شود. یک استراتژی رایج این است که داده‌ها را در چندین مرکز داده تکثیر کنید. یک مطالعه قبلی نشان می‌دهد که چگونه نتفلیکس تکثیر[4] مرکز داده چندگانه ناهم‌زمان را پیاده‌سازی می‌کند [۱۱].

---

1 multi-data center
2 Traffic redirection
3 Data synchronization
4 replication

- **تست و استقرار**: با راه‌اندازی مرکز داده چندگانه، مهم است که وب‌سایت/برنامه خود را در مکان‌های مختلف آزمایش کنید. ابزارهای استقرار خودکار برای ثابت نگه‌داشتن سرویس‌ها در تمامی مراکز داده حیاتی هستند [۱۱]. برای مقیاس‌بندی یا scale بیشتر سیستم، باید اجزای مختلف سیستم را جدا کنیم تا بتوان آنها را به طور مستقل مقیاس‌بندی کرد. صف پیام[1] یک استراتژی کلیدی است که توسط بسیاری از سیستم‌های توزیع شده در دنیای واقعی برای حل این مشکل استفاده می‌شود.

صف پیام (message queue)

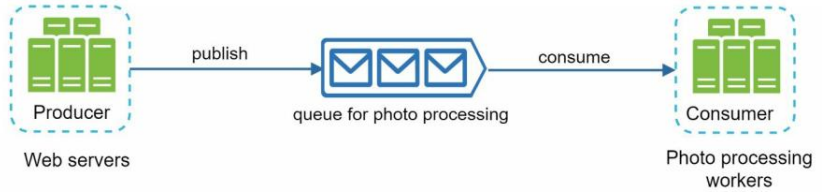
صف پیام یا message queue یک جزء بادوام و مهم است که در حافظه موقت ذخیره می‌شود و از ارتباطات ناهم‌زمان[2] پشتیبانی می‌کند و به‌عنوان یک بافر عمل می‌کند و درخواست‌های ناهم‌زمان را توزیع می‌کند. معماری اولیه صف پیام ساده است. سرویس‌های ورودی که تولیدکننده/ناشر[3] نامیده می‌شوند، پیام‌هایی را ایجاد می‌کنند و آنها را در صف پیام منتشر می‌کنند. سایر سرویس‌ها یا سرورها که مصرف‌کنندگان/مشترکین[4] نامیده می‌شوند به صف متصل می‌شوند و اقداماتی را انجام می‌دهند که توسط پیام‌ها تعریف شده است. مدل مورد تعریف در شکل ۱۷-۱ نشان‌داده‌شده است.

شکل ۱۷-۱

جداسازی[5]، صف پیام را به معماری ترجیحی برای ساختن یک اپلیکیشن مقیاس‌پذیر و قابل‌اعتماد تبدیل می‌کند. با صف پیام، زمانی که مصرف‌کننده برای پردازش آن در دسترس

1 Messaging queue
2 asynchronous
3 producers/publishers
4 consumers/subscribers
5 Decoupling

نباشد، تولیدکننده می‌تواند پیامی را به صف ارسال کند. مصرف‌کننده می‌تواند پیام‌ها را از صف بخواند حتی زمانی که تولیدکننده در دسترس نیست. مورد استفاده زیر را در نظر بگیرید: برنامه شما از سفارشی‌سازی عکس پشتیبانی می‌کند، از جمله برش، شفاف‌سازی، محوکردن، و غیره. تکمیل این وظایف سفارشی‌سازی زمان می‌برد. در شکل ۱۸-۱، وب سرورها کارهای پردازش عکس را در صف پیام منتشر می‌کنند. workerهای پردازشگر عکس‌ها، taskها را از صف پیام انتخاب می‌کنند و به طور ناهم‌زمان taskهای سفارشی سازی عکس را انجام می‌دهند. تولیدکننده و مصرف‌کننده می‌توانند به طور مستقل مقیاس شوند. هنگامی که اندازه صف بزرگ می‌شود، workerهای بیشتری برای کاهش زمان پردازش اضافه می‌شوند. اما اگر صف در بیشتر مواقع خالی باشد، می‌توان تعداد workerها را کاهش داد.



شکل ۱۸-۱

ثبت لاگ‌ها، متریک‌ها و خودکارسازی

هنگام کار با یک وب‌سایت کوچک که روی چند سرور اجرا می‌شود، log برداری، متریک‌ها و خودکارسازی روش‌های خوبی هستند؛ اما یک ضرورت نیستند. بااین‌حال، اکنون که سایت شما برای خدمت به یک تجارت بزرگ رشد کرده است، سرمایه‌گذاری در این ابزارها ضروری است.

- **Logging** : نظارت بر log ها و گزارش‌های خطا مهم است؛ زیرا به شناسایی خطاها و مشکلات در سیستم کمک می‌کند. می‌توانید گزارش‌های خطا را در هر سطح سرور نظارت کنید یا از ابزارهایی برای جمع‌آوری آنها در یک سرویس متمرکز برای جستجو و مشاهده آسان استفاده کنید.

- **متریک‌ها**: جمع‌آوری انواع مختلف متریک‌ها به ما کمک می‌کند تا بینش تجاری را به دست آوریم و وضعیت سلامت سیستم را درک کنیم. برخی از متریک‌های زیر مفید هستند:
  - متریک‌های لایه Host: به‌عنوان‌مثال مصرف منابع؛ CPU، حافظه، ورودی/خروجی دیسک و سایر موارد دیگر.
  - متریک‌های تجمعی[1]: به‌عنوان‌مثال، عملکرد کل ردیف‌های پایگاه‌داده، ردیف‌های کَش و غیره.
  - متریک‌های تجاری: کاربران فعال روزانه، هزینه‌های نگهداری، درآمد و غیره.
- **خودکارسازی**: هنگامی که یک سیستم بزرگ و پیچیده می‌شود، باید ابزارهای اتوماسیون را برای بهبود بهره‌وری بسازیم یا از آنها استفاده کنیم. ادغام پیوسته[2] یک روش مناسبی است که در آن هر کد ورود به سیستم از طریق خودکارسازی تأیید می‌شود و به تیم‌ها اجازه می‌دهد مشکلات را زودتر تشخیص دهند. علاوه بر این، خودکارسازی فرایند ساخت، آزمایش، استقرار و سایر موارد دیگر می‌تواند بهره‌وری توسعه‌دهنده را به میزان قابل‌توجهی بهبود بخشد.

استفاده از صف پیام و سایر ابزارهای دیگر

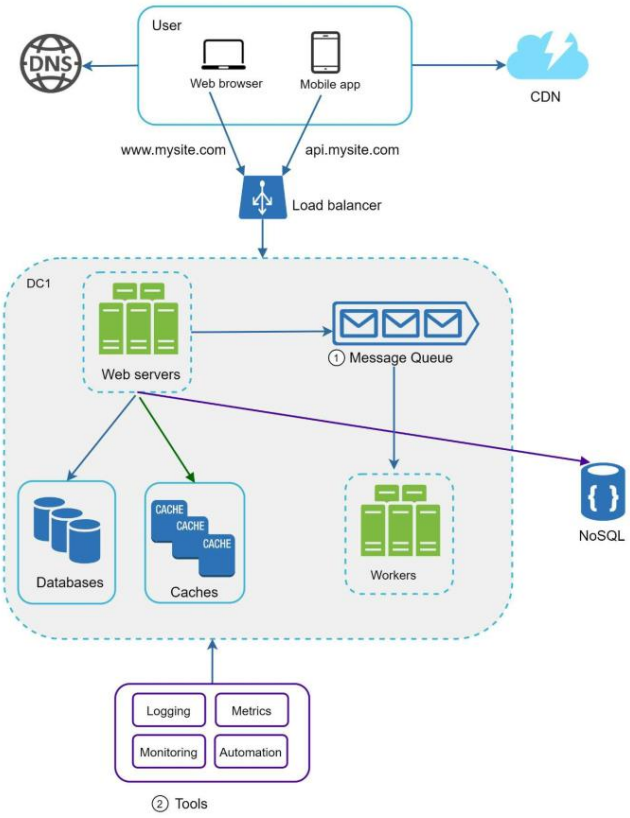
شکل ۱۹-۱ طراحی به‌روز شده را نشان می‌دهد. به دلیل محدودیت فضا، تنها یک مرکز داده در شکل نشان‌داده‌شده است.

۱. طراحی شامل یک صف پیام است که به سیستم کمک می‌کند تا پیوسته و مستحکم و انعطاف‌پذیرتر در مقابل خرابی شود.

۲. ثبت گزارش یا Logging ، نظارت یا monitoring ، متریک‌ها، و ابزارهای خودکارسازی گنجانده شده است.

---

1 Aggregated
2 Continuous integration



شکل ۱۹-۱

همان‌طور که داده‌ها هر روز رشد می‌کنند در نتیجه پایگاه‌داده شما بیش از حد بارگذاری می‌شود. زمان آن است که لایه داده را مقیاس‌دهی کنیم.

مقیاس‌پذیری پایگاه‌داده

دو رویکرد گسترده برای مقیاس‌بندی پایگاه‌داده وجود دارد: مقیاس‌بندی عمودی[1] و مقیاس‌بندی افقی[2].

1 vertical scaling
2 horizontal scaling

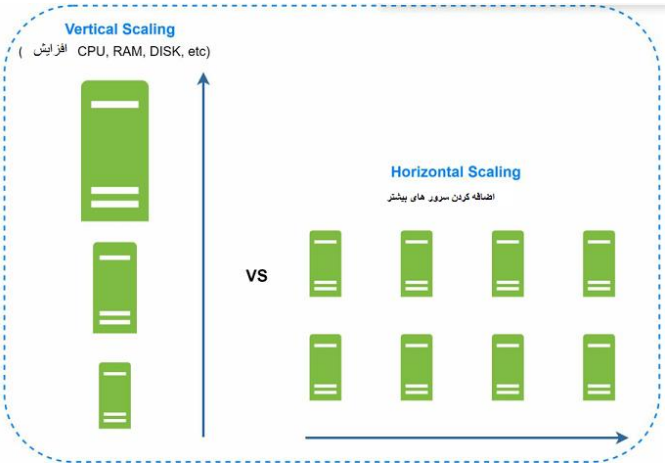
مقیاس‌پذیری عمودی

مقیاس‌پذیری عمودی که همچنین به‌عنوان *scaling up* شناخته می‌شود، مقیاس‌پذیری با اضافه‌کردن قدرت بیشتر (CPU، RAM، DISK، و غیره) به یک ماشین موجود است. سرورهای پایگاه‌داده قدرتمندی وجود دارد. طبق سرویس پایگاه‌داده رابطه‌ای آمازون (RDS) [۱۲]، می‌توانید یک سرور پایگاه‌داده با ۲۴ ترابایت رم دریافت کنید. این نوع سرور پایگاه‌داده قدرتمند می‌تواند داده‌های زیادی را ذخیره و مدیریت کند. به‌عنوان‌مثال، *stackoverflow.com* در سال ۲۰۱۳ بیش از ۱۰ میلیون بازدیدکننده منحصربه‌فرد ماهانه داشت، اما تنها یک پایگاه‌داده اصلی داشت [۱۳]. بااین‌حال، مقیاس‌دهی عمودی با برخی از اشکالات جدی همراه است:

- می‌توانید CPU، RAM و غیره بیشتری را به سرور پایگاه‌داده خود اضافه کنید، اما محدودیت‌های سخت‌افزاری وجود دارد. اگر تعداد کاربرهای زیادی دارید دیگر یک سرور تنها کافی نیست.
- بزرگ‌ترین خطرپذیری در یک نقطه‌ضعف وجود دارد.
- هزینه کلی مقیاس‌دهی عمودی بالا است. سرورهای قدرتمند بسیار گران‌تر هستند.

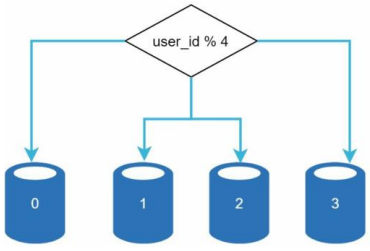
مقیاس‌پذیری افقی

مقیاس‌پذیری افقی، همچنین به‌عنوان **sharding** نیز شناخته می‌شود، تمرینی برای افزودن سرورهای بیشتر است. شکل ۲۰-۱ مقیاس‌دهی عمودی را با مقیاس‌دهی افقی مقایسه می‌کند.



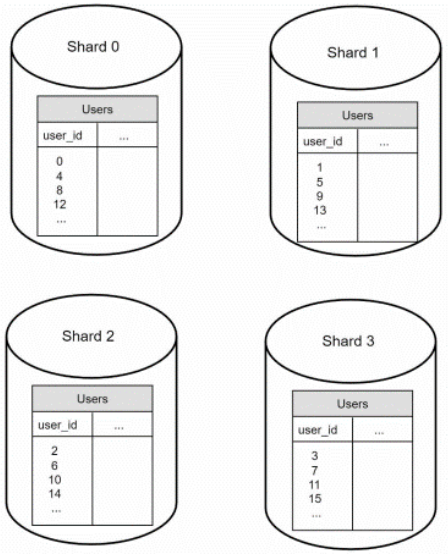
شکل ۲۰-۱

در Sharding پایگاه‌داده‌های بزرگ را به قطعات کوچک‌تر و با سهولت مدیریت بیشتر به نام Shard جدا می‌کند. هر Shard طرح و شماتیک یکسانی را به اشتراک می‌گذارد، اگرچه داده‌های واقعی هر Shard منحصربه‌فرد است. شکل ۲۱-۱ نمونه‌ای از پایگاه‌داده‌های قطعه‌ای یا Shard شده را نشان می‌دهد. داده‌های کاربرها بر اساس شناسه‌های کاربر به یک سرور پایگاه‌داده اختصاص داده می‌شود. هر زمان که به داده‌ها دسترسی پیدا می‌کنید، از یک تابع هش برای یافتن قطعه یا Shard مربوطه استفاده می‌شود. در مثال ما، $user\_id \% 4$ به‌عنوان تابع هش استفاده می‌شود. اگر نتیجه برابر با ۰ باشد، از Shard شماره ۰ برای ذخیره و واکشی داده‌ها استفاده می‌شود. اگر نتیجه برابر با ۱ باشد، از Shard شماره ۱ استفاده می‌شود. همین منطق در مورد سایر Shardها نیز صدق می‌کند.



شکل ۲۱-۱

شکل ۲۲-۱ جدول کاربرها را در پایگاه‌داده‌های قطعه یا Sharding شده نشان می‌دهد.



شکل ۲۲-۱

مهم‌ترین عاملی که در اجرای استراتژی Sharding باید در نظر گرفته شود، انتخاب کلید Sharding است. کلید شاردینگ (معروف به کلید پارتیشن) از یک یا چند ستون تشکیل شده است که نحوه توزیع داده‌ها را تعیین می‌کند. همان‌طور که در شکل ۲۲-۱ نشان‌داده‌شده است، "user_id" کلید Sharding است. یک کلید Sharding به شما امکان می‌دهد تا با مسیریابی کوئری‌های پایگاه‌داده به پایگاه‌داده صحیح هدایت شده و داده‌ها را به طور مؤثر بازیابی و اصلاح کنید. هنگام انتخاب یک کلید Sharding، یکی از مهم‌ترین معیارها انتخاب کلیدی است که بتواند داده‌ها را به طور مساوی توزیع کند. Sharding یک تکنیک عالی برای مقیاس‌دهی پایگاه‌داده است، اما راه‌حل کاملی نیست. پیچیدگی‌ها و چالش‌های جدیدی را به سیستم معرفی می‌کند:

- **مجدداً Sharding کردن داده‌ها**[1]: زمانی داده‌ها مجدداً Sharding کنید که:

۱. یک قطعه یا Shard دیگر نمی‌تواند داده‌های بیشتری را به دلیل رشد سریع نگه دارد، پس Shard مجدد داده‌ها نیاز است.

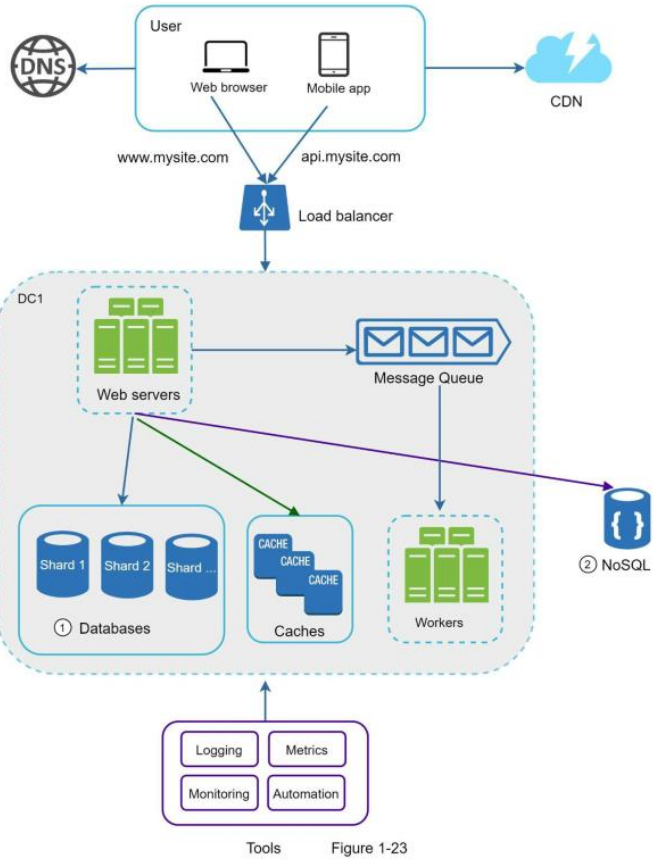
۲. برخی Shardها ممکن است به دلیل توزیع نابرابر داده‌ها، سریع‌تر از سایرین دچار از فرسودگی و ازکارافتادن شوند. هنگامی که فرسودگی Shard اتفاق می‌افتد، نیاز به‌روزرسانی عملکردهای Sharding و جابه‌جایی داده‌ها وجود دارد. هش مداوم[2] که در فصل ۵ موردبحث قرار خواهد گرفت، یک تکنیک رایج برای حل این مشکل است.

۳. **مشکل سلبریتی‌ها**[3]: به این مشکل کلید کانونی یا hotspot نیز می‌گویند. دسترسی بیش از حد به یک Shard می‌تواند باعث اضافه‌بار روی سرور شود. تصور کنید داده‌های Katy Perry، Justin Bieber و Lady Gaga همه به یک Shard خاص ختم می‌شوند. برای کاربردهای اجتماعی، آن قطعه با عملیات خواندن بیش از حد درگیر خواهد شد. برای حل این مشکل، ممکن است لازم باشد برای هر سلبریتی یک قطعه یا Shard تخصیص دهیم. هر Shard گاهی ممکن است مجدداً به پارتیشن بیشتری نیاز داشته باشد.

۴. **پیوستن و غیرنرمال سازی**[4]: زمانی که یک پایگاه‌داده در چندین سرور، چند قطعه و shard شد، انجام عملیات اتصال در میان قطعات/shard پایگاه‌داده دشوار است. یک راه‌حل متداول، غیرنرمال کردن پایگاه‌داده است تا بتوان کوئری‌ها را در یک جدول واحد انجام داد. در شکل ۲۳-۱، ما پایگاه‌های داده را برای پشتیبانی از افزایش سریع ترافیک داده‌ها shard می‌کنیم. در همان زمان، برخی از عملکردهای غیررابطه‌ای به یک ذخیره‌گاه داده NoSQL منتقل می‌شوند تا بار روی پایگاه‌داده را کاهش دهند. در اینجا مقاله‌ای وجود دارد که بسیاری از موارد استفاده از NoSQL را پوشش می‌دهد [۱۴].

---

1 Resharding data
2 Consistent hashing
3 Celebrity problem
4 Join and de-normalization



شکل ۲۳-۱

برخی اصطلاحات مورداستفاده در این کتاب

کارگر (worker)

در علوم کامپیوتر، مفهوم کارگر یا "worker" به معنای یک مؤلفه است که وظایف واقعی را انجام می‌دهد. به‌عبارت‌دیگر، worker یک وظیفه/task را دریافت می‌کند و در همان زمان، در یک نخ/thread متفاوت آن را پردازش می‌کند. وقتی worker وظیفه را به پایان می‌رساند، از طریق یک متد فراخوانی بازگشتی به شما اطلاع می‌دهد. به‌عبارت‌دیگر، یک متد ویژه که در فراخوانی اولیه فراهم شده است، فراخوانی می‌شود. به‌طورکلی، workerها به‌منظور انجام

عملیات‌های زمان‌بر یا مصرف‌کننده منابع به‌صورت ناهم‌زمان بدون نیاز به تعامل مستقیم با کاربران ضروری هستند. به‌عنوان‌مثال، در سیستم‌های مبتنی بر وب، workerها می‌توانند وظایفی مانند پردازش تصاویر، محاسبات پیچیده یا به‌روزرسانی پایگاه‌داده را به‌صورت ناهم‌زمان انجام دهند.

نقاط پایانی (Endpoints)

نقاط پایانی (Endpoints) به دستگاه‌هایی اشاره دارد که به شبکه کامپیوتری متصل می‌شوند. این دستگاه‌ها، نقاط پایانی یا اِندپوینت‌ها نامیده می‌شوند. به‌عنوان‌مثال، کامپیوترهای رومیزی، تلفن‌های هوشمند، تبلت‌ها، لپ‌تاپ‌ها و دستگاه‌های اینترنت اشیا (IoT) نمونه‌هایی از نقاط پایانی هستند. این دستگاه‌ها به شبکه متصل شده و اطلاعات را بین سیستم‌های مختلف منتقل می‌کنند.

فراداده (Metadata)

فراداده[1] یک اصطلاح است که برای توصیف داده‌ها به کار می‌رود. این اطلاعات درباره داده‌های دیگر را فراهم می‌کند، اما خود محتوای داده را نشان نمی‌دهد. به‌عبارت‌دیگر، متادیتا توضیحاتی است درباره داده‌ها که به تسهیل پیگیری و کار با داده‌های خاص کمک می‌کند. مثال‌هایی از متادیتا شامل روش ایجاد داده، هدف داده و اطلاعات توصیفی درباره یک منبع می‌شود. متادیتا می‌تواند از لایه‌ها و انواع مختلفی تشکیل شده باشد، از جمله متادیتای توصیفی، متادیتای ساختاری، متادیتای مدیریتی، متادیتای مرجع و متادیتای آماری.

رشته‌ها (Threads)

Thread (نخ یا رشته) به‌توالی محدودی از دستورالعمل‌های برنامه‌نویسی اشاره دارد. یک جریان واحد از کنترل در یک برنامه است که امکان اجرای چند وظیفه[2] را در درون یک

1 Metadata
2 Task

پردازش به وجود می‌آورد. به‌عبارت‌دیگر، هر برنامه ممکن است تعدادی پردازش مرتبط داشته باشد و هر پردازش می‌تواند دارای چند Thread باشد که این Threadها آن پردازش را اجرا می‌کنند. تمام Threadهای یک پردازش فضای آدرس مجازی و منابع سیستمی آن پردازش را با یکدیگر شریک می‌شوند. Threadها معمولاً به‌وسیله یک زمان‌بند[1] مدیریت می‌شوند. یک زمان‌بند بخش استانداردی از یک سیستم‌عامل است. برای ایجاد یک Thread ابتدا باید یک پردازش یا process ایجاد شود. پس از آن، پردازش یک Thread ایجاد می‌کند و سپس این Thread اجرا خواهد شد. این فرایند می‌تواند برای بازه زمانی کوتاه یا طولانی بسته به نوع پردازش رخ بدهد. فارغ از اینکه چه مدت‌زمان موردنیاز باشد، این مورد به وجود می‌آید که کامپیوتر دارد کارهای زیادی را در یک‌لحظه انجام می‌دهد. هر پردازش حداقل دارای یک Thread است، اما سقفی برای تعداد Threadهای یک پردازش وجود ندارد و پردازش می‌تواند هر تعداد Thread مورد نیازش را به کار گیرد. برای وظایف تخصصی، هر چه تعداد Thread بیش‌تری وجود داشته باشد، عملکرد و کارایی سیستم بهتر خواهد بود.

فیلتر بلوم (Bloom filter)

بلوم فیلتر یک داده ساختار احتمالاتی بسیار کارآمد از نظر فضای ذخیره‌سازی است. از این فیلتر می‌توان برای آزمودن عضویت یک عنصر در مجموعه استفاده کرد.

فرض کنید می‌خواهیم وجود یک عنصری را در مجموعه جستجو کنیم، اگر داده ساختار به شما جواب داد که این عضو در مجموعه وجود دارد احتمال دارد که وجود نداشته باشد. اما اگر بگوید وجود ندارد، قطعاً درست هست. براین‌اساس خطای مثبت داریم؛ ولی خطای منفی امکان‌پذیر نمی‌باشد. این داده ساختار امکان درج عنصر را به ما می‌دهد؛ ولی امکان حذف آن وجود ندارد. هرچه بیشتر عنصر به آن اضافه شود احتمال خطای مثبت بیشتر می‌شود. یعنی احتمال اینکه داده ساختار به شما بگوید عضوی در مجموعه وجود دارد؛ اما در واقع وجود

1 Scheduler

نداشته باشد بیشتر می‌شود. پیچیدگی زمانی فیلتر بلوم O(k) (در عملیات درج عنصر، جستجوی عنصر) است که K به معنی تعداد تابع درهم‌ساز[1] است.

همچنین فیلتر بلوم یک آرایه‌ی بیتی با اندازه‌ی (m) بیت است که ابتدا تمامی عناصر آن صفر مقداردهی می‌شود. پس تعداد (K) تابع درهم‌سازی در اختیار داریم. هر یک از این توابع یک عدد را به یکی از خانه‌های آرایه‌ی فیلتر بلوم نسبت می‌دهد و مقدار آن خانه را از صفر به یک تغییر می‌دهد. برای درج یک عنصر، ابتدا عدد موردنظر را به (K) تابع درهم‌سازی می‌دهیم. سپس به تعداد (K) موقعیت در آرایه، برای این عدد محل مشخص می‌شود. مقدار آرایه در موقعیت‌های مشخص شده را به یک تغییر می‌دهیم. این عمل باعث درج عنصر در فیلتر بلوم می‌شود.

برای جستجوی یک عنصر، ابتدا آن عنصر را به (K) تابع درهم‌سازی می‌دهیم. سپس مقادیر آرایه را در این (K) موقعیت بررسی می‌کنیم. اگر در تعداد یک یا بیشتری از خانه‌ها مقدار صفر باشد، فیلتر با قطعیت جواب می‌دهد که این عنصر در آرایه وجود ندارد. اما اگر تمامی آنها یک باشد، معلوم نیست که این یک شدن به‌خاطر وجود این عنصر است یا به‌خاطر درج‌های دیگر عناصر منجر به یک شدن این خانه شده است؛ بنابراین جواب می‌دهد که احتمالاً این عنصر در فیلتر بلوم حضور دارد.

پیوستگی ضعیف (Loosely Coupled)

پیوستگی ضعیف یا Loosely Coupled به‌نوعی معماری در توسعه سیستم‌های نرم‌افزاری گفته می‌شود که در آن، کامپوننت‌ها یا بخش‌های مختلف اپلیکیشن تا حد ممکن مستقل از یکدیگر خواهند بود. به‌عبارت‌دیگر، این اصطلاح به میزان ارتباط مستقیم ماژول‌های مختلف اپلیکیشن با یکدیگر اطلاق می‌گردد.

هدف اصلی رویکرد یا معماری Loosely Coupled، کاهش هرگونه ریسکی است که پس از اعمال یک‌سری تغییرات در یکی از کامپوننت‌ها ممکن است به‌صورت ناخواسته در دیگر اجزای اپلیکیشن ایجاد گردد. با استفاده از این معماری، ارتباطات داخلی کامپوننت‌ها به حداقل

1 Hash Function

رسانده می‌شود. این تضمین را ایجاد می‌کند که در صورت نیاز به ایجاد یک تغییر در بخشی از اپلیکیشن، صرفاً کامپوننت موردنظر را تغییر داده و عملکرد دیگر کامپوننت‌ها تحت‌الشعاع قرار نخواهد گرفت. همچنین، این معماری فرایند عیب زدایی[1] را به‌مراتب راحت‌تر و سریع‌تر می‌کند.

یکپارچگی تدریجی (Eventual Consistency)

یکپارچگی تدریجی[2] یکی از سه عنصر مدل BASE[3] است که به‌عنوان جایگزینی برای مدل سنتی ACID[4] در سیستم‌های توزیع‌شده مورد استفاده قرار می‌گیرد. این مدل مناسب همه‌ی انواع سیستم‌ها نیست، اما به‌طورکلی به تأخیر کم در ارسال اطلاعات، همراه با ریسک برگشت داده‌های تکراری پاسخ می‌دهد.

در مدل سازگاری یا یکپارچگی تدریجی، اگر ما دستوراتی را اجرا کنیم و سپس سیستم به مدت کافی فعال باقی بماند، می‌توانیم وضعیت داده‌ها را بدانیم. هر خواندن بعدی از آن داده، همان مقدار را باز می‌گرداند و داده‌ها در همه ذخیره‌گاه‌های داده، تقریباً یکسان و برابر است هر چند تضمینی در این مورد وجود ندارد و نیاز به‌صرف زمان جهت یکسان شدن همه داده‌ها در همه ذخیره‌گاه‌ها دارد. این مدل به‌عنوان یک روش تقریبی برای همگام‌سازی داده‌ها در سیستم‌های توزیع‌شده به‌خصوص در حالتی که کارایی سیستم مهم‌تر از سازگاری و یکپارچگی آن است، مورد استفاده قرار می‌گیرد. برای آشنایی بیشتر در این مورد به فصل ۶ مراجعه کنید.

یکپارچگی قوی (Strong Consistency)

یکپارچگی قوی یا "Strong Consistency" در حوزه برنامه‌نویسی هم‌زمان و سیستم‌های توزیع‌شده، به مدلی از یکپارچگی و سازگاری اطلاعات اشاره دارد. در این مدل، تمامی

---

1 debug
2 Eventual Consistency
3 Basically Available, Soft State, Eventually Consistent
4 Atomicity, Consistency, Isolation, Durability

دسترسی‌ها توسط تمامی فرایندهای موازی (یا گره‌ها، پردازنده‌ها و غیره) به یک ترتیب (به‌صورت متوالی) دیده می‌شوند؛ بنابراین فقط یک حالت سازگار و یکپارچه مشاهده می‌شود. به‌عبارت‌دیگر، "Strong Consistency" تضمین می‌کند که هر عمل خواندن نتیجه آخرین عمل نوشتن را برمی‌گرداند، بدون توجه به گره‌ای که عمل خواندن روی آن اجرا شده است. این معمولاً با استفاده از الگوریتم‌های اجماع مانند Paxos یا Raft انجام می‌شود. "Strong Consistency" در تلاش است تا سطح بالاتری از سازگاری و یکپارچگی را فراهم کند. بر خلاف یکپارچگی تدریجی[1] این "Strong Consistency" اطمینان می‌دهد که تمامی گره‌ها داده‌های یکسانی را بدون هیچ ناسازگاری و ناپایداری موقتی مشاهده کنند. این رویکرد به سیستم‌هایی که نیاز به سازگاری و یکپارچگی سخت‌گیرانه دارند، مانند سیستم‌های مالی یا برنامه‌های پردازش داده‌های بلادرنگ، استفاده کرد.

یکپارچگی ضعیف (Weak Consistency)

سازگاری یا یکپارچگی ضعیف، "Weak Consistency" تضمین نمی‌کند که هر عمل خواندن نتیجه آخرین عمل نوشتن را برمی‌گرداند. "Weak Consistency" در تلاش است تا سطح بالاتری از دردسترس‌بودن را فراهم کند. بر خلاف یکپارچگی قوی، "Weak Consistency" اطمینان **نمی‌دهد** که تمامی گره‌ها داده‌های یکسانی را بدون هیچ یکپارچگی موقتی مشاهده کنند. این رویکرد به سیستم‌هایی که نیاز به **دردسترس‌بودن بالا**[2] دارند، استفاده می‌شود.

بررسی ذخیره‌سازهای مختلف

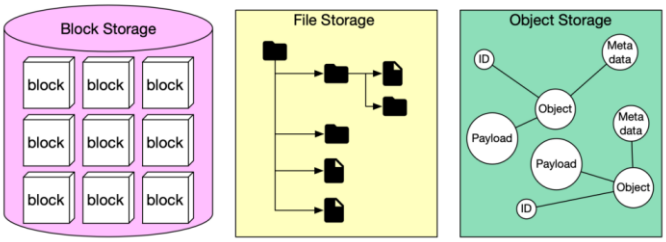
سیستم‌های ذخیره‌سازی به سه دسته کلی تقسیم می‌شوند:

- Block storage
- file storage
- object storage

---

1 Eventual Consistency
2 High avilable

نمودار زیر مقایسه سیستم‌های ذخیره‌سازی مختلف را نشان می‌دهد.



شکل ۲۴-۱

در جدول زیر به مقایسه این سه نوع ذخیره‌ساز پرداخته شده است:

	Block storage	File storage	Object storage
Mutable Content	Y	Y	N (object versioning is supported, in-place update is not)
Cost	High	Medium to high	Low
Performance	Medium to high, very high	Medium to high	Low to medium
Consistency	Strong consistency	Strong consistency	Strong consistency
Data access	SAS/iSCSI/FC	Standard file access, CIFS/SMB, and NFS	RESTful API
Scalability	Medium scalability	High scalability	Vast scalability
Good for	Virtual machines (VM), high-performance applications like database	General-purpose file system access	Binary data, unstructured data

جدول ۱-۱

ذخیره‌ساز بلوکی (Block Storage)

ذخیره‌سازی بلوکی[1] اولین نوع ذخیره‌سازهایی است که در دهه ۱۹۶۰ معرفی شد. ابزارهای ذخیره‌سازی رایج مانند هارددیسک (HDD) و درایو حالت جامد (SSD) که به‌صورت فیزیکی به سرورها متصل می‌شوند، همگی به‌عنوان ذخیره‌سازی بلوکی شناخته می‌شوند. ذخیره‌سازی بلوکی، بلوک‌های خام را به‌صورت یک حجم[2] به سرور ارائه می‌دهد. این انعطاف‌پذیرترین و کارآمدترین شکل ذخیره‌سازی است. سرور می‌تواند بلوک‌های خام را فرمت‌بندی کند و از آن‌ها به‌عنوان یک سیستم فایل استفاده نماید یا اینکه کنترل این بلوک‌ها را به یک برنامه واگذار کند. برخی از برنامه‌ها مانند پایگاه‌داده یا ماشین مجازی این بلوک‌ها را به‌صورت مستقیم مدیریت می‌کنند تا حداکثر کارایی را از آن‌ها به دست آورند. ذخیره‌سازی بلوکی محدود به ابزارهای فیزیکی متصل به سرور نیست. این نوع ذخیره‌سازی می‌تواند از طریق یک شبکه پرسرعت یا پروتکل‌های اتصال استاندارد صنعتی مانند فیبر کانال (FC) و iSCSI به سرور متصل شود. از لحاظ مفهومی، ذخیره‌سازی بلوکی متصل به شبکه نیز همچنان بلوک‌های خام را ارائه می‌دهد. برای سرورها، این روش مشابه با ذخیره‌سازی بلوکی متصل به‌صورت فیزیکی عمل می‌کند. چه به شبکه متصل باشد و چه به‌صورت فیزیکی به سیستم متصل باشند، ذخیره‌سازی بلوکی به طور کامل در اختیار یک سرور واحد قرار دارد و یک منبع اشتراکی به‌حساب نمی‌آید. همین‌طور Block Storage از متادیتا محدود استفاده می‌کند؛ اما برای عملیات خواندن/نوشتن بر روی شناسه‌های منحصربه‌فرد اختصاص‌داده‌شده به هر بلوک تکیه می‌کند. این کاهش اضافه‌بار انتقال داده را کاهش می‌دهد و به سرور اجازه می‌دهد به طور کارآمد به داده‌ها در ذخیره‌سازی بلوک دسترسی پیدا کند و آنها را بازیابی کند. به دلیل محدود بودن متادیتا ذخیره‌سازی بلوک، این نوع ذخیره‌سازی تأخیر فوق‌العاده کمی را که برای بارهای کاری که به عملکرد بالا نیاز دارند، سرعت مناسبی را ارائه می‌دهد. این برای برنامه‌های حساس به تأخیر مانند پایگاه‌های داده لازم است.

---

1 Block Storage
2 volume

ذخیره‌سازی شیء (Object storage)

ذخیره‌سازی شیء[1] که ذخیره‌ساز حبابی یا Blob Storage نیز نامیده می‌شود، یک فناوری نسبتاً جدید است. این نوع از ذخیره‌سازی با اولویت قراردادن دوام (نگهداری ایمن اطلاعات برای مدت طولانی)، مقیاس‌پذیری عظیم و هزینه پایین، به طور عمدی از کارایی (سرعت) صرف‌نظر می‌کند. این نوع ذخیره‌سازی برای داده‌هایی که زیاد مورد استفاده قرار نگرفته (به‌اصطلاح داده‌های سرد) طراحی شده و عمدتاً برای بایگانی و پشتیبان‌گیری مورد استفاده قرار می‌گیرد. در ذخیره‌سازی شیء، تمامی اطلاعات به‌صورت شیء (Object) در یک ساختار تخت (بدون سلسله‌مراتب) ذخیره می‌شوند. در نتیجه، برخلاف روش‌های سنتی، خبری از ساختار درختیِ پوشه‌ها نیست. دسترسی به داده‌ها معمولاً از طریق یک رابط برنامه‌نویسی (RESTful API) فراهم می‌شود. سرعت ذخیره‌سازی شیء نسبت به سایر انواع روش‌های ذخیره‌سازی پایین‌تر است. اکثر ارائه‌دهندگان خدمات ابری عمومی، سرویس ذخیره‌سازی شیء را ارائه می‌دهند، مانند سرویس S3 شرکت آمازون (AWS)، سرویس ذخیره‌سازی بلوکی گوگل (Google Block Storage) و سرویس ذخیره‌سازی شیء آژور (Azure Blob Storage).

ذخیره‌سازی فایل (File storage)

ذخیره‌سازی فایل[2] روی بستر ذخیره‌سازی بلوکی بنا شده است. این نوع از ذخیره‌سازی، سطح بالاتری از انتزاع[3] را برای مدیریت آسان‌تر فایل‌ها و پوشه‌ها ارائه می‌دهد. در این روش، داده‌ها به‌عنوان فایل تحت یک ساختار سلسله‌مراتبی از پوشه‌ها ذخیره می‌شوند. ذخیره‌سازی فایل، متداول‌ترین راهکار ذخیره‌سازی برای مقاصد عمومی است. این نوع از ذخیره‌سازی را می‌توان با استفاده از پروتکل‌های رایج شبکه‌ای در سطح فایل، مانند SMB/CIFS و NFS را برای تعداد زیادی از سرورها قابل‌دسترسی کرد. سرورهایی که به ذخیره‌سازی فایل دسترسی پیدا می‌کنند، نیازی به مدیریت پیچیدگی‌های مربوط به بلوک‌ها، فرمت‌بندی حجم و غیره ندارند.

---

1 Object Storage
2 File Storage
3 abstraction

سادگی ذاتیِ ذخیره‌سازی فایل، آن را به راه‌حلی ایده‌آل برای اشتراک‌گذاری تعداد زیادی از فایل‌ها و پوشه‌ها درون یک سازمان تبدیل می‌کند.

پارتیشن شبکه (Network partition)

پارتیشن شبکه (Network partition) به وضعیتی گفته می‌شود که یک شبکه کامپیوتری به دو یا چند زیرشبکه مجزا تقسیم می‌شود. این تقسیم‌بندی می‌تواند به دو صورت اتفاق بیفتد:

- **تقسیم‌بندی عمدی**: گاهی اوقات، مدیران شبکه به طور عمد شبکه را برای اهداف امنیتی، مدیریتی یا بهینه‌سازی عملکرد، به بخش‌های جداگانه تقسیم می‌کنند.
- **تقسیم‌بندی غیرمنتظره**: این حالت رایج‌تر است و زمانی رخ می‌دهد که یک مشکل فنی در سخت‌افزار یا نرم‌افزار شبکه باعث قطع ارتباط بین بخش‌های مختلف شبکه شود. این مشکلات می‌توانند شامل خرابی روتر، قطعی کابل یا ازکارافتادن سایر تجهیزات شبکه باشند.

تقسیم‌بندی شبکه می‌تواند مشکلاتی را برای سیستم‌هایی که برای برقراری ارتباط با یکدیگر در سراسر شبکه طراحی شده‌اند، ایجاد کند.

نرم‌افزارهای توزیع‌شده که برای اجرا بر روی چندین کامپیوتر در یک شبکه طراحی شده‌اند، به طور خاص در برابر پارتیشن شبکه آسیب‌پذیر هستند. این نرم‌افزارها باید به‌گونه‌ای طراحی شوند که تحمل پارتیشن را داشته باشند، یعنی حتی در صورت تقسیم‌بندی شبکه، همچنان بتوانند به‌درستی کار کنند.

محاسبات خوشه‌ای (Cluster Computing)

خوشه[1] به گروهی از رایانه‌های مستقل گفته می‌شود که به هم متصل شده‌اند و به‌عنوان یک سیستم واحد عمل می‌کنند.

مزایای استفاده از خوشه‌ها:

---

1 Cluster

- **افزایش قدرت پردازش**: با استفاده از قدرت پردازشی مجموعه‌ای از رایانه‌ها، می‌توان وظایف سنگین و پیچیده را سریع‌تر انجام داد.
- **افزایش قابلیت اطمینان**: اگر یک رایانه در خوشه از کار بیفتد، سایر رایانه‌ها می‌توانند وظایف آن را انجام دهند و از قطع‌شدن کار جلوگیری می‌شود.
- **افزایش مقیاس‌پذیری**: می‌توان به‌راحتی رایانه‌های جدیدی را به خوشه اضافه کرد تا ظرفیت پردازش و ذخیره‌سازی را افزایش داد.
- **کاهش هزینه**: استفاده از خوشه می‌تواند به‌صرفه‌تر از خرید یک رایانه بسیار قدرتمند باشد.

انواع مختلف خوشه‌ها:

- خوشه‌های با کارایی بالا[1]: این نوع خوشه‌ها برای انجام وظایف علمی و محاسباتی سنگین مانند شبیه‌سازی‌های پیچیده و تجزیه‌وتحلیل داده‌ها استفاده می‌شوند.
- خوشه‌های با دسترسی بالا[2]: این نوع خوشه‌ها برای ارائه خدمات حیاتی و حساس مانند وب‌سایت‌های بزرگ و سیستم‌های بانکی استفاده می‌شوند.
- خوشه‌های محاسبات ابری[3]: این نوع خوشه‌ها برای ارائه خدمات محاسبات ابری مانند Amazon Web Services و Microsoft Azure استفاده می‌شوند.

نتیجه‌ی نهایی

میلیون‌ها کاربر و فراتر از آن، مقیاس‌گذاری یک سیستم، یک فرایند تکرارپذیر است. تکرار آنچه در این فصل آموخته‌ایم می‌تواند ما را به جایی برساند. تا بدانیم که تنظیم دقیق‌تر و استراتژی‌های جدید برای مقیاس‌پذیری فراتر از میلیون‌ها کاربر موردنیاز است. برای مثال، ممکن است لازم باشد سیستم خود را بهینه کنید و سیستم را به سرویس‌های کوچک‌تر جدا کنید. تمام تکنیک‌های آموخته شده در این فصل باید پایه خوبی برای مقابله با چالش‌های

1 High-performance clusters
2 High-availability clusters
3 Cloud computing clusters

جدید ایجاد کند. برای پایان‌دادن به این فصل، خلاصه‌ای از نحوه مقیاس‌دهی سیستم خود برای پشتیبانی از میلیون‌ها کاربر ارائه می‌دهیم:

- لایه وب را stateless نگه دارید.
- ایجاد افزونگی یا redundancy در هر لایه.
- تا جایی که می‌توانید داده‌ها را کَش کنید.
- پشتیبانی از مراکز داده متعدد.
- میزبانی محتوی‌های ایستا در CDN.
- لایه داده‌های خود را با sharding مقیاس‌پذیر کنید.
- تقسیم لایه‌ها به سرویس‌های مستقل.
- سیستم خود را نظارت کنید و از ابزارهای خودکار استفاده کنید.

منابع و مراجع فصل ۱

[1] Hypertext Transfer Protocol:
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[2] Should you go Beyond Relational Databases?:
https://blog.teamtreehouse.com/should-you-go-beyond-relational-databases
[3] Replication: https://en.wikipedia.org/wiki/Replication_(computing)
[4] Multi-master replication: https://en.wikipedia.org/wiki/Multi-master_replication
[5] NDB Cluster Replication: Multi-Master and Circular Replication:
https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-replication-multi-master.html
[6] Caching Strategies and How to Choose the Right One:
https://codeahoy.com/2017/08/11/caching-strategies-and-how-to-choose-the-right-one/
[7] R. Nishtala, "Facebook, Scaling Memcache at," 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’13).
[8] Single point of failure: https://en.wikipedia.org/wiki/Single_point_of_failure
[9] Amazon CloudFront Dynamic Content Delivery:
https://aws.amazon.com/cloudfront/dynamic-content/
[10] Configure Sticky Sessions for Your Classic Load Balancer:
https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-sticky-sessions.html
[11] Active-Active for Multi-Regional Resiliency:
https://netflixtechblog.com/active-active-for-multi-regional-resiliency-c47719f6685b
[12] Amazon EC2 High Memory Instances:
https://aws.amazon.com/ec2/instance-types/high-memory/
[13] What it takes to run Stack Overflow:
http://nickcraver.com/blog/2013/11/22/what-it-takes-to-run-stack-overflow
[14] What The Heck Are You Actually Using NoSQL For:
http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html

فصل ۲

تخمین تقریبی و برآورد

در یک مصاحبه طراحی سیستم، گاهی اوقات از شما خواسته می‌شود که ظرفیت سیستم یا نیازمندی‌های عملکردی و اجرایی را با استفاده از یک تکنیک تخمین تقریبی و برآورد به‌عنوان back-of-the-envelope تخمین بزنید. به گفته Jeff Dean، عضو ارشد گوگل، «محاسبات مبتنی بر تخمین سریع و تقریبی[1]، تخمین‌هایی هستند که شما با استفاده از ترکیبی از آزمایش‌های فکری و اعداد، حالت عملیاتی متداولی را برای یک سیستم ایجاد می‌کنید تا راه‌حل مناسبی در مورد اینکه کدام طرح‌ها نیازهای شما را برآورده می‌کنند را پیدا کنید» [۱]. برای انجام مؤثر این تخمین به روش back-of-the-envelope باید درک خوبی از اصول مقیاس‌پذیری[2] داشته باشید و مفاهیم زیر باید به‌خوبی درک شوند: توان دو[3] [۲] و «تأخیر عددی[4]» که در واقع به معنی که حداکثر زمان تأخیر در اجرای برنامه است.

---

1 back-of-the-envelope
2 scalability
3 power of two
4 latency numbers

محاسبات توان دو

اگرچه حجم داده‌ها در سیستم‌های توزیع‌شده می‌تواند بسیار زیاد شود؛ ولی بیشتر محاسبات به اصول اولیه‌ای خلاصه می‌شود. برای به‌دست‌آوردن محاسبات صحیح، دانستن واحد حجم داده با استفاده از توان ۲ بسیار مهم است. یک بایت دنباله‌ای از ۸ بیت است. یک کاراکتر ASCII از یک بایت حافظه (۸ بیت) استفاده می‌کند. در زیر جدولی است که واحد حجم داده را توضیح می‌دهد (جدول ۱-۲).

Power	Approximate value	Full name	Short name
10	1 Thousand	1 Kilobyte	1 KB
20	1 Million	1 Megabyte	1 MB
30	1 Billion	1 Gigabyte	1 GB
40	1 Trillion	1 Terabyte	1 TB
50	1 Quadrillion	1 Petabyte	1 PB

جدول ۱-۲

محاسبات مربوط به تأخیر زمانی برنامه

به‌عنوان مثالی در مرجع [۱] از آقای Dr. Dean از مهندسان ارشد در گوگل؛ محاسبه یک عملیات معمولی کامپیوتر را در سال ۲۰۱۰ نشان می‌دهد. برخی از اعداد منسوخ شده‌اند زیرا رایانه‌ها در گذر زمان سریع‌تر و قدرتمندتر می‌شوند. بااین‌حال، آن اعداد همچنان باید بتوانند ایده‌ای از سرعت و کندی عملیات مختلف رایانه به ما ارائه دهند.

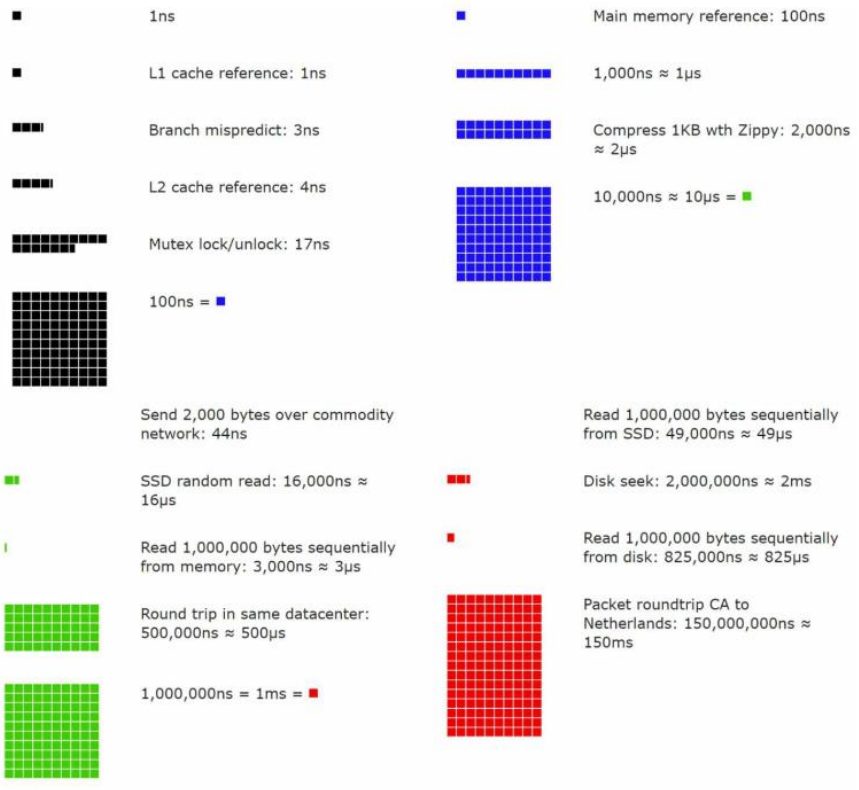
Operation name	Time
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns = 10 µs
Send 2K bytes over 1 Gbps network	20,000 ns = 20 µs
Read 1 MB sequentially from memory	250,000 ns = 250 µs
Round trip within the same datacenter	500,000 ns = 500 µs
Disk seek	10,000,000 ns = 10 ms
Read 1 MB sequentially from the network	10,000,000 ns = 10 ms
Read 1 MB sequentially from disk	30,000,000 ns = 30 ms
Send packet CA (California) ->Netherlands->CA	150,000,000 ns = 150 ms

جدول ۲-۲

نکته

$ns = nanosecond, \mu s = microsecond, ms = millisecond$
$1\ ns = 10\text{^} - 9\ seconds$
$1\ \mu s = 10\text{^} - 6\ seconds = 1{,}000\ ns$
$1\ ms = 10\text{^} - 3\ seconds = 1{,}000\ \mu s = 1{,}000{,}000\ ns$

همین‌طور ***Dr. Dean***؛ ابزاری برای تجسم اعداد ساخته است. این ابزار عامل زمان را نیز در نظر می‌گیرد. شکل ۱-۲ حداکثر تأخیر عددی[1] مشاهده شده را تا سال ۲۰۲۰ نشان می‌دهد (منبع: مرجع [۳]).



شکل ۱-۲

با تجزیه‌وتحلیل اعداد در شکل ۱-۲، به نتایج زیر می‌رسیم:

- حافظه موقت یا Memory سریع است؛ اما دیسک کند است.
- در صورت امکان از جستجو روی دیسک خودداری کنید.
- الگوریتم‌های فشرده‌سازی ساده سریع هستند.

1 latency numbers

- در صورت امکان، داده‌ها را قبل از ارسال از طریق اینترنت فشرده کنید.
- مراکز داده[1] معمولاً در مناطق مختلف هستند و ارسال داده‌ها بین آنها زمان می‌برد.

ارقام در دسترسی بودن – Availability numbers

دردسترس‌بودن بالا[2]، توانایی یک سیستم برای اجرای عملیات مداوم برای یک دوره زمانی مطلوب است. دردسترس‌بودن بالا به‌صورت درصد اندازه‌گیری می‌شود. مقدار ۱۰۰٪ به معنای سرویسی است که تعداد ۰ مرتبه ازکارافتادگی یا downtime دارد. اکثر سرویس‌ها معمولاً میزان از نظر دسترس بودن[3] بین ۹۹٪ تا ۱۰۰٪ نوسان می‌کنند. قرارداد سطح سرویس یا service level agreemen که به طور خلاصه نیز نامیده می‌شود (SLA)، یک اصطلاح رایج برای ارائه‌دهندگان این نوع سرویس‌ها است. این یک توافق نامه بین شما (ارائه‌دهنده سرویس/service provider) و کلاینت شما است و این توافقنامه به طور رسمی سطح زمان آپدیت ارائه سرویس شما را مشخص می‌کند. ارائه‌دهندگان ابری آمازون [۴]، گوگل [۵] و مایکروسافت [۶] که SLA خود را ۹۹.۹٪ یا بالاتر تعیین کردند. Uptime به طور سنتی بر حسب مضرب‌های عدد ۹ اندازه گیری می‌شود. هر چه تعداد ۹ها بیشتر باشد بهتر است. همان‌طور که در جدول ۳-۲ نشان‌داده‌شده است، تعداد رقم ۹ با زمان ازکارافتادگی[4] مورد انتظار سیستم مرتبط است.

1 Data centers
2 High availability
3 Availability
4 downtime

Availability %	Downtime per day	Downtime per year
99%	14.40 minutes	3.65 days
99.9%	1.44 minutes	8.77 hours
99.99%	8.64 seconds	52.60 minutes
99.999%	864.00 milliseconds	5.26 minutes
99.9999%	86.40 milliseconds	31.56 seconds

جدول ۳-۲

مثال: تخمین QPS[1] و storage برای سایت Twitter

لطفاً توجه داشته باشید که اعداد زیر فقط برای این تمرین هستند؛ زیرا اعداد واقعی از توییتر نیستند.

مفروضات:

- ۳۰۰ میلیون کاربر فعال ماهانه.
- ۵۰ درصد از کاربران روزانه از توییتر استفاده می‌کنند.
- کاربران به طور متوسط ۲ توییت در روز ارسال می‌کنند.
- ۱۰ درصد توییت‌ها حاوی موارد چند رسانه‌ای مثل فیلم و عکس هستند.
- داده‌ها به مدت ۵ سال ذخیره می‌شوند.

برآوردها:

تخمین کوئری در ثانیه که Query per second یا (QPS) نیز نامیده می‌شود:

- کاربران فعال روزانه[2] به‌صورت تخمینی برابر است با:

$$\text{Daily active users (DAU)} = 300\,million * 50\% = 150\,million$$

1 Query per second
2 Daily active users (DAU)

- میزان کوئری بر ثانیه یا QPS توییت‌ها و مقدار حداکثر آن به‌صورت تخمینی برابر است با:

$$Tweets\ QPS = 150\ million * 2\ tweets / 24\ hour / 3600\ seconds = \sim 3500$$

$$Peek\ QPS = 2 * QPS = \sim 7000$$

ما در اینجا فقط فضای ذخیره‌سازی رسانه[1] را تخمین می‌زنیم.

- میانگین اندازه توییت:
- tweet_id ← ۶۴ بایت
  - متن ← ۱۴۰ بایت
  - رسانه‌ها ← ۱ مگابایت
- فضای ذخیره‌سازی رسانه برابر است با:

$$Media\ storage = 150\ million * 2 * 10\% * 1\ MB = 30\ TB\ per\ day$$

- ذخیره‌سازی رسانه‌ای به مدت ۵ سال:

$$30\ TB * 365 * 5 = \sim 55\ PB$$

ترفندها و نکته‌ها

تخمین سریع و تقریبی[2] معمولاً در مورد فرایندها صحبت می‌کند. پس حل مشکل مهم‌تر از به‌دست‌آوردن نتیجه است. مصاحبه‌کنندگان ممکن است مهارت‌های حل مسئله شما را آزمایش کنند. در اینجا چند نکته برای توجه‌کردن وجود دارد:

- **گرد کردن و تقریب**. انجام عملیات پیچیده ریاضی در طول مصاحبه دشوار است. به‌عنوان‌مثال، نتیجه "۹۹۹۸۷ / ۹.۱" چیست؟ برای حل مسائل پیچیده ریاضی نیازی به صرف زمان خاصی نیست. در مصاحبه معمولاً دقت زیاد انتظار نمی‌رود. از اعداد گِرد

1 media storage
2 Back-of-the-envelope

و تقریبی به نفع خود استفاده کنید. سؤال تقسیم را می‌توان به‌صورت زیر ساده کرد: "۱۰ / ۱۰۰۰۰۰".

- **فرضیات خود را بنویسید**. ایده خوبی است که فرضیات خود را بنویسید تا بعداً به آنها مراجعه کنید.
- **واحدهای محاسباتی خود را مشخص کنید**. وقتی «۵» را می‌نویسید، یعنی ۵ کیلوبایت یا ۵ مگابایت؟ ممکن است خودتان را با این گیج کنید. واحدها را یادداشت کنید زیرا «۵ مگابایت» به رفع ابهام کمک می‌کند.
- **روش‌های تخمین متداول:** مثلاً حداکثر QPS و مقدار لحظه‌ای QPS، ذخیره‌سازی[1]، حافظه cache، تعداد سرورها و سایر موارد دیگر. می‌توانید این محاسبات را هنگام آماده‌شدن برای مصاحبه تمرین کنید. تمرین معجزه می‌کند[2].

1 storage
2 Practice makes perfect

منابع و مراجع فصل ۲

[1] J. Dean.Google Pro Tip: Use Back-Of-The-Envelope-Calculations To Choose The Best Design: http://highscalability.com/blog/2011/1/26/google-pro-tip-use-back-of-the-envelope- calculations-to-choo.html
[2] System design primer: https://github.com/donnemartin/system-design-primer
[3] Latency Numbers Every Programmer Should Know: https://colin-scott.github.io/personal_website/research/interactive_latency.html
[4] Amazon Compute Service Level Agreement: https://aws.amazon.com/compute/sla/
[5] Compute Engine Service Level Agreement (SLA): https://cloud.google.com/compute/sla
[6] SLA summary for Azure services: https://azure.microsoft.com/en-us/support/legal/sla/summary/

فصل ۳

الگویی برای مصاحبه طراحی سیستم

شما به‌تازگی فرصت یک مصاحبه جذاب از شرکت رؤیایی خود دریافت کرده‌اید. هماهنگ‌کننده استخدام برنامه‌ای برای آن روز برای شما ارسال می‌کند. با بررسی فهرست این کتاب در لحظه‌ای که چشمان شما به این فصل «مصاحبه طراحی سیستم» بیفتد بسیار خوشحال خواهید شد.

مصاحبه‌های طراحی سیستم اغلب ترسناک هستند. ممکن است به‌اندازه «طراحی یک محصول معروف X؟» مبهم باشد. سؤالات مبهم هستند و به طور غیرمنطقی گسترده به نظر می‌رسند. آشفتگی شما قابل‌درک است به‌هرحال چگونه کسی می‌تواند محصولی محبوب را در یک ساعت طراحی کند که ساخت آن وقت صدها، بلکه هزاران مهندس را صرف کرده است؟

خبر خوب این است که هیچ کس از شما انتظار غیرواقعی ندارد. طراحی سیستم در دنیای واقعی بسیار پیچیده است. برای مثال، جستجوی گوگل به طرز فریبنده‌ای ساده است. بااین‌حال، میزان فناوری که زیربنای این سادگی است واقعاً شگفت‌انگیز است. اگر کسی از شما انتظار ندارد که در یک ساعت یک سیستم واقعی طراحی کنید، پس مصاحبه طراحی سیستم چه فایده‌ای دارد؟

مصاحبه طراحی سیستم حل مسئله واقعی را شبیه‌سازی می‌کند که در آن دو همکار بر روی یک مشکل مبهم با یکدیگر همکاری می‌کنند و راه‌حلی را ارائه می‌دهند که اهداف آنها را برآورده می‌کند. مشکل پایان باز است و هیچ پاسخ کاملی وجود ندارد. طراحی نهایی در مقایسه با کاری که در فرایند طراحی انجام می‌دهید اهمیت کمتری دارد. این کار به شما امکان می‌دهد مهارت حل مسئله خود را نشان دهید، از انتخاب‌های طراحی خود دفاع کنید و به بازخوردها به شیوه‌ای سازنده پاسخ دهید.

اجازه دهید این خیال را کنار بگذاریم و آنچه را که واقعاً در سر مصاحبه‌کننده می‌گذرد بیان کنیم؛ حالتی که مصاحبه‌کننده برای ملاقات با شما وارد اتاق جلسه می‌شود را تصور کنید. هدف اصلی مصاحبه‌کننده ارزیابی دقیق توانایی‌های شماست. آخرین چیزی که او می‌خواهد این است که یک ارزیابی بی‌نتیجه ارائه دهد؛ زیرا جلسه ضعیف پیش رفته است و سیگنال‌های کافی برای جذب شما وجود ندارد. مصاحبه‌گر در مصاحبه طراحی سیستم به دنبال چیست؟ بسیاری فکر می‌کنند که مصاحبه طراحی سیستم تماماً در مورد مهارت‌های طراحی فنی یک فرد است. این بسیار بیشتر از آن است. یک مصاحبه طراحی سیستم مؤثر سیگنال‌های قوی در مورد توانایی فرد برای همکاری، کار تحت‌فشار و حل ابهامات سازنده می‌دهد. توانایی پرسیدن سؤالات خوب نیز یک مهارت ضروری است و بسیاری از مصاحبه‌کنندگان به طور خاص به دنبال این مهارت در فرد مصاحبه‌شونده هستند.

یک مصاحبه‌کننده خوب همچنین به دنبال پرچم‌های قرمز[1] است. مهندسی‌کردن زیادتر از اندازه[2]، بیش از یک بیماری واقعی برای بسیاری از مهندسان است؛ زیرا آنها از خلوص طراحی لذت می‌برند و از مزایا و معایب[3] چشم‌پوشی می‌کنند. آنها اغلب از هزینه‌های ترکیبی سیستم‌های بیش از حد مهندسی شده آگاه نیستند و بسیاری از شرکت‌ها بهای زیادی را برای این ناآگاهی می‌پردازند. به‌طورقطع نمی‌خواهید این تمایل را در مصاحبه طراحی سیستم نشان دهید. از دیگر پرچم‌های قرمز می‌توان به کوتاه‌بینی، لجبازی و سایر موارد دیگر اشاره کرد. در

1 red flags
2 Over-engineering
3 tradeoffs

این فصل به چند نکته مفید می‌پردازیم و یک چارچوب ساده و مؤثر برای حل مشکلات مصاحبه طراحی سیستم معرفی می‌کنیم.

چهار مرحله برای یک مصاحبه طراحی سیستم تأثیرگذار

هر مصاحبه طراحی سیستم متفاوت است. یک مصاحبه طراحی سیستم عالی بدون پایان است و هیچ راه‌حل نهایی وجود ندارد. بااین‌حال، مراحل و زمینه‌های مشترک وجود دارد که در هر مصاحبه طراحی سیستم باید پوشش دهید.

مرحله اول - درک مسئله و شروع به طراحی

مثال زیر را در نظر بگیرید:

"چرا ببر غرش کرد؟"

یک دست در پشت کلاس بالا آمد.

معلم پاسخ داد: "بله، جیمی؟"

"چون او گرسنه بود".

"خیلی خوب جیمی."

جیمی در طول دوران کودکی خود همیشه اولین کسی بود که در کلاس به سؤالات پاسخ می‌داد. هر وقت معلم سؤالی می‌پرسد، همیشه بچه‌ای در کلاس درس وجود دارد که دوست دارد سؤالش را بی‌تفاوت ببیند و برایش فرقی نمی‌کند جواب را بداند یا نه. آن جیمی است. جیمی یک دانش‌آموز ممتاز است. او به دانستن سریع همه پاسخ‌ها افتخار می‌کند. در امتحانات معمولاً اولین نفری است که سؤالات را تمام می‌کند. او بهترین انتخاب معلم برای هر مسابقه آکادمیک است.

درنتیجه: **مثل جیمی نباشید**.

در یک مصاحبه طراحی سیستم، دادن پاسخ سریع بدون فکرکردن، هیچ امتیازی به شما نمی‌دهد. پاسخ‌دادن بدون درک کامل از الزامات یک پرچم قرمز بزرگ است؛ زیرا این مصاحبه یک مسابقه بی‌اهمیت نیست و جواب دقیقی وجود ندارد.

بنابراین، برای ارائه راه‌حل به‌سرعت وارد عمل نشوید. کمی آهسته‌تر و عمیق فکر کنید و سؤال‌های بپرسید که الزامات و نیازمندی‌ها و مفروضات را روشن کند که این کار بسیار مهم است.

به‌عنوان یک مهندس، ما دوست داریم مشکلات سخت را حل کنیم و به طراحی نهایی بپردازیم. بااین‌حال، این رویکرد احتمالاً شما را به سمت طراحی سیستم اشتباه سوق می‌دهد. یکی از مهم‌ترین مهارت‌های یک مهندس، پرسیدن سؤالات درست، ایجاد فرضیات مناسب و جمع‌آوری تمام اطلاعات موردنیاز برای ساختن یک سیستم است؛ بنابراین، از پرسیدن سؤال نترسید.

وقتی سؤالی می‌پرسید، مصاحبه‌گر مستقیماً به سؤال شما پاسخ می‌دهد یا از شما می‌خواهد که فرضیات خود را مطرح کنید. اگر مورد دوم اتفاق افتاد، فرضیات خود را روی تخته یا کاغذ بنویسید. ممکن است بعداً به آنها نیاز پیدا کنید.

چه نوع سؤالاتی بپرسیم؟ برای درک دقیق نیازمندی‌ها سؤال بپرسید. در اینجا لیستی از سؤالات برای کمک به شما برای شروع آمده است:

- قرار است چه ویژگی‌های خاصی بسازیم؟
- محصول نهایی چند کاربر دارد؟
- شرکت پیش‌بینی می‌کند که محصول با چه سرعتی رشد داشته باشد؟ مقیاس‌های پیش‌بینی شده در ۳ ماه و ۶ ماه و یک سال چیست؟
- ابزار و فناوری[1] مورد استفاده در شرکت چیست؟ چه سرویس‌هایی از قبل وجود دارند که بتوان با استفاده از آن طراحی را ساده کنیم؟

1 stack

بررسی یک مثال:

اگر از شما خواسته شد که یک سیستم **خبرخوان**[1] طراحی کنید، باید سؤالاتی بپرسید که به شما در روشن‌کردن نیازمندی‌ها کمک کند. پس مکالمه بین شما و مصاحبه‌کننده ممکن است به این شکل باشد:

شما: آیا این یک برنامه تلفن همراه است؟ یا یک برنامه وب؟ یا هر دو؟

مصاحبه‌کننده: هر دو.

شما: مهم‌ترین ویژگی‌های محصول چیست؟

مصاحبه‌کننده: امکان ارسال پست و دیدن **خبرخوان** دوستان.

شما: آیا خبرخوان به ترتیب زمانی معکوس مرتب شده است یا به ترتیب خاصی؟

منظور ترتیب خاص به این معنی است که هر پست وزن متفاوتی دارد. به‌عنوان‌مثال، پست‌های دوستان نزدیک مهم‌تر از پست‌های یک گروه هستند.

مصاحبه‌کننده: برای ساده نگه‌داشتن همه‌چیز، اجازه دهید فرض کنیم خبرخوان بر اساس ترتیب زمانی معکوس مرتب شده است.

شما: یک کاربر چند دوست می‌تواند داشته باشد؟

مصاحبه‌کننده: ۵۰۰۰

شما: حجم ترافیک چقدر است؟

مصاحبه‌کننده: ۱۰ میلیون کاربر فعال روزانه

شما: آیا news feed می‌تواند حاوی تصاویر، ویدئوها یا فقط متن باشد؟

مصاحبه‌کننده: می‌تواند حاوی فایل‌های رسانه‌ای، از جمله تصاویر و ویدئوها باشد.

---

۱ news feed – در اینجا منظور نویسنده طراحی یک سیستم timeline مربوط به شبکه‌های اجتماعی شبیه برنامه‌هایی مثل اینستاگرام و توییتر است.

در بالا چند نمونه سؤال وجود دارد که می‌توانید از مصاحبه‌کننده خود بپرسید که در درک الزامات و روشن‌شدن ابهامات مهم است.

مرحله دوم – طراحی سطح پیشرفته

در این مرحله، هدف ما توسعه یک طرح سطح بالا و رسیدن به توافق با مصاحبه‌کننده در مورد طرح است. همکاری با مصاحبه‌کننده در طول این فرایند ایده خوبی است.

- یک ایده اولیه برای طرح ارائه دهید و بازخورد بخواهید. با مصاحبه‌کننده خود به‌عنوان یک هم تیمی رفتار کنید و با هم دیگر کار کنید. بسیاری از مصاحبه‌کنندگانِ خوب عاشق صحبت‌کردن و مشارکت هستند.
- نمودارهای مختلفی را با اجزای کلیدی روی تخته سفید یا کاغذ بکشید. این کار ممکن است شامل کلاینت‌ها (موبایل/وب)، API ها، وب سرورها، ذخیره‌سازهای داده[1]، حافظه کَش[2]، CDN، صف پیام و غیره باشد.
- محاسبات تخمینی و غیردقیق[3] را انجام دهید تا ارزیابی کنید که آیا طرح شما با محدودیت‌های مقیاس[4] دارد یا خیر. با صدای بلند فکر کنید اگر تخمین دقیق‌تری لازم است قبل از عمیق‌شدن روی آن، با مصاحبه‌کننده خود ارتباط برقرار کنید.

در صورت امکان، چند مورد استفاده ابتدایی را مرور کنید. این به شما کمک می‌کند طراحی سطح پیشرفته را قالب‌بندی کنید همچنین این احتمال وجود دارد که گزینه‌های مورد بررسی به شما در کشف موارد لبه‌ای و نقاط تاریکی که هنوز در نظر نگرفته‌اید کمک کند.

آیا باید نقاط پایانی[5] برای APIها و طراحی پایگاه‌داده را در اینجا قرار دهیم؟ جواب این سؤال بستگی به نوع مسئله دارد. برای مشکلات طراحی بزرگ مانند «طراحی موتور جستجوی Google»، این سطح از طراحی کمی بی‌ارزش و ساده است. برای مشکلی مانند طراحی

---

1 data stores
2 cache
3 back-of-the-envelope
4 scale
5 endpoints

backend برای یک بازی پوکر چندنفره، این یک رویکرد منصفانه است در نهایت توجه داشته باشید که حتماً با مصاحبه‌کننده خود ارتباط برقرار کنید.

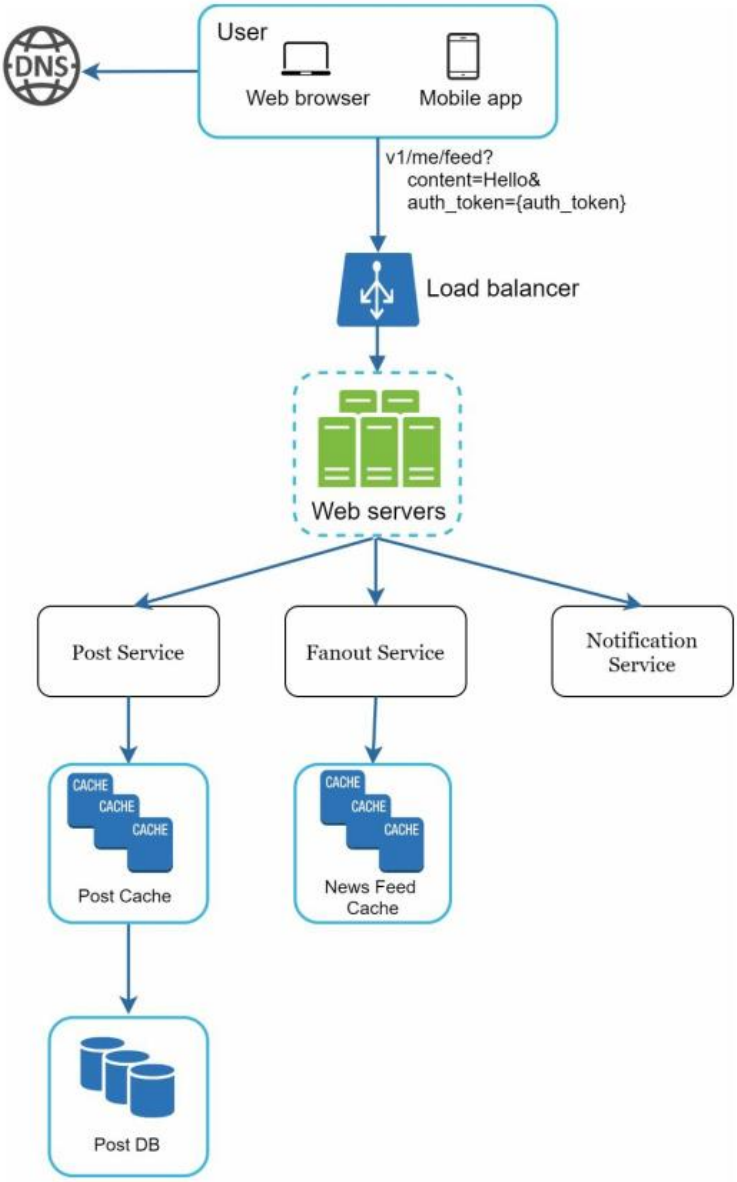
مثال

اجازه دهید از «طراحی یک سیستم News feed» برای نشان‌دادن نحوه نزدیک‌شدن به طراحی سطح پیشرفته استفاده کنیم. در اینجا نیازی به درک نحوه عملکرد سیستم ندارید. تمام جزئیات در فصل ۱۱ توضیح داده خواهد شد.

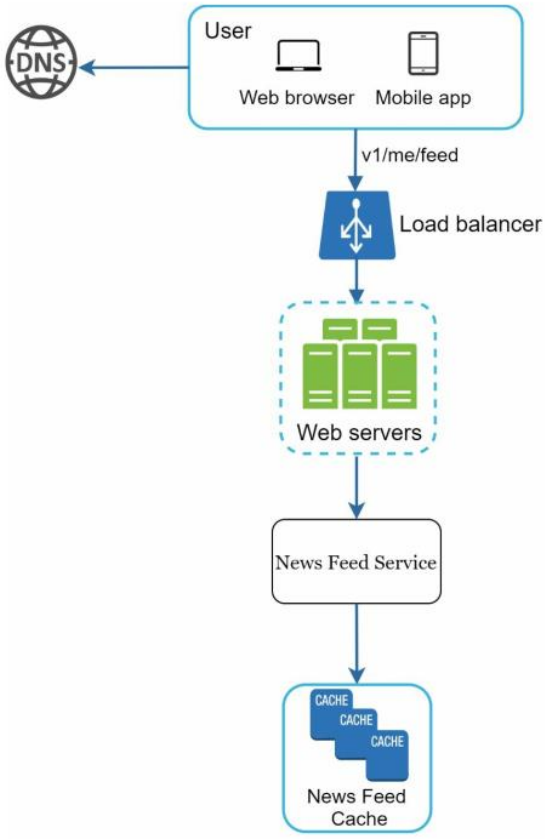
در سطح پیشرفته، طراحی به دو مسیر تقسیم می‌شود: انتشار Feed و ساخت Feed خبری (خبرخوان).

- **انتشار Feed**: وقتی کاربر پستی را منتشر می‌کند، داده‌های مربوطه در حافظه کَش/پایگاه‌داده نوشته می‌شود و پست در Feed اخبار دوستان پر می‌شود.
- **ایجاد خبرخوان**: خبرخوان با جمع‌آوری پست‌های دوستان به ترتیب زمانی معکوس ساخته می‌شود.

شکل ۱-۳ و شکل ۲-۳ به ترتیب طرح‌های سطح بالایی را برای انتشار Feed و دیاگرام ساختاری خبرخوان ارائه می‌دهد.


DNS
User
Web browser
Mobile app
v1/me/feed?
content=Hello&
auth_token={auth_token}
Load balancer
Web servers
Post Service
Fanout Service
Notification Service
CACHE
CACHE
CACHE
Post Cache
CACHE
CACHE
CACHE
News Feed Cache
Post DB

شکل ۱-۳



شکل ۲-۳

مرحله سوم - عمیق‌شدن در طراحی

در این مرحله، شما و مصاحبه‌کننده باید به اهداف زیر دست‌یافته باشید:

- در مورد اهداف کلی و محدوده ویژگی‌ها[1] به توافق رسیدید.
- طرحی در سطح پیشرفته برای ایده اصلی ترسیم کردید.
- از مصاحبه‌کننده خود در مورد طراحی سطح پیشرفته‌ی خود بازخورد دریافت کردید.
- برخی از ایده‌های اولیه که تمرکز و زمان بیشتری صرف آن شده است درباره قسمتی طرح بوده است که بر اساس بازخورد مصاحبه مطرح شده است.

1 feature scope

شما باید با مصاحبه‌کننده برای شناسایی و اولویت‌بندی اجزای معماری کار کنید. شایان‌ذکر است که هر مصاحبه متفاوت است. گاهی اوقات، مصاحبه‌کننده ممکن است به این نکته اشاره کند که دوست دارد روی طراحی سطح پیشرفته تمرکز کند. گاهی اوقات، در حین مصاحبه با مصاحبه‌شونده که دارای توان تخصصی بالا[1] است، بحث می‌تواند بر روی ویژگی‌های عملکردی سیستم متمرکز باشد که احتمالاً بر روی گلوگاه‌ها و برآورد منابع توجه دارد. در بیشتر موارد، مصاحبه‌کننده ممکن است از شما بخواهد که جزئیات برخی از اجزای سیستم را بررسی کنید. برای کوتاه‌کننده URL، جالب است که در طراحی تابع هش که URL طولانی را به یک URL کوتاه تبدیل می‌کند، متمرکز شوید. برای یک سیستم چت، نحوه کاهش تأخیر زمانی[2] و نحوه پشتیبانی از وضعیت آنلاین/آفلاین، دو موضوع جالب است.

همین‌طور مدیریت زمان ضروری است، زیرا به‌راحتی می‌توان با جزئیات غیرضروری که توانایی‌های شما را نشان نمی‌دهد مشغول شوید و زمان را هدر بدهید. شما باید مجهز به سیگنال‌هایی باشید تا به مصاحبه‌کننده خود نشان دهید و سعی کنید وارد جزئیات غیرضروری نشوید. به‌عنوان‌مثال، صحبت در مورد الگوریتم **EdgeRank** که رتبه‌بندی feed در فیس‌بوک است، در طول مصاحبه طراحی سیستم مناسب نیست؛ زیرا این کار زمان زیادی را می‌طلبد و توانایی شما را در طراحی یک سیستم مقیاس‌پذیر[3] ثابت نمی‌کند.

بررسی یک مثال

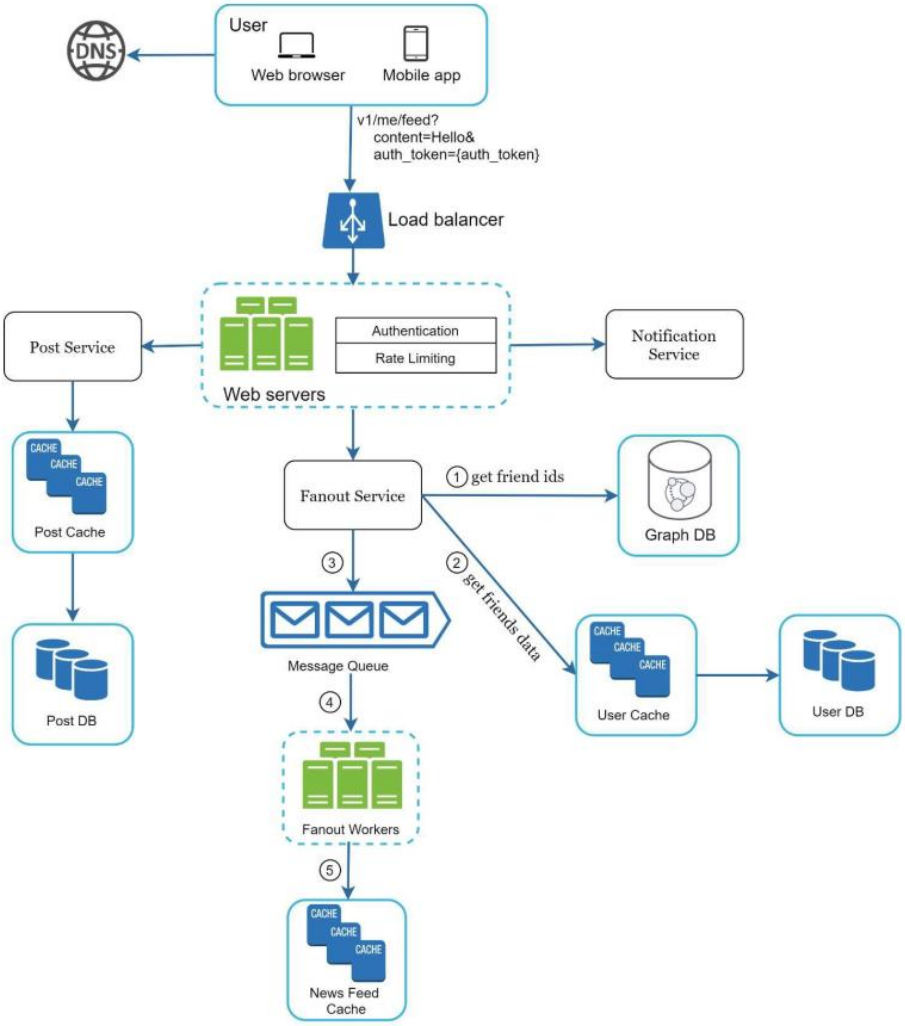
در این مرحله، ما در مورد طراحی سطح بالا برای یک سیستم خبرخوان بحث کرده‌ایم و مصاحبه‌کننده از پیشنهاد شما خوشحال است. در ادامه، دو مورد از مهم‌ترین موارد استفاده را بررسی خواهیم کرد:

۱. انتشار فید[4]
۲. دریافت اخبار فید[5]

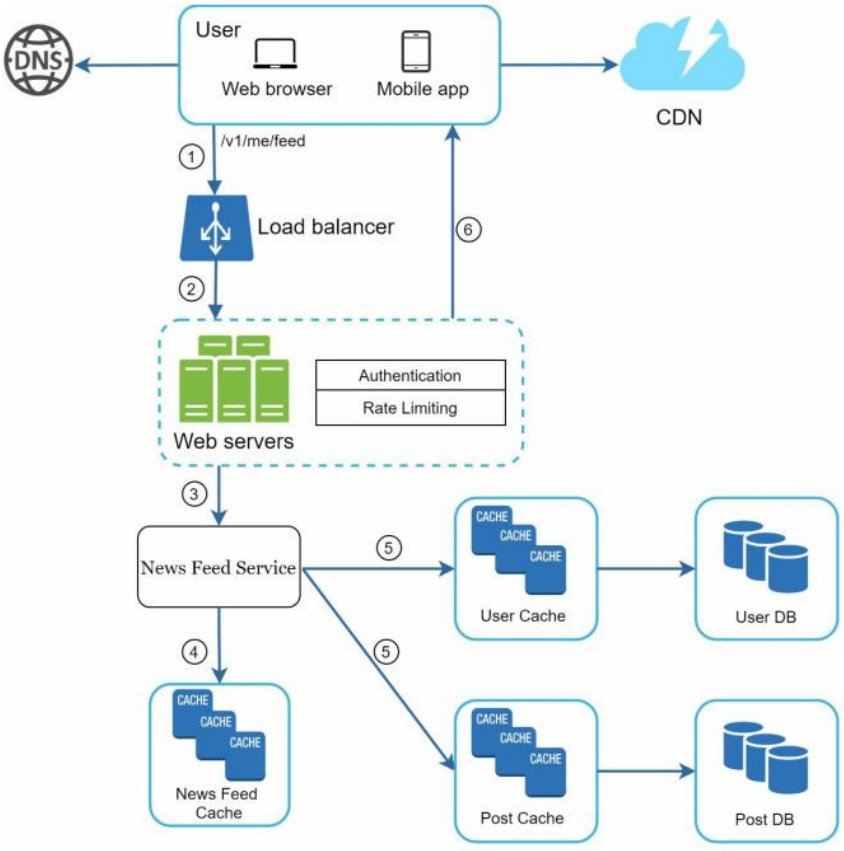
---

1 senior
2 latency
3 scalable
4 Feed publishing
5 News feed retrieval

شکل ۳-۳ و شکل ۴-۳ طرح خوبی از دو مورد استفاده شده را نشان می‌دهد که در فصل ۱۱ توضیح داده خواهد شد.



شکل ۳-۳



شکل ۴-۳

مرحله چهارم - جمع‌بندی

در این مرحله نهایی، مصاحبه‌کننده ممکن است چند سؤال دیگر از شما بپرسد یا به شما این آزادی را بدهد که در مورد سایر نکات اضافی صحبت کنید. در اینجا چند دستورالعمل وجود دارد که باید دنبال کنید:

- ممکن است مصاحبه‌کننده از شما بخواهد که گلوگاه‌های سیستم را شناسایی کرده و در مورد بهبودهای اساسی آن بحث کنید. هرگز نگویید طراحی شما کامل است و

هیچ‌چیز قابل بهبود نیست. همیشه چیزی برای بهبود وجود دارد. این یک فرصت عالی برای نشان‌دادن تفکر انتقادی خود و ایجاد یک تأثیر نهایی خوب است.

- ارائه خلاصه‌ای از طرح خود به مصاحبه‌کننده می‌تواند مفید باشد. اگر چند راه‌حل جدید پیشنهاد دهید که کاری بسیار خوب است. تازه کردن حافظه مصاحبه‌کننده می‌تواند بعد از یک جلسه طولانی مفید باشد.
- موارد خطا (خرابی سرور، ازدست‌دادن شبکه و غیره) موضوعات جالبی هستند که می‌توانید در مورد آن‌ها صحبت کنید.
- مسائل عملیاتی نیز قابل‌توجه است. چگونه متریک‌ها و لاگ‌های خطا را نظارت[1] می‌کنید؟ چگونه سیستم را راه‌اندازی کنیم؟
- نحوه مدیریت نمودار مقیاس‌دهی[2] سیستم نیز موضوع جالبی است. به‌عنوان‌مثال، اگر شما طراحی فعلی از ۱ میلیون کاربر پشتیبانی می‌کند، برای پشتیبانی از ۱۰ میلیون کاربر چه تغییراتی باید انجام دهید؟
- اگر وقت بیشتری داشتید، اصلاحات دیگری را که نیاز دارید پیشنهاد دهید.

برای جمع‌بندی، فهرستی از بایدها و نبایدها را خلاصه می‌کنیم.

بایدها

- برای واضح‌تر شدن مسئله، سؤال بپرسید و فرض خود را درست فرض نکنید.
- الزامات مسئله را درک کنید.
- هیچ پاسخ درست و بهترینی وجود ندارد. راه‌حلی که برای حل مشکلات یک استارتاپ نوپا طراحی شده است با راه‌حل یک شرکت تأسیس شده با میلیون‌ها کاربر متفاوت است. مطمئن شوید که نیازمندی‌ها را درک کرده‌اید.
- اجازه دهید مصاحبه‌کننده بداند به چه فکر می‌کنید و با او ارتباط برقرار کنید.
- در صورت امکان چندین روش مختلف پیشنهاد دهید.

---

1 monitor
2 scale

- هنگامی که با مصاحبه‌کننده خود در مورد ایده اولیه مطرح شده به توافق رسیدید، به جزئیات هر قسمت توجه بیشتر کنید. ابتدا اصلی‌ترین اجزا را طراحی کنید.
- خیلی از ایده‌ها را در برابر مصاحبه‌کننده مطرح نکنید و از نظرات مصاحبه‌کننده استفاده کنید و فرصت همکاری با مصاحبه‌کننده را از دست ندهید. در خیلی از مواقع یک مصاحبه‌کننده خوب به‌عنوان یک هم تیمی با شما کار می‌کند.
- هرگز تسلیم نشوید[1].

نبایدها

- بسیار نامطلوب است که برای سؤالات معمولی و ساده در مصاحبه آمادگی نداشته نباشید.
- بدون روشن‌کردن الزامات و نیازمندی‌ها و مفروضات وارد راه‌حل و دادن ایده نشوید.
- در ابتدا بیش از حد به جزئیات یک قسمت نپردازید. ابتدا طرح سطح بالا را ارائه دهید سپس سراغ جزئیات بروید.
- اگر گیر کردید و به مشکل خوردید از درخواست راهنمایی دریغ نکنید.
- باز هم ارتباط برقرار کنید در سکوت فکر نکنید.

فکر نکنید مصاحبه شما با ارائه طرحی که پیشنهاد دادید تمام شده است. تا زمانی که مصاحبه‌کننده شما بگوید کار شما تمام نشده است، باید کار خود ادامه بدهید و به طور مداوم بازخورد بخواهید.

تخصیص زمان برای هر مرحله

سؤالات مصاحبه طراحی سیستم معمولاً بسیار گسترده است و ۴۵ دقیقه یا یک ساعت برای پوشش کل طرح کافی نیست. در نتیجه مدیریت زمان ضروری است. برای هر مرحله چقدر باید وقت بگذارید؟ در زیر یک راهنمای بسیار تقریبی در مورد توزیع وقت خود در یک جلسه

1 Never give up

مصاحبه ۴۵ دقیقه‌ای ارائه شده است. لطفاً به یاد داشته باشید که این یک تخمین تقریبی است و توزیع زمان واقعی به دامنه مشکل و نیازهای مصاحبه‌گر بستگی دارد.

مرحله اول - درک مسئله و شروع به طراحی، بین ۳ الی ۱۰ دقیقه زمان نیاز دارد.

مرحله دوم - طراحی سطح پیشرفته، حدود ۱۰ الی ۱۵ دقیقه زمان نیاز دارد.

مرحله سوم - عمیق‌شدن در طراحی، حدود ۱۰ الی ۲۵ دقیقه زمان نیاز دارد.

مرحله چهارم - جمع‌بندی که بین ۳ - ۵ دقیقه زمان نیاز دارد.

فصل ۴

طراحی یک RATE LIMITER

در یک سیستم تحت شبکه؛ یک rate limiter یا محدودکننده نرخ برای کنترل نرخ ترافیک ارسال شده توسط یک کلاینت یا یک سرویس استفاده می‌شود. در دنیای HTTP، یک rate limiter، تعداد درخواست‌های مجاز یک کاربر یا کلاینت جهت ارسال به یک سرویس، در یک دوره مشخص را محدود می‌کند. اگر تعداد درخواست‌های API از آستانه تعیین شده توسط rate limiter بیشتر شود، همه تماس‌های اضافی مسدود می‌شوند. در اینجا چند مثال برای این مسئله وجود دارد:

- کاربر نمی‌تواند بیش از ۲ پست در ثانیه بنویسد.
- شما می‌توانید حداکثر ۱۰ حساب در روز از یک آدرس IP ایجاد کنید.
- شما نمی‌توانید بیش از ۵ بار در هفته از یک دستگاه امتیاز (جهت ثبت‌نام کاربر جدید) دریافت کنید.

در این فصل از شما خواسته شده است که یک rate limiter طراحی کنید. قبل از شروع طراحی، ابتدا به مزایای استفاده از یک rate limiter از نوع API نگاه می‌کنیم:

- **جلوگیری از آسیب‌پذیری منابع ناشی از حمله (DoS[1])**. تقریباً همه APIهای منتشر شده توسط شرکت‌های بزرگ فناوری نوعی rate limiting را اعمال می‌کنند[۱].

1 Denial of Service

به‌عنوان‌مثال، توییتر تعداد توییت‌ها را به ۳۰۰ توییت در هر ۳ ساعت محدود می‌کند [۲]. APIهای Google docs دارای محدودیت پیش‌فرض زیر هستند: به‌عنوان‌مثال تعداد ۳۰۰ درخواست خواندن برای هر کاربر در ۶۰ ثانیه بوده [۳] و همین‌طور یک rate limiter با مسدودکردن تماس‌های اضافی از حملات DoS به صورت عمدی یا غیرعمدی جلوگیری می‌کند.

- **کاهش هزینه**. محدودکردن درخواست‌های اضافی به معنای سرورهای کمتر و تخصیص منابع بیشتر به APIهای با اولویت بالا است. rate limiting برای شرکت‌هایی که از APIهای شخص ثالث غیررایگان[1] استفاده می‌کنند، بسیار مهم است. به‌عنوان‌مثال، برای APIهای خارجی زیر بر اساس هر تماس، هزینه دریافت می‌کنید: بررسی اعتبار، ایجاد یک پرداخت مالی، بازیابی سوابق سلامتی[2] و غیره. محدودکردن تعداد تماس‌ها برای کاهش هزینه‌ها ضروری است.
- جلوگیری از بارگذاری بیش از حد[3] سرورها برای کاهش بار روی سرور، یک rate limiter برای فیلترکردن درخواست‌های اضافی ناشی از ربات‌ها یا رفتار نادرست کاربران استفاده می‌شود.

مرحله ۱ - درک مسئله و شروع به طراحی

محدودکننده نرخ را می‌توان با استفاده از الگوریتم‌های مختلف پیاده‌سازی کرد که هر کدام مزایا و معایب خود را دارند. تعاملات بین یک مصاحبه‌کننده و یک کاندید کمک می‌کند تا نوع rate limiters را که در تلاش برای ایجاد آن هستیم را روشن و واضح کنیم.

کاندید: قرار است چه نوع rate limiter ای را طراحی کنیم؟ آیا این یک rate limiter سمت کلاینت است یا rate limiter سمت API سرور؟

مصاحبه‌کننده: سؤال مناسبی پرسیده شد. ما روی rate limiter API سمت سرور تمرکز می‌کنیم.

---

1 paid third party APIs
2 retrieve health records
3 overload

کاندید: آیا rate limiter درخواست‌های API[1] را بر اساس IP، شناسه کاربری یا سایر ویژگی‌ها محدود می‌کند؟

مصاحبه‌کننده: محدودکننده درخواست‌ها باید به‌اندازه کافی انعطاف‌پذیر باشد تا از مجموعه‌های مختلف قوانین[2] rate limiter پشتیبانی کند.

کاندید: مقیاس[3] سیستم چقدر است؟ آیا برای یک استارت‌آپ کوچک یا یک شرکت بزرگ با کاربران زیاد ساخته شده است؟

مصاحبه‌کننده: سیستم باید قادر به رسیدگی به تعداد زیادی درخواست باشد.

کاندید: آیا این سیستم در یک محیط توزیع‌شده کار خواهد کرد؟

مصاحبه‌کننده: بله.

کاندید: آیا rate limiter یک سرویس جداگانه است یا باید در کد برنامه پیاده‌سازی شود؟

مصاحبه‌کننده: این تصمیم در طراحی به عهده شماست.

کاندید: آیا باید به کاربرانی که با نرخ خاصی در حال درخواست‌کردن هستند، نزدیک‌شدن به آستانه محدود یا مسدودشدن درخواست‌های آنها را اطلاع دهیم؟

مصاحبه‌کننده: بله

نیازمندی‌ها و الزامات

در اینجا خلاصه‌ای از الزامات سیستم آمده است:

- درخواست‌های بیش از حدِ[4] مجاز را به طور دقیق محدود کنید.
- **زمان تأخیر کم**[5]. rate limiter نباید زمان پاسخ‌دهی HTTP را کاهش دهد.
- برنامه تاحدامکان از حافظه کمتری استفاده کند.

---

1 API requests
2 rules
3 scale
4 excessive requests
5 Low latency

- **rate limiter توزیع شده**. rate limiter را می‌توان در چندین سرور یا فرایند به اشتراک گذاشت.
- **رسیدگی به استثناها یا خطاها**. هنگامی که درخواست‌های کاربران به از آستانه مجاز بیشتر شده و در نتیجه محدود شده‌اند، در این حالت باید پیام واضحی جهت اطلاع از محدود یا مسدودشدن درخواست‌ها به کاربر اعلام کنید.
- **تحمل خرابی بالا**[1]. اگر مشکلی در rate limiter وجود داشته باشد (مثلاً یک سرور کَش آفلاین شود)، بر کل سیستم تأثیر نگذارد.

مرحله ۲ – طراحی سطح پیشرفته

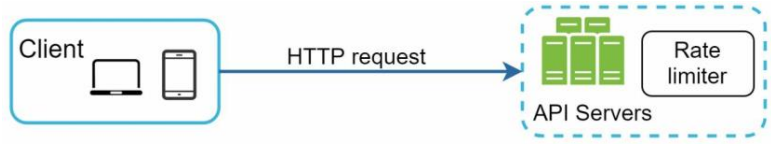
بگذارید کارها را ساده نگه داریم و از یک مدل ابتدایی کلاینت و سرور برای ارتباط استفاده کنیم.

محدودیت نرخ را در کجا قرار دهیم؟

به طور مستقیم، شما می‌توانید یک rate limiter را در سمت کلاینت یا سمت سرور پیاده‌سازی کنید.

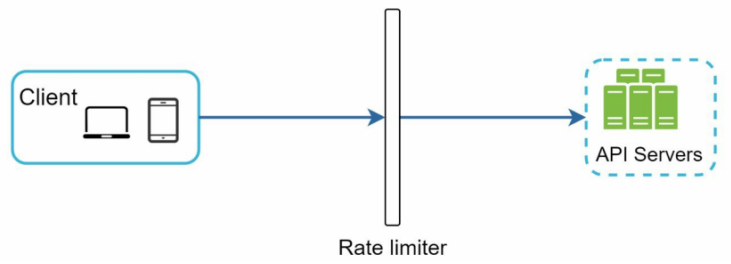
- پیاده‌سازی سمت کلاینت؛ به‌طورکلی، کلاینت مکان غیرقابل‌اعتمادی برای اعمال rate limiter است؛ زیرا درخواست‌های کلاینت می‌تواند به‌راحتی توسط عوامل مخرب جعل یا دست‌کاری شود. علاوه بر این، ممکن است کنترلی بر اجرای فرایندها در سمت کلاینت نداشته باشیم.
- پیاده‌سازی سمت سرور شکل ۱-۴ یک rate limiter را نشان می‌دهد که در سمت سرور قرار داده شده است.

1 High fault tolerance



شکل ۱-۴

علاوه بر پیاده‌سازی rate limiter در سمت کلاینت یا سمت سرور، یک راه جایگزین نیز وجود دارد. به‌جای قراردادن یک rate limiter در API سرور، ما یک middleware rate limiter[۱] ایجاد می‌کنیم که درخواست‌ها به API ها را همان‌طور که در شکل ۲-۴ نشان‌داده‌شده است، کنترل و محدود می‌کند.

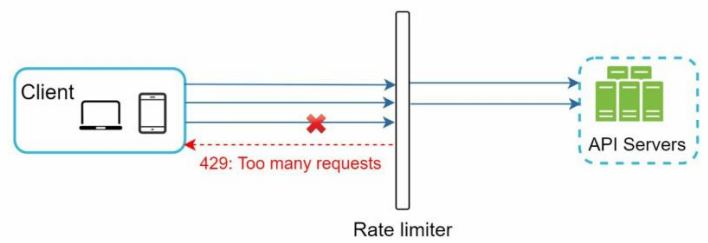


شکل ۲-۴

بگذارید از نمونه‌ای در شکل ۳-۴ استفاده کنیم تا نشان دهیم که چگونه rate limiting در این طرح کار می‌کند. فرض کنید API ما ۲ درخواست در هر ثانیه اجازه می‌دهد و یک کلاینت در یک ثانیه تعداد ۳ درخواست به سرور ارسال می‌کند. دو درخواست اول به سرورهای API هدایت می‌شوند. بااین‌حال، middleware rate limiter سومین درخواست را محدود می‌کند و کد وضعیت HTTP 429 را بر می‌گرداند. کد وضعیت HTTP 429 حالتی را نشان می‌دهد که کاربر درخواست‌های زیادی ارسال کرده است.

---

۱ محدود کننده نرخ میانی



شکل ۳-۴

میکروسرویس‌های ابری[1] [۴] به طور گسترده‌ای محبوب شده‌اند و rate limiting معمولاً در یک مؤلفه به نام API gateway[2] پیاده سازی می‌شود. API gateway یک سرویس کاملا مدیریت شده است که از محدود کننده‌های نرخ[3]، ترمینال‌های SSL، احراز هویت[4]، لیست سفیدی[5] از IPها، سرویس دادن به محتوای ثابت و غیره پشتیبانی می‌کند. در حال حاضر، فقط باید بدانیم که API gateway یک middleware[6] است که از rate limiting پشتیبانی می‌کند.

در حین طراحی یک rate limiter، یک سؤال مهم که باید از خود بپرسیم این است: rate limiting در کجا باید پیاده‌سازی شود؟ در سمت سرور؟ یا در یک gateway؟ هیچ پاسخ مطلقی وجود ندارد. این به تکنولوژی و فناوری[7] فعلی، منابع انسانی متخصص و پیش‌زمینه یا دانش لازم در مهندسی نرم‌افزار، اولویت‌ها، اهداف و سایر موارد مربوط به شرکت شما بستگی دارد. در اینجا چند دستورالعمل کلی وجود دارد:

- تکنولوژی و فناوری‌های فعلی خود را ارزیابی کنید، مانند زبان برنامه‌نویسی، سرویس حافظه کَش و غیره. مطمئن شوید که زبان برنامه‌نویسی فعلی شما برای اجرای محدودیت نرخ یا rate limiter در سمت سرور مناسب است.

---

1 Cloud microservices

۲ دروازه API

3 rate limiter
4 authentication
5 IP whitelisting

۶ میان افزار

7 stack

- الگوریتم‌های rate limiter را که متناسب با نیازهای کسب‌وکار شما است را شناسایی کنید. وقتی همه‌چیز را در سمت سرور پیاده‌سازی می‌کنید، در نتیجه کنترل کامل الگوریتم خود را دارید. بااین‌حال، اگر از gatewayهای شخص ثالث[1] استفاده می‌کنید، ممکن است انتخاب شما محدود باشد.
- اگر قبلاً از معماری میکروسرویس استفاده کرده‌اید و یک API gateway را برای انجام احراز هویت[2]، لیست سفیدی از IPها و غیره در طراحی گنجانده‌اید، می‌توانید یک rate limiter به API gateway اضافه کنید.
- ایجاد سرویس rate limiter خود به‌صرف زمان نیاز دارد. اگر منابع مهندسی کافی برای اجرای rate limiter ندارید، یک API gateway تجاری گزینه بهتری است.

الگوریتم‌هایی برای محدودیت نرخ

می‌توان یک Rate limiting با استفاده از الگوریتم‌های مختلف پیاده‌سازی کرد و هر کدام مزایا و معایب مشخصی دارند. اگرچه در این فصل بر روی الگوریتم‌ها تمرکز نمی‌کند ولی درک آنها در سطح بالا به انتخاب الگوریتم یا ترکیبی از الگوریتم‌ها متناسب با موارد استفاده ما کمک می‌کند. در اینجا لیستی از الگوریتم‌های محبوب آمده است:

- سطل توکن (Token bucket)
- سطل نشتی (Leaking bucket)
- شمارنده پنجره ثابت (Fixed window counter)
- لاگ پنجره کشویی (Sliding window log)
- شمارنده پنجره کشویی (Sliding window counter)

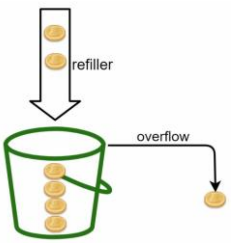
1 third-party
2 authentication

الگوریتم سطل توکن

الگوریتم Token bucket[۱] به طور گسترده برای rate limiting استفاده می‌شود. این الگوریتم ساده است و به‌خوبی درک می‌شود و معمولاً توسط شرکت‌های اینترنتی استفاده می‌شود. هر دو شرکت Amazon [۵] و Stripe [۶] از این الگوریتم برای کاهش درخواست‌های API خود استفاده می‌کنند.

الگوریتم Token bucket به‌صورت زیر عمل می‌کند:

- در واقع Token bucket شامل ظرف، پیمانه یا سطلی است که دارای ظرفیت از پیش تعریف شده است. توکن‌ها به‌صورت دوره‌ای با نرخ‌های از پیش تعیین شده در سطل یا bucket قرار می‌گیرند. پس از پر شدن سطل، هیچ توکن دیگری اضافه نمی‌شود. همان‌طور که در شکل ۴-۴ نشان‌داده‌شده است، ظرفیت سطل توکن برابر ۴ است. پرکننده سطل در هر ثانیه ۲ توکن را در سطل قرار می‌دهد. پس از پر شدن سطل، توکن‌های اضافی سرریز می‌شوند.



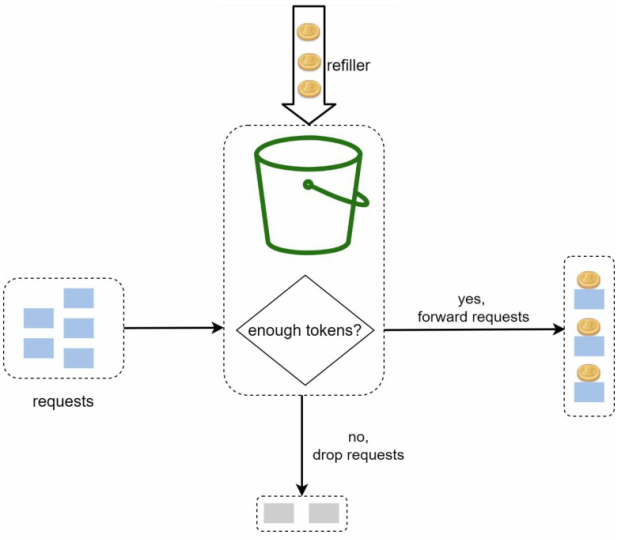
شکل ۴-۴

- هر درخواست یک توکن مصرف می‌کند. وقتی درخواستی می‌رسد، بررسی می‌کنیم که آیا توکن‌های کافی در سطل وجود دارد یا خیر. شکل ۵-۴ نحوه عملکرد آن را توضیح می‌دهد.
- اگر توکن‌های کافی وجود داشته باشد، برای هر درخواست یک توکن بیرون می‌آوریم و درخواست انجام می‌شود.

---

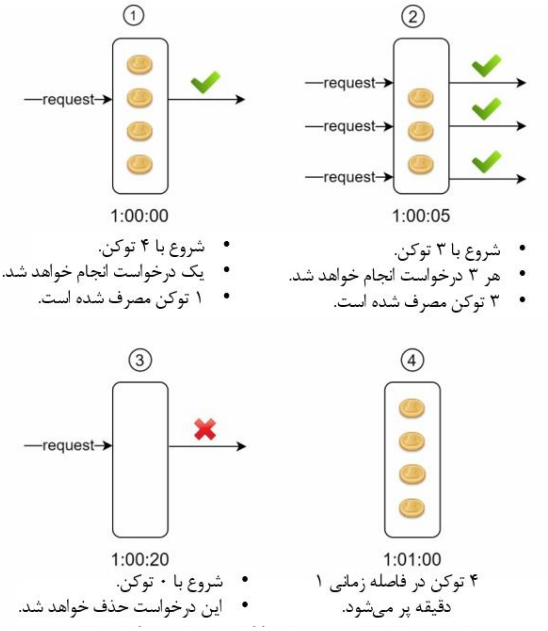
۱ سطل نشانه

- اگر توکن‌های کافی وجود نداشته باشد، درخواست حذف می‌شود.



شکل ۵-۴

شکل ۶-۴ که چگونگی مصرف توکن را نشان می‌دهد، که شامل پر کردن مجدد سطل و کار کردن منطق rate limiter مورد استفاده است. در این مثال، اندازه سطل توکن برابر ۴ است و نرخ پر کردن مجدد به میزان ۴ عدد در هر دقیقه است.



شکل ۶-۴

الگوریتم Token bucket دو پارامتر می‌گیرد:

- اندازه سطل[1]: حداکثر تعداد توکن‌های مجاز در سطل.
- نرخ پر کردن مجدد[2]: تعداد توکن‌هایی که در هر ثانیه در سطل قرار می‌گیرند.

سؤالی که به وجود می‌آید این است که: به چه تعدادی سطل نیاز داریم؟ جواب این سؤال سرراست نیست و به قوانین محدودکننده نرخ[3] بستگی دارد. در اینجا چند مثال هستند.

- معمولاً وجود bucketهای مختلف برای نقاط انتهایی[4] یک API مختلف ضروری است. به‌عنوان‌مثال، در یک اپلیکیشن شبکه اجتماعی یک کاربر مجاز است در هر ثانیه ۱ پست بگذارد، ۱۵۰ دوست در روز اضافه کند و ۵ پست در هر ثانیه را لایک کند، پس به ۳ سطل یا bucket جداگانه برای هر کاربر موردنیاز است.

---

1 Bucket size
2 Refill rate
3 rate-limiting rules
4 endpoints

- اگر نیاز داریم که درخواست‌ها را بر اساس آدرس‌های IP محدود کنیم، هر آدرس IP به یک سطل نیاز دارد.
- اگر سیستم حداکثر ۱۰۰۰۰ درخواست در ثانیه را اجازه دهد، منطقی است که یک سطل سراسری مشترک[1] بین همه درخواست‌ها موجود باشد.

مزایا:

- پیاده‌سازی الگوریتم آسان است.
- مصرف حافظه مناسب است.
- استفاده از Token bucket اوج مصرف ترافیک لحظه‌ای[2] برای دوره‌های کوتاه‌مدت را اجازه می‌دهد. یک درخواست[3] می‌تواند به راه خود ادامه بدهد تا زمانی که توکن‌هایی برای این درخواست باقی‌مانده باشند.

معایب:

- دو پارامتر مهم در این الگوریتم برابر است با اندازه سطل[4] و نرخ پر کردن[5] توکن است. بااین‌حال، تنظیم صحیح آنها ممکن است چالش‌برانگیز باشد.

الگوریتم سطل نشتی دار

الگوریتم سطل نشتی دار یا Leaking bucket مشابه الگوریتم Token bucket است با این تفاوت که درخواست‌ها با نرخ ثابتی پردازش می‌شوند. معمولاً با ساختار داده صف‌های FIFO[6] اجرا می‌شود. الگوریتم به‌صورت زیر کار می‌کند:

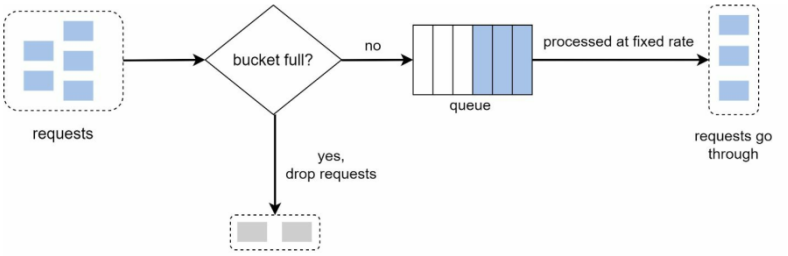
- هنگامی که درخواستی می‌رسد، سیستم بررسی می‌کند که آیا صف پر است یا خیر. اگر پر نباشد، درخواست به صف اضافه می‌شود.
- در غیر این صورت درخواست منتفی می‌شود.

---

1 global bucket shared
2 burst of traffic
3 request
4 bucket size
5 refill rate
6 first-in-first-out

- درخواست‌ها از صف بیرون کشیده می‌شوند و در فواصل زمانی معین پردازش می‌شوند.

شکل ۷-۴ نحوه عملکرد الگوریتم را توضیح می‌دهد.



شکل ۷-۴

الگوریتم Leaking bucket دو پارامتر زیر را می‌گیرد:

• **اندازه سطل**[1]: برابر با اندازه صف است. صف درخواست‌ها را برای پردازش با نرخ ثابت نگه می‌دارد.

• **نرخ خروجی**[2]: تعیین می‌کند که چه تعداد درخواست را می‌توان با نرخ ثابت، معمولاً در چند ثانیه را پردازش کرد.

Shopify که یک شرکت تجارت الکترونیکی است، از Leaking bucket برای محدودکردن نرخ استفاده می‌کند [۷].

مزایا و معایب این الگوریتم به شرح زیر است:

مزایا:

- باتوجه‌به اندازه محدود صف، میزان مصرف حافظه مناسب است.
- درخواست‌ها با نرخ ثابت پردازش می‌شوند، بنابراین برای موارد استفاده‌ای که نرخ خروجی پایدار موردنیاز است، این روش مناسب است.

1 Outflow rate
2 Outflow rate

معایب:

- انبوهی از ترافیک لحظه‌ای[1]، صف را با درخواست‌های قدیمی پر می‌کند و اگر آن درخواست‌ها به‌موقع پردازش نشوند، درخواست‌های جدیدتر دچار rate limit خواهند شد.
- دو پارامتری که در این الگوریتم وجود دارد (نرخ خروجی و اندازه سطل) و ممکن است تنظیم صحیح آنها آسان نباشد.

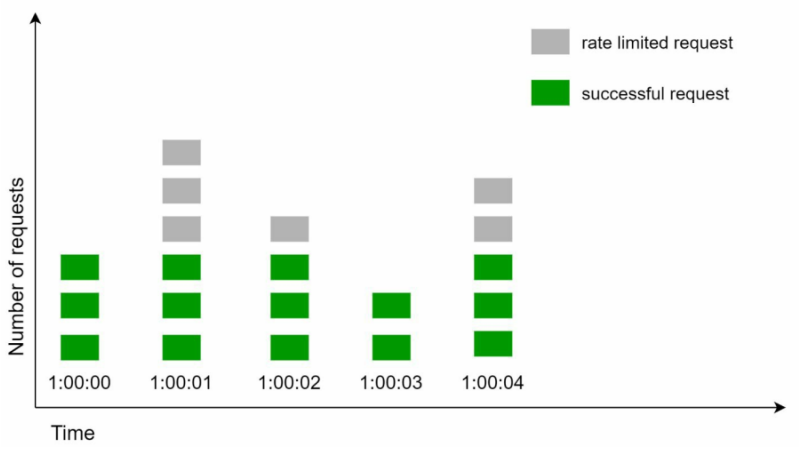
الگوریتم شمارنده پنجره ثابت (Fixed window counter algorithm)

الگوریتم شمارنده پنجره ثابت[2] به شرح زیر عمل می‌کند:

- این الگوریتم جدول زمانی[3] را به پنجره‌های زمانی[4] با اندازه ثابت تقسیم می‌کند و برای هر پنجره یک شمارنده اختصاص می‌دهد.
- هر درخواست شمارنده را یک عدد افزایش می‌دهد.
- هنگامی که شمارنده به آستانه از پیش تعریف شده رسید، درخواست‌های جدید حذف می‌شوند تا زمانی که یک پنجره زمانی جدید شروع شود.

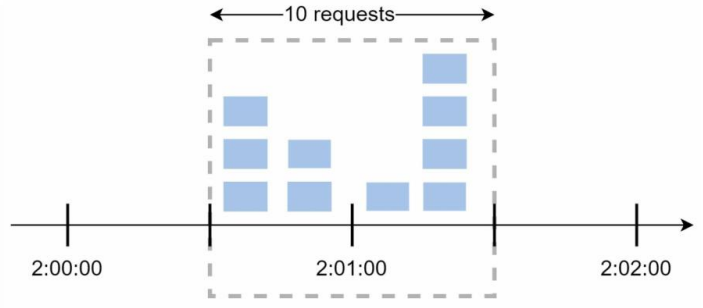
اجازه دهید از یک مثال عینی استفاده کنیم تا ببینیم چگونه کار می‌کند. در شکل ۸-۴ واحد زمان بر حسب ۱ ثانیه است و سیستم اجازه حداکثر ۳ درخواست در ثانیه را می‌دهد. در هر پنجره یک‌ثانیه‌ای، اگر بیش از ۳ درخواست دریافت شود آنگاه درخواست‌های اضافی همان‌طور که در شکل ۸-۴ نشان‌داده‌شده است، حذف می‌شوند.

1 burst of traffic
2 Fixed window counter algorithm
3 timeline
4 time windows



شکل ۸-۴

مشکل اصلی این الگوریتم این است که یک حجم ترافیکی شدید در لبه‌های مربوط به پنجره‌های زمانی می‌تواند باعث درخواست‌هایی بیشتر از حدِ مجاز شود. حالا بیایید به یک مورد خاص برای درک بهتر این موضوع نگاه کنیم:



شکل ۹-۴

در شکل ۹-۴، سیستم حداکثر ۵ درخواست در دقیقه را مجاز می‌سازد و مقدار باقی‌مانده بر حسب دقیقه و به‌صورت یک عدد گِرد شده، بازنشانی می‌شود. همان‌طور که مشاهده شد، پنج درخواست بین ساعت ۲:۰۰:۰۰ تا ۲:۰۱:۰۰ و پنج درخواست دیگر بین ساعت ۲:۰۱:۰۰ تا ۲:۰۲:۰۰ وجود دارد. برای پنجره به طول زمانی یک دقیقه‌ای بین ساعت ۲:۰۰:۳۰ تا ۲:۰۱:۳۰، حدود ۱۰ درخواست ارسال می‌شود. این دوبرابر درخواست‌های مجاز است.

مزایا:

- مصرف منابع حافظه‌ای مناسب است.
- درک آسان عملکرد.
- قابلیت بازنشانی مقدار باقی‌مانده در پایان یک واحد از پنجره زمانی که مناسب برای استفاده در حالت‌های خاص است.

معایب:

- افزایش ترافیک لحظه‌ای در لبه‌های پنجره می‌تواند باعث درخواست‌های بیشتر از حد مجاز شود.

الگوریتم لاگ پنجره کشویی

همان‌طور که قبلاً بحث شد، الگوریتم شمارنده پنجره ثابت[1] یک مشکل اساسی دارد: در واقع این الگوریتم اجازه می‌دهد تا درخواست‌های بیشتری در لبه‌های یک پنجره انجام شود. الگوریتم لاگ پنجره کشویی[2] این مشکل را برطرف می‌کند و به‌صورت زیر عمل می‌کند:

- این الگوریتم timestamp[3]های یک درخواست را پیگیری می‌کند. داده‌های timestamp معمولاً در حافظه کَش ذخیره می‌شوند، مثل مجموعه‌هایِ مرتب شده[4] در Redis [۸].
- وقتی درخواست جدیدی وارد شد، تمام timestampهای قدیمی را حذف کنید. Timestampهای قدیمی به Timestampهایی گفته می‌شود که مربوط به پنجره زمانی قبل از پنجره جاری باشند.
- timestamp مربوط به درخواست جدید را به لاگ اضافه کنید.
- اگر اندازه لاگ برابر یا کمتر از تعداد مجاز درخواست‌ها باشد، درخواست پذیرفته می‌شود. در غیر این صورت درخواست رد می‌شود.

---

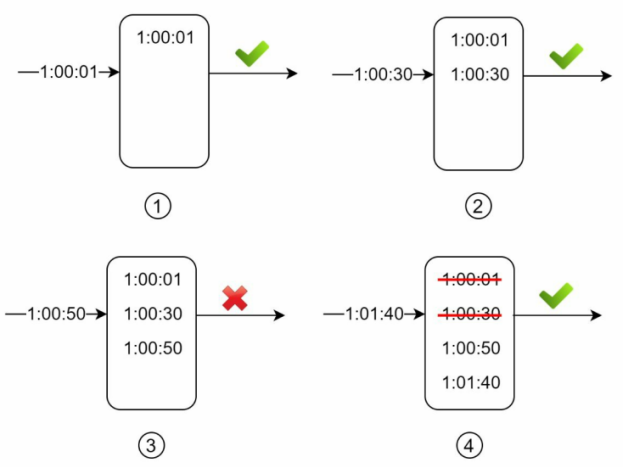
1 fixed window counter

2 Sliding window log algorithm

۳- برچسب زمان یا Timestamp یک توالی از نویسه‌ها یا اطلاعات رمزگذاری شده‌است که هنگام وقوع یک رویداد خاص مشخص می‌گردد که معمولاً شامل تاریخ و ساعت روز یا بخش دقیقی از یک بازه زمانی (ثانیه) خاص می‌شود.

4 sorted sets

ما الگوریتم را با مثالی که در شکل ۱۰-۴ نشان‌داده‌شده است توضیح می‌دهیم.



شکل ۱۰-۴

در این مثال، rate limiter اجازه ۲ درخواست در دقیقه را می‌دهد. معمولاً timestampهای لینوکس به‌صورت لاگ ذخیره می‌شوند. بااین‌حال، نمایش زمان قابل خواندن توسط انسان در مثال ما برای خوانایی بهتر استفاده شده است.

- هنگامی که یک درخواست جدید در ساعت ۱:۰۰:۰۱ می‌رسد، لاگ خالی است. بنابراین، درخواست مجاز است.
- یک درخواست جدید در ساعت ۱:۰۰:۳۰ می‌رسد، timestamp در ۱:۰۰:۳۰ در لاگ درج می‌شود. پس از درج، اندازه لاگ برابر ۲ شده است و بزرگتر از تعداد مجاز نیست. بنابراین، درخواست مجاز است.
- یک درخواست جدید در ساعت ۱:۰۰:۵۰ می‌رسد و timestamp در لاگ درج می‌شود. پس از درج، اندازه لاگ برابر ۳ شده است و این مقدار بزرگ‌تر از اندازه مجاز ۲ که در نظر گرفته شده بوده است؛ بنابراین این درخواست رد می‌شود حتی اگر timestamp در لاگ باقی بماند.
- یک درخواست جدید در ساعت ۱:۰۱:۴۰ می‌رسد. درخواست‌هایی در محدوده (۱:۰۰:۴۰ الی ۱:۰۱:۴۰) در آخرین بازه زمانی ممکن هستند، اما درخواست‌هایی که

قبل از ساعت ۱:۰۰:۴۰ ارسال می‌شوند قدیمی هستند. دو timestamp قدیمی، ۱:۰۰:۰۱ و ۱:۰۰:۳۰ از لاگ حذف می‌شوند. پس از عملیات حذف، اندازه لاگ ۲ می‌شود؛ بنابراین درخواست پذیرفته می‌شود.

مزایا:

- طراحی Rate limit اجرا شده توسط این الگوریتم بسیار دقیق است. در هر پنجره متحرک[1]، درخواست‌ها از حد مجاز تجاوز نمی‌کنند.

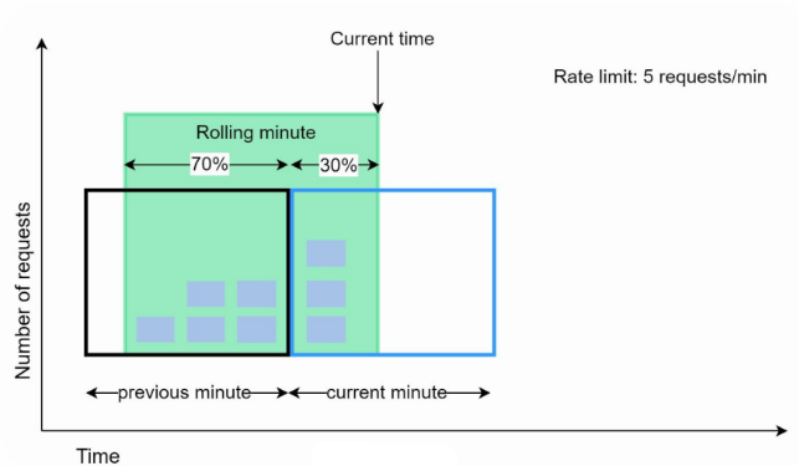
معایب:

- الگوریتم حافظه زیادی مصرف می‌کند؛ زیرا حتی اگر درخواستی رد شود، ممکن است timestamp آن همچنان در حافظه ذخیره شود.

الگوریتم شمارنده پنجره کشویی

الگوریتم شمارنده پنجره کشویی یا Sliding window counter یک رویکرد ترکیبی است که شمارنده پنجره ثابت[2] و لاگ پنجره کشویی[3] را ترکیب می‌کند. این الگوریتم را می‌توان با دو رویکرد متفاوت پیاده‌سازی کرد. ما در این بخش یکی از پیاده‌سازی‌ها را توضیح خواهیم داد و در پایانِ بخش مرجعی را برای اجزای دیگر ارائه می‌دهیم. شکل ۱۱-۴ نحوه عملکرد این الگوریتم را نشان می‌دهد.

1 rolling window
2 fixed window counter
3 sliding window log



شکل ۱۱-۴

فرض کنید rate limiter حداکثر ۷ درخواست در دقیقه را مجاز می‌کند و ۵ درخواست در دقیقه قبل و ۳ درخواست در دقیقه فعلی وجود دارد. برای درخواست جدیدی که در دقیقه کنونی به موقعیت ۳۰٪ می‌رسد، تعداد درخواست‌ها در پنجره چرخشی[1] با استفاده از فرمول زیر محاسبه می‌شود:

- درخواست‌ها در پنجره فعلی + درخواست‌ها در پنجره قبلی × درصد همپوشانی پنجره چرخشی و پنجره قبلی.
- با استفاده از این فرمول: ۶.۵ = ۰.۷٪ × ۵ + ۳ که معادل ۶.۵ درخواست است را دریافت می‌کنیم. بسته به مورد استفاده، عدد را می‌توان به بالا یا پایین گرد کرد. در مثال ما، به ۶ گرد شده است.

از آنجایی که rate limiter حداکثر اجازه ۷ درخواست در دقیقه را می‌دهد، درخواست فعلی می‌تواند انجام شود. بااین‌حال، پس از دریافت یک درخواست دیگر، به حد مجاز خواهد رسید. به دلیل محدودیت‌های کتاب، در اینجا به اجزای دیگر نمی‌پردازیم. خوانندگان علاقه‌مند باید به مطالب مرجع [۹] مراجعه کنند. این الگوریتم کامل نیست. مزایا و معایب دارد.

1 rolling window

مزایا

- افزایش لحظه‌ای ترافیک[1] را کاهش می‌دهد؛ زیرا نرخ بر اساس میانگین نرخ پنجره قبلی است.
- مصرف مناسب منابع حافظه.

معایب

- این روش تنها برای پنجره زمانی که در آن نگاه به درخواست‌های گذشته[2] که خیلی سخت‌گیرانه نباشد، کاربرد دارد. این روش تقریبی از نرخ واقعی درخواست‌ها ارائه می‌دهد زیرا فرض می‌کند درخواست‌ها در پنجره زمانی قبلی به طور مساوی توزیع شده‌اند. بااین‌حال، این مشکل ممکن است به آن بدی که به نظر می‌رسد نباشد. طبق آزمایش‌هایی که توسط CloudFlare انجام شده است، تنها ۰/۰۰۳ درصد از درخواست‌ها در میان ۴۰۰ میلیون درخواست به‌اشتباه مجاز یا محدود شده‌اند [۱۰].

معماری سطح پیشرفته

ایده اصلی الگوریتم‌های rate limiter ساده است. در سطح بالا، ما به یک شمارنده برای پیگیری تعداد درخواست‌های ارسال شده از یک کاربر، آدرس IP و غیره نیاز داریم. اگر شمارنده بزرگ‌تر از حد مجاز باشد، پس درخواست غیرمجاز است.

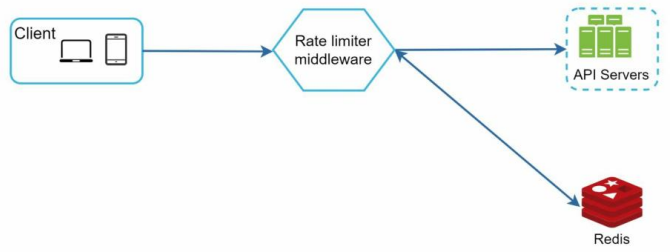
شمارنده‌ها را در کجا ذخیره کنیم؟ استفاده از پایگاه‌داده به دلیل کندی دسترسی به دیسک ایده خوبی نیست. کش درون حافظه[3] به این دلیل انتخاب شده است که سریع است و از استراتژی انقضا مبتنی بر زمان[4] پشتیبانی می‌کند. به‌عنوان‌مثال، Redis [۱۱] یک گزینه محبوب برای اجرای محدودیت نرخ است. Redis یک حافظه ذخیره‌سازی است که دو دستور را ارائه می‌دهد: INCR و EXPIRE

1 spikes in traffic
2 look back window
3 In-memory cache
4 time-based expiration strategy

• **افزایش مقدار صحیح کلید (INCR)**: مقدار شمارنده ذخیره شده را به‌اندازه ۱ افزایش می‌دهد.

• **تاریخ انقضا (EXPIRE)**: برای شمارنده زمان انقضا تعیین می‌کند. اگر زمان انقضا به پایان برسد، شمارنده به طور خودکار حذف می‌شود.

شکل ۱۲-۴ معماری سطح بالا برای rate limiting را نشان می‌دهد و این کار به شرح زیر است:



شکل ۱۲-۴

• کلاینت درخواستی را به میان‌افزار محدودی کننده نرخ[1] می‌فرستد.

• میان‌افزار محدودی کننده نرخ، شمارنده را از سطل[2] مربوطه در Redis واکشی کرده و سپس آن را بررسی می‌کند که آیا به حد مجاز رسیده است یا خیر.

• در صورت رسیدن به حد مجاز، درخواست رد می‌شود.

• در صورت عدم رسیدن به حد مجاز، درخواست به سرورهای API ارسال می‌شود. در همین حال، سیستم شمارنده را افزایش می‌دهد و آن را در Redis ذخیره می‌کند.

مرحله ۳ - عمیق‌شدن در طراحی

طراحی سطح بالا موجود در شکل ۱۲-۴ به سؤالات زیر پاسخ نمی‌دهد:

• قوانین[3] rate limiter چگونه ایجاد می‌شوند؟ قوانین کجا ذخیره می‌شوند؟

---

1 Rate Limiting Middleware
2 bucket
3 rules

• چگونه به درخواست‌هایی رسیدگی کنیم که rate-limit شده‌اند؟

در این بخش ابتدا به سؤالات مربوط به قوانین rate limiter پاسخ می‌دهیم و سپس به بررسی استراتژی‌های رسیدگی به درخواست‌های rate-limit شده می‌پردازیم. در نهایت، ما محدودیت نرخ در محیط توزیع شده با قابلیت‌هایی مثل طراحی دقیق، بهینه‌سازی عملکردی و نظارت[1] را مورد بحث قرار خواهیم داد.

قوانین Rate limiting

Lyft[2] اجزای rate limiter خود را به‌صورت منبع‌باز [۱۲] قرار داده است. ما به داخل مؤلفه و اجزای آن نگاه می‌کنیم و به چند نمونه از قوانین rate limiter نگاه می‌کنیم:

```
domain: messaging
descriptors:
 - key: message_type
  Value: marketing
  rate_limit:
    unit: day
    requests_per_unit: 5
```

در مثال بالا، سیستم به‌گونه‌ای پیکربندی شده است که حداکثر ۵ پیام بازاریابی در روز را مجاز می‌کند. در اینجا یک مثال دیگر وجود دارد:

```
domain: auth
descriptors:
- key: auth_type
  Value: login
  rate_limit:
    unit: minute
    requests_per_unit: 5
```

1 monitoring

2 یک شرکت تاکسی اینترنتی است

این rule نشان می‌دهد که کاربرها مجاز به ورود[1] بیش از ۵ بار در ۱ دقیقه نیستند. Ruleها معمولاً در فایل‌های پیکربندی نوشته شده و روی دیسک ذخیره می‌شوند.

rate-limit بیش از حد مجاز

درصورتی‌که درخواستی دچار rate-limit شده باشد، APIها کد پاسخ HTTP 429 (درخواست‌های بسیار زیاد) را به کلاینت بر می‌گردانند. بسته به موارد استفاده، ممکن است درخواست‌هایی با نرخ محدود شده را در نوبت قرار دهیم تا بعداً پردازش شوند. برای مثال، اگر نرخ برخی از سفارش‌ها به دلیل بارگذاری بیش از حد سیستم محدود شده باشد، ممکن است آن سفارش‌ها را نگه داریم تا بعداً پردازش شوند.

هدرهای Rate limiter

کاربر چگونه متوجه می‌شود که در حال محدودشدن و rate-limit است؟ چگونه یک کاربر تعداد درخواست‌های باقیمانده مجاز را قبل از محدودشدن می‌داند؟ پاسخ در هدرهای پاسخ HTTP نهفته است. rate limiter هدرهای HTTP زیر را به کلاینت‌ها برمی‌گرداند:

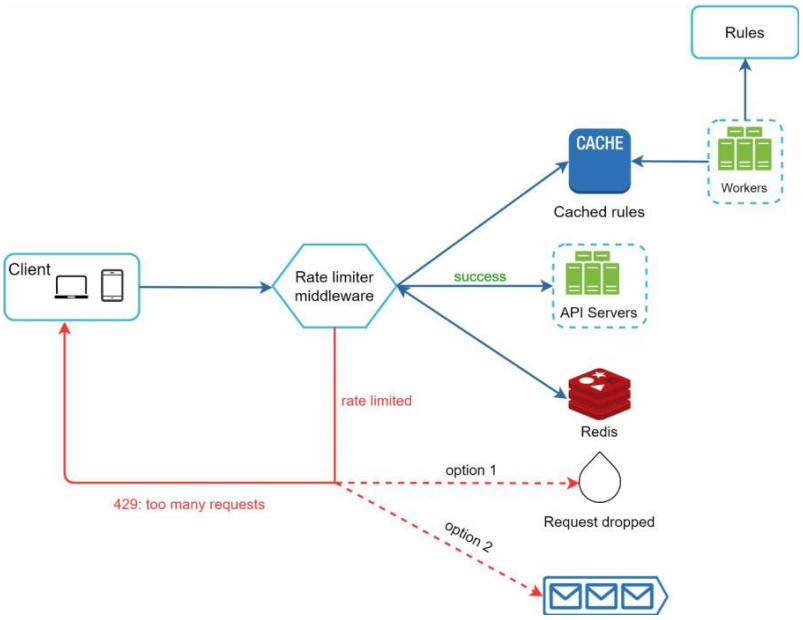
X-Ratelimit-Remaining: بیان‌کننده تعداد باقیمانده درخواست‌های مجاز در یک پنجره زمانی است.
X-Ratelimit-Limit: نشان می‌دهد که کاربر می‌تواند در هر پنجره زمانی چند تماس برقرار کند.
X-Ratelimit-Retry-After: تعداد ثانیه‌هایی که باید منتظر بمانید تا بتوانید دوباره درخواستی را بدون درنگ درخواست دهید. زمانی که کاربر درخواست‌های زیادی ارسال کرده باشد، خطای ۴۲۹ که به معنی درخواست بیش از حد مجاز است و یک هدر با کلیدواژه X-Ratelimit- Retry-After به کلاینت برگردانده می‌شود.

---

1 login

جزئیات طراحی

شکل ۱۳-۴ طراحی دقیق سیستم را نشان می‌دهد.



شکل ۱۳-۴

- قوانین یا ruleها روی دیسک ذخیره می‌شوند. Workerها اغلب قوانین را از دیسک بیرون می‌کشند و آنها را در حافظه کَش ذخیره می‌کنند.
- هنگامی که یک کلاینت درخواستی را به سرور ارسال می‌کند، ابتدا درخواست به Rate limiter middleware ارسال می‌شود.
- Rate limiter middleware قوانین را از حافظه کَش بارگذاری می‌کند و شمارنده‌ها به همراه آخرین زمان درخواست را از کش Redis واکشی می‌کند. بر اساس پاسخ، rate limiter تصمیم می‌گیرد:
- اگر درخواست rate-limit نشده نباشد، به سرورهای API ارسال می‌شود.
- اگر درخواست rate-limit شده باشد، rate limiter خطای ۴۲۹ که به معنی درخواست بیش از حد مجاز است را اعلام می‌کند. دراین‌بین، درخواست یا حذف می‌شود یا به صف جهت نوبت‌دهی ارسال می‌شود.

استفاده از محدودکننده نرخ در یک محیط توزیع شده

ساختن یک rate limiter که در یک محیط سرور که به‌تنهایی کار می‌کند، کار سختی نیست. بااین‌حال، مقیاس‌بندی[1] سیستم برای پشتیبانی از چندین سرور و threadهای هم‌زمان آن داستان متفاوتی است.

در این وضعیت دو چالش مهم وجود دارد:

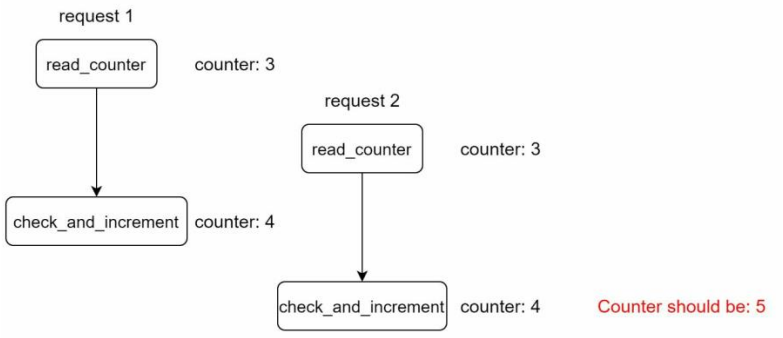
- وضعیت رقابتی[2]
- مشکل همگام‌سازی[3]

وضعیت رقابتی

همان‌طور که قبلاً بحث شد، rate limiter در سطح بالا به شرح زیر عمل می‌کند:

- مقدار شمارنده را از Redis بخواند.
- بررسی می‌کند که آیا (شمارنده + ۱) از حد آستانه فراتر رفته است یا خیر.
- اگر این‌طور نیست، مقدار شمارنده را ۱واحد در Redis افزایش دهید.

وضعیت رقابتی می‌تواند در یک محیط‌هایی با قابلیت بالایِ اجرایِ هم‌زمان[4] فرایندها اتفاق بیفتد همان‌طور که در شکل ۱۴-۴ نشان‌داده‌شده است.



شکل ۱۴-۴

1 scaling
2 Race condition
3 Synchronization
4 highly concurrent environment

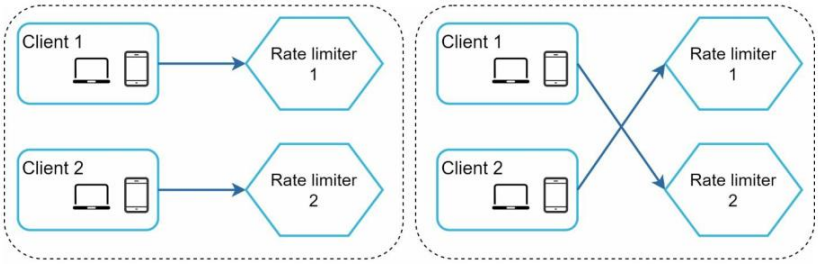
فرض کنید مقدار شمارنده در Redis برابر مقدار ۳ باشد. اگر دو درخواست هم‌زمان مقدار شمارنده را بخوانند قبل از اینکه هر یک از آنها مقدار را برگرداند، هر کدام شمارنده را برابر ۱ واحد افزایش می‌دهند و بدون بررسی thread دیگر آن را دوباره می‌نویسند. هر دو درخواست (threadها) معتقدند که مقدار شمارنده صحیح ۴ را دارند. بااین‌حال، مقدار شمارنده صحیح باید ۵ باشد.

قفل‌ها[1] واضح‌ترین راه‌حل برای حل وضعیت رقابتی هستند. بااین‌حال، قفل‌ها به طور قابل‌توجهی سرعت سیستم را کاهش می‌دهند. دو استراتژی معمولاً برای حل این مشکل استفاده می‌شود: اسکریپت‌نویسی با Lua [۱۳] و ساختار داده مجموعه‌های مرتب شده[2] در Redis در مرجع شماره [۸] موجود است. برای خوانندگان علاقه‌مند به این استراتژی‌ها، به منابع مربوطه [۸] [۱۳] مراجعه کنید.

مسائل هم‌زمانی

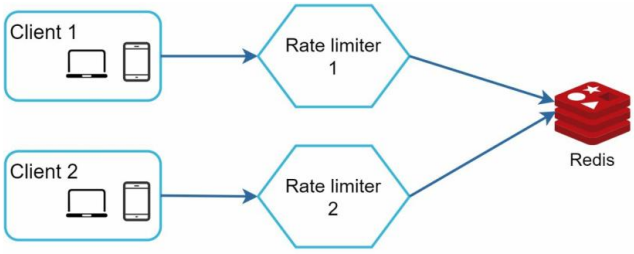
همگام‌سازی[3] عامل مهم دیگری است که در یک محیط توزیع شده باید در نظر گرفته شود. برای پشتیبانی از میلیون‌ها کاربر، یک سرور rate limiter به‌تنهایی ممکن است برای مدیریت ترافیک کافی نباشد. هنگامی که چندین سرور rate limiter استفاده می‌شود که درنتیجه همگام‌سازی بین آن‌ها لازم است. به‌عنوان‌مثال، در سمت چپ شکل ۱۵-۴، کلاینت ۱ درخواست‌ها را به rate limiter 1 ارسال می‌کند و کلاینت ۲ درخواست‌ها را به rate limiter 2 ارسال می‌کند. از آنجایی که لایه وب stateless است، کلاینت‌ها می‌توانند همان‌طور که در سمت راست شکل ۱۵-۴ نشان‌داده‌شده است درخواست‌ها را به یک rate limiter متفاوت ارسال کنند. اگر همگام‌سازی صورت نگیرد، rate limiter 1 حاوی هیچ داده‌ای درباره کلاینت ۲ نیست؛ بنابراین منطق rate limiter نمی‌تواند به‌درستی کار کند.

1 Locks
2 sorted sets data structure
3 Synchronization



شکل ۱۵-۴

یک راه‌حل ممکن استفاده از sticky sessions[1] است که به کلاینت اجازه می‌دهد ترافیک را به همان rate limiter موردنظر ارسال کند. این راه‌حل به‌هیچ‌وجه توصیه نمی‌شود؛ زیرا مقیاس‌پذیر و انعطاف‌پذیر نیست. یک رویکرد بهتر استفاده از ذخیره‌گاه‌های داده متمرکز[2] مانند Redis است. این طرح در شکل ۱۶-۴ نشان‌داده‌شده است.



شکل ۱۶-۴

بهینه‌سازی‌های عملکردی

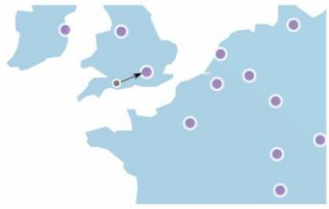
بهینه‌سازی عملکردی یک موضوع رایج در مصاحبه‌های طراحی سیستم است. ما دو حوزه را برای بهبود این مسئله پوشش خواهیم داد.

نخست، راه‌اندازی چند مرکز داده برای rate limiter بسیار مهم است، زیرا تأخیر زمانی بالایی برای کاربرانی که دور از مرکز داده هستند، انتظار می‌رود. اکثر ارائه‌دهندگان خدمات ابری مکان‌های سرور لبه[3] زیادی را در سراسر جهان ایجاد می‌کنند. به‌عنوان‌مثال، از تاریخ

---

1 جلسات چسبنده
2 centralized data stores
3 edge server

۲۰/۲۰۲۰/ ۵، Cloudflare دارای ۱۹۴ سرور لبه توزیع شده جغرافیایی است [۱۴]. ترافیک به طور خودکار به نزدیک‌ترین سرور لبه هدایت می‌شود تا تأخیر کاهش یابد.



شکل ۱۷-۴ – منبع [۱۰]

دوم، همگام‌سازی داده‌ها با یک مدل یکپارچگی تدریجی[1]. اگر در مورد مدل‌های سازگاری و یکپارچگی تدریجی چیزی نمی‌دانید، بهتر است به بخش «یکپارچگی یا consistency» در "فصل ۶: طراحی ذخیره‌ساز Key-value" مراجعه کنید.

نظارت (Monitoring)

پس از فعال‌سازی rate limiter در مکان موردنظر سیستم نرم‌افزاری، جمع‌آوری داده‌های تحلیلی برای بررسی اینکه آیا rate limiter کارآمد و تأثیرگذار است یا خیر، موردی مهم است. در درجه اول، ما می‌خواهیم مطمئن شویم:

• الگوریتم rate limiter مؤثر است.

• قوانین rate limiter مؤثر هستند.

برای مثال، اگر قوانین rate limiter بسیار سخت‌گیرانه و محدود باشند، بسیاری از درخواست‌های معتبر حذف می‌شوند. در این مورد، ما می‌خواهیم کمی از سخت‌گیری و پیچیدگی قوانین را کاهش دهیم. در مثالی دیگر، متوجه می‌شویم که rate limiter ما زمان‌هایی که ترافیک شدید و ناگهانی افزایش می‌یابد، ناکارآمد می‌شود. مانند فروش لحظه‌ای و با تخفیف یک محصول خاص (مثلاً حراج کفش یا حراج جمعه سیاه) که سیل عظیم درخواست‌ها روی سرور و اپلیکیشن ایجاد می‌شود. در این سناریو، ممکن است الگوریتمی را

1 eventual consistency

برای پشتیبانی از ترافیک شدید و لحظه‌ای[1] جایگزین کنیم. الگوریتم‌های Token bucket[2] در اینجا مناسب است.

مرحله ٤ - جمع‌بندی

در این فصل، الگوریتم‌های مختلف rate limiting و جوانب مثبت و منفی آن‌ها را مورد بحث قرار دادیم. الگوریتم‌های مورد بحث عبارت‌اند از:

- سطل توکن (Token bucket)
- سطل نشتی (Leaking bucket)
- پنجره ثابت (Fixed window)
- ورود به سیستم پنجره کشویی (Sliding window log)
- شمارنده پنجره کشویی (Sliding window counter)

سپس، معماری سیستم، rate limiter در یک محیط توزیع شده به همراه، بهینه‌سازی‌های عملکردی و نظارت[3] را مورد بحث قرار دادیم. مشابه هر سؤال مصاحبه طراحی سیستم، نکات دیگری نیز وجود دارد که می‌توانید در صورت اجازه به آنها اشاره کنید:

- rate limiting سخت در مقابل نرم.
  - **سخت**: تعداد درخواست‌ها نمی‌تواند از مقدار آستانه تجاوز کند.
  - **نرم**: درخواست‌ها می‌توانند برای مدت کوتاهی از مقدار آستانه تجاوز کنند.
- **rate limiting در سطوح مختلف**. در این فصل، ما فقط در مورد محدودیت نرخ در سطح لایه برنامه (HTTP: لایه ۷) صحبت کردیم. امکان اعمال محدودیت نرخ در سایر لایه‌ها در سطح شبکه وجود دارد. به‌عنوان‌مثال، شما می‌توانید با استفاده از Iptables[4] [۱۵] (IP: لایه ۳)، محدودیت نرخ را به آدرس‌های IP اعمال کنید. توجه

---

1 burst traffic

2 سطل/پیمانه توکن

3 monitoring

4 جداول IPها

کنید که مدل اتصال سیستم‌های باز (مدل OSI) دارای ۷ لایه است [۱۶]: لایه ۱: لایه فیزیکی، لایه ۲: لایه Data link، لایه ۳: لایه شبکه، لایه ۴: لایه Transport، لایه ۵: لایه Session، لایه ۶: لایه Presentation، لایه ۷: لایه Application.

- بهتر است تا جای ممکن از محدودیت نرخ[1] اجتناب کنید. با رعایت بهترین شیوه‌ها، کلاینت خود را طراحی کنید
  - برای جلوگیری از برقراری تماس‌های مکرر با API سمت سرور از حافظه کَش مربوط به کلاینت استفاده کنید.
  - محدودیت‌ها را درک کنید و درخواست‌های زیادی را در یک بازه زمانی کوتاه ارسال نکنید.
  - کدی را برای یافتن استثناها یا خطاها اضافه کنید تا کلاینت شما بتواند به‌خوبی بتواند خطاها را بازیابی کند.
  - اضافه‌کردن زمان بازگشت به‌اندازه کافی برای امتحان مجدد منطق[2] پیاده‌سازی شده را در نظر بگیرید.

1 Rate Limiting
2 retry logic

منابع و مراجع فصل ٤

[1] Rate-limiting strategies and techniques: https://cloud.google.com/solutions/rate-limitingstrategies-techniques
[2] Twitter rate limits: https://developer.twitter.com/en/docs/basics/rate-limits
[3] Google docs usage limits: https://developers.google.com/docs/api/limits
[4] IBM microservices: https://www.ibm.com/cloud/learn/microservices
[5] Throttle API requests for better throughput: https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-requestthrottling.html
[6] Stripe rate limiters: https://stripe.com/blog/rate-limiters
[7] Shopify REST Admin API rate limits: https://help.shopify.com/en/api/reference/restadmin-api-rate-limits
[8] Better Rate Limiting With Redis Sorted Sets: https://engineering.classdojo.com/blog/2015/02/06/rolling-rate-limiter/
[9] System Design — Rate limiter and Data modelling: https://medium.com/@saisandeepmopuri/system-design-rate-limiter-and-data-modelling- 9304b0d18250
[10] How we built rate limiting capable of scaling to millions of domains: https://blog.cloudflare.com/counting-things-a-lot-of-different-things/
[11] Redis website: https://redis.io/
[12] Lyft rate limiting: https://github.com/lyft/ratelimit
[13] Scaling your API with rate limiters: https://gist.github.com/ptarjan/e38f45f2dfe601419ca3af937fff574d#request-rate-limiter
[14] What is edge computing: https://www.cloudflare.com/learning/serverless/glossary/whatis-edge-computing/
[15] Rate Limit Requests with Iptables: https://blog.programster.org/rate-limit-requests-withiptables
[16] OSI model: https://en.wikipedia.org/wiki/OSI_model#Layer_architecture

فصل ۵

طراحی سیستم هش مداوم

برای دستیابی به مقیاس‌دهی افقی یا horizontal scaling مناسب، توزیع requests/data[۱] به طور مؤثر و یکنواخت در بین سرورها مهم است. Consistent hashing یا هش مداوم یک تکنیک رایج برای دستیابی به این هدف است. اما ابتدا اجازه دهید نگاهی عمیق به مشکل بیندازیم.

مشکل مجدد هش‌کردن[۲]

اگر n سرور کَش[۳] دارید، یک راه متداول برای توزیع بار[۴] استفاده از روش هش زیر است:

$$serverIndex = hash(key) \% N, where\ N\ is\ the\ size\ of\ the\ server\ pool.$$

که N برابر تعداد سرورهای مورد استفاده برای هش است.

اجازه دهید از یک مثال برای توضیح نحوه عملکرد آن استفاده کنیم. همان‌طور که در جدول ۵-۱ نشان‌داده‌شده است، ما ۴ سرور و ۸ کلیدهای رشته‌ای[۵] به همراه هش آنها داریم.

---

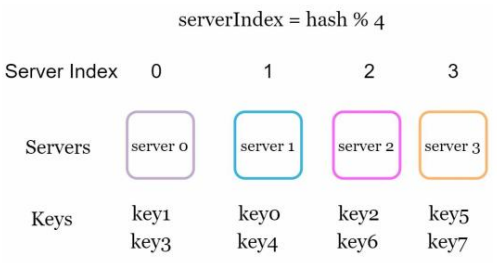
۱ داده/درخواست‌ها

2 rehashing
3 cache
4 balance the load
5 string keys

key	hash	hash % 4
key0	18358617	1
key1	26143584	0
key2	18131146	2
key3	35863496	0
key4	34085809	1
key5	27581703	3
key6	38164978	2
key7	22530351	3

جدول ۱-۵

برای واکشی سروری که یک کلید در آن ذخیره شده است، عملیات ماژول شده $f(key)\ \%\ 4$ را انجام می‌دهیم. برای مثال، $hash(key0)\ \%\ 4\ =\ 1$ به این معنی است که یک کلاینت باید با سرور ۱ تماس بگیرد تا داده‌های کَش شده را واکشی کند. شکل ۱-۵ توزیع کلیدها را بر اساس جدول ۱-۵ نشان می‌دهد.



شکل ۱-۵

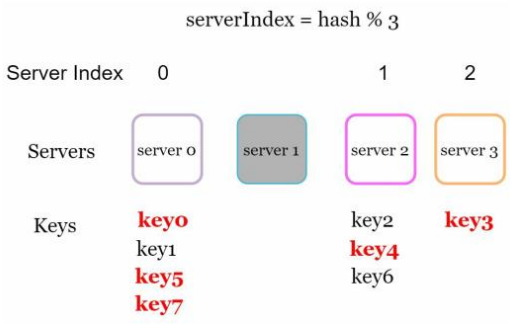
این رویکرد زمانی به‌خوبی کار می‌کند که اندازه مجموعه سرورها ثابت باشد و توزیع داده‌ها یکنواخت باشد. بااین‌حال، هنگامی که سرورهای جدید اضافه می‌شوند یا سرورهای موجود

حذف می‌شوند، مشکلات ایجاد می‌شود. به‌عنوان‌مثال، اگر سرور ۱ آفلاین شود، اندازه مخزن یا تعداد سرورها برابر ۳ می‌شود. با استفاده از همان تابع هش، مقدار هش یکسانی را برای یک کلید دریافت می‌کنیم. اما اعمال عملیات ماژولار به ما سرور ایندکس‌های[1] متفاوتی می‌دهد زیرا تعداد سرورها تا حد ۱ عدد کاهش می‌یابد. نتایجی را که در جدول ۲-۵ نشان‌داده‌شده است با اعمال هش ٪۳ مقداری برابر جدول زیر را دریافت می‌کنیم:

key	Hash	hash % 3
key0	18358617	0
key1	26143584	0
key2	18131146	1
key3	35863496	2
key4	34085809	1
key5	27581703	0
key6	38164978	1
key7	22530351	0

جدول ۲-۵

شکل ۲-۵ توزیع جدید کلیدها را بر اساس جدول ۲-۵ نشان می‌دهد.



شکل ۲-۵

1 indexes

همان‌طور که در شکل ۲-۵ نشان‌داده‌شده است، بیشتر کلیدها، نه فقط آنهایی که در ابتدا در سرور آفلاین (سرور ۱) ذخیره شده‌اند، دوباره توزیع می‌شوند. این بدان معنی است که وقتی سرور ۱ آفلاین می‌شود، اکثر کاربرهای cache برای واکشی داده‌ها به سرورهای اشتباهی متصل می‌شوند. این باعث طوفانی از فقدان حافظه کَش می‌شود. هش کردن مداوم[1] یک تکنیک مؤثر برای کاهش این مشکل است.

هش مداوم (Consistent hashing)

به نقل از ویکی‌پدیا: هش مداوم نوع خاصی از درهم‌سازی است، به‌طوری که وقتی اندازه جدول هش تغییر می‌کند و از هش مداوم استفاده می‌شود، تنها کلیدهای k/n باید به طور متوسط دوباره نگاشت[2] شوند، جایی که k برابر تعداد کلیدها است و n برابر تعداد خانه یا اسلات‌ها[3] است. در مقابل، در بیشتر جداول هش سنتی، تغییر در تعداد اسلات‌های آرایه باعث می‌شود تقریباً همه کلیدها دوباره نگاشت شوند [۱].

فضا و حلقه هش

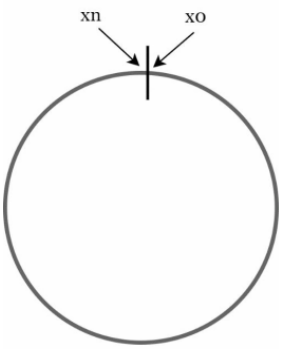
اکنون ما تعریف هش مداوم را درک می‌کنیم، اجازه دهید بفهمیم که چگونه کار می‌کند. فرض کنید SHA-1 به‌عنوان تابع هش f استفاده می‌شود و محدوده خروجی تابع هش عبارت است از: x0، x1، x2، x3، ...، xn. در رمزنگاری، فضای هش SHA-1 از ۰ به (۱-(۲^۱۶۰)) می‌رود. این بدان معناست که x0 مربوط به ۰ است و xn مربوط به (۱-(۲^۱۶۰)) است پس همه مقادیر هش دیگر در بین ۰ و (۱-(۲^۱۶۰)) قرار می‌گیرند.

1 Consistent hashing
2 remapped
3 slots

شکل ۳-۵ فضای هش را نشان می‌دهد.

شکل ۳-۵

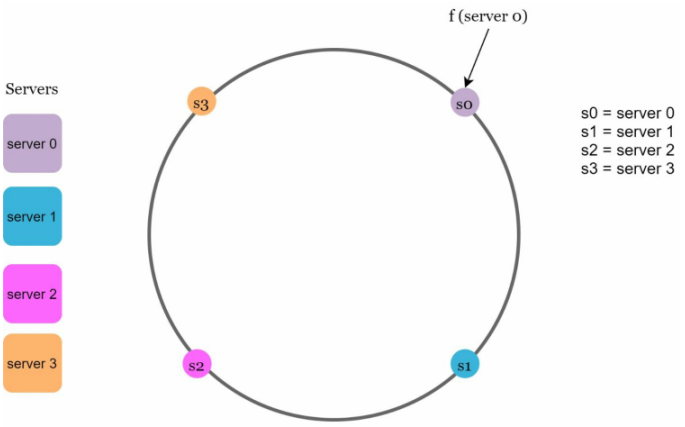
با جمع‌آوری هر دو سر منحنی، یک حلقه هش همان‌طور که در شکل ۴-۵ نشان‌داده‌شده را دریافت می‌کنیم:



شکل ۴-۵

سرورهای هش

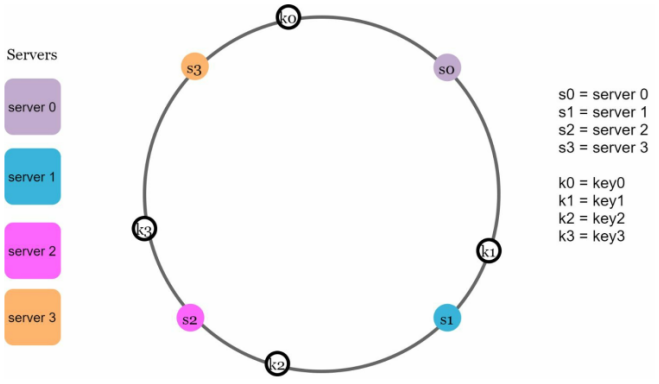
با استفاده از همان تابع هش f، سرورها را بر اساس IP یا نام سرور بر روی حلقه نگاشت می‌کنیم. شکل ۵-۵ نشان می‌دهد که ۴ سرور بر روی حلقه هش نگاشت شده‌اند.



شکل ۵-۵

کلیدهای هش

بدترین نکته‌ای که قابل‌ذکر است این است که تابع هش استفاده شده در اینجا با «مشکل هش‌کردن مجدد[1]» متفاوت است و هیچ عملیات ماژول شده‌ای وجود ندارد. همان‌طور که در شکل ۶-۵ نشان‌داده‌شده است، ۴ کلید حافظه کَش (key0، key1، key2 و key3) روی حلقه هش قرار می‌گیرند.

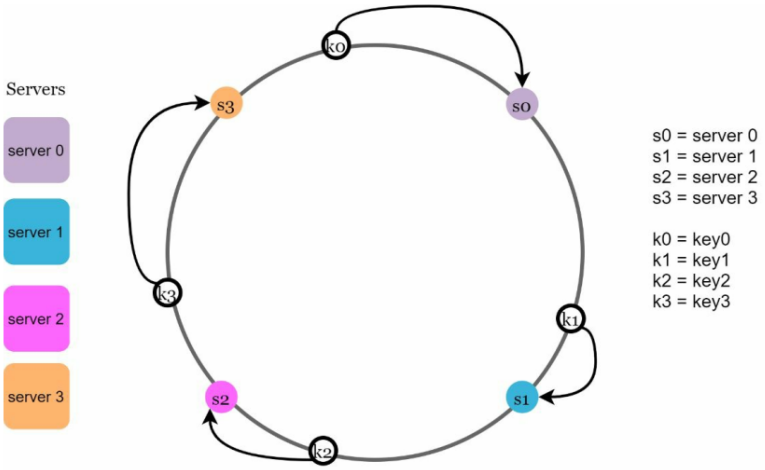


شکل ۶-۵

1 the rehashing problem

جستجوی سرور – Server lookup

برای تعیین اینکه یک کلید در کدام سرور ذخیره شده است، از موقعیت کنونی کلید روی حلقه در جهت عقربه‌های ساعت حرکت می‌کنیم تا سرور پیدا شود. شکل ۷-۵ این فرایند را توضیح می‌دهد. در جهت عقربه‌های ساعت، key0 در سرور ۰ ذخیره می‌شود. key1 در سرور ۱ ذخیره می‌شود. key2 در سرور ۲ و key3 در سرور ۳ ذخیره می‌شود.

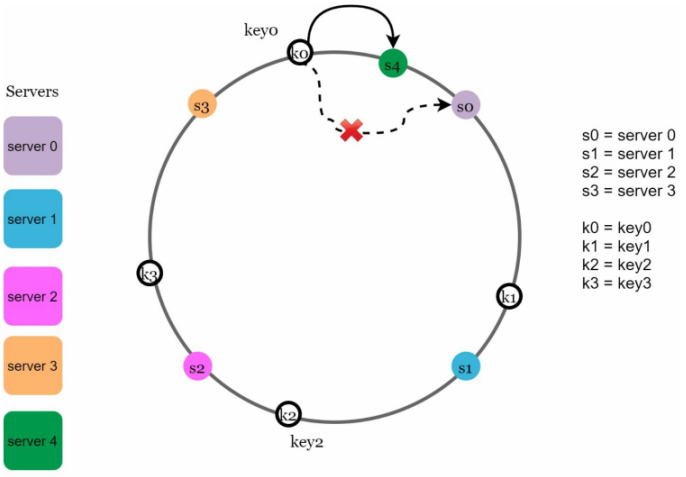


شکل ۷-۵

اضافه‌کردن یک سرور

با استفاده از منطقی که در بالا توضیح داده شد، افزودن یک سرور جدید فقط به توزیع مجدد کسری[1] از کلیدها نیاز دارد. در شکل ۸-۵، پس از اضافه شدن یک سرور جدید به‌عنوان‌مثال سرور شماره ۴، تنها key0 نیاز به توزیع مجدد دارد. k1، k2 و k3 در همان سرورها باقی می‌مانند. بیایید نگاه دقیقی به این منطق داشته باشیم. قبل از اضافه شدن سرور ۴، key0 در سرور ۰ ذخیره می‌شود. اکنون، key0 در سرور ۴ ذخیره می‌شود زیرا سرور ۴ اولین سروری است که با رفتن در جهت عقربه‌های ساعت از موقعیت key0 روی حلقه با آن روبرو می‌شود. کلیدهای دیگر بر اساس الگوریتم هش مداوم توزیع مجدد نمی‌شوند.

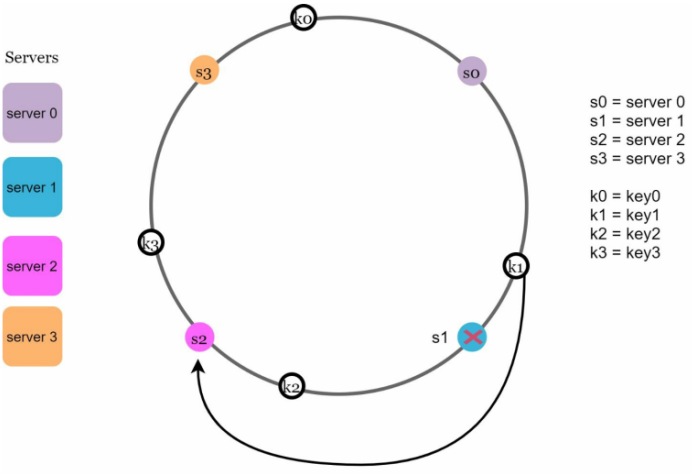
1 fraction



شکل ۸-۵

حذف یک سرور

هنگامی که یک سرور حذف می‌شود، تنها بخش کوچکی از کلیدها نیاز به توزیع مجدد با روش هش مداوم دارند. در شکل ۹-۵، هنگامی که سرور ۱ حذف می‌شود، فقط کلید ۱ باید به سرور ۲ نگاشت شود. بقیه کلیدها تحت تأثیر قرار نمی‌گیرند.



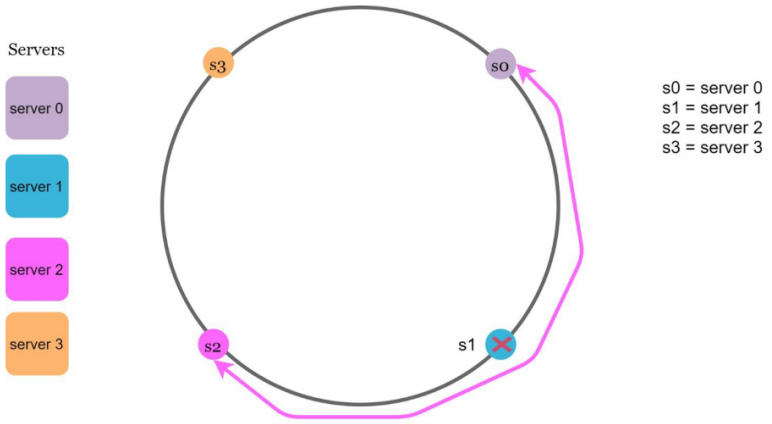
شکل ۹-۵

دو مبحث اساسی در این رویکرد

الگوریتم هش مداوم توسط Karger و همکاران در MIT [۱] معرفی شد. مراحل اساسی آن عبارت‌اند از:

- نگاشت[1] سرورها و کلیدها روی حلقه با استفاده از یک تابع هش توزیع شده یکنواخت انجام شود.
- برای اینکه بفهمید یک کلید به کدام سرور نگاشت شده است، از موقعیت کلید در جهت عقربه‌های ساعت حرکت کنید تا اولین سرور روی حلقه پیدا شود.

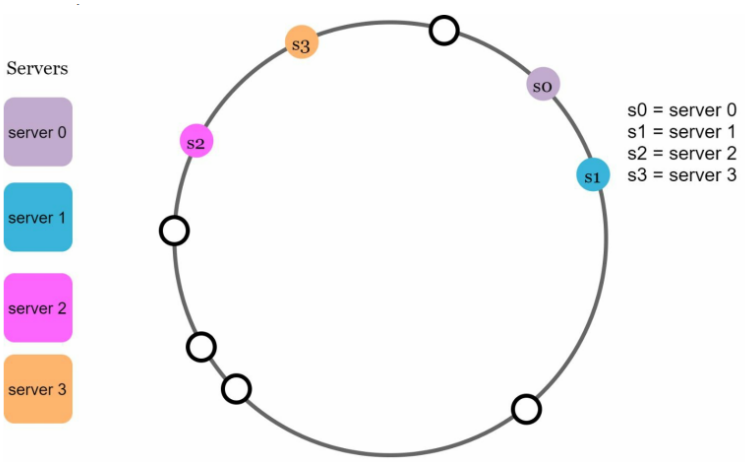
دو مشکل با این رویکرد شناسایی می‌شود. **نخست**، غیرممکن است که اندازه یکسانی از پارتیشن‌ها را روی حلقه برای همه سرورها نگه دارید، باتوجه‌به اینکه یک سرور می‌تواند اضافه یا حذف شود و پارتیشن فضای هش[2] بین سرورهای مجاور است. پس این امکان وجود دارد که اندازه پارتیشن‌های موجود در حلقه اختصاص‌داده‌شده به هر سرور بسیار کوچک یا نسبتاً بزرگ باشد. در شکل ۱۰-۵، اگر s1 حذف شود، پارتیشن s2 (که با فلش‌های دو طرفه برجسته شده است) دو برابر بزرگتر از پارتیشن s0 و s3 است.



شکل ۱۰-۵

1 Map
2 hash space

دوم، امکان توزیع کلید غیریکنواخت روی حلقه وجود دارد. به‌عنوان‌مثال، اگر سرورها به موقعیت‌های فهرست شده در شکل ۱۱-۵ نگاشت شوند، بیشتر کلیدها در سرور ۲ ذخیره می‌شوند. بااین‌حال، سرور ۱ و سرور ۳ هیچ داده‌ای ندارند.



شکل ۱۱-۵

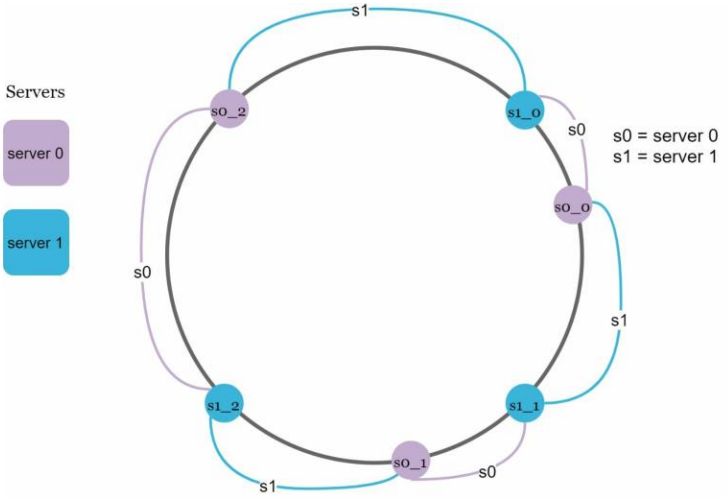
برای حل این مشکلات از تکنیکی به نام گره‌های مجازی[1] یا تکثیر[2] استفاده می‌شود.

گره مجازی (Virtual nodes)

یک گره مجازی به گره واقعی اشاره دارد و هر سرور با چندین گره مجازی روی حلقه نشان داده می‌شود. در شکل ۱۲-۵، هر دو سرور ۰ و سرور ۱ دارای ۳ گره مجازی هستند. مقدار ۳ به طور دلخواه انتخاب شده است و در سیستم‌های موجود در دنیای واقعی، تعداد گره‌های مجازی بسیار بیشتر است. به جای استفاده از s0، ما s0_0، s0_1 و s0_2 را برای نشان دادن سرور ۰ در حلقه داریم. به طور مشابه، s1_0، s1_1 و s1_2 نشان دهنده سرور ۱ در حلقه هستند. با گره‌های مجازی، هر سرور مسئول پارتیشن‌های متعدد است. پارتیشن‌ها (لبه‌ها) با

1 Virtual nodes
2 replica

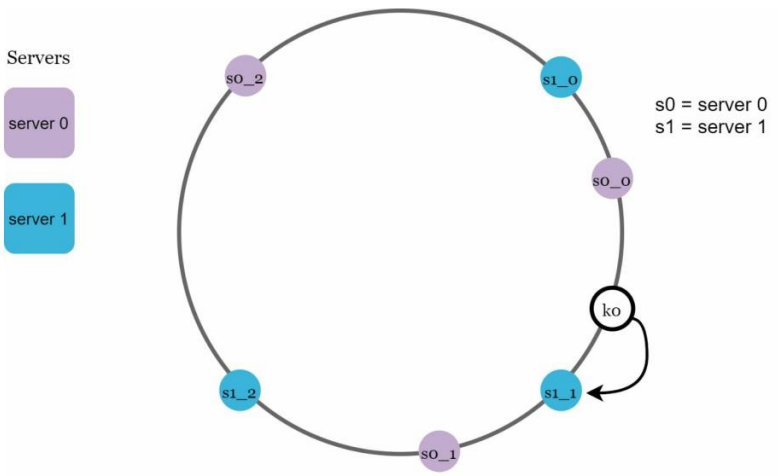
برچسب[1] s0 توسط سرور ۰ مدیریت می‌شوند. از طرف دیگر پارتیشن‌های دارای برچسب s1 توسط سرور ۱ مدیریت می‌شوند.



شکل ۱۲-۵

برای اینکه بفهمیم یک کلید در کدام سرور ذخیره شده است، از محل کلید در جهت عقربه‌های ساعت حرکت می‌کنیم و اولین گره مجازی را که روی حلقه با آن مواجه می‌شویم پیدا می‌کنیم. در شکل ۱۳-۵، برای اینکه بفهمیم در کدام سرور k0 ذخیره شده است، از محل k0 در جهت عقربه‌های ساعت می‌رویم و گره مجازی s1_1 را پیدا می‌کنیم که به سرور ۱ اشاره دارد.

1 label



شکل ۱۳-۵

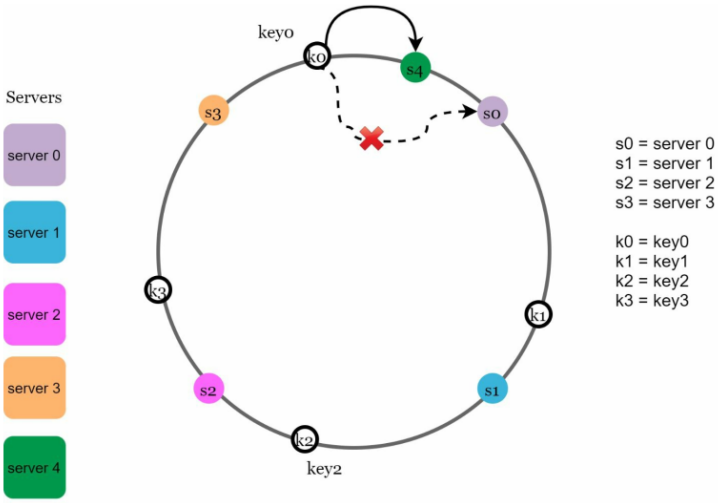
با افزایش تعداد گره‌های مجازی، توزیع کلیدها متعادل‌تر[1] می‌شود. به این دلیل که انحراف استاندارد[2] با افزایش گره‌های مجازی، کوچک‌تر می‌شود و منجر به توزیع متعادل داده می‌شود. انحراف استاندارد نحوه پخش داده‌ها را اندازه‌گیری می‌کند. نتیجه یک آزمایش انجام شده توسط تحقیقات آنلاین [۲] نشان می‌دهد که با یکصد یا دویست گره مجازی، انحراف معیار بین ۵٪ (برای ۲۰۰ گره مجازی) و ۱۰٪ (برای ۱۰۰ گره مجازی) از میانگین است. زمانی که تعداد گره‌های مجازی را افزایش دهیم، انحراف معیار کمتر خواهد بود. بااین‌حال، فضاهای بیشتری برای ذخیره داده‌های مربوط به گره‌های مجازی موردنیاز است. این یک مبادله[3] است و ما می‌توانیم تعداد گره‌های مجازی را متناسب با نیازهای سیستم خود تنظیم کنیم.

پیداکردن کلیدهای تأثیرپذیر

هنگامی که یک سرور اضافه یا حذف می‌شود، کسری از داده‌ها باید دوباره توزیع شود. چگونه می‌توانیم محدوده آسیب‌دیده را برای توزیع مجدد کلیدها پیدا کنیم؟ در شکل ۱۴-۵، سرور

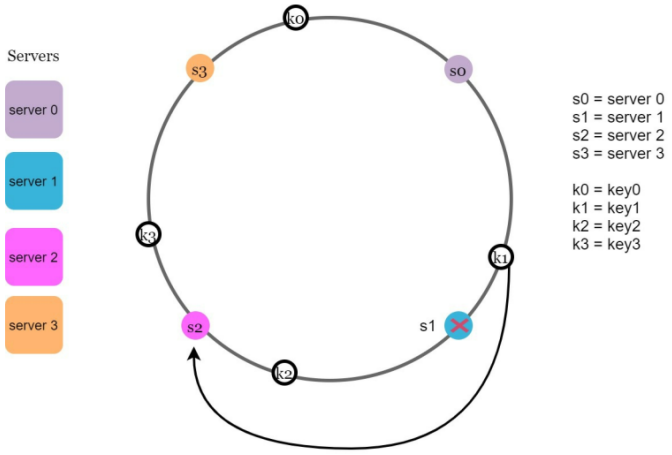
1 balanced
2 standard deviation
3 tradeoff

شماره ۴ به حلقه اضافه شده است. محدوده تحت تأثیر از s4 (گره جدید اضافه شده) شروع می‌شود و در خلاف جهت عقربه‌های ساعت در اطراف حلقه حرکت می‌کند تا زمانی که یک سرور دیگر (s3) پیدا شود. بنابراین، کلیدهایی که بین s3 و s4 قرار دارند باید در s4 توزیع شوند.



شکل ۱۴-۵

هنگامی که یک سرور (s1) همان‌طور که در شکل ۱۵-۵ نشان‌داده‌شده است حذف می‌شود، محدوده تحت تأثیر از s1 (گره حذف شده) شروع می‌شود و در خلاف جهت عقربه‌های ساعت در اطراف حلقه حرکت می‌کند تا زمانی که یک سرور پیدا شود (s0). بنابراین، کلیدهای واقع بین s0 و s1 باید دوباره به s2 توزیع شوند.



شکل ۱۵-۵

جمع‌بندی

در این فصل، ما یک بحث عمیق در مورد هش مداوم داشتیم، از جمله اینکه چرا به آن نیاز است و چگونه کار می‌کند. مزایای هش مداوم عبارت‌اند از:

- هنگامی که سرورها اضافه یا حذف می‌شوند، کلیدهای مینیمم[1] شده مجدداً توزیع می‌شوند.
- مقیاس افقی[2] آسان است؛ زیرا داده‌ها به طور یکنواخت توزیع می‌شوند.
- مشکل کلید کانونی[3] را کاهش می‌دهد. دسترسی بیش از حد به یک قطعه[4] خاص می‌تواند باعث اضافه‌بار سرور شود. تصور کنید داده‌های خواننده Katy Perry، Justin Bieber و Lady Gaga همه به یک shard ختم می‌شوند. هش مداوم با توزیع یکنواخت داده‌ها به کاهش مشکل کمک می‌کند.

1 Minimized key
2 scale horizontally
3 hotspot
4 shard

هش مداوم به طور گسترده در سیستم‌های موجود در دنیای واقعی استفاده می‌شود، از جمله برخی موارد قابل‌توجه:

- جزء پارتیشن‌بندی پایگاه‌داده Dynamo در آمازون است [۳]
- پارتیشن‌بندی داده‌ها در سراسر خوشه[1] در Apache Cassandra [۴]
- برنامه گفتگوی Discord [۵]
- شبکه تحویل محتوا Akamai [۶]
- متعادل‌کننده بار شبکه Maglev [۷]

1 cluster

منابع و مراجع فصل ۵

[1] Consistent hashing: https://en.wikipedia.org/wiki/Consistent_hashing
[2] Consistent Hashing: https://tom-e-white.com/2007/11/consistent-hashing.html
[3] Dynamo: Amazon’s Highly Available Key-value Store: https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
[4] Cassandra - A Decentralized Structured Storage System: http://www.cs.cornell.edu/Projects/ladis2009/papers/Lakshman-ladis2009.PDF
[5] How Discord Scaled Elixir to 5,000,000 Concurrent Users: https://blog.discord.com/scaling-elixir-f9b8e1e7c29b
[6] CS168: The Modern Algorithmic Toolbox Lecture #1: Introduction and Consistent Hashing: http://theory.stanford.edu/~tim/s16/l/l1.pdf
[7] Maglev: A Fast and Reliable Software Network Load Balancer: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44824.pdf

فصل ۶

طراحی پایگاه‌داده KEY-VALUE

یک ذخیره‌ساز[1] کلید-مقدار که گاهی به آن پایگاه‌داده key-value نیز گفته می‌شود، یک پایگاه‌داده غیررابطه‌ای است. هر شناسه منحصربه‌فرد به‌عنوان یک کلید با مقدار مربوط به آن ذخیره می‌شود. این جفت داده به‌عنوان یک جفت "key-value" شناخته می‌شود. در یک جفت key-value، کلید باید منحصربه‌فرد باشد و مقدار مرتبط با کلید از طریق کلید قابل‌دسترسی باشد. کلیدها می‌توانند متن ساده یا مقادیر هش شده باشند. به دلایل عملکردی معمولاً یک کلید کوتاه بهتر عمل می‌کند. کلیدها چه شکلی هستند؟ در اینجا چند نمونه هستند:

- کلید متن ساده[2]: last_logged_in_at
- کلید هش شده[3]: 253DDEC

---

1 store
2 Plain text key
3 Hashed key

مقدار[1] در یک key-value می‌تواند رشته‌ها، لیست‌ها، اشیا[2] و غیره باشد. این مقدار معمولاً در ذخیره‌ساز key-value مانند Amazon dynamo [1]، Memcached [2]، Redis [۳] و غیره، به‌عنوان یک object غیر قابل رویت در نظر گرفته می‌شود. در زیر یک قطعه داده در یک ذخیره‌ساز key-value آمده است:

Key	value
145	john
147	bob
160	Julia

جدول ۱-۶

در این فصل، از شما خواسته می‌شود که یک ذخیره‌ساز key-value طراحی کنید که از عملیات زیر پشتیبانی کند:

- نوشتن با put(key, value) به‌عنوان واردکردن مقادیر مرتبط با کلید
- خواندن با get(key) به‌عنوان خواندن مقادیر مرتبط با کلید

درک مسئله و شروع به طراحی

هیچ طراحی کامل و بی‌نقصی وجود ندارد. هر طرح به تعادل خاصی در رابطه با استفاده از خواندن، نوشتن و حافظه دست می‌یابد. معاوضه یا مزایا و معایب[3] دیگری که باید انجام شود بین سازگاری[4] و دردسترس‌بودن[5] این نوع پایگاه‌داده است. در این فصل، ما یک ذخیره‌ساز[6] key-value طراحی می‌کنیم که شامل ویژگی‌های زیر است:

- اندازه یک جفت key-value کوچک باشد: کمتر از ۱۰ کیلوبایت.
- قابلیت ذخیره داده‌های بزرگ را داشته باشد.

---

1 value
2 objects
3 tradeoff
4 consistency
5 availability
6 store

- دردسترس‌بودن بالا[1]: سیستم با سرعت مناسبی پاسخ می‌دهد، حتی در هنگام خرابی.
- مقیاس‌پذیری بالا[2]: سیستم را می‌توان برای پشتیبانی از بهره‌برداری از مجموعه‌داده‌های بزرگ مقیاس‌دهی کرد.
- مقیاس‌دهی خودکار[3]: افزودن/حذف سرورها باید بر اساس ترافیک و به‌صورت خودکار باشد.
- یکپارچگی یا سازگاری[4] قابل تنظیم.
- زمان تأخیر[5] کم.

یک سرور ذخیره‌ساز key-value منفرد

ایجاد یک ذخیره‌ساز key-value که در یک سرور واحد قرار دارد آسان است. یک رویکرد شهودی، ذخیره جفت‌های key-value در یک جدول هش است که همه‌چیز را در حافظه نگه می‌دارد. حتی اگر دسترسی به حافظه سریع است، به دلیل محدودیت فضا، جادادن همه‌چیز در حافظه ممکن است غیرممکن باشد. دو بهینه‌سازی را می‌توان برای جا دادن داده‌های بیشتر در یک سرور انجام داد:

- فشرده‌سازی داده‌ها.
- فقط داده‌های استفاده شده را در حافظه موقت و بقیه را روی دیسک ذخیره کنید.

حتی با این بهینه‌سازی‌ها، یک سرور می‌تواند خیلی سریع به ظرفیت نهایی خود برسد. برای پشتیبانی از کلان‌داده[6] به یک ذخیره‌ساز key-value توزیع شده نیاز است.

---

1 High availability
2 High scalability
3 Automatic scaling
4 consistency
5 latency
6 big data

ذخیره‌ساز توزیع‌شده‌ی key-value

به یک ذخیره‌ساز key-value توزیع شده، جدول هش توزیع شده نیز گفته می‌شود که جفت‌های key-value را در بسیاری از سرورها توزیع می‌کند. هنگام طراحی یک سیستم توزیع شده، مهم است که قضیه CAP[1] که به معنی « یکپارچگی، دردسترس‌بودن، تحمل پارتیشن» را درک کنید.

بررسی تئوری CAP

قضیه CAP بیان می‌کند که غیرممکن است که یک سیستم توزیع شده به طور هم‌زمان بیش از دو مورد از این سه ضمانت را ارائه دهد: که این موارد شامل «یکپارچگی، دردسترس‌بودن و تحمل پارتیشن» است. اجازه دهید چند تعریف ارائه دهیم.

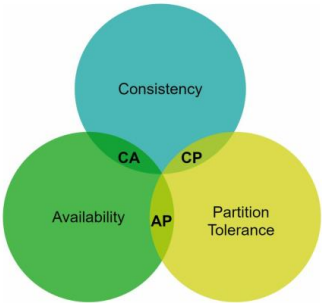
- **یکپارچگی**[2]: یکپارچگی به این معنی است که همه کلاینت‌ها بدون توجه به اینکه به کدام گره متصل می‌شوند، داده‌های یکسانی را در یک‌زمان مشاهده می‌کنند.
- **دردسترس‌بودن**[3]: دردسترس‌بودن به این معنی است که هر کلاینت که داده‌ای را درخواست می‌کند، حتی اگر برخی از گره‌ها[4] خاموش باشند، پاسخ را دریافت می‌کند.

- **آستانه تحمل پارتیشن**[5]: یک پارتیشن نشان‌دهنده قطع ارتباط بین دو گره است. تحمل پارتیشن به این معنی است که سیستم با برخلاف پارتیشن‌های شبکه به کار خود ادامه می‌دهد.

قضیه CAP بیان می‌کند که یکی از سه ویژگی باید قربانی شود تا از ۲ خاصیت از ۳ خاصیت که در شکل ۱-۶ نشان‌داده‌شده است، پشتیبانی شود.

---

1 Consistency, Availability, Partition Tolerance
2 Consistency
3 Availability
4 nodes
5 Partition Tolerance



شکل ۱-۶

امروزه ذخیره‌سازهای key-value بر اساس دو ویژگی از CAP که پشتیبانی می‌کنند طبقه‌بندی می‌شوند:

- **سیستم‌های CP (یکپارچگی و تحمل پارتیشن):** یک ذخیره‌ساز key-value از نوع CP که از یکپارچگی و تحمل پارتیشن را پشتیبانی می‌کند درحالی‌که در گزینه دسترس بودن را قربانی می‌کند.
- **سیستم‌های AP (دردسترس‌بودن و تحمل پارتیشن):** یک ذخیره‌سازهای key-value از نوع AP که از قابلیت دسترسی و تحمل پارتیشن را پشتیبانی می‌کند و درعین‌حال یکپارچگی را قربانی می‌کند.
- **سیستم‌های CA (یکپارچگی و دردسترس‌بودن):** یک ذخیره‌سازهای key-value از نوع CA از یکپارچگی و دردسترس‌بودن[1] پشتیبانی می‌کند درحالی‌که تحمل پارتیشن را قربانی می‌کند. از آنجایی که خرابی‌های شبکه اجتناب‌ناپذیر است، یک سیستم توزیع شده باید پارتیشن شبکه را تحمل کند؛ بنابراین، یک سیستم CA نمی‌تواند در اپلیکیشن‌های دنیای واقعی وجود داشته باشد.

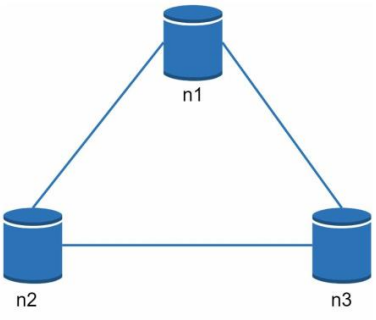
آنچه در بالا خواندید بیشتر به‌عنوان مقدمات و تعریف است. برای درک آسان‌تر، اجازه دهید به چند مثال عینی نگاهی بیندازیم. در سیستم‌های توزیع شده، داده‌ها معمولاً چندین بار

1 availability

تکرار و تکثیر[1] می‌شوند. فرض کنید داده‌ها بر روی سه گره مشابه، n1، n2 و n3 همانند شکل ۲-۶ تکثیر می‌شوند.

وضعیت ایده‌آل

در وضعیت ایده‌آل، پارتیشن شبکه[2] هرگز اتفاق نمی‌افتد. داده‌های نوشته شده به n1 به طور خودکار به n2 و n3 تکرار می‌شود. در نتیجه هم یکپارچگی و هم دردسترس‌بودن به دست می‌آید.

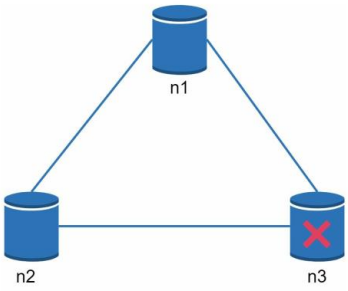


شکل ۲-۶

در یک سیستم توزیع‌شده، نمی‌توان از استفاده‌ی پارتیشن‌ها اجتناب کرد و زمانی که یک مشکل پارتیشن اتفاق می‌افتد، باید بین یکپارچگی و دردسترس‌بودن یکی را انتخاب کنیم. در شکل ۳-۶، مورد n3 پایین می‌آید و به مشکل می‌خورد و نمی‌تواند با n1 و n2 ارتباط برقرار کند. اگر کلاینت‌ها، داده‌ها را به روی n1 یا n2 بنویسند، داده‌ها را نمی‌توان به n3 تکثیر[3] کرد. اگر داده روی n3 نوشته شود؛ اما هنوز به n1 و n2 منتشر و تکثیر نشده باشد، n1 و n2 داده‌های قدیمی و فرسوده خواهند داشت.

---

1 replicated
2 network partition
3 propagated



شکل ۳-۶

اگر یکپارچگی را بر دردسترس‌بودن (سیستم CP) انتخاب کنیم، باید تمام عملیات نوشتن را روی n1 و n2 مسدود کنیم تا از ناسازگاری[1] داده‌ها در بین این سه سرور جلوگیری کنیم که باعث می‌شود سیستم در دسترس نباشد. سیستم‌های بانکی معمولاً نیازمندی‌های یکپارچگی بسیار بالایی[2] دارند. برای مثال، برای یک سیستم بانکی بسیار مهم است که به‌روزترین اطلاعات موجودی را نمایش دهد. اگر ناسازگاری به دلیل پارتیشن شبکه رخ دهد، سیستم بانک قبل از رفع ناسازگاری، یک خطا را بر می‌گرداند. بااین‌حال، اگر دردسترس‌بودن را به‌جای سازگاری یا یکپارچگی (سیستم AP) انتخاب کنیم، سیستم همچنان درخواست‌های خواندن را می‌پذیرد، حتی گاهی ممکن است داده‌های قدیمی را برگرداند. برای موارد مربوط به نوشتن داده‌ها، n1 و n2 به پذیرش نوشتن ادامه می‌دهند و زمانی که مشکل پارتیشن شبکه حل شد، داده‌ها با n3 همگام‌سازی می‌شوند. انتخاب CAP مناسب، تضمین می‌کند که بهترین گزینه مورد استفاده شما باشد و گام مهمی در ایجاد یک ذخیره‌ساز key-value توزیع شده است. حالا می‌توانید این موضوع را با مصاحبه‌کننده خود در میان بگذارید و سیستم را بر اساس آن طراحی کنید.

اجزای سیستم

در این بخش، مولفه‌های اصلی و تکنیک‌های مورد استفاده برای ایجاد یک ذخیره‌ساز key-value را مورد بحث قرار خواهیم داد:

---

1 inconsistency
2 extremely high consistent

- Data partition – پارتیشن‌بندی داده‌ها
- تکثیر داده‌ها (Data replication)
- یکپارچگی (Consistency)
- وضوح غیر یکپارچه (Inconsistency resolution)
- رسیدگی به شکست‌ها (Handling failures)
- دیاگرام معماری سیستم (System architecture diagram)
- مسیر نوشتن (Write path)
- مسیر خواندن (Read path)

محتوی زیر عمدتاً مبتنی بر سه سیستم ذخیره‌ساز key-value محبوب در زیر است: Dynamo [4] و Cassandra [۵] و BigTable [۶].

پارتیشن‌بندی داده‌ها

برای برنامه‌های بزرگ، واردکردن مجموعه‌داده‌های بزرگ به‌صورت کامل در یک سرور غیرممکن است. ساده‌ترین راه برای انجام این کار، تقسیم داده‌ها به پارتیشن‌ها بخش‌های کوچک‌تر و ذخیره آنها در چندین سرور است. هنگام پارتیشن‌بندی داده‌ها دو چالش وجود دارد:

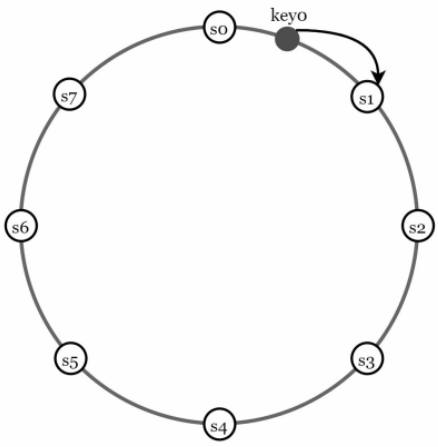
- داده‌ها را در چندین سرور به طور مساوی توزیع کنید.
- جابه‌جایی داده‌ها را هنگام اضافه یا حذف گره‌ها[1] به حداقل برسانید.

هش مداوم[2] که در فصل ۵ مورد بحث قرار گرفت، یک تکنیک عالی برای حل این مشکلات است. اجازه دهید نحوه عملکرد هش مداوم در سطح بالا را دوباره بررسی کنیم.

- ابتدا سرورها روی یک حلقه هش قرار می‌گیرند. در شکل ۴-۶، هشت سرور، که با s0، s1، ...، s7 نشان‌داده‌شده اند، روی حلقه هش قرار می‌گیرند.

1 nodes
2 Consistent hashing

- در مرحله بعد، یک کلید روی همان حلقه هش می‌شود و در اولین سروری که هنگام حرکت در جهت عقربه‌های ساعت با آن مواجه می‌شود، ذخیره می‌شود. به‌عنوان‌مثال، key0 با استفاده از این منطق در s1 ذخیره می‌شود.



شکل ۴-۶

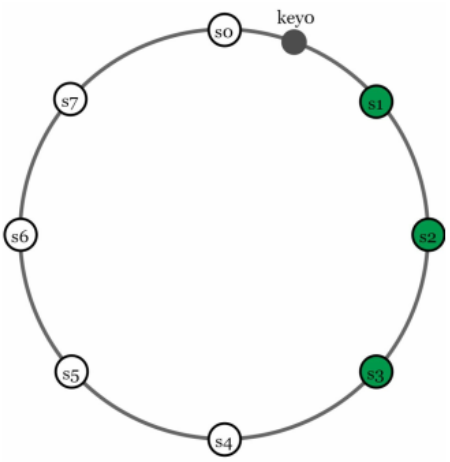
استفاده از هش مداوم برای پارتیشن‌بندی داده‌ها دارای مزایای زیر است:

- **مقیاس‌بندی خودکار**[1]: بسته به بارگذاری، سرورها می‌توانند به طور خودکار اضافه و حذف شوند.
- **ناهمگنی**[2]: تعداد گره‌های مجازی برای یک سرور متناسب با ظرفیت سرور است. به‌عنوان‌مثال، سرورهایی با ظرفیت بالاتر با گره‌های مجازی بیشتری تخصیص داده می‌شوند.

1 Automatic scaling
2 Heterogeneity

تکثیر داده‌ها (Data replication)

برای دستیابی به دسترسی[1] و قابلیت اطمینان[2] بالا، داده‌ها باید به‌صورت ناهم‌زمان[3] روی N سرور تکثیر[4] شوند، جایی که N یک پارامتر قابل تنظیم است. این تعداد N برای سرورها با استفاده از منطق زیر انتخاب می‌شوند: پس از اینکه یک کلید به موقعیتی در حلقه هش نگاشت شد، از آن موقعیت در جهت عقربه‌های ساعت حرکت کنید و اولین N سرور را روی حلقه برای ذخیره کپی‌های داده انتخاب کنید. در شکل ۵-۶ برای (N = 3) در حالتی key0 در s1، s2 و s3 تکرار شده است.



شکل ۵-۶

با گره‌های مجازی[5] که اولین گره N در حلقه هست و ممکن است متعلق به کمتر از N سرور فیزیکی باشد. پس برای جلوگیری از این خطا، ما فقط سرورهای منحصربه‌فرد را در حین اجرای منطق حرکت در جهت عقربه‌های ساعت انتخاب می‌کنیم. گره‌ها در یک مرکز داده[6] اغلب به دلیل قطع برق، مشکلات شبکه، بلایای طبیعی و غیره به طور هم‌زمان از کار می‌افتند.

---

1 availability
2 reliability
3 asynchronously
4 replicated
5 virtual nodes
6 data centers

برای اطمینان بهتر، کپی‌ها یا تکثیرها در مراکز داده متمایز قرار می‌گیرند و مراکز داده از طریق شبکه‌های پرسرعت به هم متصل می‌شوند.

یکپارچگی (Consistency)

به نقل از ویکی‌پدیا: یکپارچگی داده خاصیتی است که تضمین می‌کند که پایگاه‌داده همیشه پس از انجام یک تراکنش از یک حالت معتبر به حالت معتبر دیگری می‌رود؛ به این معنی که هم قبل و هم بعد از انجام تراکنش، خواص پایگاه‌داده (از جمله محدودیت‌ها، triggerها و روابط بین کلیدهای خارجی و کلیدهای اصلی) یکسان بوده و به‌درستی رعایت می‌شوند.

از آنجایی که داده‌ها در چندین گره تکثیر می‌شوند، باید در بین موارد تکثیر شده همگام[1] شوند. اجماع حدنصاب[2] می‌تواند ثبات لازم را برای عملیات خواندن و نوشتن تضمین کند. اجازه دهید ابتدا چند تعریف ارائه کنیم.

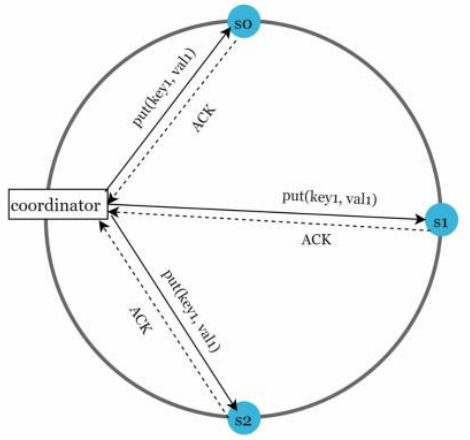
N = تعداد تکثیرها[3]

W = حدنصاب نوشتن به‌اندازه W. برای اینکه یک عملیات نوشتن موفق در نظر گرفته شود، عملیات نوشتن باید از طریع تعداد تکثیرهایW تأیید شود.

R = حدنصاب خواندن به‌اندازه R. برای اینکه عملیات خواندن موفقیت‌آمیز در نظر گرفته شود، عملیات خواندن باید منتظر پاسخ‌های حداقل R تعداد از replicaها/کپی‌ها باشد.

مثال زیر را در شکل ۶-۶ با N = 3 در نظر بگیرید

1 synchronized
2 Quorum consensus
3 replicas



تصویر ۶-۶

ACK= acknowledgment

مقدار W = 1 به این معنی نیست که داده‌ها روی یک سرور نوشته می‌شوند. به‌عنوان‌مثال، با پیکربندی در شکل ۶-۶، داده‌ها در s0، s1 و s2 تکرار و تکثیر[1] می‌شوند. پس W = 1 به این معنی است که هماهنگ کننده[2] باید حداقل یک تاییدیه قبل از اینکه عملیات نوشتن به‌عنوان موفقیت آمیز در نظر گرفته شود را دریافت کند. به‌عنوان‌مثال، اگر از s1 یک تایید دریافت کنیم، دیگر نیازی نیست منتظر تاییدیه‌های s0 و s2 باشیم. یک هماهنگ‌کننده به‌عنوان یک پروکسی بین کاربر و گره‌ها عمل می‌کند.

پیکربندی W، R و N یک مبادله معمولی بین تأخیر[3] و یکپارچگی [4] است. اگر W = 1 یا R = 1، یک عملیات به‌سرعت برگردانده می‌شود؛ زیرا یک هماهنگ‌کننده فقط باید منتظر پاسخ از هر یک از replicaها باشد. اگر W یا R > 1، سیستم یکپارچگی بهتری را ارائه می‌دهد. بااین‌حال، کوئری‌ها کندتر خواهد بود؛ زیرا هماهنگ‌کننده باید منتظر پاسخ از کندترین replica/ کپی‌ها باشد. اگر W + R > N، یکپارچگی قوی[5] تضمین می‌شود؛ زیرا باید حداقل

1 replicated
2 coordinator
3 latency
4 consistency
5 strong consistency

یک گره همپوشانی وجود داشته باشد که آخرین داده‌ها را برای اطمینان از سازگاری و یکپارچگی آن در نظر داشته باشد.

چگونه N، W و R را برای موارد استفاده خود، پیکربندی کنیم؟ در اینجا برخی از تنظیمات ممکن است:

اگر R = 1 و W = N باشد، سیستم برای خواندن سریع بهینه شده است.

اگر W = 1 و R = N باشد، سیستم برای نوشتن سریع بهینه شده است.

اگر W + R > N، یکپارچگی قوی تضمین می‌شود (معمولاً N = 3، W = R = 2).

اگر W + R <= N، یکپارچگی قوی تضمین نمی‌شود.

بسته به نیاز، می‌توانیم مقادیر W، R، N را تنظیم کنیم تا به سطح مطلوبی از ثبات دست یابیم.

مدل‌های یکپارچگی

مدل یکپارچگی[1] یکی دیگر از عوامل مهمی است که باید در طراحی ذخیره‌ساز key-value در نظر گرفت. یک مدل یکپارچه، درجه یکپارچگی داده‌ها را تعریف می‌کند و طیف گسترده‌ای از مدل‌های یکپارچگی وجود دارد:

• **یکپارچگی قوی**[2]: هر عملیات خواندن مقدار داده را مطابق با جدیدترین داده‌ها که تازه نوشته شده‌اند را بر می‌گرداند. پس کاربر هرگز داده‌های قدیمی را نمی‌بیند.

• **یکپارچگی ضعیف**[3]: عملیات خواندن که ممکن است لحظات دیگر انجام شود احتمال دارد به‌روزترین و جدیدترین مقدار داده را نبینند.

• **یکپارچگی تدریجی**[4]: این شکل خاصی از یکپارچگی ضعیف است. در این حالت باتوجه‌به صرف زمان کافی، همه به‌روزرسانی‌ها منتشر می‌شوند و همه کپی‌ها[5] در نهایت کاملاً یکپارچه و هماهنگ هستند.

---

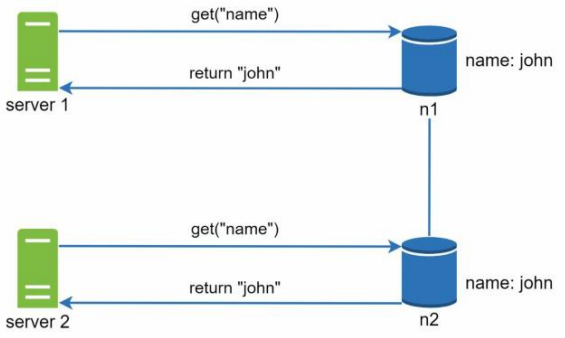
1 Consistency models
2 Strong consistency
3 Weak consistency
4 Eventual consistency
5 replicas

یکپارچگی قوی معمولاً با مجبور کردن یک replica/کپی به قبول‌نکردن خواندن/نوشتن جدید تا زمانی که هر replica بر روی نوشتن فعلی توافق نکرده باشد، به دست می‌آید. این رویکرد برای سیستم‌های با دسترسی بالا[1] ایده‌آل نیست زیرا می‌تواند عملیات جدید را مسدود کند. Dynamo و Cassandra یکپارچگی تدریجی[2] را اتخاذ می‌کنند که شبیه مدل یکپارچگی توصیه شده ما برای ذخیره‌ساز key-value است. با نوشتن‌های هم‌زمان در ذخیره‌ساز، یکپارچگی تدریجی به مقادیر داده ناهماهنگ و ناسازگار اجازه می‌دهد تا وارد سیستم شوند و کاربر را مجبور به خواندن این مقادیر برای تطبیق بیشتر کند. بخش بعدی توضیح می‌دهد که چگونه با کمک روش تطبیق[3] در کنار روش نسخه‌دهی[4] می‌توان چه کاری را انجام داد.

راه‌حل غیریکپارچگی: نسخه‌سازی

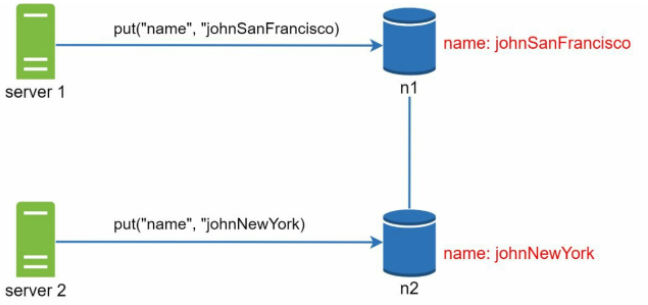
تکثیر داده‌ها یا Replication دسترسی بالایی را به همراه دارد؛ اما باعث ناهماهنگی بین replicaها می‌شود. قفل‌های نسخه‌سازی و برداری[5] برای حل مشکلات ناهماهنگی داده‌ها استفاده می‌شوند. نسخه‌دهی به این معنی است که هر تغییر داده را به‌عنوان یک نسخه تغییرناپذیر جدید از داده‌ها در نظر بگیرید. قبل از اینکه در مورد نسخه دهی صحبت کنیم، اجازه دهید از یک مثال برای توضیح چگونگی ناهماهنگی استفاده کنیم: همان‌طور که در شکل ۷-۶ نشان‌داده‌شده است، هر دو گره replica شامل n1 و n2 دارای یک مقدار هستند. اجازه دهید این مقدار را مقدار اصلی بنامیم. سرور ۱ و سرور ۲ مقدار یکسانی را برای عملیات get(“name”) دریافت می‌کنند.

1 highly available
2 Eventual consistency
3 reconciliation
4 versioning
5 Versioning and vector locks



شکل ۶-۷

در مرحله بعد، سرور ۱ ، نام را به "johnSanFrancisco" تغییر می‌دهد، و سرور ۲ نام را به "johnNewYork" تغییر می‌دهد همان‌طور که در شکل ۶-۸ نشان‌داده‌شده است. این دو تغییر به طور هم‌زمان انجام می‌شود. اکنون، مقادیر متناقضی داریم که نسخه‌های v1 و v2 نامیده می‌شوند.



شکل ۶-۸

در این مثال، مقدار اصلی را می‌توان نادیده گرفت؛ زیرا تغییرات بر اساس آن انجام شده است. بااین‌حال، هیچ راه روشنی برای حل تعارض و تناقض دو نسخه آخر وجود ندارد. برای حل این مشکل، ما به یک سیستم نسخه‌دهی نیاز داریم که بتواند تضادها[1] را شناسایی کرده و تضادها را با هم تطبیق[2] دهد. ساعت برداری[3] یک تکنیک رایج برای حل این مشکل است. اجازه دهید نحوه کار ساعت‌های برداری را بررسی کنیم.

---

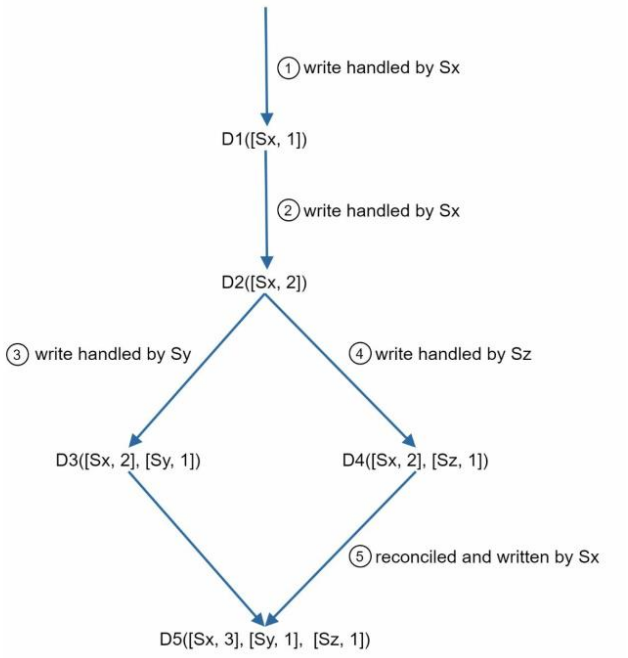
1 conflict
2 reconcile
3 vector clock

ساعت برداری یک جفت [سرور، نسخه] است که با یک آیتم داده مرتبط است. می‌توان از آن برای بررسی اینکه آیا یک نسخه قبل، موفق یا در تضاد با نسخه‌های دیگر است استفاده می‌شود.

فرض کنید یک ساعت برداری با ([S1, v1] , [S2, v2] , …, [Sn, vn]) D نشان داده می‌شود که در آن D یک آیتم داده، v1 یک شمارنده نسخه و s1 یک شماره سرور و موارد دیگر است. اگر مورد D روی سرور Si نوشته شده باشد، سیستم باید یکی از وظایف زیر را انجام دهد:

- در صورت وجود vi آنگاه [Si, vi] را افزایش دهید.
- در غیر این صورت، یک ورودی جدید ایجاد کنید [Si, 1].

منطق انتزاعی فوق با یک مثال عینی همان‌طور که در شکل ۹-۶ نشان‌داده‌شده است توضیح داده شده است.



شکل ۹-۶

مرجع [۴]

۱. یک سرویس کلاینت یک مورد داده D1 را در سیستم می‌نویسد و این مرحله نوشتن توسط سرور Sx که اکنون دارای ساعت برداری [(Sx, 1)]D1 است، مدیریت و کنترل می‌شود.

۲. یک کلاینت دیگر آخرین مقدار D1 را می‌خواند و آن را به مقدار D2 به‌روزرسانی می‌کند و سپس دوباره می‌نویسد. D2 از D1 پایین می‌آید بنابراین D1 را بازنویسی می‌کند. فرض کنید نوشتن توسط همان سرور Sx انجام می‌شود که اکنون ساعت برداری D2 ([Sx, 2]) دارد.

۳. یک کلاینت دیگر آخرین D2 را می‌خواند، آن را به D3 به‌روزرسانی می‌کند و دوباره می‌نویسد.

فرض کنید نوشتن توسط سرور Sy اداره می‌شود که اکنون دارای ساعت برداری D3 ([Sx, 2], [Sy, 1]) است.

۴. یک کلاینت دیگر آخرین D2 را می‌خواند، آن را به D4 به‌روزرسانی می‌کند و دوباره می‌نویسد. فرض کنید نوشتن توسط سرور Sz انجام می‌شود که اکنون دارای D4 ([Sx, 2], [Sz, 1]) است.

۵. وقتی کلاینت دیگری D3 و D4 را می‌خواند، یک تضاد در داده‌ها را کشف می‌کند که ناشی از تغییر یک مورد خاص داده D2 توسط Sy و Sz است. تضاد توسط کلاینت حل می‌شود و داده‌های به‌روز شده به سرور ارسال می‌شود. فرض کنید نوشتن توسط Sx انجام می‌شود که اکنون دارای D5 ([Sx, 3], [Sy, 1], [Sz, 1]) است. به‌زودی نحوه تشخیص تضاد[1] را توضیح خواهیم داد.

با استفاده از ساعت‌های برداری، به‌راحتی می‌توان تشخیص داد که نسخه X یک والد[2] (بدون تضاد) از نسخه Y است، اگر شمارنده‌های نسخه برای هر شرکت‌کننده در ساعت برداری Y

1 conflict
2 ancestor

بزرگ‌تر یا مساوی با نسخه X باشد. برای مثال، ساعت برداری ([s0, 1], [s1, 1])D والدِ ([s0, 1], [s1, 2])D است؛ بنابراین هیچ تعارضی ثبت نمی‌شود.

به طور مشابه، اگر شرکت‌کننده‌ای در ساعت برداری Y وجود داشته باشد که شمارنده‌ای کمتر از شمارنده متناظر آن در X داشته باشد، می‌توانید بگویید که نسخه X خواهر و برادر Y است (تضاد وجود دارد). به‌عنوان‌مثال، دو مورد زیر ساعت‌های برداری نشان می‌دهد که یک تضاد وجود دارد: ([s0, 1], [s1, 2])D و ([s0, 2], [s1, 1])D.

حتی اگر ساعت‌های بُرداری می‌توانند تضادها را حل کنند، دو جنبه منفی قابل‌توجه وجود دارد.

ابتدا، ساعت‌های برداری به کلاینت پیچیدگی می‌بخشند، زیرا نیاز به پیاده‌سازی منطق حل تضاد دارد.

دوم، جفت‌های [server: version] در ساعت برداری می‌توانند به‌سرعت رشد کنند. برای رفع این مشکل، یک آستانه برای طول آن تعیین می‌کنیم و اگر از حد مجاز بیشتر شود، قدیمی‌ترین جفت‌ها حذف می‌شوند. این کار می‌تواند منجر به ناکارآمدی در مرحله تطبیق[1] شود، زیرا نمی‌توان رابطه نسل‌ها و والدها را به طور دقیق تعیین کرد. بااین‌حال، بر اساس مقاله Dynamo [۴]، آمازون هنوز با این مشکل در تولید مواجه نشده است؛ بنابراین احتمالاً برای اکثر شرکت‌ها راه‌حل قابل‌قبولی است.

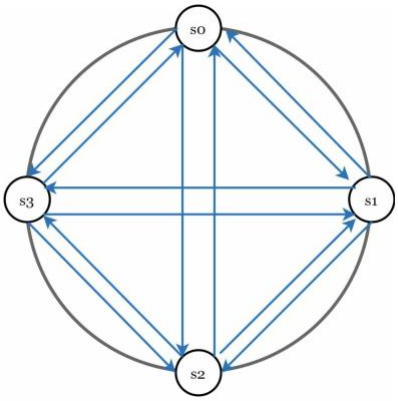
رسیدگی به شکست‌ها

مانند هر سیستم بزرگ در مقیاس بالا، خرابی‌ها نه‌تنها اجتناب‌ناپذیر بلکه رایج هستند. رسیدگی به سناریوهای شکست بسیار مهم است. در این بخش ابتدا به معرفی تکنیک‌هایی برای تشخیص خرابی‌ها و شکست‌ها می‌پردازیم. سپس، استراتژی‌های رایج حل شکست را مرور می‌کنیم.

1 reconciliation

شـنـاسـایـی خـطـا

در یک سیستم توزیع شده، باور اینکه یک سرور از کارافتاده است، زیرا سرور دیگری چنین می‌گوید دلیل کافی‌ای نیست. برای این تشخیص معمولاً حداقل به دو منبع اطلاعات مستقل نیاز دارد تا وضعیت سلامتی یک سرور را مشخص کند. همان‌طور که در شکل ۱۰-۶ نشان‌داده‌شده است، پخش چندگانه[1] یک راه‌حل ساده است. بااین‌حال، زمانی که سرورهای زیادی در سیستم هستند، این روش ناکارآمد است.



شکل ۱۰-۶

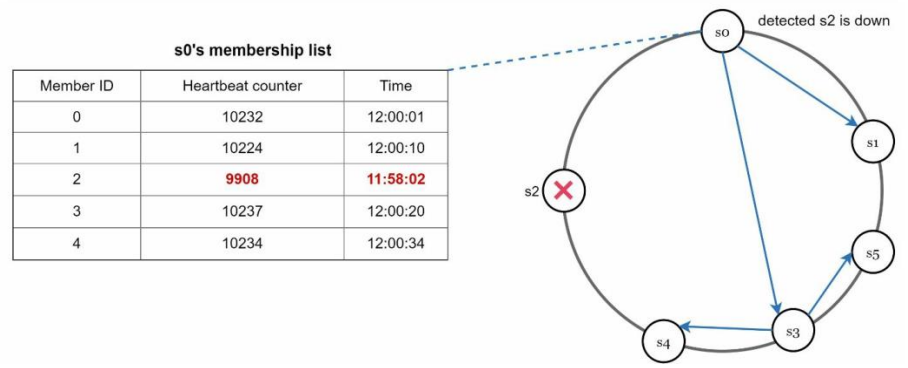
راه‌حل بهتر استفاده از روش‌های غیرمتمرکز تشخیص خرابی مانند پروتکل شایعات[2] است. پروتکل Gossip به شرح زیر عمل می‌کند:

- هر گره یک لیست عضویت گره دارد که شامل شناسه اعضا و شمارنده ضربان قلب[3] است.
- هر گره به طور دوره‌ای شمارنده heartbeat خود را افزایش می‌دهد.
- هر گره به طور دوره‌ای heartbeat را به مجموعه‌ای از گره‌های تصادفی ارسال می‌کند که به نوبه خود به مجموعه دیگری از گره‌ها انتشار می‌یابد.

---

1 multicasting
2 gossip
3 heartbeat

- هنگامی که گره‌ها heartbeat دریافت می‌کنند، لیست عضویت به آخرین اطلاعات به‌روزرسانی می‌شود.
- اگر heartbeat بیش از دوره‌های از پیش تعریف شده افزایش نیافته باشد، عضو مربوط به آن به‌عنوان آفلاین در نظر گرفته می‌شود.



شکل ۱۱-۶

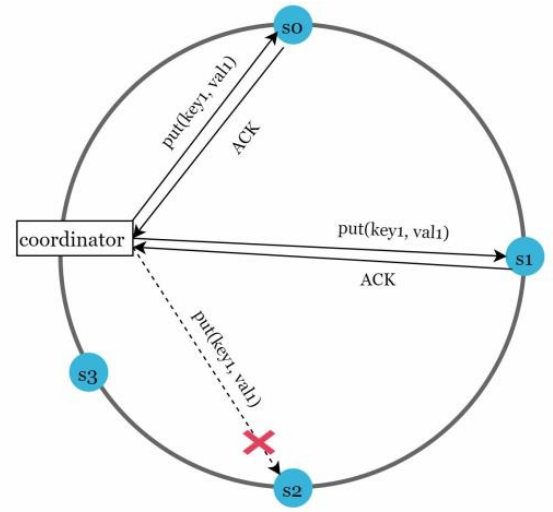
همان‌طور که در شکل ۱۱-۶ نشان‌داده‌شده است:

- گره s0 یک لیست عضویت گره نشان‌داده‌شده در سمت چپ را نگهداری می‌کند.
- گره s0 متوجه می‌شود که شمارشگر heartbeat گره s2 (شناسه عضو = ۲) برای مدت طولانی افزایش نیافته است.
- گره s0 ضربان قلب یا heartbeat را که شامل اطلاعات s2 است به مجموعه‌ای از گره‌های تصادفی ارسال می‌کند. هنگامی که گره‌های دیگر تأیید کردند که شمارشگر heartbeat مربوط به s2 برای مدت طولانی به‌روزرسانی نشده است آنگاه گره s2 علامت‌گذاری می‌شود و این اطلاعات به گره‌های دیگر منتشر می‌شود.

رسیدگی به خرابی‌های موقت

پس از شناسایی خرابی‌ها از طریق پروتکل gossip، سیستم باید مکانیسم‌های خاصی را برای اطمینان از دردسترس‌بودن[1] مستقر کند. در رویکرد حدنصاب سخت‌گیرانه[2]، عملیات خواندن و نوشتن می‌تواند مسدود شود، همان‌طور که در بخش اجماع حدنصاب[3] نشان‌داده‌شده است. تکنیکی به نام «حدنصاب نامرتب[4]» [۴] برای بهبود دردسترس‌بودن استفاده می‌شود. به جای اجرای شرط حد نصاب، سیستم اولین سرورهای سالم W را برای نوشتن و اولین سرورهای سالم R را برای خواندن در حلقه هش انتخاب می‌کند و سرورهای آفلاین نادیده گرفته می‌شوند. اگر سروری به دلیل خرابی شبکه یا سرور در دسترس نباشد، سرور دیگری درخواست‌ها را به طور موقت پردازش می‌کند. هنگامی که سرور ازکارافتاده، دوباره فعال می‌شود، پس تغییرات برای دستیابی به یکپارچگی داده‌ها به عقب بازگردانده[5] می‌شوند. به این فرایند «hinted handoff[6]» می‌گویند. از آنجایی که s2 در شکل ۱۲-۶ در دسترس نیست، خواندن و نوشتن توسط s3 به طور موقت مدیریت می‌شود. وقتی s2 دوباره آنلاین شد، s3 داده‌ها را به s2 بر می‌گرداند.

1 availability
2 strict quorum
3 quorum consensus
4 sloppy quorum
5 pushed back
۶ در لغت به معنی «انتقال اشاره‌ای» است ولی یک اصطلاح فنی است و توصیه می شود که ترجمه نشود و در ادامه توضیح داده می‌شود.



شکل ۱۲-۶

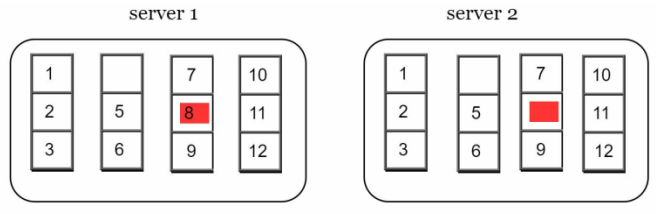
رسیدگی به خرابی‌های دائمی

از روش hinted handoff برای رسیدگی به خرابی‌های موقت استفاده می‌شود. اگر replica برای همیشه در دسترس نباشد چه؟ برای رسیدگی به چنین وضعیتی، ما یک پروتکل ضد آنتروپی را برای همگام نگه‌داشتن کپی‌ها پیاده‌سازی می‌کنیم. ضد آنتروپی شامل مقایسه هر تکه داده روی replicaها و به‌روزرسانی هر replica به جدیدترین نسخه است. درخت مرکل[1] برای تشخیص ناهماهنگی و به حداقل رساندن مقدار داده‌های منتقل شده استفاده می‌شود. به نقل از ویکی‌پدیا [۷]: «درخت هش یا درخت مرکل درختی است که در آن هر گرهِ غیربرگی با یک هش برچسب‌گذاری[2] شده یا مقادیر آن (در مورد برگ‌ها) در گره‌های فرزند برچسب‌گذاری می‌شود. درختان هش امکان تأیید کارآمد و ایمن محتویات ساختارهای داده بزرگ را فراهم می‌کنند.

1 Merkle tree
2 labels

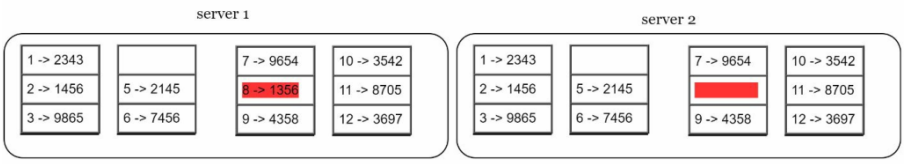
با فرض اینکه فضای کلید[1] از ۱ تا ۱۲ باشد، مراحل زیر نحوه ساخت درخت مرکل را نشان می‌دهد. کادرهای برجسته نشان دهنده ناهماهنگی و ناسزگاری در داده‌ها است.

مرحله ۱: همان‌طور که در شکل ۱۳-۶ نشان‌داده‌شده است، فضای کلید را به پیمانه‌ها (در مثال ما ۴ قسمت) تقسیم کنید. یک پیمانه به‌عنوان گره سطح ریشه برای نگهداری عمق محدود درخت استفاده می‌شود.



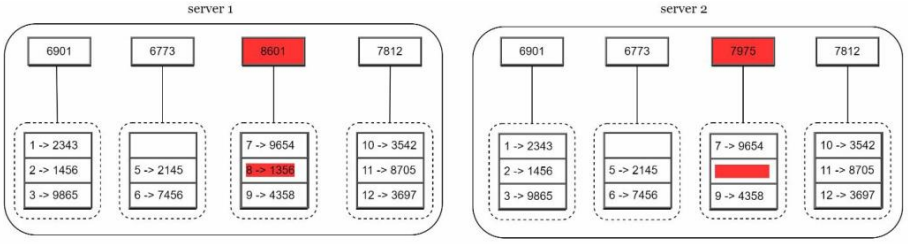
شکل ۱۳-۶

مرحله ۲: پس از ایجاد پیمانه‌ها، هر کلید را با استفاده از روش هش یکنواخت[2] در یک پیمانه هش کنید (شکل ۱۴-۶).



شکل ۱۴-۶

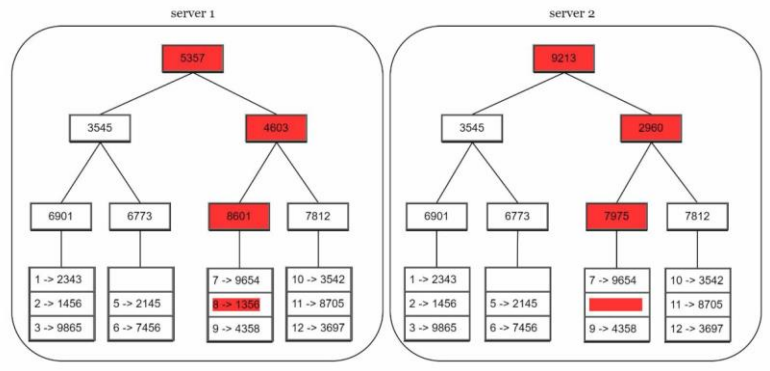
مرحله ۳: یک گره هش در هر پیمانه ایجاد کنید (شکل ۱۵-۶).



شکل ۱۵-۶

1 key space
2 uniform hashing

مرحله ۴: با محاسبه هش‌های فرزندان، درخت را به سمت بالا و تا ریشه بسازید. مشابه تصویر زیر در شکل ۱۶-۶.



شکل ۱۶-۶

برای مقایسه دو درخت مرکل، با مقایسه هش‌های ریشه شروع کنید. اگر هش ریشه مطابقت داشته باشد، هر دو سرور داده‌های یکسانی دارند. اگر هش‌های ریشه با هم مخالف باشند، هش‌های فرزند سمت چپ و سپس هش‌های فرزند راست مقایسه می‌شوند. می‌توانید درخت را پیمایش کنید تا ببینید کدام پیمانه‌ها هماهنگ نیستند و فقط آن پیمانه‌ها را همگام‌سازی[1] کنید. با استفاده از درخت مرکل، مقدار داده‌ای که برای همگام‌سازی لازم است، متناسب با تفاوت‌های بین دو کپی است و نه با مقدار داده‌های موجود در آنها. در سیستم‌های دنیای واقعی، اندازه پیمانه بسیار بزرگ است. به‌عنوان‌مثال، یک پیکربندی ممکن به‌عنوان‌مثال؛ برابر با یک میلیون پیمانه در هر یک میلیارد کلید است، بنابراین هر پیمانه فقط شامل ۱۰۰۰ کلید است.

رسیدگی به خرابی‌های مرکز داده

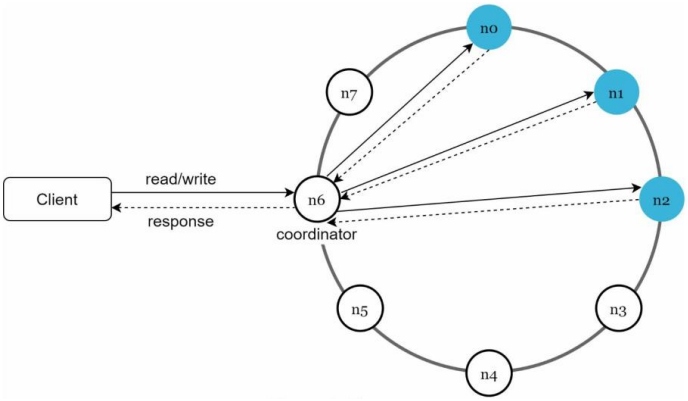
خرابی‌های مربوط مرکز داده[2] ممکن است به دلیل قطع برق، قطع شبکه، بلایای طبیعی و سایر دلایل دیگر اتفاق بیفتد. برای ایجاد سیستمی که قادر به مدیریت خرابی‌های مرکز داده

1 synchronized
2 data center

باشد، مهم است که داده‌ها را در چندین مرکز داده تکرار یا تکثیر[1] کنید. حتی اگر یک مرکز داده کاملاً آفلاین باشد، کاربران همچنان می‌توانند از طریق مراکز داده دیگر به داده‌ها دسترسی داشته باشند.

نمودار معماری سیستم

اکنون که ملاحظات فنی مختلف را در طراحی یک ذخیره‌ساز key-value مورد بحث قرار دادیم، می‌توانیم تمرکز خود را روی نمودار معماری که در شکل ۱۷-۶ نشان‌داده‌شده است تغییر دهیم.



شکل ۱۷-۶

ویژگی‌های اصلی معماری به شرح زیر است:

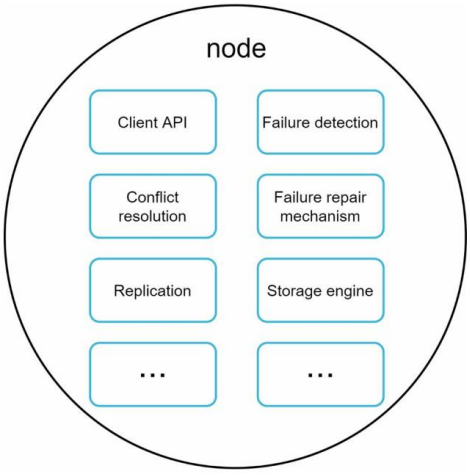
- کاربران از طریق APIهای ساده با ذخیره‌ساز key-value ارتباط برقرار می‌کنند: به‌عنوان‌مثال به کمک دستورات get(key) و put (key, value).
- هماهنگ‌کننده[2] گره‌ای است که به‌عنوان یک پروکسی یا رابط بین کاربر و ذخیره‌ساز key-value عمل می‌کند.
- گره‌ها بر روی یک حلقه با استفاده از هش مداوم[3] توزیع می‌شوند.

---

1 replicate
2 coordinator
3 consistent hashing

- سیستم کاملاً غیرمتمرکز[1] است؛ بنابراین اضافه‌کردن و جابه‌جایی گره‌ها می‌تواند خودکار باشد.
- داده‌ها در چندین گره تکثیر[2] می‌شوند.
- هیچ نقطه شکست واحدی[3] وجود ندارد؛ زیرا هر گره دارای مجموعه‌ای از مسئولیت‌ها است.

همان‌طور که طراحی غیرمتمرکز است، هر گره وظایف و تسک‌های زیادی را همان‌طور که در شکل ۱۸-۶ ارائه شده است انجام می‌دهد.



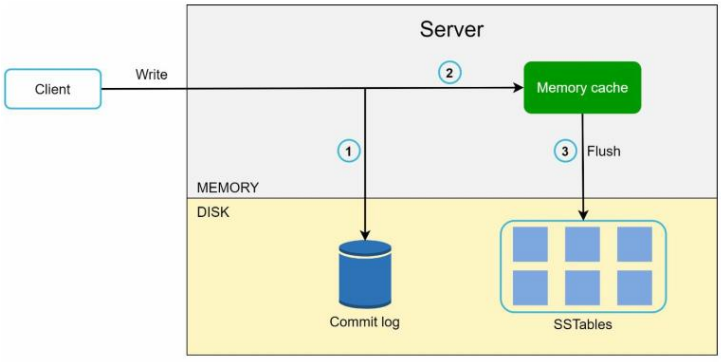
شکل ۱۸-۶

مسیر نوشتن

شکل ۱۹-۶ توضیح می‌دهد که پس از ارسال درخواست نوشتن به یک گره خاص چه اتفاقی می‌افتد. لطفاً توجه داشته باشید که طرح‌های پیشنهادی برای مسیرهای نوشتن یا خواندن بر اساس معماری Cassandra [۸] تا حدی مقدماتی هستند.

---

1 decentralized
2 replicated
3 single point of failure



شکل ۱۹-۶

تصویر ۱۹-۶ به‌صورت زیر توضیح داده شده است:

۱. درخواست نوشتن در یک فایل log ادامه دارد.

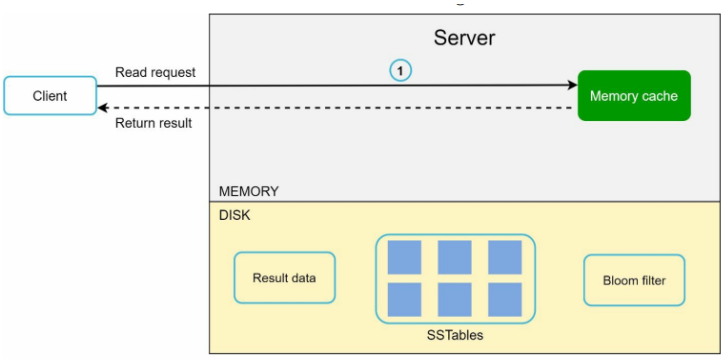
۲. داده‌ها در حافظه کَش[1] ذخیره می‌شوند.

۳. هنگامی که حافظه کَش پر است یا به آستانه از پیش تعریف شده می‌رسد، داده‌ها به SSTable [۹] روی دیسک منتقل می‌شوند. **نکته**: جدول رشته‌های مرتب[2] شده (SSTable) یک لیست مرتب شده از جفت‌های <key, value> است. برای خوانندگانی که علاقه‌مند به یادگیری بیشتر در مورد SStable هستند، به مرجع [۹] مراجعه کنید.

مسیر خواندن

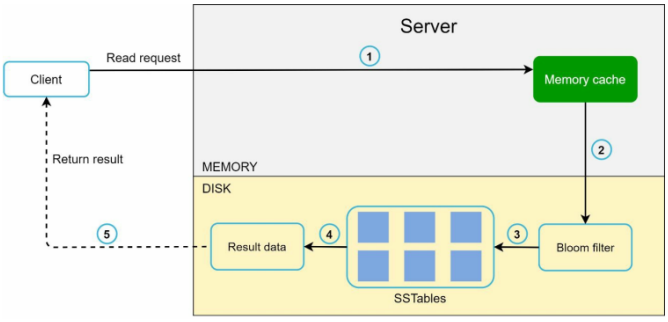
پس از اینکه درخواست خواندن به یک گره خاص هدایت شد، ابتدا بررسی می‌کند که آیا داده‌ها در حافظه کَش هستند یا خیر. در این صورت، داده‌ها همان‌طور که در شکل ۲۰-۶ نشان‌داده‌شده است به کاربر بازگردانده می‌شود.

1 cache
2 sorted-string table (SSTable)



شکل ۲۰-۶

اگر داده در حافظه نباشد، در عوض از دیسک بازیابی می‌شود. ما به یک روش کارآمد نیاز داریم تا بفهمیم کدام SSTable حاوی کلید است. Bloom filter [۱۰] معمولاً برای حل این مشکل استفاده می‌شود. مسیر خواندن در شکل ۲۱-۶ نشان‌داده‌شده است که داده‌ها در حافظه نیستند.



شکل ۲۱-۶

۱. سیستم ابتدا بررسی می‌کند که آیا داده‌ها در حافظه کَش هستند یا خیر. اگر این‌طور نیست پس به مرحله شماره ۲ بروید.

۲. اگر داده‌ها در حافظه نباشد، سیستم Bloom filter را بررسی می‌کند.

۳. Bloom filter برای تعیین اینکه کدام SSTables ممکن است حاوی کلید باشد استفاده می‌شود.

۴. SSTables نتیجه‌ای از مجموعه‌داده‌ها را بر می‌گرداند.

۵. نتیجه مجموعه‌داده‌ها به کلاینت بازگردانده می‌شود.

نـتیجه‌گیری

این فصل مفاهیم و تکنیک‌های زیادی را پوشش می‌دهد. برای تازه کردن مطالب مطرح شده در این فصل، جدول زیر ویژگی‌ها و تکنیک‌های متناظر مورد استفاده برای ذخیره‌ساز key-value توزیع‌شده را خلاصه می‌کند.

راه‌حل	مشکل/هدف
از هش مداوم برای توزیع بار بین سرورها استفاده کنید	قابلیت ذخیره کلان‌داده
تکثیر داده‌ها راه‌اندازی مرکز داده چندگانه	دردسترس‌بودن بالا جهت خواندن داده‌ها
نسخه‌سازی و حل تعارض با ساعت‌های برداری	دردسترس‌بودن بالا جهت نوشتن داده‌ها
هش مداوم	پارتیشن کردن مجموعه‌داده‌ها
هش مداوم	مقیاس‌پذیری افزایشی
هش مداوم	ناهمگن بودن
حدنصاب اجماع	پایداری قابل تنظیم
حدنصاب تصادفی و hinted handoff	رسیدگی به خرابی‌های موقت
درخت مرکل	رسیدگی به خرابی‌های دائمی
حدنصاب تصادفی و hinted handoff	رسیدگی به خرابی مرکز داده

جدول ۲-۶

منابع و مراجع فصل ٦

[1] Amazon DynamoDB: https://aws.amazon.com/dynamodb/
[2] memcached: https://memcached.org/
[3] Redis: https://redis.io/
[4] Dynamo: Amazon’s Highly Available Key-value Store: https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
[5] Cassandra: https://cassandra.apache.org/
[6] Bigtable: A Distributed Storage System for Structured Data: https://static.googleusercontent.com/media/research.google.com/en//archive/bigtableosdi06.pdf
[7] Merkle tree: https://en.wikipedia.org/wiki/Merkle_tree
[8] Cassandra architecture: https://cassandra.apache.org/doc/latest/architecture/
[9] SStable: https://www.igvita.com/2012/02/06/sstable-and-log-structured-storage-leveldb/
[10] Bloom filter https://en.wikipedia.org/wiki/Bloom_filter

فصل ۷

طراحی تولیدکننده UNIQUE ID توزیع‌شده

در این فصل از شما خواسته می‌شود که یک تولیدکننده شناسه منحصربه‌فرد[1] در سیستم‌های توزیع شده طراحی کنید. اولین فکر شما ممکن است استفاده از یک کلید اولیه با ویژگی «افزایش خودکار[2]» در یک پایگاه‌داده ساده باشد. بااین‌حال، روش افزایش خودکار در یک محیط توزیع شده کار نمی‌کند؛ زیرا یک سرور شامل یک پایگاه‌داده واحد به‌اندازه کافی بزرگ نیست و ایجاد شناسه‌های منحصربه‌فرد در چندین پایگاه‌داده با حداقل تأخیر بین تولید و هماهنگی بین آن‌ها چالش‌برانگیز است.

در اینجا چند نمونه از شناسه‌های منحصربه‌فرد آورده شده است:

user_id
1227238262110117894
1241107244890099715
1243643959492173824
1247686501489692673
1567981766075453440

تصویر ۱-۷

1 unique ID generator
2 auto_increment

مرحله ۱ – درک مسئله و شروع به طراحی

پرسیدن سؤالات واضح، اولین قدم برای مقابله با هر سؤال طراحی سیستم است.

در اینجا نمونه‌ای از تعامل شما و مصاحبه‌کننده آورده شده است:

کاندید: شناسه‌های منحصربه‌فرد[1] چه ویژگی‌هایی دارند؟

مصاحبه‌کننده: شناسه‌ها باید منحصربه‌فرد و قابل مرتب‌سازی باشند.

کاندید: برای هر رکورد جدید، آیا IDها یا شناسه‌ها به‌اندازه ۱ واحد افزایش[2] می‌یابند؟

مصاحبه‌کننده: شناسه‌ها بر حسب زمان افزایش می‌یابد، اما نه لزوماً فقط ۱ واحد افزایش می‌یابد. شناسه‌هایی که در بعد از ظهر ایجاد می‌شوند معمولاً بزرگتر از مواردی هستند که در صبح همان روز ایجاد می‌شوند.

کاندید: آیا شناسه‌ها فقط حاوی مقادیر عددی هستند؟

مصاحبه‌کننده: بله، درست است.

کاندید: طول شناسه‌های موردنیاز چیست؟

مصاحبه‌کننده: شناسه‌ها باید در حدود ۶۴ بیت اندازه داشته باشند.

کاندید: مقیاس یا scale سیستم چقدر است؟

مصاحبه‌کننده: سیستم باید بتواند ۱۰۰۰۰، ID یا شناسه در ثانیه تولید کند. در بالا تعدادی از نمونه سؤالاتی وجود دارد که می‌توانید از مصاحبه‌کننده خود بپرسید. درک الزامات و روشن‌شدن ابهامات برای این سؤال مصاحبه بسیار مهم است.

در نهایت موارد ضروری به شرح زیر ذکر شده است:

- شناسه‌ها یا IDها باید منحصربه‌فرد باشند.
- شناسه‌ها فقط مقادیر عددی هستند.
- شناسه‌ها در ۶۴ بیت قرار می‌گیرند.
- شناسه‌ها بر اساس تاریخ مرتب می‌شوند.

---

1 unique ID
2 increment

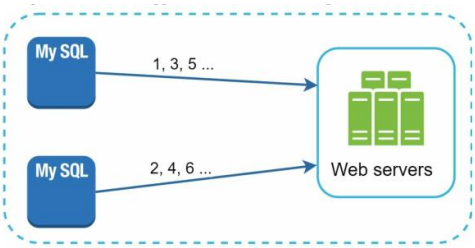
- امکان تولید بیش از ۱۰۰۰۰، unique ID در هر ثانیه موردنیاز است.

مرحله ۲ - طراحی سطح پیشرفته

برای تولید شناسه‌های منحصربه‌فرد یا unique IDها در سیستم‌های توزیع شده می‌توان از گزینه‌های مختلفی استفاده کرد. گزینه‌های در نظر گرفته شده عبارت‌اند از:

- Multi-Master Replication
- Universally unique identifier (UUID)
- Ticket server
- رویکرد Twitter snowflake

همان‌طور که در شکل ۲-۷ نشان‌داده‌شده است، اولین رویکرد مورد نظر ما استفاده از روش، Multi-Master Replication است.



شکل ۲-۷

این رویکرد از ویژگی افزایش خودکار[1] پایگاه‌داده استفاده می‌کند. به‌جای اینکه ID یا شناسه بعدی را ۱ واحد افزایش دهیم، آن را با k واحد افزایش می‌دهیم که k تعداد سرورهای پایگاه‌داده در حال استفاده است. همان‌طور که در شکل ۲-۷ نشان‌داده‌شده است، شناسه بعدی که باید تولید شود برابر با شناسه قبلی در همان سرور به اضافه ۲ است. این برخی از مشکلات مقیاس‌پذیری[2] را حل می‌کند زیرا شناسه‌ها می‌توانند با تعداد سرورهای پایگاه‌داده مقیاس[3] شوند. بااین‌حال، این استراتژی دارای چند اشکال اساسی است:

---

1 auto_increment
2 scalability
3 scale

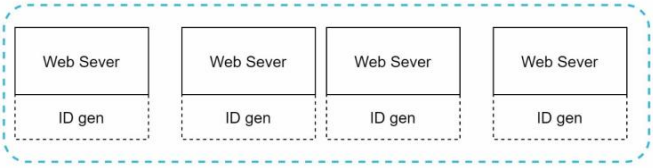
- مقیاس‌پذیری با مرکز داده‌های[1] مختلف دشوار است.
- شناسه‌ها با بر اساس زمان در چندین سرور مختلف افزایش نمی‌یابند.
- هنگامی که یک سرور اضافه یا حذف می‌شود، پارامتر مقیاس‌پذیری وضعیت خوبی ندارد.

شناسه‌های منحصربه‌فرد – UUID

در واقع UUID راه آسان دیگری برای به‌دست‌آوردن شناسه‌های منحصربه‌فرد است. UUID یک عدد ۱۲۸ بیتی است که برای شناسایی اطلاعات در سیستم‌های کامپیوتری استفاده می‌شود. UUID از نظر آماری احتمال تصادم[2] (وجود خروجی یکسان در برابر یک ورودی متفاوت) بسیار کمی دارد. به نقل از ویکی‌پدیا: پس از تولید ۱ میلیارد UUID در هر ثانیه به مدت تقریباً ۱۰۰ سال، احتمال ایجاد یک نسخه تکراری در خروجی به ۵۰٪ می‌رسد [۱]. در اینجا نمونه‌ای از UUID آمده است:

UUID: 09c93e62-50b4-468d-bf8a-c07e1040bfb2

همین‌طور UUIDها می‌توانند به طور مستقل و بدون هماهنگی بین سرورها تولید شوند. شکل ۳-۷ طراحی UUIDها را نشان می‌دهد.



شکل ۳-۷

در این طراحی، هر وب سرور شامل یک تولیدکننده شناسه[3] است و یک وب سرور به طور مستقل وظیفه تولید IDها را بر عهده دارد.

---

1 data center
2 collusion
3 ID generator

مزایا:

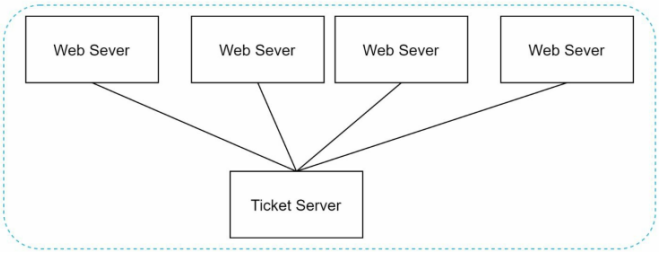
- ایجاد UUID ساده است. هیچ هماهنگی بین سرورها لازم نیست، بنابراین هیچ مشکلی در همگام‌سازی[1] وجود نخواهد داشت.
- مقیاس‌کردن[2] سیستم آسان است؛ زیرا هر وب سرور مسئول تولید شناسه‌هایی است که استفاده از آن می‌کند. تولیدکننده شناسه می‌تواند به‌راحتی با وب سرورها مقیاس‌دهی شود.

معایب:

- شناسه‌ها ۱۲۸ بیت هستند، اما نیاز ما ۶۴ بیت است.
- شناسه‌ها بر اساس گذشت زمان افزایش نمی‌یابند.
- شناسه‌ها باید غیرعددی باشند.

Ticket Server

استفاده از Ticket Serverها یکی دیگر از راه‌های جالب برای تولید شناسه‌های منحصربه‌فرد است. وب‌سایت معروف Flicker یک روش Ticket Server برای تولید کلید اولیه[3] توزیع شده را توسعه داده است[۲]. تصویر زیر نشان می‌دهد که این سیستم چگونه کار می‌کند.



شکل ۴-۷

این ایده مبتنی بر استفاده از ویژگی متمرکز «**افزایش خودکار**» در یک سرور پایگاه‌داده منفرد

1 synchronization
2 scale
3 primary key

(Ticket Server) است. برای کسب اطلاعات بیشتر در این مورد، به مقاله وبلاگ مهندسی flicker [۲] مراجعه کنید.

مزایا:

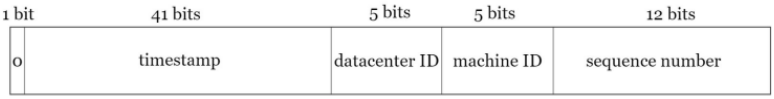
- شناسه‌های آن عددی است.
- پیاده‌سازی آن آسان است و برای کاربردهای کوچک تا متوسط کار می‌کند.

معایب:

تنها نقطه‌ضعف Ticket Server منفرد به این معنی است که اگر Ticket Server از کار بیفتد، تمام سیستم‌هایی که به آن وابسته هستند با مشکل مواجه می‌شوند. برای جلوگیری از یک نقطه‌ضعف، می‌توانیم چندین Ticket Server راه‌اندازی کنیم. بااین‌حال، این چالش‌های جدیدی مانند همگام‌سازی[1] داده‌ها را به همراه خواهد داشت.

Twitter snowflake

رویکردهای ذکر شده در بالا به ما ایده‌هایی درباره نحوه عملکرد سیستم‌های مختلف تولیدکننده شناسه[2] می‌دهد. بااین‌حال، هیچ یک از آنها الزامات خاص ما را برآورده نمی‌کند؛ بنابراین، ما به رویکرد دیگری نیاز داریم. سیستم unique ID generation مورد استفاده در توییتر به نام **"snowflake"** [۳] که بسیار جالب و الهام‌بخش است و می‌تواند نیازهای موردنظر ما را برآورده کند. در این حالت روش تقسیم و حل[3] به کمک ما می‌آید. به‌جای تولید مستقیم شناسه، یک شناسه را به بخش‌های مختلف تقسیم می‌کنیم. شکل ۵-۷ طرح یک شناسه از نوع ۶۴ بیتی را نشان می‌دهد.



شکل ۵-۷

1 synchronization
2 ID generation
3 Divide and conquer

هر بخش در زیر توضیح داده شده است.

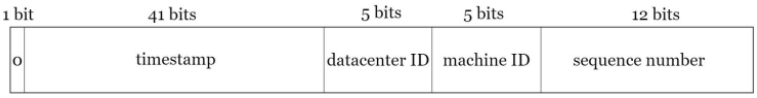
- **بیت علامت**[1]: دارای ۱ بیت است که همیشه ۰ خواهد بود. این بیت برای استفاده‌های بعدی رزرو شده است. به طور بالقوه می‌توان از آن برای تمایز بین اعداد علامت دار و بدون علامت استفاده کرد.
- **Timestamp**: دارای ۴۱ بیت است. برحسب میلی ثانیه بوده و از epoch یا epoch اصلاح شده استفاده می‌کند. ما از زمان و epoch پیش‌فرض ۱۲۸۸۸۳۴۹۷۴۶۵۷، معادل ۰۴ نوامبر ۲۰۱۰، ۰۱:۴۲:۵۴ UTC استفاده می‌کنیم.
- **شناسه مرکز داده**[2]: دارای ۵ بیت است، که به ما ۳۲ = ۲ ^ ۵ مرکز داده[3] را می‌دهد.
- **شناسه ماشین**[4]: دارای ۵ بیت است، که به ما ۳۲ = ۲ ^ ۵ ماشین را در هر مرکز داده می‌دهد.
- **شماره دنباله**[5]: دارای ۱۲ بیت است. برای هر ID تولید شده در آن پردازش/ماشین که شماره دنباله را ۱ واحد افزایش می‌دهد. این عدد در هر میلی ثانیه به ۰ بازنشانی می‌شود.

مرحله ۳ – عمیق‌شدن در طراحی

در طراحی سطح بالا، ما گزینه‌های مختلفی را برای طراحی یک تولیدکننده شناسه‌های منحصربه‌فرد در سیستم‌های توزیع شده مورد بحث قرار دادیم. ما رویکردی را که مبتنی بر تولیدکننده معماری snowflake در توییتر است را انتخاب می‌کنیم. بهتر است عمیقاً در طراحی فرو برویم. برای تازه کردن حافظه، نمودار طراحی در زیر نمایش‌داده‌شده است.

---

1 Sign bit
2 Datacenter ID
3 Datacenter
4 Machine ID
5 Sequence number

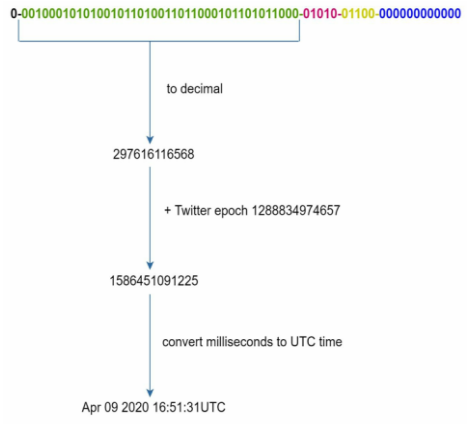


شکل ۶-۷

شناسه‌ها یا IDهای مربوط به مراکز داده[1] و machine IDها در زمان شروع به کار سیستم انتخاب می‌شوند و معمولاً پس از راه‌اندازی سیستم ثابت می‌شوند. هر گونه تغییر در شناسه مرکز داده‌ها و شناسه ماشین‌ها معمولاً نیاز به بررسی دقیق و مجدد دارد؛ زیرا تغییر تصادفی در این مقادیر می‌تواند منجر به تداخل شناسه‌ها شود. زمانی که تولیدکننده شناسه[2] در حال اجرا است، Timestamp و دنباله عددی[3] تولید می‌شوند.

بررسی Timestamp

مهم‌ترین بیت از بین ۴۱ بیت را قسمت timestamp را تشکیل می‌دهند. همان‌طور که timestamp با گذشت زمان رشد می‌کنند، شناسه‌ها بر اساس زمان قابل مرتب‌سازی هستند. شکل ۷-۷ نمونه‌ای از نحوه تبدیل نمایش باینری به UTC را نشان می‌دهد. همچنین می‌توانید UTC را با استفاده از روشی مشابه به نمایش باینری تبدیل کنید.



شکل ۷-۷

1 Datacenter
2 ID generator
3 sequence numbers

حداکثر مقدار timestamp که می‌توان در ۴۱ بیت نمایش داد برابر با مقدار:
(۲۱۹۹۰۲۳۲۵۵۵۵۵۱= ۲ ^ ۴۱ – ۱) میلی‌ثانیه (ms) است که به ما مقداری حدودی برابر حالت زیر را می‌دهد:

$$\sim 69\ years = 2199023255551\ ms\ /\ 1000\ seconds\ /\ 365\ days\ /\ 24\ hours\ /\ 3600\ seconds.$$

این بدان معناست که ID generator به مدت ۶۹ سال کار خواهد کرد و داشتن یک epoch اصلاح شده نزدیک به تاریخ امروز، زمان سرریز را به تأخیر می‌اندازد. پس از ۶۹ سال، ما به epoch time نیاز خواهیم داشت یا تکنیک‌های دیگری را برای تغییر شناسه‌ها اتخاذ می‌کنیم.

دنباله‌ی عددی

مقدار و ظرفیت دنباله‌ی عددی برابر ۱۲ بیت است که به ما ۴۰۹۶=۲ ^ ۱۲ ترکیب می‌دهد. این فیلد به‌صورت پیش‌فرض برابر ۰ است مگر اینکه بیش از یک ID در یک میلی‌ثانیه در همان سرور ایجاد شود. در تئوری، یک ماشین می‌تواند حداکثر ۴۰۹۶ شناسه یا ID جدید را در هر میلی‌ثانیه پشتیبانی کند.

مرحله ٤ – جمع‌بندی

در این فصل، ما رویکردهای مختلفی را برای طراحی یک unique ID generator مورد بحث قرار دادیم:

- multimaster replication
- UUID
- ticket server
- Twitter snowflake-like unique ID generator

ما حالت snowflake را انتخاب می‌کنیم؛ زیرا از همه موارد استفاده ما را پشتیبانی می‌کند و در یک محیط توزیع شده مقیاس‌پذیر است.

اگر زمان اضافی در پایان مصاحبه وجود دارد، در اینجا چند نکته اضافی برای صحبت وجود دارد:

- **همگام‌سازی زمان و ساعت**. در این طراحی، فرض می‌کنیم که سرورهای تولید ID دارای ساعت یکسانی هستند. این فرض ممکن است زمانی که یک سرور بر روی چندین هسته اجرا می‌شود درست نباشد. همین چالش در سناریوهای چند ماشینی وجود دارد. راه‌حل‌های همگام‌سازی زمان خارج از محدوده این کتاب هستند. بااین‌حال، درک وجود مشکل مهم است.

  Time Protocol محبوب‌ترین راه‌حل برای این مشکل است. برای خوانندگان علاقه‌مند، به مطالب مرجع [۴] مراجعه کنید.
- **بهینه‌کردن طول قسمت‌ها**[1]. به‌عنوان‌مثال، دنباله اعداد کمتر اما timestamp بیشتر برای برنامه‌هایی که هم‌زمان[2] کمی داشته؛ ولی طولانی‌مدت کار می‌کنند، بسیار مؤثر هستند.
- **دردسترس‌بودن بالا**[3]. از آنجایی که یک ID generator یک سیستم بسیار ضروری و مهم است، باید سطح دسترسی بالایی داشته باشد.

1 Section length tuning
2 concurrency
3 High availability

منابع و مراجع فصل ۷

[1] Universally unique identifier:
https://en.wikipedia.org/wiki/Universally_unique_identifier
[2] Ticket Servers: Distributed Unique Primary Keys on the Cheap:
https://code.flickr.net/2010/02/08/ticket-servers-distributed-unique-primary-keys-on-thecheap/
[3] Announcing Snowflake:
https://blog.twitter.com/engineering/en_us/a/2010/announcingsnowflake.html
[4] Network time protocol:
https://en.wikipedia.org/wiki/Network_Time_Protocol

فصل ۸

طراحی یک کوتاه‌کننده URL

در این فصل، به یک سؤال مصاحبه طراحی سیستمی جالب و کلاسیک می‌پردازیم: آن هم طراحی یک سرویس کوتاه‌کننده URL[1] مانند **tinyurl** است.

مرحله ۱ - درک مسئله و شروع به طراحی

سؤالات مصاحبه طراحی سیستم عمداً با پایان باز تمام می‌شوند. برای طراحی یک سیستمی که به خوب طراحی شده، پرسیدن سؤالات شفاف‌سازی جهت شناختن بهتر صورت‌مسئله بسیار مهم است.

کاندید: آیا می‌توانید مثالی از نحوه عملکرد یک کوتاه‌کننده URL بزنید؟

مصاحبه‌کننده: URL ای مثل آدرس زیر را فرض کنید که طولانی است

https://www.example.com/q=chatsystem&c=loggedin&v=v3&l=long

سرویس شما یک نام دیگر و آدرس کوتاه شده‌ی دیگری به شکل

https://tinyurl.com/y7keocw

ایجاد می‌کند. اگر که روی آن لینک کلیک کنید شما را به آدرس اصلی هدایت[2] می‌کند.

کاندید: حجم ترافیک مصرفی چقدر است؟

مصاحبه‌کننده: ۱۰۰ میلیون URL در روز تولید می‌شود.

---

1 URL shortening
2 redirects

کاندید: طول URL کوتاه شده چقدر است؟

مصاحبه‌کننده: تا حد امکان کوتاه.

کاندید: چه کاراکترهایی در URL کوتاه شده مجاز هستند؟

مصاحبه‌کننده: URL کوتاه شده می‌تواند ترکیبی از اعداد (۰-۹) و کاراکترها (a-z، A-Z) باشد.

کاندید: آیا URLهای کوتاه شده را می‌توان حذف یا به‌روزرسانی کرد؟

مصاحبه‌کننده: برای سادگی، فرض می‌کنیم که URLهای کوتاه شده را نمی‌توان حذف یا به‌روزرسانی کرد.

موارد استفاده اساسی در اینجا آمده است:

۱. کوتاه‌کردن[1] URL: دریافت یک URL طولانی ← URL بسیار کوتاهتری را بر می‌گرداند.

۲. تغییر مسیر[2] URL: دریافت یک URL کوتاهتر ← هدایت به URL اصلی.

۳. دردسترس‌بودن بالا، مقیاس‌پذیری، و ملاحظات تحمل خطا.

تـخـمین مسـئلـه

- عملیات نوشتن: ۱۰۰ میلیون URL در روز تولید می‌شود.
- عملیات نوشتن در ثانیه: ۱۱۶۰ = ۳۶۰۰ / ۲۴ / میلیون ۱۰۰
- عملیات خواندن: با فرض نسبت عملیات خواندن به عملیات نوشتن حدود ۱۰ به ۱ است، عملیات خواندن در ثانیه: ۱۱۶۰۰ = ۱۰ * ۱۱۶۰
- با فرض اینکه سرویس کوتاه‌کننده URL به مدت ۱۰ سال اجرا می‌شود، این بدان معناست که ما باید (۳۵۶ میلیارد رکورد = ۱۰ * ۳۶۵ * میلیون ۱۰۰) را پشتیبانی کنیم.
- فرض کنید طول متوسط URL صد باشد.
- نیاز به ذخیره‌سازی بیش از ۱۰ سال:

---

1 URL shortening
2 URL redirectin

ترابایت ۳۶۵ = سال ۱۰ * بایت ۱۰۰ * میلیارد ۳۶۵

بسیار مهم است که پیش‌فرض‌ها و موارد محاسباتی را با مصاحبه‌کننده خود طی کنید تا هر دوی شما یک دیدگاه داشته باشید.

مرحله ۲ - طراحی سطح پیشرفته

در این بخش، نقاط پایانی[1] API، تغییر مسیر URL، و جریان‌های کوتاه‌کردن URL را مورد بحث قرار می‌دهیم.

API Endpoints

نقاط پایانی API، ارتباط بین کلاینت و سرور را تسهیل می‌کند. ما APIها را به سبک معماری REST طراحی خواهیم کرد. اگر با API restful آشنا نیستید، می‌توانید به منبع [۱] مراجعه کنید. یک کوتاه‌کننده URL اولیه به دو API endpoints نیاز دارد.

۱. **کوتاه‌کننده URL**[2]. برای ایجاد یک URL کوتاه جدید؛ کلاینت یک درخواست POST ارسال را می‌کند که حاوی یک پارامتر آن هم URL طولانی و اصلی ما است. API به شکل زیر است:

```
POST api/v1/data/shorten
• request parameter: {longUrl: longURLString}
• return shortURL
```

۲. تغییر مسیر URL. برای تغییر مسیر یک URL کوتاه به URL طولانی مربوطه، کلاینت یک درخواست GET شبیه کد زیر را ارسال می‌کند.

```
GET api/v1/shortUrl
    • Return longURL for HTTP redirection
```

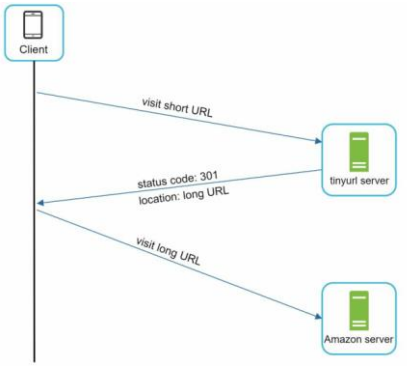
1 API endpoints
2 URL shortening

تغییر مسیر URL

شکل ۱-۸ نشان می‌دهد که وقتی یک tinyurl را در مرورگر وارد می‌کنید چه اتفاقی می‌افتد. هنگامی که سرور درخواست tinyurl را دریافت کرد، URL کوتاه را با تغییر مسیر[1] ۳۰۱ به URL طولانی تغییر می‌دهد.

شکل ۱-۸

ارتباط دقیق بین کلاینت‌ها و سرورها در شکل ۲-۸ نشان‌داده‌شده است.



شکل ۲-۸

یکی از مواردی که در اینجا قابل‌بحث است، تغییر مسیر ۳۰۱ در مقابل تغییر مسیر ۳۰۲ است.

1 redirect

redirect 301:

تغییر مسیر ۳۰۱ نشان می‌دهد که URL درخواستی «به طور دائم» به URL طولانی منتقل شده است. از آنجایی که برای همیشه redirect می‌شود، مرورگر پاسخ را در حافظه کَش نگه می‌دارد و درخواست‌های بعدی برای همان URL به سرویس کوتاه‌کردن URL ارسال نمی‌شود. در عوض، درخواست‌ها مستقیماً به سرور مربوط به URL طولانی هدایت می‌شوند.

redirect 302:

تغییر مسیر ۳۰۲ به این معنی است که URL «به طور موقت» به URL طولانی منتقل می‌شود، به این معنی که درخواست‌های بعدی برای همان URL ابتدا به سرویس کوتاه‌کردن URL ارسال می‌شود. سپس، آنها به سرور URL طولانی هدایت می‌شوند.

هر روش تغییر مسیر، جوانب مثبت و منفی خود را دارد. اگر اولویت کاهش بار روی سرور است، استفاده از تغییر مسیر ۳۰۱ منطقی است زیرا تنها اولین درخواست از همان URL به سرورهای کوتاه‌کننده URL ارسال می‌شود. بااین‌حال، اگر تجزیه‌وتحلیل مهم است، تغییر مسیر ۳۰۲ انتخاب بهتری است زیرا می‌تواند نرخ کلیک و منبع کلیک را راحت‌تر ردیابی کند.

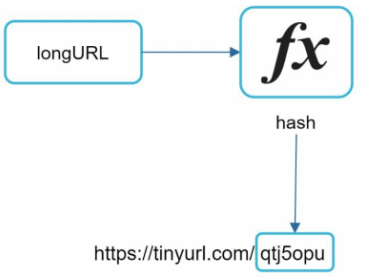
ساده‌ترین راه برای پیاده‌سازی تغییر مسیر URL استفاده از جداول هش[1] است. با فرض اینکه جدول هش جفت‌های <shortURL، longURL> را ذخیره می‌کند، تغییر مسیر URL را می‌توان با موارد زیر پیاده‌سازی کرد:

- دریافت longURL: longURL = hashTable.get(shortURL)
- پس از دریافت longURL، تغییر مسیر URL را انجام دهید.

1 hash table

کوتاه‌کردن آدرس URL

فرض کنید URL کوتاه به این صورت است: **www.tinyurl.com/{hashValue}**. برای پشتیبانی از مورد استفاده کوتاه‌کردن URL، باید تابع هش fx را پیدا کنیم که یک URL طولانی را به hashValue نگاشت می‌کند، همان‌طور که در شکل ۳-۸ نشان‌داده‌شده است.



شکل ۳-۸

تابع هش باید شرایط زیر را برآورده کند:

- هر longURL باید به یک hashValue هش شود.
- هر hashValue را می‌توان به longURL نگاشت کرد.

طراحی دقیق برای تابع هش در مرحله سوم که مرحله عمیق‌شدن در طراحی نام دارد مورد بحث قرار گرفته است.

مرحله ۳ - عمیق‌شدن در طراحی

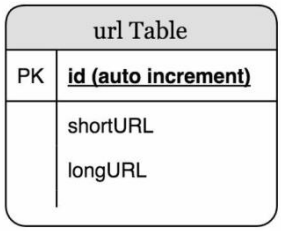
تابه‌حال ما در مورد طراحی سطح پیشرفته کوتاه‌کردن URL و تغییر مسیر URL بحث کرده‌ایم. در این بخش، ما عمیقاً به موارد زیر می‌پردازیم: مدل داده[۱]، تابع هش[۲]، کوتاه‌کردن[۳] URL و تغییر مسیر[۴] URL.

---

1 data model
2 hash function
3 URL shortening
4 URL redirecting

مـــــــدل داده‌ای

در طراحی سطح پیشرفته، همه‌چیز در یک جدول هش ذخیره می‌شود. این نقطه شروع خوبی است. بااین‌حال، این رویکرد برای سیستم‌های دنیای واقعی امکان‌پذیر نیست؛ زیرا منابع حافظه محدود و گران هستند. یک گزینه بهتر این است که نگاشت <shortURL, longURL> را در یک پایگاه‌داده رابطه‌ای ذخیره کنید. شکل ۴-۸ طراحی ساده جدول پایگاه‌داده را نشان می‌دهد. نسخه ساده شده جدول شامل ۳ ستون است: id، shortURL، longURL.



شکل ۴-۸

تـــابـــع هـــش

تابع Hash برای هش‌کردن یک URL طولانی به یک URL کوتاه که به‌عنوان hashValue نیز شناخته می‌شود، استفاده می‌شود.

طول مقدار هش

مقدار هش یا hashValue از کاراکترهایی از [۰-۹, a-z, A-Z] تشکیل شده است که شامل ۶۲ = ۲۶ + ۲۶ + ۱۰ کاراکتر ممکن است. برای محاسبه طول hashValue، کوچکترین n را به‌طوری که $62^n \geq 365$ میلیارد باشد را پیدا کنید. این سیستم باید تا ۳۶۵ میلیارد URL را بر اساس تخمین اولیه پشتیبانی کند. جدول ۱-۸ طول hashValue و حداکثر تعداد URLهایی که می‌تواند پشتیبانی کند را نشان می‌دهد.

N	Maximal number of URLs
1	62^1 = 62
2	62^2 = 3,844
3	62^3 = 238,328
4	62^ 4 = 14,776,336
5	62^5 = 916,132,832
6	62^6 = 56,800,235,584
7	62^7 = 3,521,614,606,208 = ~3.5 trillion

جدول ۱-۸

وقتی $n = 7$ در نتیجه ($62 \text{ ^ } n = \sim 3.5\ trillion$) پس ۳.۵ تریلیون برای نگهداری ۳۶۵ میلیارد URL کافی است، بنابراین طول hashValue برابر ۷ است.

ما دو نوع توابع هش را برای کوتاه‌کننده URL بررسی خواهیم کرد. اولین مورد «هش + دقتِ تصادم[1]» و مورد دوم «تبدیل مبنا[2] ۶۲» است. بگذارید یک به یک آنها را بررسی کنیم.

هش + دقتِ تصادم

در علم رمزنگاری، «برخورد یا تصادم هش» زمانی اتفاق می‌افتد که دو داده متفاوت در یک جدول هش، یک مقدار یکسانی از هش را در خروجی به اشتراک می‌گذارند این مقدار هش از یک تابع هش گرفته می‌شود که یک ورودی داده را می‌گیرد و یک رشته بیت با طول ثابت بر می‌گرداند. با وجود اینکه الگوریتم‌های هش با هدف مقاومت در برابر تصادم طراحی شده‌اند، گاهی اوقات ممکن است داده‌های مختلفی را به یک مقدار هش مشابه نگاشت کنند (به دلیل اصل لانه‌کبوتری).

برای کوتاه‌کردن یک URL طولانی، باید یک تابع هش را پیاده‌سازی کنیم که یک URL طولانی را به یک رشته ۷ کاراکتری هش می‌کند. یک راه‌حل ساده استفاده از توابع هش

1 Hash + collision resolution
2 base 62 conversion

شناخته شده مانند CRC32، MD5 یا SHA-1 است. جدول زیر نتایج هش را پس از اعمال توابع هش مختلف در این URL مقایسه می‌کند:

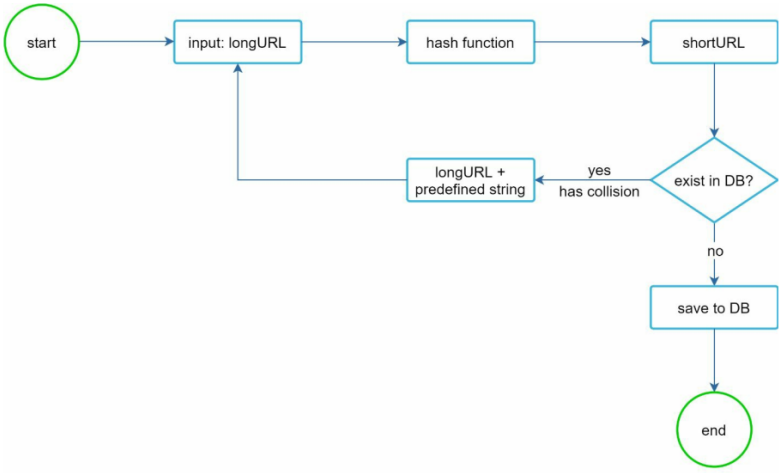
https://en.wikipedia.org/wiki/Systems_design

Hash function	Hash value (Hexadecimal)
CRC32	5cb54054
MD5	5a62509a84df9ee03fe1230b9df8b84e
SHA-1	0eeae7916c06853901d9ccbefbfcaf4de57ed85b

جدول ۲-۸

همان‌طور که در جدول ۲-۸ نشان‌داده‌شده است، حتی کوتاهترین مقدار هش (از CRC32) بسیار طولانی است (بیش از ۷ کاراکتر). چگونه می‌توانیم آن را کوتاه‌تر کنیم؟

اولین رویکرد جمع‌آوری ۷ کاراکتر اول یک مقدار هش است. بااین‌حال، این روش می‌تواند منجر به تصادم هش[1] شود. برای حل برخوردها یا تصادم‌های هش، می‌توانیم به‌صورت بازگشتی یک رشته از پیش تعریف شده جدید را اضافه کنیم تا زمانی که هیچ تصادم دیگری کشف نشود. این فرایند در شکل ۵-۸ توضیح داده شده است.



شکل ۵-۸

1 hash collisions در واقع

این روش می‌تواند تصادم را از بین ببرد. بااین‌حال، کوئری روی پایگاه‌داده برای بررسی وجود یک shortURL برای هر درخواست سنگین است. تکنیکی به نام فیلترهای bloom می‌تواند عملکرد را بهبود بخشد. فیلتر bloom یک تکنیک احتمالی مناسب در فضای [1]حالت برای آزمایش اینکه آیا یک عنصر، عضوی از یک مجموعه است یا خیر، را فراهم می‌کند.

تبدیل Base 62

تبدیل مبنا[2] روش دیگری است که معمولاً برای کوتاه‌کننده‌های URL استفاده می‌شود. تبدیل مبنا کمک می‌کند تا عدد مشابهی را بین سیستم‌های مختلف نمایش اعداد تبدیل کند. تبدیل مبنای ۶۲ زمانی استفاده می‌شود که ۶۲ کاراکتر ممکن برای hashValue وجود دارد. اجازه دهید از یک مثال برای توضیح نحوه عملکرد این تبدیل استفاده کنیم:

تبدیل $_{10}$ ۱۱۱۵۷ بر پایه ۶۲ در زیر نشان داده است:

($_{10}$ ۱۱۱۵۷ نشان دهنده عدد ۱۱۱۵۷ در سیستم مبنای ۱۰ است).

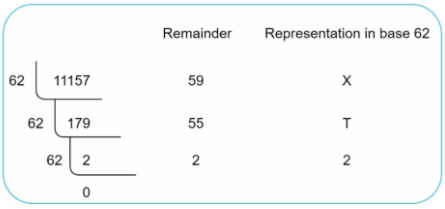
• باتوجه‌به اسمی که دارد، base 62 راهی برای استفاده از ۶۲ کاراکتر برای رمزگذاری است. نگاشت‌ها عبارتند از:

$$0-0,\ldots,9-9,10-a,11-b,\ldots,35-z,36-A,\ldots,61-Z$$

که در آن 'a' مخفف ۱۰، 'Z' مخفف ۶۱ است و و برای سایر حروف‌ها و رقم‌ها دیگر مشابه چنین نگاشتی را داریم، به‌عنوان‌مثال:

$$11157_{10} = 2\ x\ 62^2 + 55\ x\ 61^1 + 59\ x\ 62^0 = [2, 55, 59\ ] \rightarrow [2, T, X]\ base\ 62$$

شکل ۶-۸ روند تغییر مبنا را نشان می‌دهد.



شکل ۶-۸

1 space-efficient probabilistic technique
2 Base

- بنابراین، URL کوتاه برابر https://tinyurl.com/2TX است

مقایسه این دو رویکرد

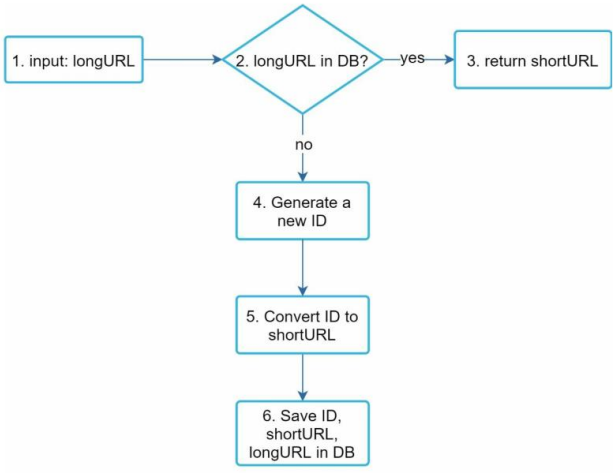
جدول ۳-۸ تفاوت‌های این دو رویکرد را نشان می‌دهد.

تبدیل مبنی ٦٢	هش + دقت تصادم
طول URL کوتاه، ثابت نیست و با شناسه بالا می‌رود	طول URL ثابت و کوتاه
این گزینه به یک تولید کننده شناسه منحصر به فرد بستگی دارد.	نیازی به تولید کننده شناسه منحصر به فرد ندارد.
وقوع تصادم غیر ممکن است زیرا شناسه منحصر به فرد است	احتمال وقوع تصادم وجود دارد و باید مشکل حل شود.
اگر شناسه برای ورودی جدید به مقدار ۱ واحد افزایش یابد، به راحتی می‌توان URL کوتاه بعدی موجود را کشف کرد. این می‌تواند یک نگرانی امنیتی باشد.	تشخیص URL کوتاه بعدی ممکن نیست زیرا وابستگی به شناسه وجود ندارد.

جدول ۳-۸

عمیق‌شدن در طراحی کوتاه‌کننده URL

به‌عنوان یکی از قطعات اصلی سیستم، ما می‌خواهیم مسیر کوتاه‌کردن URL از نظر منطقی ساده و کاربردی باشد. در این طراحی از تبدیل مبنای ۶۲ استفاده شده است. برای نشان دادن این مسیر، نمودار زیر را می‌سازیم (شکل ۷-۸).



شکل ۷-۸

۱. longURL ورودی این نمودار است.
۲. سیستم بررسی می‌کند که longURL در پایگاه‌داده موجود است یا خیر.
۳. اگر چنین است، به این معنی است که longURL قبلاً به shortURL تبدیل شده است. در این حالت، shortURL را از پایگاه‌داده واکشی کرده و به کلاینت برگردانید.
۴. اگر این‌طور نیست، پس longURL جدید است. یک شناسه منحصربه‌فرد[1] جدید (کلید اصلی[2]) توسط تولیدکننده شناسه منحصر[3] به فرد تولید می‌شود.
۵. شناسه (ID) را به shortURL با تبدیل Base 62 تبدیل کنید.
۶. یک ردیف در پایگاه‌داده جدید با ID، shortURL و longURL ایجاد کنید.

برای درک آسان‌تر این مسیر، اجازه دهید به یک مثال عینی نگاه کنیم.

- با فرض longURL ورودی زیر:

  https://en.wikipedia.org/wiki/Systems_design
- Unique ID generator ID مقداری را برمی‌گرداند: ۲۰۰۹۲۱۵۶۷۴۹۳۸.
- با استفاده از تبدیل مبنای ۶۲، شناسه را به shortURL تبدیل کنید. شناسه (۲۰۰۹۲۱۵۶۷۴۹۳۸) به "zn9edcu" تبدیل شده است.
- ID، shortURL و longURL را همان‌طور که در جدول ۴-۸ نشان‌داده‌شده است در پایگاه‌داده ذخیره کنید.

id	shortURL	longURL
2009215674938	zn9edcu	https://en.wikipedia.org/wiki/Systems_design

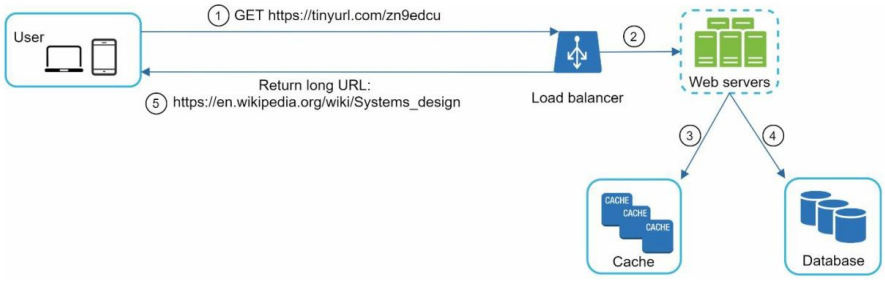
جدول ۴-۸

1 unique ID
2 primary key
3 unique ID generator

در این قسمت یادآوری «تولیدکننده شناسه منحصربه‌فرد توزیع شده[1]» یک مورد باارزش جهت بررسی و توجه است. عملکرد اصلی آن تولید شناسه‌های منحصربه‌فرد سراسری[2] است که برای ایجاد URLهای کوتاه استفاده می‌شود. در یک محیط به‌شدت توزیع شده، پیاده‌سازی یک تولیدکننده شناسه منحصربه‌فرد کاری چالش‌برانگیز است. خوشبختانه، ما قبلاً چند راه‌حل را در «فصل ۷: طراحی یک تولیدکننده شناسه منحصربه‌فرد در سیستم‌های توزیع شده» مورد بحث قرار داده‌ایم.

عمیق‌شدن در URL redirect

شکل ۸-۸ طراحی دقیق تغییر دهنده مسیر[3] URL را نشان می‌دهد. از آنجایی که تعداد موارد مربوط به خواندن‌ها بیشتر از نوشتن است، نگاشت <shortURL, longURL> برای بهبود عملکرد در حافظه کَش ذخیره می‌شود.



شکل ۸-۸

جریان تغییر مسیر URL به‌صورت زیر خلاصه می‌شود:

۱. کاربر روی پیوند URL کوتاه کلیک می‌کند: https://tinyurl.com/zn9edcu

۲. متعادل‌کننده بار[4] درخواست را به سرورهای وب ارسال می‌کند.

۳. اگر یک shortURL از قبل در حافظه کش موجود باشد، پس longURL را مستقیماً برگردانید.

---

1 distributed unique ID generator
2 generate globally unique ID
3 redirecting
4 load balancer

۴. اگر shortURL در حافظه کَش موجود نباشد، longURL را از پایگاه‌داده واکشی کنید. اگر در پایگاه‌داده موجود نباشد، احتمالاً کاربر یک shortURL نامعتبر وارد کرده است.
۵. در نهایت این longURL به کاربر برگردانده می‌شود.

مرحله ٤- جمع‌بندی

در این فصل، ما در مورد طراحی API، مدل داده، تابع هش، کوتاه‌کردن URL و تغییر مسیر URL صحبت کردیم.

اگر در پایان مصاحبه زمان اضافی وجود دارد، در اینجا چند نکته جهت صحبت اضافی وجود دارد.

- **محدودکننده نرخ**[1]: یک مشکل امنیتی احتمالی که ممکن است با آن مواجه شویم این است که کاربران مخرب تعداد بسیار زیادی درخواست کوتاه‌کردن URL ارسال می‌کنند. محدودکننده نرخ به فیلترکردن درخواست‌ها بر اساس آدرس IP یا سایر قوانین فیلتر کمک می‌کند.
- **مقیاس‌پذیری وب سرور**[2]: ازآنجایی‌که لایه وب stateless است، مقیاس‌پذیری سطح وب با افزودن یا حذف وب سرورها آسان است.
- **مقیاس‌بندی پایگاه‌داده**[3]: تکثیر[4] پایگاه‌داده و شاردینگ[5] تکنیک‌های رایج هستند.
- **تجزیه‌وتحلیل**: داده‌ها برای موفقیت تجاری اهمیت فزاینده‌ای دارند. ادغام یک راه‌حل تجزیه‌وتحلیل برای کوتاه‌کننده URL می‌تواند به پاسخ به سؤالات مهمی مانند: تعداد افرادی که روی یک لینک کلیک می‌کنند کمک کند؟ چه زمانی روی لینک کلیک می‌کنند؟ و غیره.

---

1 Rate limiter
2 Web server scaling
3 Database scaling
4 replication
۵ شاردینگ(sharding) پایگاه داده یک روش برای توزیع یک مجموعه داده‌ی بزرگ بر روی چندین پایگاه داده است که می‌توانند در چندین ماشین مختلف ذخیره شوند. این کار باعث می‌شود که مجموعه داده‌های بزرگ به قطعات کوچکتر تقسیم شوند و در چندین گره داده ذخیره شوند، که در نتیجه ظرفیت کلی ذخیره‌سازی سیستم را افزایش می‌دهد. شاردینگ یکی از روش‌های مقیاس‌پذیری افقی یا افزایش مقیاس خارجی است.

- دردسترس‌بودن[1]، یکپارچگی [2] و قابلیت اطمینان[3]. این مفاهیم هسته اصلی موفقیت هر سیستم بزرگی هستند. ما آنها را در فصل ۱ به‌تفصیل مورد بحث قرار دادیم، لطفاً این موضوعات را در صورت فراموشی، یادآوری کنید.

1 Availability
2 consistency
3 reliability

منابع و مراجع فصل ۸

[1] A RESTful Tutorial: https://www.restapitutorial.com/index.html
[2] Bloom filter: https://en.wikipedia.org/wiki/Bloom_filter

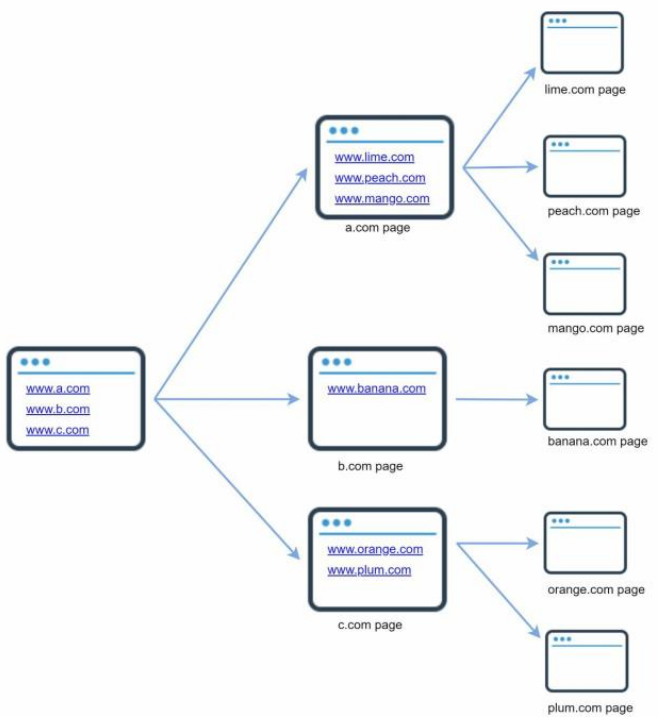
فصل ۹

طراحی WEB CRAWLER

در این فصل، ما بر طراحی خزنده وب (Web Crawler) تمرکز می‌کنیم: این مورد یک سؤال مصاحبه طراحی سیستم بسیار جالب و کلاسیک هست.

خزنده وب یا web crawler که به‌عنوان یک ربات یا spider شناخته می‌شود که طرز گسترده‌ای توسط موتورهای جستجو برای کشف محتوای جدید یا به‌روز شده در وب استفاده می‌شود. محتوا می‌تواند یک صفحه وب، یک تصویر، یک ویدئو، یک فایل PDF و غیره باشد. یک خزنده وب با جمع‌آوری چند صفحه وب شروع می‌کند و سپس پیوندهای موجود در آن صفحات را برای جمع‌آوری محتوای جدید دنبال می‌کند. شکل ۱-۹ یک مثال تصویری از فرایند خزیدن[1] را نشان می‌دهد.

1 crawl



شکل ۱-۹

خزنده (crawler) برای اهداف بسیاری استفاده می‌شود:

ایندکس موتور جستجو[1]: این رایج‌ترین مورد استفاده از خزنده‌ها است. یک خزنده صفحات وب را جمع‌آوری می‌کند تا یک ایندکس‌دهی داخلی[2] برای موتورهای جستجو ایجاد کند. به‌عنوان‌مثال، Googlebot خزنده وب پشت موتور جستجوی گوگل است.

وب آرشیو[3]: این کار فرایند جمع‌آوری اطلاعات از وب برای نگهداری داده‌ها برای استفاده‌های بعدی است. به‌عنوان‌مثال، بسیاری از کتابخانه‌های مطرح در جهان از خزنده‌ها برای آرشیو وب‌سایت‌ها استفاده می‌کنند. نمونه‌های قابل‌توجه کتابخانه کنگره ایالات متحده [۱] و آرشیو وب اتحادیه اروپا [۲] است.

---

1 Search engine indexing
2 local index
3 WEB ARCHIVING

وب‌کاوی[1]: رشد انفجاری وب فرصتی بی‌سابقه برای داده‌کاوی ارائه می‌دهد. وب‌کاوی به کشف دانش مفید از اینترنت کمک می‌کند. برای مثال، شرکت‌های مالی مهم و برتر از خزنده‌ها برای دانلود جلسات سهام‌داران و گزارش‌های سالانه آن‌ها برای یادگیری نکات کلیدی و روش‌های جدید و اطلاعات در مورد وضعیت شرکت‌ها استفاده می‌کنند.

وب مانیتورینگ[2]. خزنده‌ها به نظارت بر نقض حق چاپ و علائم تجاری از طریق اینترنت کمک می‌کنند. به‌عنوان‌مثال، Digimarc [۳] از خزنده‌ها برای کشف آثار و گزارش‌های دزدی ادبی و علمی و آثار هنری استفاده می‌کند.

پیچیدگی توسعه یک خزنده وب بستگی به مقیاسی[3] دارد که ما قصد پشتیبانی از آن را داریم که حتی می‌تواند یک پروژه کوچک دانشگاهی باشد که تکمیل آن فقط چند ساعت طول می‌کشد یا یک پروژه غول‌پیکر که نیاز به بهبود مستمر از یک تیم مهندسی حرفه‌ای دارد؛ بنابراین، ما قابلیت مقیاس‌دهی و ویژگی‌های قابل پشتیبانی را در زیر بررسی می‌کنیم.

مرحله ۱ - درک مسئله و شروع به طراحی

الگوریتم‌های اصلی یک خزنده وب ساده است:

۱. باتوجه‌به مجموعه‌ای از URLها، تمام صفحات وب آدرس داده شده توسط URLها را دانلود کنید.
۲. همه URLها را از این صفحات وب استخراج کنید.
۳. همه URLهای جدید را به لیست URLهایی که باید دانلود شوند اضافه کنید. این ۳ مرحله را تکرار کنید.

آیا یک خزنده وب واقعاً به‌سادگی این الگوریتم اصلی کار می‌کند؟ نه دقیقاً. طراحی یک خزنده وب با مقیاس‌پذیری بالا یک کار بسیار پیچیده است. بعید است که کسی بتواند یک خزنده وب عظیم در طول مدت مصاحبه طراحی کند. قبل از پرش به طرح، باید سؤالاتی را برای درک الزامات و تعیین محدوده طراحی بپرسیم:

---

1 Web mining
2 Web monitoring
3 QUERY PER SECOND

کاندید: هدف اصلی خزنده چیست؟ آیا برای ایندکس موتورهای جستجو، داده‌کاوی یا چیز دیگری استفاده می‌شود؟

مصاحبه‌کننده: ایندکس موتورهای جستجو.

کاندید: خزنده وب در ماه چند صفحه وب را جمع‌آوری می‌کند؟

مصاحبه‌کننده: ۱ میلیارد صفحه در هر ماه.

کاندید: چه نوع محتوایی شامل می‌شود؟ فقط HTML یا سایر انواع محتوا مانند PDF و تصاویر نیز می‌شود؟

مصاحبه‌کننده: فقط HTML.

کاندید: آیا صفحات وب جدید اضافه شده یا ویرایش شده را در نظر بگیریم؟

مصاحبه‌کننده: بله، باید صفحات وب تازه اضافه شده یا ویرایش شده را در نظر بگیریم.

کاندید: آیا باید صفحات HTML خزیده شده از وب را ذخیره کنیم؟

مصاحبه‌کننده: بله، تا ۵ سال.

کاندید: چگونه صفحات وب با محتوای تکراری را مدیریت کنیم؟

مصاحبه‌کننده: صفحاتی که محتوای تکراری دارند باید نادیده گرفته شوند.

در بالا تعدادی از نمونه سؤالاتی وجود دارد که می‌توانید از مصاحبه‌کننده خود بپرسید. درک الزامات و روشن‌شدن ابهامات مهم است. حتی اگر از شما خواسته شود که یک محصول ساده مانند خزنده وب طراحی کنید، ممکن است شما و مصاحبه‌کننده فرضیات یکسانی نداشته باشید. علاوه بر قابلیت‌هایی که باید با مصاحبه‌کننده خود توضیح دهید، در واقع بسیار مهم است که ویژگی‌های زیر که مربوط به یک خزنده وب باکیفیت است را یادداشت کنید:

- **مقیاس‌پذیری**[1]: وب بسیار بزرگ است. میلیاردها صفحه وب وجود دارد. خزنده وب باید با استفاده از روش‌های موازی‌سازی[2] بسیار کارآمد باشد.

1 Scalability
2 parallelization

- **مقاوم‌بودن**[1]: وب پر از تله است. HTMLهای بدطراحی‌شده و ناکارآمد، سرورهای که پاسخی نمی‌دهند، خرابی، لینک‌های مخرب و غیره که همه‌وهمه بسیار رایج هستند. خزنده باید تمام آن موارد را کنترل و بررسی کند.
- **رفتارهای مؤدبانه**[2]: خزنده نباید در یک بازه زمانی کوتاه درخواست‌های زیادی از یک وب‌سایت ارائه کند.
- **توسعه‌پذیری**[3]: سیستمی انعطاف‌پذیر است که حداقل تغییرات لازم برای پشتیبانی از انواع محتوای جدید موردنیاز را داشته باشد. به‌عنوان‌مثال، اگر بخواهیم در آینده فایل‌های تصویری را crawl کنیم، نباید نیازی به طراحی مجدد کل سیستم داشته باشیم.

تخمین طراحی یک خزنده وب

برآوردهای زیر بر اساس بسیاری از مفروضات است و مهم است که پاسخ مصاحبه‌کننده در یک صفحه برآورد کنید.

به یاد داریم که QPS[4] به معنی تعداد کوئری‌ها در ثانیه بوده و مقدار حداکثر آن را $Peak\ QPS$ تعریف کردیم.

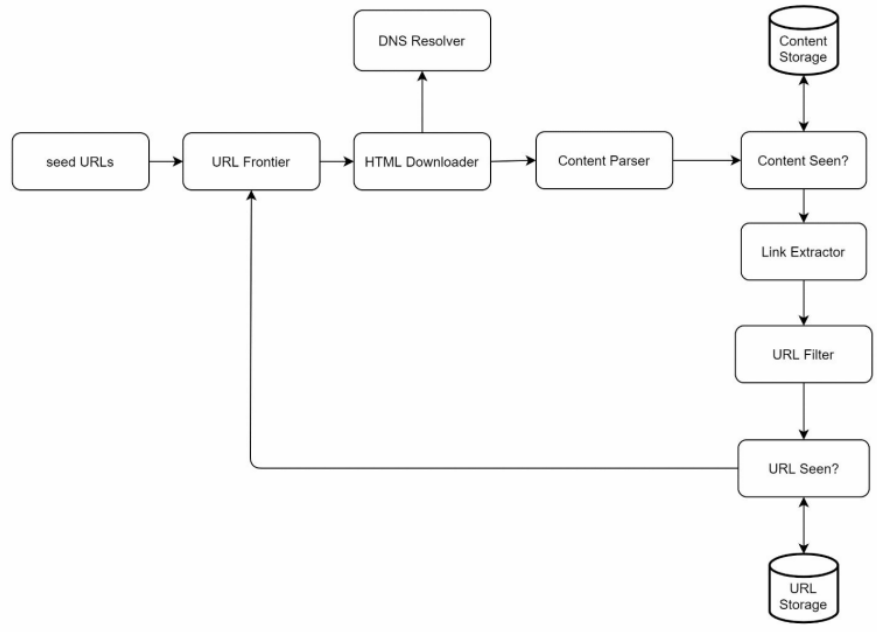
- فرض کنید هر ماه ۱ میلیارد صفحه وب دانلود می‌شود.
- QPS: $1{,}000{,}000{,}000 / 30\ days / 24\ hours / 3600\ seconds = \sim 400\ pages\ per\ second$
- $Peak\ QPS = 2 * QPS = 800$
- فرض کنید اندازه متوسط صفحه وب 500k است.

1 Robustness
2 Politeness
3 Extensibility
4 QUERY PER SECOND

- ۱ میلیارد صفحه 500k x ← 500 ترابایت فضای ذخیره‌سازی در ماه می‌خواهد. اگر در مورد واحدهای ذخیره‌سازی دیجیتال دچار ابهام هستید، مجدداً بخش «توان دو[1]» در فصل ۲ را مرور کنید.
- با فرض ذخیره داده‌ها به مدت پنج سال، ۵۰۰ ترابایت × ۱۲ ماه × ۵ سال = ۳۰ پتابایت. یک فضای ذخیره‌سازی حدود ۳۰ PB برای ذخیره محتوای پنج ساله موردنیاز است.

مرحله ۲ - طراحی سطح پیشرفته

پس از مشخص‌شدن الزامات طراحی، به سمت طراحی سطح پیشرفته می‌رویم. با الهام از مطالعات قبلی در مورد خزیدن وب [۴] [۵]، ما یک طراحی سطح پیشرفته را همان‌طور که در شکل ۲-۹ نشان‌داده‌شده است، پیشنهاد می‌کنیم.



شکل ۲-۹

1 Power of 2

ابتدا، ما هر جزء طراحی را بررسی می‌کنیم تا عملکرد آنها را درک کنیم. سپس، روند کار خزنده را مرحله‌به‌مرحله بررسی می‌کنیم.

واحد Seed URLs

یک خزنده وب از URLهای اولیه به‌عنوان نقطه شروع برای فرایند خزیدن استفاده می‌کند. به‌عنوان‌مثال، برای خزیدن تمام صفحات وب از وب‌سایت یک دانشگاه، یک راه ساده برای انتخاب URLهای اولیه استفاده از نام دامنه دانشگاه است. برای خزیدن در کل وب، باید در انتخاب URLهای اولیه خلاق باشیم. یک URL اولیه خوب به‌عنوان نقطه شروع خوبی است که خزنده می‌تواند از آن برای عبور از لینک‌های زیادی که ممکن است استفاده کند بهره‌برداری کند. استراتژی کلی این است که کل فضای URL را به فضای کوچک‌تر تقسیم کنید. اولین رویکرد پیشنهادی مبتنی بر موقعیت مکانی است؛ زیرا کشورهای مختلف ممکن است وب‌سایت‌های مطرح متفاوتی داشته باشند. راه دیگر این است که URLهای موردنظر را بر اساس موضوعات انتخاب کنید. به‌عنوان‌مثال، ما می‌توانیم فضای URL را به دسته یا گروه‌های شامل: خرید، ورزش، مراقبت‌های بهداشتی و غیره تقسیم کنیم. از شما انتظار نمی‌رود که پاسخ کاملی بدهید. فقط عمیق‌تر فکر کنید.

واحد URL Frontier

اکثر خزنده‌های وب که طراحی جدیدتری دارند، حالت خزیدن را به دو قسمت اصلی تقسیم می‌کنند:

- آماده برای دانلودشدن.
- دانلود شده.

ساختاری که URLهایی را که در صف دانلود و سپس صف ذخیره داده‌ها قرار می‌دهد را URL Frontier می‌نامند. شما می‌توانید به این صف FIFO[1] اشاره کنید.

1 First-in-First-out

واحد HTML Downloader

دانلود کننده HTML، صفحات وب را از اینترنت دانلود می‌کند. این URLها توسط URL Frontier ارائه یا تهیه می‌شوند.

واحد DNS Resolver

برای دانلود یک صفحه وب، یک URL باید به یک آدرس IP ترجمه شود. دانلود کننده HTML با DNS Resolver تماس می‌گیرد تا آدرس IP مربوطه را برای URL دریافت کند. به‌عنوان‌مثال، URL مربوط به www.wikipedia.org در تاریخ 2019/5/3 به آدرس IP مانند 198.35.26.96 تبدیل شده است.

واحد Content Parser

پس از دانلود یک صفحه وب، باید آن را تجزیه و اعتبارسنجی کرد؛ زیرا صفحات وب ناقص و خراب می‌توانند مشکلات زیادی را ایجاد کنند و فضای ذخیره‌سازی را هدر دهند. پیاده‌سازی تجزیه‌کننده محتوا[1] در سرور crawl، روند خزیدن[2] را آهسته می‌کند؛ بنابراین تجزیه‌کننده محتوا یک قسمت جداگانه در این طرح است.

واحد Content Seen?

تحقیقات آنلاین [۶] نشان می‌دهد که ۲۹٪ از صفحات وب محتوای تکراری هستند، که ممکن است باعث شود یک محتوا چندین بار ذخیره شود. ما گزینه «آیا محتوا دیده شده است؟[3]» را معرفی می‌کنیم که ساختار داده‌ای برای حذف موارد غیرلازم و غیر قابل استفاده در داده‌ها و کوتاه‌کردن زمان پردازش آن است و به شناسایی محتوای جدیدی که قبلاً در سیستم ذخیره شده است کمک می‌کند. برای مقایسه دو سند HTML، می‌توانیم آنها را کاراکتر به کاراکتر مقایسه کنیم. بااین‌حال، این روش کند و زمان بر است، به خصوص زمانی که میلیاردها صفحه

1 content parser
2 crawling
3 Content Seen

وب درگیر هستند. یک راه کارآمد برای انجام این کار، مقایسه مقادیر هش (hash) دو صفحه وب است [۷].

واحد Content Storage

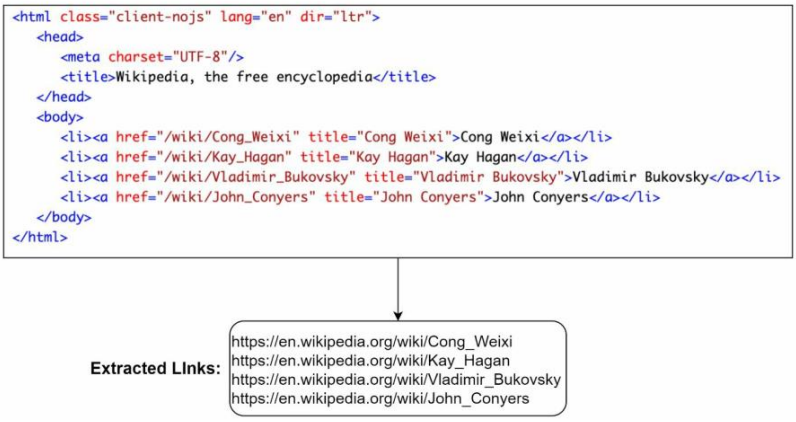
این یک سیستم ذخیره‌سازی برای ذخیره محتوای HTML است. انتخاب سیستم ذخیره‌سازی به عواملی مانند نوع داده، اندازه داده، دفعات دسترسی، طول عمر و ... بستگی دارد. برای این حالت از هر دو سخت‌افزار دیسک و حافظه موقت[1] یا کَش استفاده می‌شود.

• بیشتر محتوا بر روی دیسک ذخیره می‌شود؛ زیرا مجموعه‌داده آن‌قدر بزرگ است که در حافظه موقت جا نمی‌شود.

• محتوای محبوب یا مطرح برای کاهش تأخیر در حافظه موقت نگهداری می‌شود.

واحد URL Extractor

استخراج‌کننده URL پیوندها را از صفحات HTML تجزیه و استخراج می‌کند. شکل ۳-۹ نمونه‌ای از فرایند استخراج لینک را نشان می‌دهد. مسیرهای نسبی با افزودن پیشوند "https://en.wikipedia.org" به URLهای مطلق تبدیل می‌شوند.



شکل ۳-۹

1 memory

واحد URL Filter

فیلتر URL انواع خاصی از محتوا را تأیید و پردازش می‌کند و انواع پسوند فایل، لینک‌های خطا و آدرس‌های اینترنتی را در URLهای که در "لیست سیاه[1]" هستند را حذف می‌کند.

واحد URL Seen

«آیا URL دیده شده است؟[2]» یک ساختار داده‌ای است که URLهایی را که درگذشته پردازش شده است یا در Frontier در صف پردازش قرار داده شده‌اند را ردیابی می‌کند. «آیا URL دیده شده است؟» به جلوگیری از اضافه‌کردن چندین بار URL یکسان کمک می‌کند؛ زیرا این امر می‌تواند بار سرور را افزایش دهد و باعث ایجاد حلقه‌های بی‌نهایت تکراری شود. فیلتر Bloom و جدول هش[3] تکنیک‌های رایج برای پیاده‌سازی «آیا URL دیده شده است؟» هستند. ما در اینجا به اجرای دقیق بلوم فیلتر و جدول هش نمی‌پردازیم. برای اطلاعات بیشتر به منابع مرجع [۴] [۸] مراجعه کنید.

واحد URL Storage

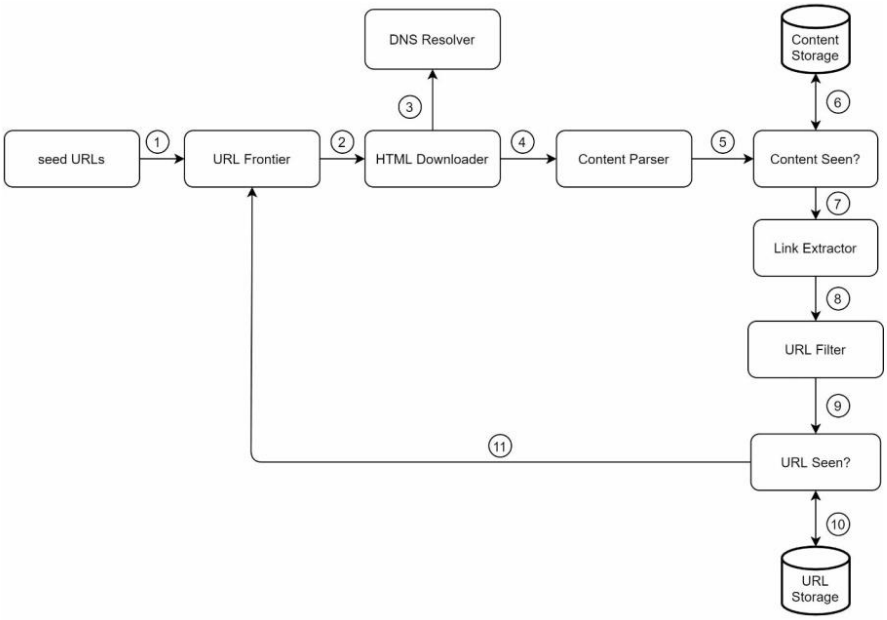
مورد Storage URL در واقع URLهای قبلاً بازدید شده را ذخیره می‌کند. تا کنون، ما در مورد هر جزء سیستم بحث کرده‌ایم. در مرحله بعد، آنها را کنار هم قرار می‌دهیم تا روند کار را توضیح دهیم.

رویه‌کاری خزنده وب

برای توضیح بهتر رویه‌کاری در یک خزنده وب، دنباله‌ای از عددها در نمودار طراحی اضافه می‌شوند که در شکل ۴-۹ نشان‌داده‌شده است.

---

1 blacklisted
2 URL Seen?
3 hash table



شکل ۴-۹

مرحله ۱: seed URLها را به URL Frontier اضافه کنید.

مرحله ۲: HTML Downloader لیستی از URLها را از URL Frontier دریافت می‌کند.

مرحله ۳: HTML Downloader آدرس IP URLها را از DNS Resolver دریافت می‌کند و شروع به دانلود می‌کند.

مرحله ۴: Content Parser صفحات HTML را تجزیه‌وتحلیل می‌کند و بررسی می‌کند که آیا صفحات نامعتبر هستند.

مرحله ۵: پس از تجزیه و تأیید محتوا به بلوک «محتوا دیده می‌شود؟[1]» ارسال می‌شود.

مرحله ۶: مؤلفه «Content Seen» بررسی می‌کند که آیا یک صفحه HTML قبلاً در حافظه وجود دارد یا خیر.

1 Content Seen?

- اگر صفحه HTML مورد هدف در فضای ذخیره‌سازی موجود بوده باشد، به این معنی است که همان محتوا در یک URL دیگر قبلاً پردازش شده است. در این حالت این صفحه HTML حذف می‌شود.
- اگر صفحه HTML مورد هدف در فضای ذخیره‌سازی موجود نباشد، پس سیستم قبلاً همان محتوا را پردازش نکرده است. در نهایت محتوا به Link Extractor ارسال می‌شود.

مرحله ۷: استخراج کننده لینک یا Link Extractor، لینک‌ها را از صفحات HTML استخراج می‌کند.

مرحله ۸: لینک‌های استخراج شده به URL filter منتقل می‌شوند.

مرحله ۹: پس از links filter، ادامه کار به قسمت "URL Seen?" ارسال می‌شوند.

مرحله ۱۰: مؤلفه «URL Seen» بررسی می‌کند که آیا URL از قبل در فضای ذخیره‌سازی وجود دارد یا خیر، اگر جواب مثبت باشد، قبلاً پردازش شده است و هیچ کاری لازم نیست انجام شود.

مرحله ۱۱: اگر URL قبلا پردازش نشده باشد، درنتیجه به URL Frontier اضافه می‌شود.

مرحله ۳ - عمیق‌شدن در طراحی

تابه‌حال ما در مورد طراحی سطح بالا بحث کرده‌ایم. در ادامه، مهم‌ترین اجزا و تکنیک‌های اساسی و ساختاری را به طور عمیق مورد بحث قرار خواهیم داد:

- الگوریتم جستجوی عمق اول[1] (DFS) در مقابل الگوریتم جستجوی سطح اول[2] (BFS)
- URL frontier
- HTML Downloader
- مقاوم‌بودن[3]

1 Depth-first search (DFS)
2 Breadth-first Search (BFS)
3 Robustness

- توسعه‌پذیری[1]
- شناسایی محتوای نامعتبر و خطا ساز و اجتناب از پردازش کردن آن‌ها.

DFS در مقابل BFS

شما می‌توانید کل ساختار وب را به‌عنوان یک گراف جهت‌دار در نظر بگیرید که در آن صفحات وب به‌عنوان گره‌ها[2] و لینک‌ها (URL) به‌عنوان یال‌ها[3] عمل می‌کنند. فرایند crawl را می‌توان به‌عنوان عبور از یک نمودار هدایت شده از یک صفحه وب به صفحه دیگر فرض کرد. دو الگوریتم متداول پیمایش نمودار DFS و BFS هستند. بااین‌حال، DFS معمولاً انتخاب خوبی نیست؛ زیرا عمق DFS می‌تواند بسیار عمیق و طولانی باشد. پس BFS معمولاً توسط خزنده‌های وب استفاده می‌شود و توسط صف FIFO[4] پیاده‌سازی می‌شود. در یک صف FIFO، آدرس‌ها یا URLهای بدون ترتیب و درهم تبدیل به آدرس‌ها یا URLهای مرتب و منظم شده و سپس در صف قرار می‌گیرند. اما این پیاده‌سازی دو مشکل دارد:

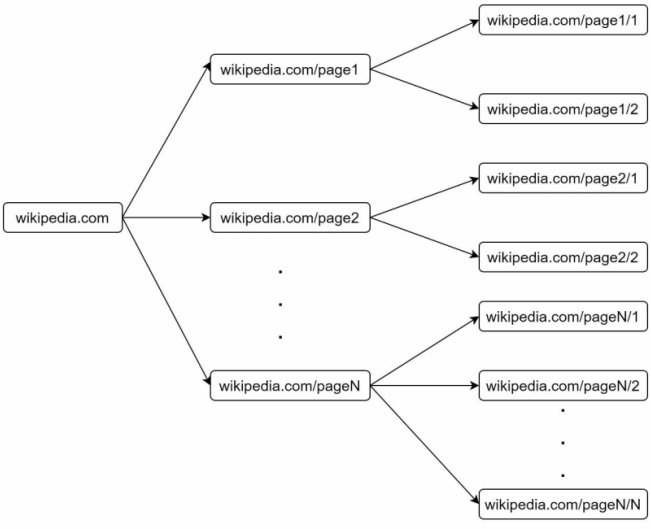
اکثر پیوندهای یک صفحه وب به همان Host پیوند داده می‌شوند. در شکل ۵-۹، به‌عنوان‌مثال تمام پیوندهای موجود در wikipedia.com پیوندهای داخلی هستند و خزنده وب را مشغول پردازش URLهای یک دامنه خاص مثلاً (wikipedia.com) می‌کند. هنگامی که crawler سعی می‌کند صفحات وب را به‌صورت موازی بارگیری کند، سرورهای ویکی‌پدیا پر از درخواست[5] می‌شوند. این به‌عنوان یک حالت «نامحترنامه[6]» در نظر گرفته می‌شود و ممکن است دامنه مورد crawler را محدود[7] کرده و درخواست‌ها را مسدود کند.

- استاندارد BFS اولویت و ترتیب URL را در نظر نمی‌گیرد. وسعت وب بسیار بزرگ است و هر صفحه از کیفیت و اهمیت یکسانی برخوردار نیست، بنابراین، ممکن است بخواهیم

---

1 Extensibility
2 nodes
3 edges
4 First-in-First-out
5 request
6 impolite
7 rate limit

URLها را باتوجه‌به رتبه صفحه[1]، ترافیک وب، تعداد دفعات به‌روزرسانی و دلایل دیگر اولویت‌بندی کنیم.



شکل ۵-۹

بررسی URL frontier

همین‌طور URL frontier به رفع این مشکلات کمک می‌کند. URL frontier یک ساختار داده‌ای است که URLهایی را برای دانلود ذخیره می‌کند. URL frontier یک جزء مهم برای اطمینان از رفتارهای محترمانه[2]، اولویت‌بندی URLها و به‌روز بودن URLها است. چند مقاله قابل‌توجه در مورد URL frontier در مرجع [۵] [۹] ذکر شده است. یافته‌های حاصل از این مقالات به شرح زیر است:

1 page ranks
2 politeness

رفتار محترمانه (Politeness)

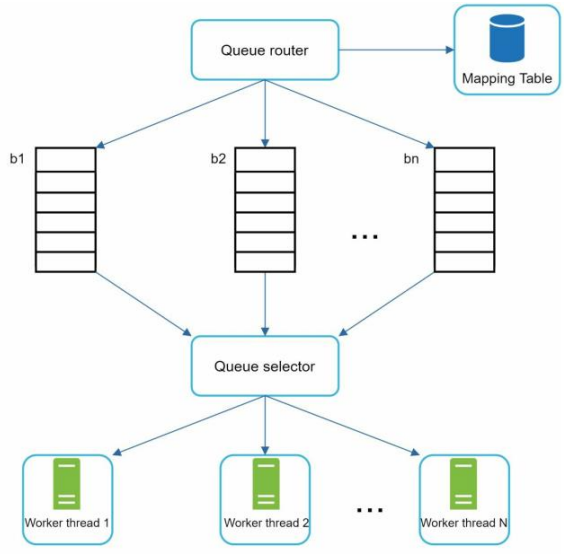
به‌طورکلی، یک خزنده وب باید در مدت‌زمان کوتاه از ارسال درخواست‌های[1] بیش از حد به سروری یا دامنه یا host یکسان و مشابه خودداری کند. ارسال درخواست‌های بیش از حد به‌عنوان "impolite" یا حتی به‌عنوان حمله denial-of-service[2] یا به‌اختصار (DOS) در نظر گرفته می‌شود. به‌عنوان‌مثال، بدون هیچ محدودیتی، خزنده می‌تواند هزاران درخواست را در هر ثانیه به یک وب‌سایت ارسال کند. این کار می‌تواند سرورهای وب را تحت‌تأثیر قرار دهد. ایده کلی اجرای politeness این است که هر بار یک صفحه را از یک host دانلود کنید. می‌توان یک تأخیر بین دو تسک مربوط به دانلود[3] اضافه کرد. در این حالت برای کنترل محدودیت ناشی از politeness با حفظ نگاشت‌ها و حالت‌های مختلف از نام وب‌سایت‌ها و آدرس‌های اینترنتی برای دانلود صفحه‌ها، درخواست‌ها در چندین thread یا worker اجرا می‌شود. هر thread دانلودکننده، یک صف FIFO جداگانه دارد و فقط URLهای به‌دست‌آمده از آن صف را دانلود می‌کند. شکل ۶-۹ طرحی را نشان می‌دهد که politeness را مدیریت می‌کند.

---

1 requests

۲ ارسال حجم عظیمی از ترافیک به سمت هدف، مانند بسته‌های IP یا درخواست‌های HTTP، تا زمانی که سرور قادر به رسیدگی به آنها نباشد.

3 download tasks



شکل ۶-۹

- مسیریاب صف[1]: تضمین می‌کند که هر صف (b1، b2، bn ...) فقط حاوی URLهایی از همان host باشد.
- جدول نگاشت[2]: هر host را به یک صف اختصاص می‌دهد.

Host	Queue
wikipedia.com	b1
apple.com	b2
...	...
nike.com	bn

جدول ۱-۹

- **صف FIFO مربوط b1, b2 تا bn:** هر صف حاوی URLهایی از همان host است.
- **انتخاب‌گر صف[3]:** هر worker thread به یک صف FIFO اختصاص داده می‌شود و فقط URLها را از آن صف به‌خصوص دانلود می‌کند. منطق نحوه انتخاب توسط صف به کمک انتخاب‌گر صف انجام می‌شود.

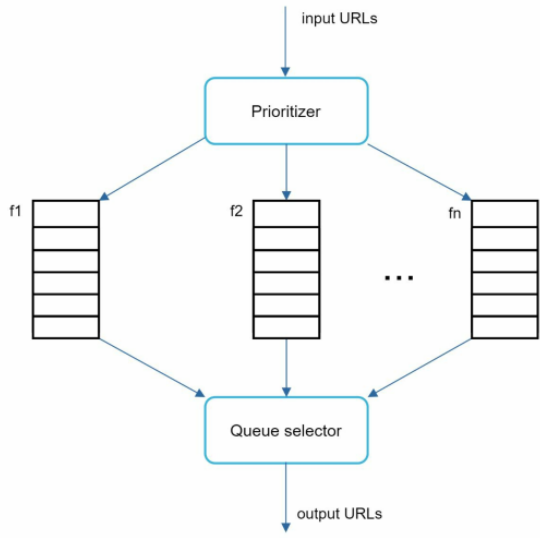
---

1 Queue router
2 Mapping table
3 Queue selector

- **1 Worker thread تا N:** یک Worker Thread صفحات وب را پشت‌سرهم از ردیف host دانلود می‌کند. همین‌طور می‌توان یک تأخیر بین دو دانلود تسک[1] اضافه کرد.

اولویت (Priority)

یک پست تصادفی از یک «تالار گفتگو[2]» درباره محصولات شرکت اپل وزن بسیار متفاوتی با پست‌های صفحه اصلی اپل دارد. حتی اگر هر دوکلمه کلیدی "Apple" را دارند، پس برای خزنده وب منطقی است که ابتدا صفحه اصلی اپل را crawl کند. ما URLها را بر اساس اهمیت آن اولویت‌بندی می‌کنیم که می‌تواند با رتبه صفحه[3] [۱۰]، ترافیک وب‌سایت، تعداد دفعات به‌روزرسانی، و غیره اندازه‌گیری شود. «اولویت ساز[4]» مؤلفه‌ای است که اولویت‌بندی URL را مدیریت می‌کند. برای اطلاعات عمیق در مورد این مفهوم به مرجع [۵] [۱۰] مراجعه کنید. شکل ۷-۹ طرحی را نشان می‌دهد که اولویت URL را مدیریت می‌کند.



شکل ۷-۹

1 download tasks
2 discussion forum
3 PageRank
4 Prioritizer

اولویت‌بندی: URLها را به‌عنوان ورودی می‌گیرد و اولویت‌ها را محاسبه می‌کند.

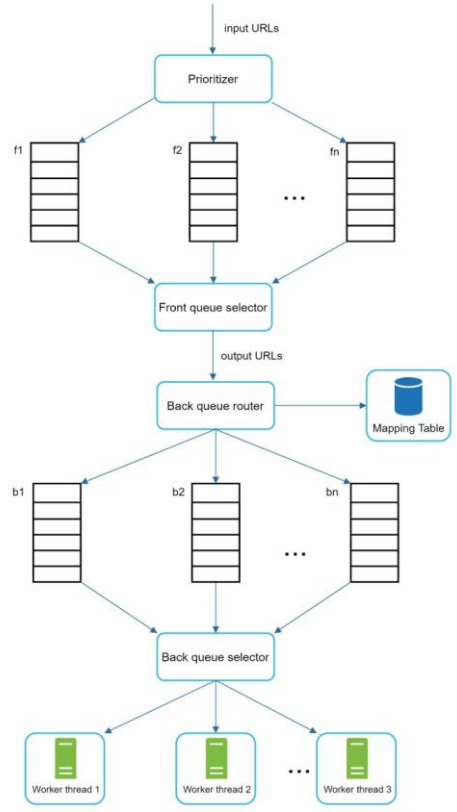
صف f1 تا fn: هر صف دارای یک اولویت است. صف‌های با اولویت بالا با احتمال بیشتری انتخاب می‌شوند.

انتخابگر صف[1]: به‌صورت تصادفی یک صف را انتخاب کنید، به‌گونه‌ای که احتمال انتخاب صف‌های با اولویت بالاتر بیشتر باشد.

شکل ۸-۹ طراحی URL frontier را نشان می‌دهد و شامل دو ماژول است:

صف‌های جلویی: اولویت‌بندی را مدیریت می‌کند.

صف‌های پشتی: که politeness را مدیریت می‌کند.



تصویر ۸-۹

1 Queue selector

تازه‌سازی (Freshness)

صفحات وب به طور مداوم اضافه، حذف و ویرایش می‌شوند. یک خزنده وب باید به طور دوره‌ای صفحات دانلود شده را خزیدن مجدد[1] کند تا مجموعه داده‌های ما را تازه نگه دارد. خزیدن مجدد، همه URLها زمان‌بر بوده و منابع زیادی را درگیر می‌کند. چند استراتژی برای بهینه‌سازی در تازه‌سازی لینک‌ها و URLها به شرح زیر ذکر شده است:

- خزیدن مجدد بر اساس تاریخچه به‌روزرسانی صفحات وب.
- URLها را اولویت‌بندی کند و صفحات مهم را در ابتدای کار و بیشتر مرور کند.

ذخیره‌سازی برای URL Frontier

مبحث crawl در دنیای واقعی برای موتورهای جستجو به طور خاص مورد استفاده قرار می‌گیرد، تعداد URLها در frontier می‌تواند بیش از صدها میلیون نمونه باشد [۴]. قراردادن همه‌چیز در حافظه موقت، به هیچ وجه بادوام و مقیاس‌پذیر[2] نیست، همین‌طور نگه‌داشتن همه‌چیز در دیسک بسیار نامطلوب است و این فقط به دلیل کندی دیسک نیست؛ بلکه به دلیل این است که می‌تواند به گلوگاهی[3] برای crawler تبدیل شود. پس یک رویکرد ترکیبی را باید در نظر بگیریم. اکثر URLهاروی دیسک ذخیره می‌شوند، بنابراین مشکلی بابت فضای ذخیره‌سازی وجود ندارد. برای کاهش هزینه خواندن از دیسک و نوشتن روی دیسک، بافرهایی را در حافظه موقت برای در صف قراردادن یا از صف خارج‌کردن[4] داده‌ها، نگهداری می‌کنیم که در نهایت داده‌ها درون بافر به‌صورت دوره‌ای روی دیسک نوشته می‌شوند.

---

1 Recrawl
2 SCALABLE
3 BOTTLENECK
4 ENQUEUE/DEQUEUE

HTML Downloader

HTML Downloader صفحات وب را با استفاده از پروتکل HTTP از اینترنت دانلود می‌کند. قبل از بحث در مورد HTML Downloader، ابتدا به پروتکل حذف روبات‌ها[1] نگاه می‌کنیم.

بررسی Robots.txt

Robots.txt که Robots Exclusion Protocol یا پروتکل حذف روبات‌ها نامیده می‌شود، استانداردی است که توسط وب‌سایت‌ها برای برقراری ارتباط با خزنده‌ها استفاده می‌شود و مشخص می‌کند که خزنده‌ها مجاز به دانلود چه صفحاتی هستند. قبل از تلاش برای خزیدن یک وب‌سایت، خزنده ابتدا باید robots.txt مربوطه خود را بررسی کند و قوانین آن را دنبال کند. برای جلوگیری از بارگیری مجدد فایل robots.txt باید نتیجه‌های موجود در فایل را کَش می‌کنیم. فایل به‌صورت دوره‌ای دانلود و در حافظه کَش ذخیره می‌شود. در اینجا یک قطعه از فایل robots.txt از https://www.amazon.com/robots.txt گرفته شده است. برخی از دایرکتوری‌ها مانند creatorhub برای ربات گوگل غیرمجاز هستند. مانند موارد زیر:

```
User-agent: Googlebot
Disallow: /creatorhub*/
Disallow: /rss/people/*/reviews
Disallow: /gp/pdp/rss/*/reviews
Disallow: /gp/cdp/member-reviews/
Disallow: /gp/aw/cr/
```

علاوه بر robots.txt و بهینه‌سازی عملکردی[2] مفهوم مهم دیگری وجود دارد که در بحث HTML downloader پوشش خواهیم داد.

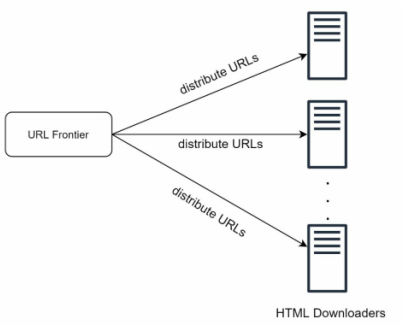
1 ROBOTS EXCLUSION PROTOCOL
2 PERFORMANCE OPTIMIZATION

بهینه‌سازی عملکردی

در زیر لیستی از بهینه‌سازی عملکردی برای HTML downloader آمده است.

۱. خزیدن توزیع شده (Distributed crawl)

برای دستیابی به کارایی بالا، crawl jobs در چندین سرور توزیع می‌شوند و هر سرور چندین thread را اجرا می‌کند. فضای URL به قطعات کوچک‌تر تقسیم می‌شود؛ بنابراین هر دانلود کننده مسئول زیرمجموعه‌ای از URLها است. شکل ۹-۹ نمونه‌ای از خزیدن توزیع شده[1] را نشان می‌دهد.



شکل ۹-۹

۲. Cache DNS Resolver

DNS Resolver یک گلوگاه برای خزنده‌ها است؛ زیرا درخواست‌های DNS ممکن است به دلیل ماهیت هم‌زمان[2] بسیاری از رابط‌های DNS زمان‌بر باشد. زمان پاسخ‌دهی DNS بین ۱۰ تا ۲۰۰ میلی ثانیه است. هنگامی که یک درخواست به DNS توسط یک thread خزنده[3] انجام می‌شود، سایر threadها تا زمانی که اولین درخواست تکمیل شود مسدود می‌شوند. نگهداری از DNS cache برای جلوگیری از تماس مکرر DNS[4] یک تکنیک

1 DISTRIBUTED CRAWL
2 synchronous
3 crawler thread
4 calling DNS frequently

مؤثر برای بهینه سازی سرعت است. حافظه کَش از DNS ما نام دامنه را در نگاشت کننده آدرس IP نگه می‌دارد و به‌صورت دوره‌ای توسط cron job[۱] به‌روزرسانی می‌شود.

۳. Locality

سرورهای خزنده را به‌صورت جغرافیایی در سروهای مختلف توزیع کنید. هنگامی که سروهای خزنده به host یا وب‌سایت‌های مورد هدف نزدیک‌تر هستند، خزنده‌ها زمان دانلود سریع‌تری را تجربه می‌کنند. طراحی خزنده بر اساس محل جغرافیایی سرور برای اکثر اجزای سیستم‌های آن اعمال می‌شود و این شامل مواردی مثل سرورهای خزنده، کَش، صف، ذخیره‌سازی و غیره است.

۴. Timeout

برخی از وب سرورها به‌کندی پاسخ می‌دهند یا ممکن است اصلاً هیچ پاسخ ندهند. برای جلوگیری از زمان انتظار طولانی، حداکثر زمان انتظار مشخص شده در نظر گرفته شده است. اگر host موردنظر در مدت‌زمان از پیش تعریف شده پاسخ ندهد، خزنده ادامه کار را متوقف می‌کند و برخی از صفحات دیگر را crawl می‌کند.

مقاوم‌بودن (Robustness)

علاوه بر بهینه‌سازی عملکردی[۲]، مقاوم‌بودن[۳] نیز یک نکته بسیار مهم است. ما چند روش برای بهبود مقاوم کردن سیستم ارائه می‌دهیم:

هش‌کردن مداوم[۴]: این به توزیع بارها بین دانلودکننده[۵] صفحه‌های وب کمک می‌کند. یک سرور **دانلود کننده** جدید را می‌توان با استفاده از روش «هش‌کردن مداوم» اضافه یا حذف کرد. برای جزئیات بیشتر به فصل ۵ مراجعه کنید.

---

۱ کرون جاب ابزاری است که در سیستم عامل‌های مبتنی بر یونیکس مانند لینوکس برای زمان‌بندی و اجرای خودکار وظایف به صورت دوره‌ای و در زمان‌های مشخص استفاده می‌شود.

2 performance optimization
3 Robustness
4 Consistent hashing
5 downloader

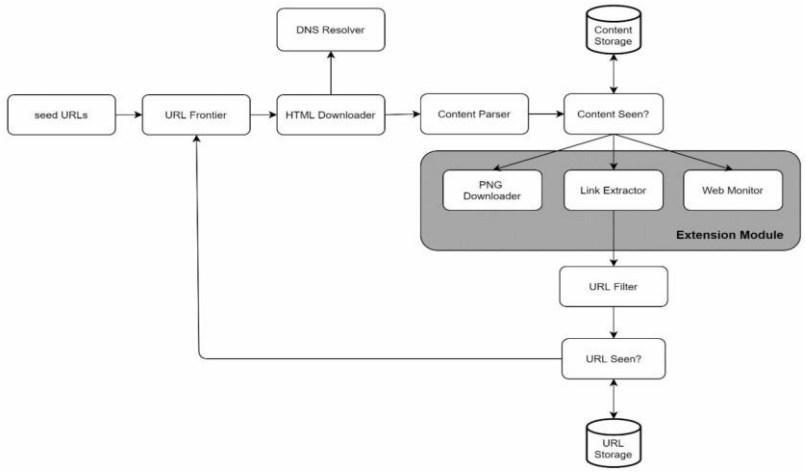
ذخیره وضعیت خزنده وب و داده‌های آن: برای محافظت در برابر خرابی‌ها، وضعیت خزنده وب و داده‌های آن را در یک سیستم ذخیره‌سازی نوشته می‌شوند. با بارگیری وضعیت‌ها و داده‌های ذخیره‌شده، می‌توان یک خزیدن ناقص و نصفه انجام یا متوقف شده را به‌راحتی از نو راه‌اندازی کرد.

رسیدگی به خطاها و استثناها: خطاها در یک سیستم در مقیاس بزرگ اجتناب‌ناپذیر و رایج هستند. خزنده‌ی وب باید استثنائات را باظرافت و بدون ازکارانداختن سیستم مدیریت کند.

اعتبارسنجی داده‌ها[1]: این یک اقدام مهم برای جلوگیری از خطاهای سیستم است.

توسعه‌پذیری (Extensibility)

همان‌طور که تقریباً هر سیستمی در حال تکامل است، یکی از اهداف طراحی این است که سیستم را به‌اندازه کافی انعطاف‌پذیر کند تا از انواع محتوای جدید پشتیبانی کند. خزنده را می‌توان با وصل کردن ماژول‌های جدید گسترش داد. شکل ۱۰-۹ نحوه افزودن ماژول‌های جدید را نشان می‌دهد.



شکل ۱۰-۹

1 Data validation

- ماژول PNG Downloader برای دانلود فایل‌های PNG وصل شده است.
- ماژول Web Monitor برای نظارت بر وب و جلوگیری از نقض حق چاپ و علائم تجاری اضافه شده است.

شناسایی و جلوگیری از محتوای مشکل‌ساز

این بخش در مورد شناسایی و پیشگیری از محتوای اضافی، بی‌معنی یا مضر بحث می‌کند.

۱. محتوای اضافی

همان‌طور که قبلاً گفته شد، تقریباً ۳۰٪ از صفحات وب تکراری هستند. هش‌ها یا checksumها به شناسایی موارد تکراری کمک می‌کنند [۱۱].

۲. تله‌های عنکبوتی

تله‌های عنکبوتی[1] یک صفحه وب است که باعث ایجاد مشکل برای یک خزنده وب در یک حلقه بی‌نهایت می‌شود. برای مثال، ساختار دایرکتوری عمیق بی‌نهایت به‌صورت زیر فهرست شده است:

www.spidertrapexample.com/foo/bar/foo/bar/foo/bar/…

چنین تله‌های عنکبوتی را می‌توان با تنظیم حداکثر طول برای URLها برطرف کرد. بااین‌حال، هیچ راه‌حل یکسانی برای تشخیص تله‌های عنکبوت وجود ندارد. شناسایی وب‌سایت‌های حاوی تله‌های عنکبوت به دلیل تعداد موارد غیرعادی زیادی از صفحات وب که در چنین وب‌سایت‌هایی کشف شده‌اند، آسان است. توسعه الگوریتم‌های خودکار برای جلوگیری از تله‌های عنکبوت دشوار است. بااین‌حال، کاربر می‌تواند به‌صورت دستی یک تله عنکبوت را تأیید و شناسایی کند و یا آن وب‌سایت‌ها را از خزنده حذف کند یا برخی از فیلترهای URL سفارشی را اعمال کند.

1 Spider traps

۳. داده‌های نویزی

برخی از محتواها ارزش کمی دارند یا اصلاً ارزشی ندارند، مانند تبلیغات، تکه کدها، URLهای هرزنامه و غیره. این محتواها برای خزنده‌ها مفید نیستند و در صورت امکان باید حذف شوند.

مرحله ٤ – جمع‌بندی

در این فصل، ابتدا ویژگی‌های یک خزنده‌ی وب خوب را مورد بحث قرار دادیم: مقیاس‌پذیری[1]، politeness، توسعه‌پذیری[2] و مقاوم‌بودن[3]. سپس، طرحی را پیشنهاد کردیم و اجزای کلیدی آن را مورد بحث قرار دادیم. ساختن یک خزنده وب با قابلیت مقیاس‌پذیری به‌هیچ‌وجه یک کار ساده و بی‌اهمیت نیست؛ زیرا وسعت وب بسیار بزرگ و پر از تله است. با وجود اینکه ما موضوعات زیادی را پوشش داده‌ایم، هنوز بسیاری از نکات مربوط به این مبحث را نمی‌توانیم پوشش دهیم:

- **رِندرِ سمت سرور**[4]: وب‌سایت‌های متعددی از اسکریپت‌هایی مانند جاوا اسکریپت، AJAX و غیره برای ایجاد لینک در لحظه استفاده می‌کنند. اگر صفحات وب را مستقیماً دانلود و آنالیز کنیم، نمی‌توانیم پیوندهای ایجاد شده به‌صورت پویا را بازیابی کنیم. برای حل این مشکل، ابتدا رِندر سمت سرور (که رندر پویا نیز نامیده می‌شود) را قبل از تجزیه یک صفحه انجام می‌دهیم [۱۲].
- **فیلترکردن صفحات ناخواسته**: با ظرفیت ذخیره‌سازی و منابع محدود جهت خزیدن وب، وجود یک قسمت ضد هرزنامه[5] جهت شناسایی و فیلترکردن صفحات هرزنامه و کیفیت پایین مفید است [۱۳] [۱۴].

---

1 scalability
2 extensibility
3 robustness
4 Server-side rendering
5 anti-spam

- **شاردینگ[1] و تکثیر[2] پایگاه‌داده**: تکنیک‌هایی مانند تکثیر و شاردینگ برای بهبود دردسترس‌بودن[3]، مقیاس‌پذیری و قابلیت اطمینان[4] در لایه داده استفاده می‌شود.
- **مقیاس‌پذیری افقی**[5]: برای خزیدن در مقیاس بزرگ، صدها یا حتی هزاران سرور برای انجام دانلود تسک[6]موردنیاز است. نکته کلیدی این است که سرورها را در وضعیت stateless که در فصل ۱ توضیح داده شده است، نگه دارید.
- **دردسترس‌بودن**[7]، یکپارچگی[8] **و قابلیت اطمینان**[9]: این مفاهیم هسته اصلی موفقیت هر سیستم بزرگی هستند. ما این مفاهیم را به‌تفصیل در فصل ۱ مورد بحث قرار دادیم. حالا باید حافظه خود را در این موضوعات تجدید کنید.
- **تجزیه‌وتحلیل و آنالیز**: جمع‌آوری و تجزیه‌وتحلیل داده‌ها بخش مهمی از هر سیستمی است؛ زیرا داده‌ها جزء کلیدی برای تنظیم دقیق یا بهینه‌سازی هستند.

---

1 sharding
2 replication
3 availability
4 reliability
5 Horizontal scaling
6 download tasks
7 Availability
8 consistency
9 reliability

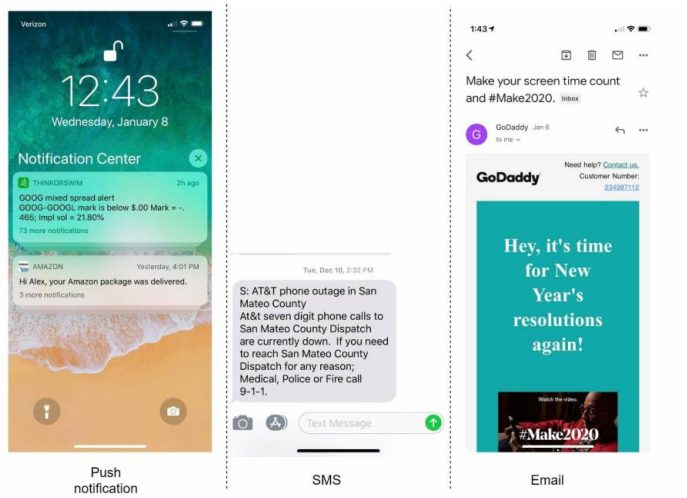
منابع و مراجع فصل ۹

[1] US Library of Congress: https://www.loc.gov/websites/
[2] EU Web Archive: http://data.europa.eu/webarchive
[3] Digimarc: https://www.digimarc.com/products/digimarc-services/piracy-intelligence
[4] Heydon A., Najork M. Mercator: A scalable, extensible web crawler World Wide Web, 2 (4) (1999), pp. 219-229
[5] By Christopher Olston, Marc Najork: Web Crawling.http://infolab.stanford.edu/~olston/publications/crawling_survey.pdf
[6] 29% Of Sites Face Duplicate Content Issues: https://tinyurl.com/y6tmh55y
[7] Rabin M.O., et al. Fingerprinting by random polynomials Center for Research in Computing Techn., Aiken Computation Laboratory, Univ. (1981)
[8] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors,"Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970.
[9] Donald J. Patterson, Web Crawling:https://www.ics.uci.edu/~lopes/teaching/cs221W12/slides/Lecture05.pdf
[10] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," Technical Report, Stanford University,1998.
[11] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7), pages 422--426, July 1970.
[12] Google Dynamic Rendering: https://developers.google.com/search/docs/guides/dynamic-rendering
[13] T. Urvoy, T. Lavergne, and P. Filoche, "Tracking web spam with hidden style similarity," in Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web, 2006.
[14] H.-T. Lee, D. Leonard, X. Wang, and D. Loguinov, "IRLbot: Scaling to 6 billion pages and beyond," in Proceedings of the 17th International World Wide Web Conference, 200

فصل ۱۰
طراحی سیستم Notification

یک سیستم اعلان یا notification در سال‌های اخیر به یک ویژگی بسیار محبوب برای بسیاری از برنامه‌ها تبدیل شده است. یک اعلان/ نوتیفیکیشن اطلاعات مهمی مانند اخبار فوری، به‌روزرسانی‌های محصول، رویدادها، پیشنهادها و غیره را به کاربر هشدار می‌دهد. این مورد به بخشی ضروری از زندگی روزمره ما تبدیل شده است. در این فصل از شما خواسته می‌شود که یک سیستم اطلاع‌رسانی طراحی کنید.

نوتیفیکیشن چیزی فراتر از نوتیفیکیشن موبایل است. سه نوع فرمت معروف نوتیفیکیشن مواردی مثل: موبایل push notification، پیام کوتاه و ایمیل هستند. شکل ۱-۱۰ نمونه‌ای از هر یک از این اعلان‌ها را نشان می‌دهد.



شکل ۱-۱۰

مرحله ۱ - درک مسئله و شروع به طراحی

ساختن یک سیستم مقیاس‌پذیر که میلیون‌ها نوتیفیکیشن در روز ارسال می‌کند کار آسانی نیست. این نیاز به درک عمیقی از notification اعلان دارد. سؤال مصاحبه عمداً به‌صورت باز و مبهم طراحی شده است و این مسئولیت شماست که سؤالاتی را برای روشن‌شدن شرایط بپرسید.

کاندید: سیستم چه نوع اعلان‌هایی را پشتیبانی می‌کند؟

مصاحبه‌کننده: push notification، پیام کوتاه و ایمیل.

کاندید: آیا این یک سیستم بلادرنگ است؟

مصاحبه‌کننده: بگذارید بگوییم این یک سیستم بلادرنگ نُرم[1] است. ما می‌خواهیم یک کاربر در سریع‌ترین زمان نوتیفیکیشن‌ها را دریافت کند. بااین‌حال، اگر سیستم تحت بار کاری بالا باشد، تأخیر جزئی قابل‌قبول است.

کاندید: دستگاه‌های پشتیبانی شده کدام‌اند؟

1 soft real-time system

مصاحبه‌کننده: دستگاه‌های iOS، دستگاه‌های اندروید و لپ‌تاپ/دسکتاپ.

کاندید: چه چیزی باعث ایجاد اعلان‌ها می‌شود؟

مصاحبه‌کننده: اعلان‌ها را می‌توان توسط برنامه‌های کلاینت فعال کرد. آنها همچنین می‌توانند در سمت سرور برنامه‌ریزی شوند.

کاندید: آیا کاربران می‌توانند از دریافت نوتیفیکیشن‌ها انصراف دهند؟

مصاحبه‌کننده: بله، کاربرانی که انصراف می‌دهند دیگر اعلان دریافت نخواهند کرد.

کاندید: روزانه چند اعلان ارسال می‌شود؟

مصاحبه‌کننده: ۱۰ میلیون اعلان موبایل، ۱ میلیون پیام کوتاه و ۵ میلیون ایمیل.

مرحله ۲ – طراحی سطح پیشرفته

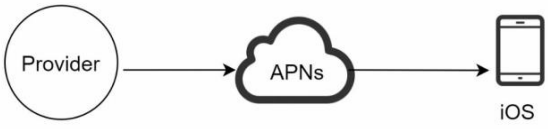
این بخش طراحی سطح پیشرفته‌ای را نشان می‌دهد که از انواع مختلف اعلان پشتیبانی می‌کند: نوتیفیکیشن در iOS، نوتیفیکیشن در اندروید، پیام کوتاه و ایمیل. ساختار آن به‌صورت زیر است:

- انواع مختلف نوتیفیکیشن
- مسیر جمع‌آوری اطلاعات تماس
- مسیر ارسال/دریافت نوتیفیکیشن

انواع مختلف نوتیفیکیشن

ما با بررسی نحوه عملکرد هر نوع اعلان در سطح بالا شروع می‌کنیم.

push notification در iOS



شکل ۲-۱۰

برای ارسال push notification در iOS به سه جزء اصلی نیاز داریم:

- **ارائه‌دهنده**[1]: یک ارائه‌دهنده درخواست‌های اطلاع‌رسانی یا نوتیفیکیشن را به APNS[2] می‌فرستد. برای ایجاد یک اعلان، ارائه‌دهنده داده‌های زیر را ارائه می‌دهد:
- **رمز دستگاه**[3]: این یک شناسه منحصربه‌فرد است که برای ارسال نوتیفیکیشن‌ها استفاده می‌شود.
- **Payload**: یک ساختار JSON است که حاوی payload اعلان است. به‌عنوان‌مثال:

```
{
  "aps" : {
        "alert" : {
           "title" : "Game Request",
           "body" : "Bob wants to play chess",
           "action-loc-key" : "PLAY"
        },
        "badge" : 5
     }
}
```

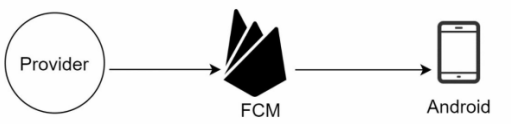
- APNS: این یک سرویس از با قابلیت دسترسی یا کنترل از راه دور[4] است که توسط اپل برای push notifications به دستگاه‌های iOS ارائه می‌شود.
- دستگاه iOS: سرویس‌گیرنده و کلاینت نهایی است که push notifications را دریافت می‌کند.

push notification در Android

اندروید مسیرهای نوتیفیکیشن مشابه مسیر iOS را اتخاذ می‌کند؛ ولی به‌جای استفاده از APN از FCM[5] برای ارسال نوتیفیکیشن‌ها به دستگاه‌های اندرویدی استفاده می‌شود.

---

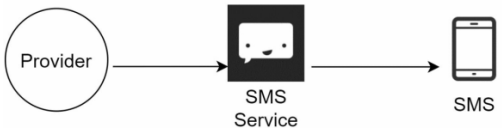
1 Provider
2 Apple Push Notification Service
3 Device token
4 remote
5 Firebase Cloud Messaging



شکل ۳-۱۰

SMS message – پیام کوتاه

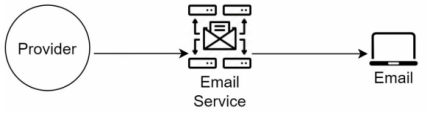
برای پیام‌های SMS، از سرویس‌های SMS شخص ثالث مانند Twilio، Nexmo و بسیاری دیگر معمولاً استفاده می‌شود. بیشتر آنها سرویس‌های تجاری هستند.



شکل ۴-۱۰

Email– نامه الکترونیکی

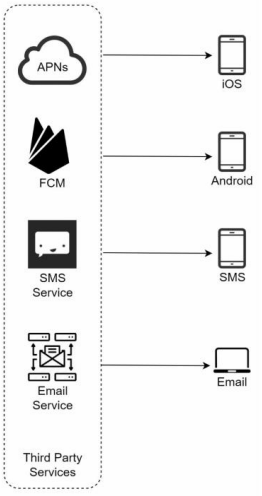
اگرچه شرکت‌ها می‌توانند سرورهای ایمیل خود را راه‌اندازی کنند، بسیاری از آنها خدمات ایمیل تجاری را انتخاب می‌کنند. Sendgrid و Mailchimp از جمله محبوب‌ترین سرویس‌های ایمیل هستند که نرخ تحویل و تجزیه‌وتحلیل داده بهتری را ارائه می‌دهند.



شکل ۵-۱۰

شکل ۶-۱۰ طراحی را پس از گنجاندن تمام سرویس‌های شخص ثالث[1] نشان می‌دهد.

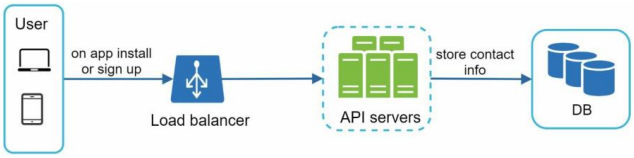
1 third-party services



شکل ۶-۱۰

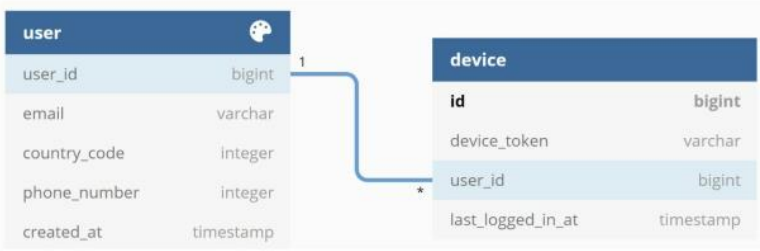
مسیر جمع‌آوری اطلاعات تماس

برای ارسال نوتیفیکیشن‌ها باید tokenها یا شناسه‌های منحصربه‌فرد دستگاه تلفن همراه، شماره‌تلفن یا آدرس ایمیل را جمع‌آوری کنیم. همان‌طور که در شکل ۷-۱۰ نشان‌داده‌شده است، زمانی که کاربر برنامه ما را نصب می‌کند یا برای اولین بار ثبت نام می‌کند، سرورهای API اطلاعات تماس کاربر را جمع آوری کرده و در پایگاه‌داده ذخیره می‌کنند.



شکل ۷-۱۰

شکل ۷-۱۰ جدول‌های پایگاه‌داده ساده شده را برای ذخیره اطلاعات تماس نشان می‌دهد. آدرس‌های ایمیل و شماره تلفن در جدول "user" ذخیره می‌شوند، درحالی‌که توکن‌های دستگاه در جدول "Device" ذخیره می‌شوند. یک کاربر می‌تواند چندین دستگاه داشته باشد، که نشان می‌دهد یک نوتیفیکیشن می‌تواند به همه دستگاه‌های کاربر ارسال شود.



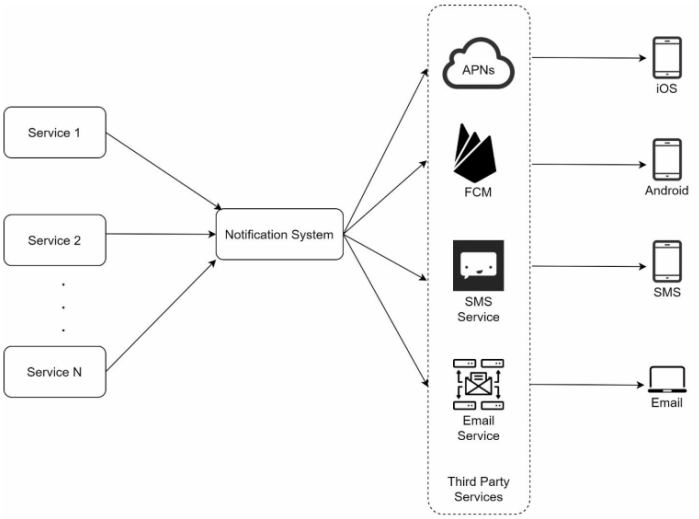
شکل ۸-۱۰

مسیر ارسال و دریافت اعلان‌ها

ابتدا طرح اولیه را ارائه خواهیم کرد. سپس چند بهینه‌سازی پیشنهاد می‌کنیم.

معماری مورد استفاده در طراحی سطح پیشرفته

شکل ۹-۱۰ طراحی را نشان می‌دهد و هر جزء سیستم در زیر توضیح داده شده است.



شکل ۹-۱۰

سرویس یک به چند: یک سرویس می‌تواند یک میکروسرویس[۱]، یک cron job[۲] یا یک سیستم توزیع شده باشد که رویدادهای ارسال نوتیفیکیشن را آغاز می‌کند. به‌عنوان‌مثال، یک سرویس صورت‌حساب مالی که ایمیل‌هایی را برای یادآوری پرداخت سررسید کلاینت‌ها ارسال می‌کند یا یک وب‌سایت فروشگاه اینترنتی به مشتری خود از طریق پیامک می‌گوید که بسته‌های آنها فردا در چه زمانی تحویل داده خواهند شد.

سرویس اعلانات[۳]:

در واقع notification system مرکز ارسال/دریافت نوتیفیکیشن‌ها است. برای شروع یک کار ساده، فقط از یک سرور notification استفاده می‌شود. این کار APIهایی را برای سرویس‌های یک به چند فراهم می‌کند و notification payload را برای سرویس‌های شخص ثالث ایجاد می‌کند.

سرویس‌های شخص ثالث[۴]: سرویس‌های شخص ثالث مسئول ارائه اعلان‌ها به کاربران هستند. در حین ادغام با سرویس‌های شخص ثالث، باید به مسئله توسعه‌پذیری[۵] توجه بیشتری داشته باشیم. توسعه‌پذیری خوب به معنای یک سیستم منعطف است که می‌تواند به‌راحتی یک سرویس شخص ثالث را متصل یا جدا کند. ملاحظات مهم دیگر این است که یک سرویس شخص ثالث ممکن است در برخی مناطق یا در آینده نزدیک در دسترس نباشد. به‌عنوان‌مثال، FCM در چین در دسترس نیست؛ بنابراین از سرویس‌های شخص ثالث جایگزین مانند Jpush، PushY و غیره در آنجا استفاده می‌شود.

:iOS, Android, SMS, Email

کاربران، نوتیفیکیشن‌ها را به روش‌های مختلفی در دستگاه‌های خود دریافت می‌کنند.

---

1 micro-service

۲ - cron job یک وظیفه‌ی زمان‌بندی شده در سیستم‌عامل‌های شبیه به یونیکس است که به کاربران اجازه می‌دهد تا دستورات یا اسکریپت‌های خاصی را در فواصل زمانی مشخصی به صورت خودکار اجرا کنند.

3 Notification system

4 Third-party services

5 extensibility

سه مشکل اساسی در این طراحی مشخص شده است:

- **یک نقطه شکست[1] (SPOF)**: یک سرور اعلان که به‌صورت منفرد کار می‌کند به معنای احتمال وقوع یک ضعف یا SPOF است.
- **پیچیدگی در مقیاس‌پذیری**: در یک سیستم نوتیفیکیشن همه‌چیز مربوط به push notifications را در یک سرور مدیریت می‌کند. مقیاس‌کردن پایگاه‌های داده، حافظه‌های کَش و اجزای مختلف پردازش اعلان‌ها به طور مستقل چالش‌برانگیز است.
- **گلوگاه عملکردی[2]**: پردازش و ارسال نوتیفیکیشن‌ها می‌تواند مصرف منابع شدیدی داشته باشد. به‌عنوان‌مثال، ساخت صفحات HTML و انتظار برای پاسخ از سرویس‌های شخص ثالث ممکن است زمان‌بر باشد. مدیریت همه‌چیز در یک سیستم می‌تواند منجر به اضافه‌بار سیستم، به‌ویژه در ساعات اوج مصرف شود.

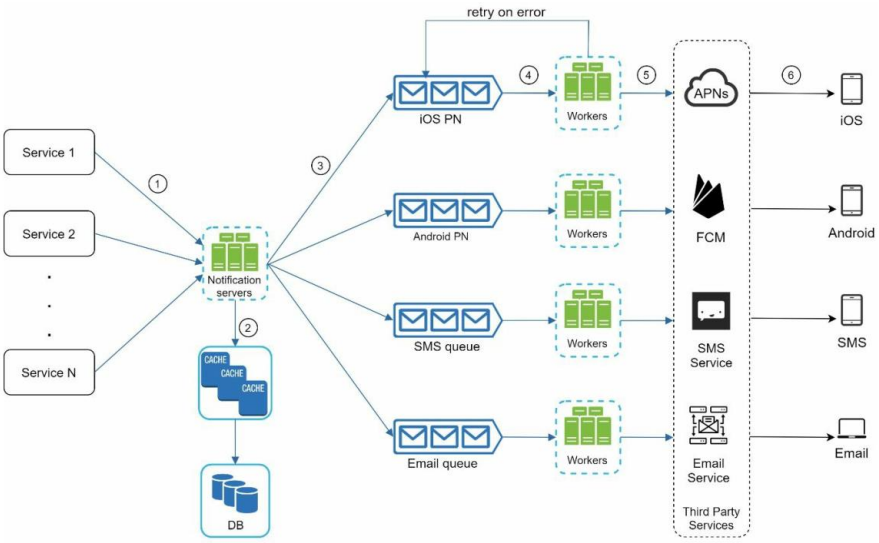
طراحی سطح پیشرفته (بهبودیافته)

پس از بررسی چالش‌ها در طراحی اولیه، طراحی را به شرح زیر بهبود می‌دهیم:

- پایگاه‌داده و حافظه کش را از سرور notification خارج کنید.
- سرورهای notification بیشتری اضافه کنید و مقیاس افقی خودکار[3] را تنظیم کنید.
- صف‌های پیام را برای جداسازی اجزای سیستم معرفی کنید.

شکل ۱۰-۱۰ بهبود طراحی سطح بالا را نشان می‌دهد.

1 Single point of failure
2 Performance bottleneck
3 automatic horizontal scaling



شکل ۱۰-۱۰

بهترین مسیر برای بررسی دیاگرام شکل ۱۰-۱۰ در قسمت بالا و از چپ به راست است:

سرویس‌های یک به چند: این‌ها سرویس‌های مختلفی را نشان می‌دهند که اعلان‌ها یا نوتیفیکیشن‌ها را از طریق APIهای ارائه شده توسط سرورهای نوتیفیکیشن ارسال می‌کنند.

سرورهای Notification: این سرورهای کاربردهای زیر را دارند:

- ارائه API برای سرویس‌های ارسال نوتیفیکیشن‌ها. این APIها فقط به‌صورت داخلی یا توسط کلاینت‌ها تأیید شده برای جلوگیری از هرزنامه قابل‌دسترسی هستند.
- اعتبارسنجی‌های اولیه را برای تأیید ایمیل‌ها، شماره تلفن‌ها و غیره انجام دهید.
- از پایگاه‌داده یا حافظه کَش برای واکشی داده‌های موردنیاز برای ارائه و پردازش نوتیفیکیشن‌ها استفاده کنید.
- داده‌های نوتیفیکیشن را برای پردازش موازی در صف‌های پیام قرار دهید.

در اینجا نمونه‌ای از API برای ارسال ایمیل آورده شده است:

POST https://api.example.com/v/sms/send

Request body

```
{
  "to": [
    {
      "user_id": 123456
    }
  ],
  "from": {
    "email": "from_address@example.com"
  },
  "subject": "Hello, World!",
  "content": [
    {
      "type": "text/plain",
      "value": "Hello, World!"
    }
  ]
}
```

حافظه کَش[1]: اطلاعات کاربر، اطلاعات دستگاه، الگوهای اعلان[2] در حافظه کَش هستند.

DB: داده‌های مربوط به کاربر، اعلان‌ها، تنظیمات و غیره را ذخیره می‌کند.

صف‌های پیام[3]: این صف‌ها وابستگی بین اجزا را حذف می‌کنند. هنگامی که حجم بالایی از نوتیفیکیشن‌ها ارسال می‌شود، صف‌های پیام به‌عنوان بافر عمل می‌کنند. پس هر نوع نوتیفیکیشن با یک صف پیام مجزا تخصیص داده می‌شود، بنابراین قطع‌شدن یک سرویس شخص ثالث بر انواع دیگر نوتیفیکیشن‌ها تأثیر نمی‌گذارد.

Workers: Workerها لیستی از سرورهایی هستند که رویدادهای نوتیفیکیشن[4] را از صف پیام‌ها بیرون می‌کشند و به سرویس‌های شخص ثالث مربوطه ارسال می‌کنند.

سرویس‌های شخص ثالث[5]: قبلاً در طرح اولیه توضیح داده شده است.

iOS، اندروید، پیامک، ایمیل: قبلاً در طرح اولیه توضیح داده شده است.

---

1 cache
2 notification templates
3 Message queues
4 notification events
5 Third-party services

در مرحله بعد، اجازه دهید بررسی کنیم که چگونه هر مؤلفه با هم دیگر کار می‌کند تا یک اعلان ارسال کند:

۱. یک سرویس APIهای ارائه شده توسط سرورهای notification را برای ارسال notificationها فراخوانی می‌کند.
۲. سرورهای notification معمولاً metadataهایی مانند اطلاعات کاربر، توکن دستگاه و notificationها را از کَش یا پایگاه‌داده واکشی می‌کنند.
۳. یک رویداد نوتیفیکیشن برای پردازش به صف مربوطه ارسال می‌شود. به‌عنوان‌مثال، یک رویداد نوتیفیکیشن iOS به صف PN iOS ارسال می‌شود.
۴. workerها رویدادهای نوتیفیکیشن را از صف‌های پیام بیرون می‌کشند.
۵. workerها نوتیفیکیشن‌ها را به سرویس‌های شخص ثالث ارسال می‌کنند.
۶. سرویس‌های شخص ثالث نوتیفیکیشن‌ها را به دستگاه‌های کاربر ارسال می‌کنند.

مرحله ۳ – بررسی عمیق‌تر

در طراحی سطح پیشرفته، انواع مختلف نوتیفیکیشن‌ها، مسیرهای جمع‌آوری اطلاعات تماس و مسیرهای ارسال/دریافت اعلان را مورد بحث قرار دادیم. ما موارد زیر را در اینجا بررسی خواهیم کرد:

- قابلیت اطمینان‌پذیری[1].
- مؤلفه و ملاحظات دیگر: الگوی نوتیفیکیشن[2]، تنظیمات اعلان‌ها، محدودیت نرخ[3]، مکانیسم امتحان مجدد[4]، امنیت در انتشار اعلان‌ها[5]، نظارت بر اعلان‌هایی که در صف جهت انتشار قرار دارند و ردیابی رویداد[6].
- به‌روزرسانی طراحی.

---

1 Reliability
2 notification template
3 rate limiting
4 retry
5 push notifications
6 event tracking

اطمینان‌پذیری – Reliability

هنگام طراحی یک سیستم اطلاع‌رسانی در محیط‌های توزیع شده، باید به چند سؤال مهم در مورد اطمینان‌پذیری یا قابل‌اعتماد بودن سیستم پاسخ دهیم.

چگونه باید ازدست‌رفتن اطلاعات و داده‌ها جلوگیری کنیم؟

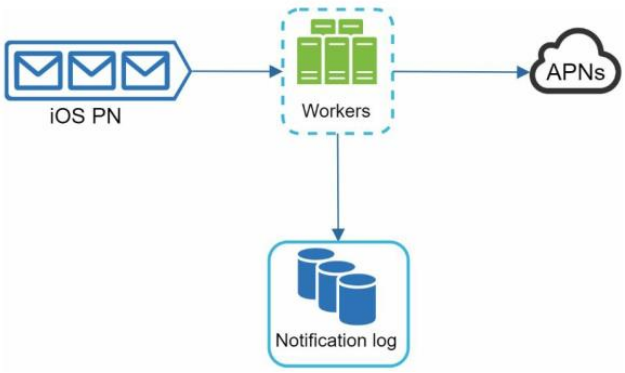
یکی از مهم‌ترین الزامات در یک سیستم‌های اعلان این است که داده‌های خود را از دست ندهد. نوتیفیکیشن‌ها معمولاً می‌توانند به تأخیر افتاده یا دوباره سفارش داده شوند، اما هرگز گم نمی‌شوند. برای برآورده‌کردن این نیاز برای سیستم نوتیفیکیشن‌ها، داده‌های نوتیفیکیشن‌ها را در یک پایگاه‌داده باقی می‌ماند و مکانیزم امتحان مجدد[1] را پیاده‌سازی می‌کند. همان‌طور که در شکل ۱۱-۱۰ نشان‌داده‌شده است، پایگاه‌داده مخصوص log سرویسِ نوتیفیکیشن برای پایداری داده‌ها یا در اصطلاح data persistence گنجانده شده است.

یادآوری:

Data persistence که به معنای تداوم/ پایداری داده‌ها پس از بسته‌شدن برنامه‌ای که آن‌ها را ایجاد کرده است. برای این اتفاق، داده‌ها باید در حافظه‌ی غیرفراری نوشته شوند. حافظه‌ی غیرفرار نوعی حافظه است که می‌تواند اطلاعات را به مدت طولانی نگه دارد، حتی اگر برنامه دیگری در حال اجرا نباشد. وقتی داده‌ها دائمی شوند، به این معناست که دقیقاً همان داده‌ها می‌توانند توسط یک برنامه دیگر بازیابی شوند و هیچ‌چیز ازمیان‌رفته نیست. این مسئله برای بهبود تجربه کاربری بسیار مهم است. به‌عنوان‌مثال، اگر یک برنامه از شما در ابتدای اجرا اطلاعاتی مانند نام و آدرس ایمیل خود را بخواهد، اگر باید برنامه را ببندید و بعداً دوباره باز کنید، نیازی به واردکردن دوباره اطلاعات نداشته باشید.

---

1 retry mechanism



شکل ۱۱-۱۰

آیا گیرندگان پیام هر کدام دقیقاً یک‌بار پیام را دریافت می‌کنند؟

جواب کوتاه است، نه! اگرچه نوتیفیکیشن در بیشتر مواقع دقیقاً یک‌بار ارسال می‌شود، ماهیت توزیع شده می‌تواند منجر به نوتیفیکیشن‌های تکراری شود. برای کاهش وقوع تکرار، یک مکانیسم dedupe[۱] را معرفی می‌کنیم و هر مورد خرابی را بادقت مدیریت می‌کنیم. در اینجا یک منطق ساده dedupe وجود دارد:

هنگامی که یک رویداد اعلان برای اولین‌بار می‌آید، با بررسی شناسه رویداد[۲] بررسی می‌کنیم که آیا قبلاً دیده شده است یا خیر. اگر قبلاً دیده شده باشد دور انداخته می‌شود. در غیر این صورت، نوتیفیکیشنی را ارسال خواهیم کرد. برای اینکه خوانندگان علاقه‌مند بفهمند که چرا نمی‌توانیم دقیقاً یک‌بار ارسال و تحویل دهی نوتیفیکیشن‌ها داشته باشیم، به مرجع [۵] مراجعه کنید.

---

۱ **Dedupe** به معنای حذف تکرارها است. این کلمه مختصری برای توصیف فرآیندی است که در آن داده‌های تکراری از یک مجموعه‌ی داده حذف می‌شوند. به عبارت دیگر، **dedupe** به شما کمک می‌کند تا اطلاعات تکراری را از یک جدول یا مجموعه‌ی داده‌ها حذف کنید. مثلاً، از طریق **dedupe** می‌توانید اسامی و آدرس‌های تکراری را از یک جدول حذف کنید.

2 event ID

مؤلفه و ملاحظات دیگر

ما درباره نحوه جمع‌آوری اطلاعات تماس کاربر[1]، ارسال و دریافت اعلان‌ها بحث کرده‌ایم. پیچیدگی‌های سیستم نوتیفیکیشن خیلی بیشتر از این است. در اینجا ما اجزای اضافی از جمله استفاده مجدد از الگو، تنظیمات نوتیفیکیشن، ردیابی رویداد، نظارت بر سیستم، محدودکردن نرخ و غیره را مورد بحث قرار می‌دهیم.

الگوهای نوتیفیکیشن‌ها

یک سیستم notification بزرگ روزانه میلیون‌ها اعلان ارسال می‌کند و بسیاری از این اعلان‌ها از فرمت مشابهی پیروی می‌کنند. الگوهای اعلان[2] برای جلوگیری از ایجاد هر اعلان از ابتدا معرفی شده‌اند. یک الگوی اعلان یک اعلان از پیش فرمت‌بندی شده برای ایجاد اعلان منحصربه‌فرد شما با سفارشی کردن پارامترها، استایل، پیوندهای ردیابی و غیره است. در اینجا یک الگوی نمونه از اعلان‌های آورده شده است.

```
BODY:
You dreamed of it. We dared it. [ITEM NAME] is back — only until [DATE].
CTA:
Order Now. Or, Save My [ITEM NAME]
```

مزایای استفاده از الگوهای نوتیفیکیشن شامل نگهداری از قالب ثابت و یکسان، کاهش خطاهای جزئی و صرفه‌جویی در زمان است.

تنظیمات نوتیفیکیشن‌ها

کاربران معمولاً روزانه نوتیفیکیشن‌های بسیار زیادی دریافت می‌کنند و به‌راحتی می‌توانند احساس آشفتگی کنند؛ بنابراین بسیاری از وب‌سایت‌ها و برنامه‌ها به کاربران کنترل دقیقی بر

1 collect user contact info
2 Notifications template

تنظیمات نوتیفیکیشن می‌دهند. این اطلاعات در جدول تنظیمات نوتیفیکیشن‌ها با فیلدهای زیر ذخیره می‌شود:

```
user_id bigInt
channel varchar # push notification, email or SMS
opt_in boolean # opt-in to receive notification
```

محدودیت نرخ (Rate limiting)

برای جلوگیری از آشفته شدن کاربران به‌خاطر نوتیفیکیشن‌های زیاد، می‌توانیم تعداد اعلان‌هایی را که کاربر می‌تواند دریافت کند محدود کنیم. این کار خیلی مهم است؛ زیرا کاربرها می‌توانند اعلان‌ها را در صورت ارسال زیاد آن‌ها، غیرفعال کنند.

مکانیزم امتحان مجدد (Retry mechanism)

هنگامی که یک سرویس شخص ثالث نتواند نوتیفیکیشن‌های خود را ارسال کند، نوتیفیکیشن‌ها برای امتحان مجدد به صف پیام اضافه می‌شود. اگر مشکل برطرف نشد، یک هشدار برای توسعه‌دهندگان ارسال می‌شود.

مبحث امنیت در ارسال اعلان‌ها

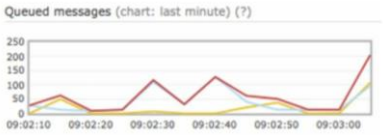
برای برنامه‌های تحت سیستم‌عامل‌های iOS یا Android از appKey و appSecret برای ایمن کردن push notification APIs استفاده می‌شوند [۶]. فقط کلاینت‌ها تأیید شده یا تأیید شده مجاز به ارسال اعلان با استفاده از APIهای موردنظر ما هستند. افراد علاقه‌مند به این موضوع باید به مطالب مرجع [۶] مراجعه کنند.

نظارت بر نوتیفیکیشن‌های صف شده

یک معیار کلیدی برای مبحث نظارت بر اعلان‌ها، بررسی تعداد کل اعلان‌های موجود در صف است. اگر تعداد اعلان‌های در صف[1] زیاد باشد، رویدادهای اعلان توسط workerها به‌اندازه کافی سریع پردازش نمی‌شوند. برای جلوگیری از تأخیر در تحویل نوتیفیکیشن‌ها، به

1 queued notifications

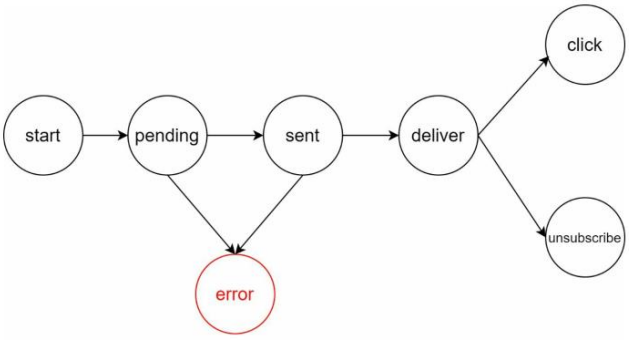
workerها بیشتری نیاز است. شکل ۱۲-۱۰ (ارجاع به منبع به [۷]) نمونه‌ای از پیام‌ها را در صف پردازش نشان می‌دهد.



شکل ۱۲-۱۰

ردیابی رویدادها

معیارها و متریک‌های نوتیفیکیشن، مانند نرخ باز[1]، نرخ کلیک و تعامل در درک رفتارهای کلاینت موارد مهمی هستند. سرویس Analytics ردیابی رویدادها را پیاده‌سازی می‌کند. یکپارچگی بین سیستم اطلاع‌رسانی و سرویس تجزیه‌وتحلیل معمولاً موردنیاز است. شکل ۱۳-۱۰ نمونه‌ای از رویدادهایی را نشان می‌دهد که ممکن است برای اهداف تحلیلی ردیابی شوند.



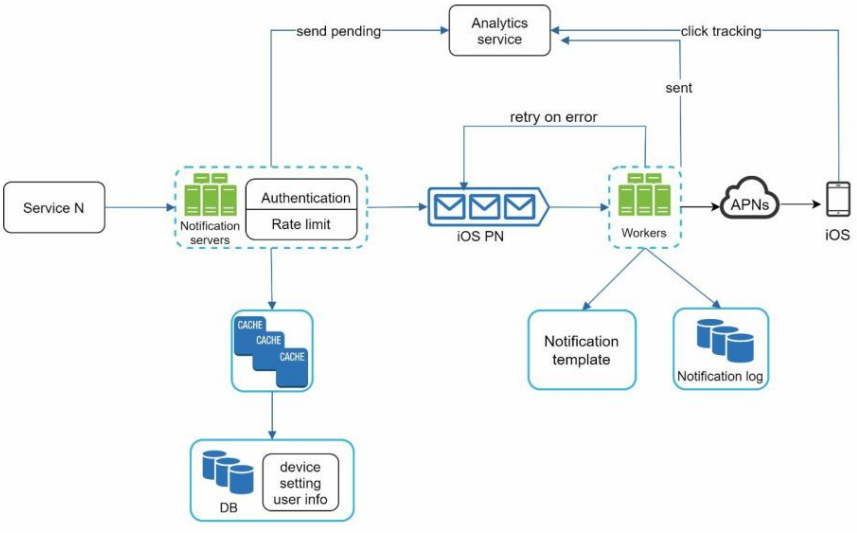
شکل ۱۳-۱۰

به‌روزرسانی طراحی

با کنار هم قراردادن همه اجزا، شکل ۱۴-۱۰ طراحی سیستم اطلاع رسانی[2] به‌روز شده را نشان می‌دهد.

---

1 open rate
2 notification system



شکل ۱۴-۱۰

در این طرح اجزای جدید بسیاری در مقایسه با طرح قبلی اضافه شده است.

- سرورهای نوتیفیکیشن[1] به دو ویژگی حیاتی دیگر مجهز هستند: احراز هویت[2] و محدودکردن نرخ[3].
- ما همچنین یک مکانیسم امتحان مجدد[4] برای رسیدگی به خطاهای نوتیفیکیشن اضافه می‌کنیم. اگر سیستم نتواند نوتیفیکیشن‌ها را ارسال کند، آنها دوباره در صف پیام‌رسانی قرار می‌گیرند و workerها برای تعداد دفعات از پیش تعریف‌شده دوباره تلاش می‌کنند.
- علاوه بر این، الگوهای نوتیفیکیشن یک فرایند ایجاد نوتیفیکیشن‌های پایدار و کارآمد را ارائه می‌دهند.
- در نهایت، سیستم‌های نظارتی و ردیابی برای بررسی سلامت سیستم و بهبودهای آینده اضافه شده است.

---

1 notification servers
2 authentication
3 rate-limiting
4 retry

مرحله ۴ - جمع‌بندی

نوتیفیکیشن‌ها غیرقابل‌صرف نظر هستند؛ زیرا اطلاعات مهمی را در جریان قرار می‌دهند. یک نوتیفیکیشن می‌تواند در مورد فیلم مورد علاقه شما در نتفلیکس یا یک ایمیل در مورد تخفیف‌ها باشد. حتی مواردی مانند محصولات جدید یا پیامی در مورد تأیید پرداخت خرید آنلاین شما یا پیام کوتاه درباره رمز عبور و رمز بانک و غیره را شامل باشد.

در این فصل، طراحی یک سیستم اطلاع‌رسانی مقیاس‌پذیر[1] را توضیح دادیم که از چندین فرمت نوتیفیکیشن پشتیبانی می‌کند: push notification، پیام کوتاه و ایمیل. همچنین از صف‌های پیام[2] برای جداسازی اجزای سیستم در قسمت طراحی سطح پیشرفته استفاده کردیم، ما جزئیات و بهینه‌سازی‌های بیشتری را بررسی کردیم.

قابلیت اطمینان: ما یک مکانیسم قوی برای تلاش مجدد[3]، جهت به‌حداقل‌رساندن میزان خطاها پیشنهاد کردیم.

امنیت: از هر دوی AppKey/appSecret استفاده می‌شود تا اطمینان حاصل شود که فقط کاربران تأیید شده می‌توانند اعلان ارسال کنند.

ردیابی و نظارت: این موارد در هر مرحله از مسیر اطلاع‌رسانی یا نوتیفیکیشن برای ثبت آمارهای مهم اجرا می‌شوند.

تنظیمات کاربر: کاربران ممکن است از دریافت اعلان‌ها انصراف دهند. سیستم ما در قدم اول، تنظیمات کاربر را قبل از ارسال نوتیفیکیشن‌ها بررسی می‌کند.

محدودکردن نرخ: کاربران از محدود بودن متوالی در تعداد نوتیفیکیشن‌هایی که دریافت می‌کنند قدردانی خواهند کرد.

---

1 scalable notification system
2 message queues
3 retry

منابع و مراجع فصل ۱۰

[1] Twilio SMS: https://www.twilio.com/sms
[2] Nexmo SMS: https://www.nexmo.com/products/sms
[3] Sendgrid: https://sendgrid.com/
[4] Mailchimp: https://mailchimp.com/
[5] You Cannot Have Exactly-Once Delivery: https://bravenewgeek.com/you-cannot-have-exactly-once-delivery/
[6] Security in Push Notifications: https://cloud.ibm.com/docs/services/mobilepush?topic=mobile-pushnotification-security-in-push-notifications
[7] RadditMQ: https://bit.ly/2sotIa6

فصل ۱۱

طراحی سیستم شبکه اجتماعی

در این فصل از شما خواسته شده است که یک سیستم خوراک خبری یا فید خبررسانی[1] که اساس یک اپلیکیشن شبکه اجتماعی است را طراحی کنید. فید خبری چیست؟ باتوجه‌به صفحه راهنمای فیس‌بوک: «فید خبری؛ لیستی است که دائماً از داستان‌ها[2] که در وسط صفحه اصلی مرورگر یا اپلیکیشن مورد استفاده شما به‌روزرسانی می‌شود. News Feed شامل به‌روزرسانی‌های وضعیت، عکس‌ها، ویدئوها، پیوندها، فعالیت برنامه‌ها و لایک‌های افراد، صفحات و گروه‌هایی است که در فیس‌بوک دنبال می‌کنید».

این یک سؤال مصاحبه محبوب است. سؤالات مشابهی که معمولاً پرسیده می‌شود عبارت‌اند از: طراحی فید خبری فیس‌بوک، فید اینستاگرام، جدول‌زمانی[3] توییتر و غیره.

---

1 feed system
2 stories
3 timeline

شکل ۱-۱۱

منبع:https://bit.ly/2Vv9JCm

مرحله ۱ – درک مسئله و شروع به طراحی

اولین مجموعه سؤال‌ها در جهت شفاف‌سازی این است که بفهمید مصاحبه‌کننده وقتی از شما می‌خواهد یک سیستم فید خبری طراحی کنید چه چیزی در ذهن دارد. حداقل باید بفهمید که برنامه موردنظر از چه ویژگی‌هایی پشتیبانی کنید. در اینجا نمونه‌ای از تعامل کاندید-مصاحبه‌کننده آورده شده است:

کاندید: آیا این یک برنامه تلفن همراه است؟ یا یک برنامه وب؟ یا هر دو؟

مصاحبه‌کننده: هر دو.

کاندید: ویژگی‌های مهم این برنامه چیست؟

مصاحبه‌کننده: کاربر می‌تواند یک پست منتشر کند و پست‌های دوستان خود را در صفحه فید خبری ببیند.

کاندید: آیا فید اخبار بر اساس ترتیب زمانی معکوس یا هر ترتیب خاصی مانند امتیاز موضوع مرتب شده است؟ به‌عنوان‌مثال، پست‌های دوستان نزدیک شما امتیازات بالاتری دارد.

مصاحبه‌کننده: برای ساده نگه‌داشتن همه‌ی اجزا، اجازه دهید فرض کنیم فید بر اساس ترتیب زمانی معکوس مرتب شده است.

کاندید: یک کاربر چند دوست می‌تواند داشته باشد؟

مصاحبه‌کننده: ۵۰۰۰

کاندید: حجم ترافیک مصرفی چقدر است؟

مصاحبه‌کننده: ۱۰ میلیون کاربر فعال در روز - DAU[1]

کاندید: آیا فید می‌تواند حاوی تصاویر، ویدئوها باشد؟ آیا فقط شامل متن است؟

مصاحبه‌کننده: برنامه می‌تواند حاوی فایل‌های چندرسانه‌ای از جمله تصاویر و ویدئوها باشد.

اکنون که شما الزامات را جمع‌آوری کرده‌اید، ما بر روی طراحی سیستم تمرکز می‌کنیم.

مرحله ۲ - طراحی سطح پیشرفته

این طرح به دو مسیر مختلف تقسیم می‌شود: انتشار فید و ساخت فید خبری.

انتشار فید[2]: زمانی که کاربر پستی را منتشر می‌کند، داده‌های مربوطه در حافظه کَش/موقت و پایگاه‌داده نوشته می‌شود. در نتیجه یک پست در فید خبری دوستانش نشان داده می‌شود.

ساخت News Feed: برای سادگی، فرض کنیم فید اخبار با جمع‌آوری پست‌های دوستان به ترتیب زمانی معکوس[3] ساخته شده است.

Newsfeed APIs

APIهای فید خبری راه‌های اصلی برای ارتباط کلاینت‌ها با سرورها هستند. این APIها مبتنی بر HTTP هستند که به کلاینت‌ها اجازه اقدامات موردنظر را می‌دهند که این اقدامات شامل پُست‌کردن status، بازیابی فید اخبار، افزودن دوستان و غیره می‌شود. ما دو API مهم را مورد بحث قرار می‌دهیم: API انتشار فید و API بازیابی فید خبری.

API انتشار فید

برای انتشار یک پست، یک درخواست HTTP POST به سرور ارسال می‌شود. به عنوان مثال API ای در زیر نشان‌داده‌شده است:

---

1 daily active users
2 feed publishing
3 reverse chronological

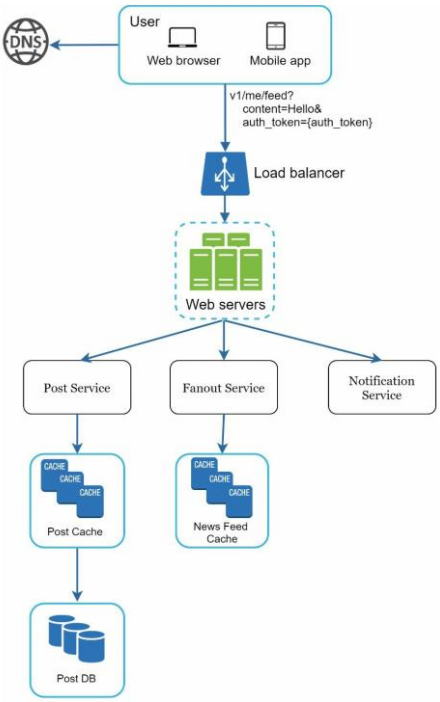
POST /v1/me/feed

پارامترها:

- محتوا[1]: در واقع محتوا شامل متنِ پست است.
- auth_token: برای احراز هویت درخواست‌های API استفاده می‌شود.

API بازیابی فید خبری

شکل ۲-۱۱ طراحی سطح پیشرفته مسیر انتشار فید را نشان می‌دهد.



شکل ۲-۱۱

کاربر: کاربر می‌تواند فیدهای خبری را در مرورگر یا برنامه تلفن همراه مشاهده کند. یک کاربر از طریق API، پستی با محتوای «Hello» می‌گذارد:

/v1/me/feed?content=Hello&auth_token={auth_token}

1 content

متعادل‌کننده بار[1]: ترافیک را به وب سرورها توزیع می‌کند.

وب سرورها: وب سرورها ترافیک را به سرویس‌های داخلی مختلف هدایت می‌کنند.

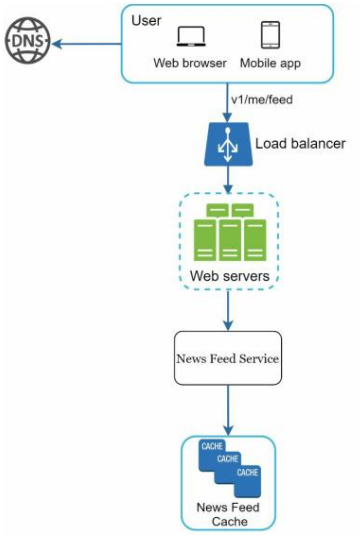
سرویس پُست: مربوط به پایداری داده‌های سرویس پُست[2] در پایگاه‌داده و حافظه کَش است.

سرویس Fanout: محتوای جدید را به فید اخبار دوستان ارسال می‌کند. اطلاعات فید خبری برای بازیابی سریع در حافظه کَش ذخیره می‌شود.

سرویس اعلان[3]: به دوستان اطلاع می‌دهد که محتوای جدید در دسترس است و نوتیفیکیشن را منتشر می‌کند.

ساخت فید خبری

در این بخش به نحوه ساخت فید خبری در پشت‌صحنه می‌پردازیم. شکل ۳-۱۱ طراحی سطح پیشرفته را نشان می‌دهد:



شکل ۳-۱۱

1 Load balancer
2 persist
3 Notification service

- **کاربر**: کاربر درخواستی برای بازیابی فید خبری خود ارسال می‌کند. درخواست به این شکل است:

/v1/me/feed

- **متعادل‌کننده بار**: متعادل‌کننده بار ترافیک را به سرورهای وب هدایت می‌کند.
- **وب سرورها**: سرورهای وب درخواست‌ها را به سرویس فید خبری هدایت[1] می‌کنند.
- **سرویس فید خبری**: این سرویس، فید اخبار را از حافظه کَش واکشی می‌کند.
- **حافظه کَش مربوط به فید خبری**: شناسه‌های فید خبری را که برای پردازش و ارائه فید خبری لازم است را ذخیره می‌کند.

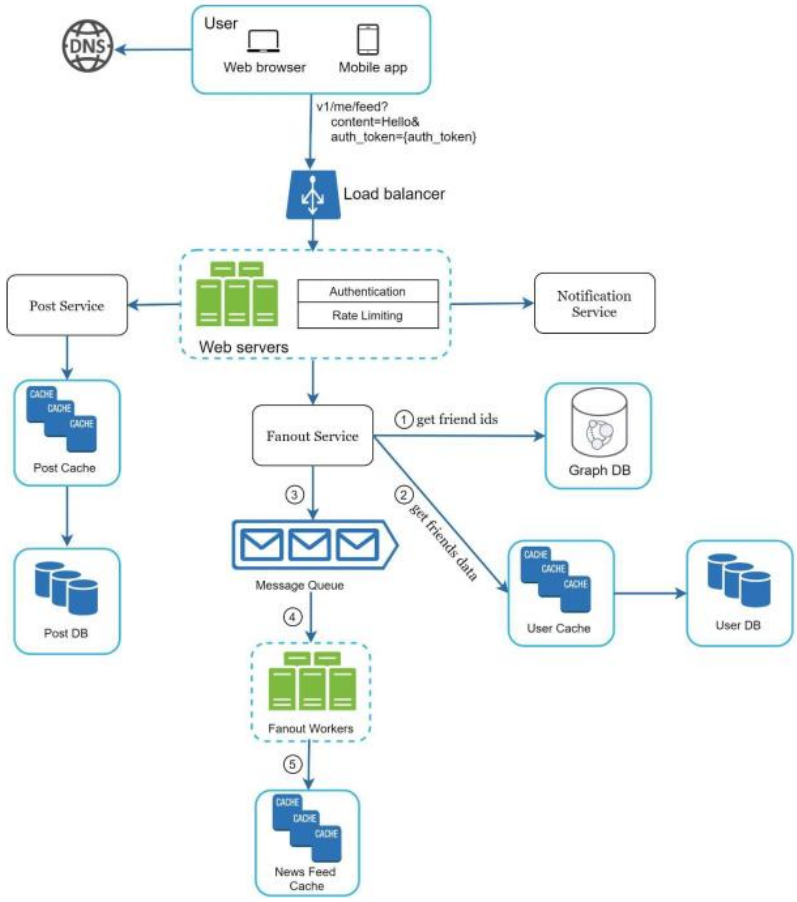
مرحله ۳ - بررسی عمیق‌تر

طراحی سطح پیشرفته به طور خلاصه دو جریان را شامل می‌شود: انتشار فید و ساخت فید خبری. در اینجا، ما آن موضوعات را عمیق‌تر مورد بحث قرار می‌دهیم.

بررسی عمیق‌تر فید خبری

شکل ۴-۱۱ طرح دقیق برای انتشار فید را نشان می‌دهد. ما بیشتر مولفه‌ها را در طراحی سطح پیشرفته مورد بحث قرار داده ایم و روی دو جزء تمرکز خواهیم کرد: وب سرورها و سرویس fanout.

1 redirect



شکل ۴-۱۱

وب ســـــرورهـــا

وب سرورها علاوه بر برقراری ارتباط با کلاینت‌ها، احراز هویت[1] و محدودیت نرخ[2] را اعمال می‌کنند. فقط کاربرانی که با auth_token معتبر وارد شده‌اند اجازه دارند پست بگذارند. این سیستم تعداد پست‌هایی را که کاربر می‌تواند در یک بازه زمانی خاص ارسال کند را محدود می‌کند که این مورد برای جلوگیری از هرزنامه و محتوای توهین‌آمیز حیاتی است.

1 authentication
2 rate-limiting

سرویس Fanout

Fanout فرایند ارسال یک پست به همه دوستان است. دو نوع مدل fanout عبارت‌اند از: fanout در نوشتن (که مدل push نیز نامیده می‌شود) و fanout در خواندن (که مدل pull نامیده می‌شود). هر دو مدل مزایا و معایب دارند. ما گردش کار آنها را توضیح می‌دهیم و بهترین رویکرد را برای پشتیبانی از سیستم خود بررسی می‌کنیم.

Fanout در حالت نوشتن. با این رویکرد، فید خبری در طول زمان نوشتن از قبل محاسبه می‌شود. یک پست جدید بلافاصله به دوستان تحویل داده می‌شود و بلافاصله بعد از انتشار پست در حافظه کَش ذخیره می‌شود.

مزایا:

- فید اخبار در زمان واقعی تولید می‌شود و می‌تواند بلافاصله برای دوستان ارسال شود.
- واکشی فید اخبار سریع است؛ زیرا فید اخبار در طول زمان نوشتن از قبل محاسبه شده است.

معایب:

- اگر کاربر دوستان زیادی داشته باشد، واکشی لیست دوستان و ایجاد فید خبری برای همه آنها کند و زمان بر است که به آن مشکل hotkey می‌گویند.
- برای کاربران غیرفعال یا کسانی که به‌ندرت وارد سیستم می‌شوند، از قبل محاسبه‌شدن فید خبری، منابع محاسباتی را تلف می‌کند.

Fanout در حالت خواندن. فید اخبار در زمان خواندن آن تولید می‌شود. این یک مدل بر اساس تقاضا[1] است. زمانی که کاربر صفحه اصلی خود را بارگیری می‌کند، پست‌های اخیر از سرور کشیده[2] می‌شوند.

مزایا:

1 on-demand model
2 pulled

- برای کاربران غیرفعال یا کسانی که به‌ندرت وارد سیستم می‌شوند، fanout در خواندن بهتر عمل می‌کند؛ زیرا منابع محاسباتی را برای آنها هدر نمی‌دهد.
- داده‌ها به دوستان ارسال نمی‌شوند، بنابراین هیچ مشکل hotkey ای وجود ندارد.

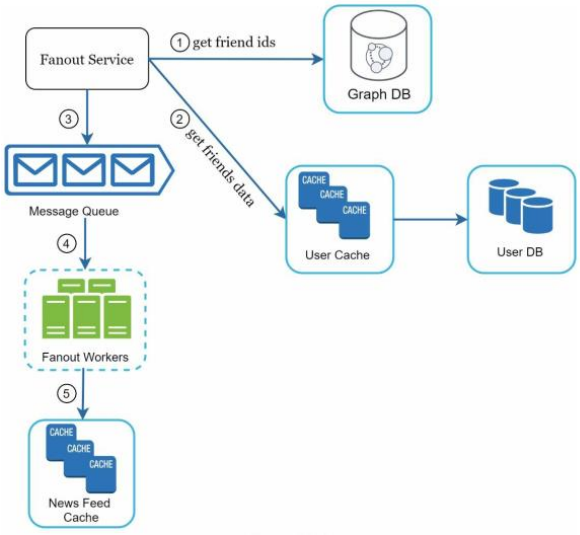
معایب:

- واکشی فید اخبار آهسته است؛ زیرا فید اخبار از قبل محاسبه نشده است.

ما یک رویکرد ترکیبی را برای به‌دست‌آوردن مزایای هر دو رویکرد و جلوگیری از مشکلات در آنها اتخاذ می‌کنیم. ازآنجایی‌که واکشی سریع فید اخبار بسیار مهم است، ما از یک push model برای اکثر کاربران استفاده می‌کنیم. برای افراد مشهور کاربرانی که دوستان/ دنبال‌کنندگان زیادی دارند، ما به دنبال‌کنندگان[1] اجازه می‌دهیم محتوای خبری را بر اساس تقاضای خود تهیه کنند تا از بار اضافی[2] روی سیستم جلوگیری کنند. هش‌کردن مداوم[3] یک تکنیک مفید برای کاهش مشکل کلید hotkey است؛ زیرا به توزیع یکنواخت درخواست‌ها/داده‌ها[4] کمک می‌کند.

اجازه دهید همان‌طور که در شکل ۵-۱۱ نشان‌داده‌شده است نگاهی دقیق به سرویس fanout بیندازیم.

1 followers
2 overload
3 Consistent hashing
4 requests/data



شکل ۵-۱۱

سرویس fanout به شرح زیر عمل می‌کند:

۱. شناسه دوستان را از پایگاه‌داده گرافی[1] واکشی می‌کند. پایگاه‌داده‌های گرافی برای مدیریت روابط دوستان[2] و پیشنهاد افراد دیگر جهت دوست‌یابی[3] مناسب هستند. خوانندگان علاقه‌مندی که مایل به کسب اطلاعات بیشتر در مورد این مفهوم هستند باید به مطالب مرجع [۲] مراجعه کنند.

۲. اطلاعات دوستان را از کَش کاربر[4] دریافت کنید. سپس سیستم، دوستان را بر اساس تنظیمات کاربری فیلتر می‌کند. به‌عنوان‌مثال، اگر فردی را بی‌صدا[5] کنید، دیگر پست‌های او در فید خبری شما نشان داده نمی‌شود، حتی اگر هنوز دوست هستید. یکی دیگر از دلایلی که ممکن است پست‌ها نشان داده نشوند این است که کاربر می‌تواند به طور انتخابی اطلاعات را با دوستان خاصی به اشتراک بگذارد یا آن را از افراد دیگر پنهان کند.

---

1 graph database
2 friend relationship
3 friend recommendations
4 user cache
5 mute

۳. لیست دوستان و شناسه پست جدید را به صف پیام ارسال کنید.

۴. Workerها، Fanout داده‌ها را از صف پیام دریافت می‌کنند و داده‌های فید اخبار را در حافظه کَش مربوط به فید اخبار ذخیره می‌کنند. می‌توانید کَش فید اخبار را به‌عنوان یک جدول نگاشت شده[1] <post_id, user_id> در نظر بگیرید. هر زمان که یک پست جدید ایجاد شود، همان‌طور که در شکل ۶-۱۱ نشان‌داده‌شده است، به جدول فید اخبار اضافه می‌شود. اگر کل اطلاعات کاربر به همراه جزئیات پست‌ها را در حافظه کَش ذخیره کنیم، حافظه مصرفی می‌تواند بسیار زیاد شود؛ بنابراین فقط شناسه‌ها ذخیره می‌شوند. برای کوچک نگه‌داشتن اندازه حافظه، یک محدودیت قابل تنظیم تعیین می‌کنیم. احتمال اینکه کاربر در میان هزاران پست در فید خبری گَردش کند، بسیار اندک است. اکثر کاربران فقط به جدیدترین مطالب علاقه‌مند هستند، بنابراین میزان ازدست‌دادن حافظه کَش پایین است.

۵. <post_id, user_id > را در کَش فید خبری ذخیره کنید. شکل ۶-۱۱ نمونه‌ای از نحوه ذخیره فید اخبار در حافظه کَش را نشان می‌دهد.

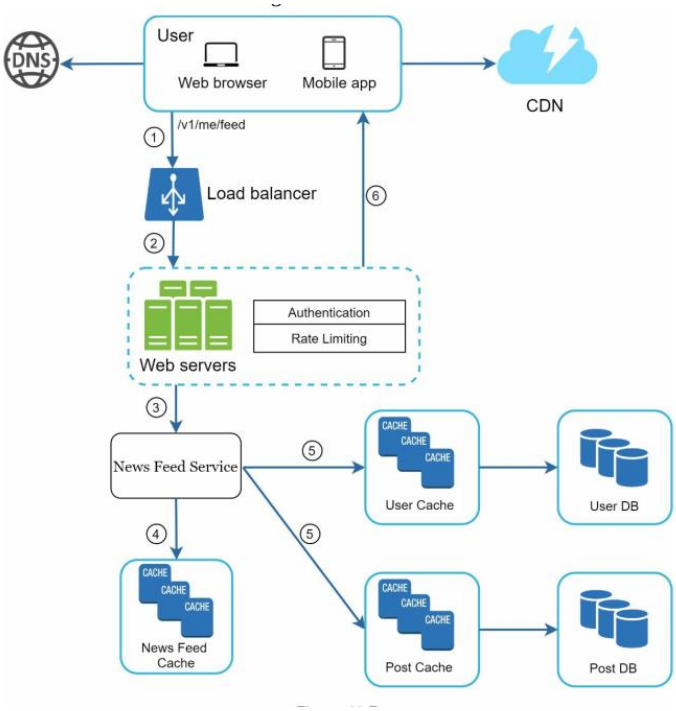
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id

تصویر ۶-۱۱

1 mapping table

بررسی عمیق در بازیابی فید خبری

شکل ۷-۱۱ طراحی دقیق برای بازیابی فید اخبار را نشان می‌دهد.



شکل ۷-۱۱

همان‌طور که در شکل ۷-۱۱ نشان‌داده‌شده است، محتوای چند رسانه‌ای (تصاویر، فیلم‌ها و غیره) برای بازیابی سریع در CDN ذخیره می‌شود. اجازه دهید نگاه کنیم که چگونه یک مشتری فید اخبار را بازیابی می‌کند.

۱. یک کاربر درخواستی برای بازیابی فید خبری خود ارسال می‌کند. درخواست به این شکل است: v1/me/feed/
۲. متعادل‌کننده بار درخواست‌ها را دوباره به سرورهای وب توزیع می‌کند.
۳. وب سرورها سرویس فید اخبار را برای واکشی فیدهای خبری فراخوانی می‌کنند.

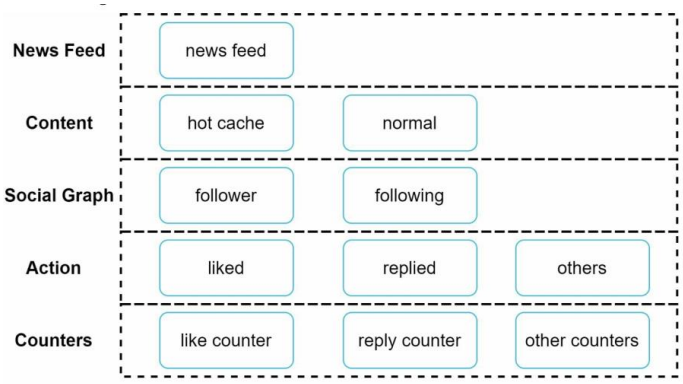
۴. سرویس فید اخبار یک لیست شناسه پست[1] را از حافظه کَش فید اخبار دریافت می‌کند.

۵. فید خبری یک کاربر چیزی بیش از فهرستی از شناسه‌های فید است. این شامل نام کاربری، تصویر پروفایل، محتوای پست، تصویر پست و غیره است؛ بنابراین سرویس فید اخبار، تمامی جزئیات مربوط به کاربر و پست‌هایش (حافظه کَش کاربر و حافظه کَش پُست) را از حافظه کَش واکشی می‌کند تا فید خبری کاملاً مناسب و بهینه‌ای را ایجاد کند.

۶. فید خبری کاملاً بهینه شده در قالب JSON برای ارائه به کلاینت بازگردانده می‌شود.

معماری کَش

کَش برای یک سیستم فید خبری بسیار مهم است. طبق شکل ۸-۱۱ لایه کَش را به ۵ لایه تقسیم می‌کنیم.



شکل ۸-۱۱

- **News Feed**: شناسه فیدهای خبری را ذخیره می‌کند.
- **Content**: هر داده پست شده‌ای را ذخیره می‌کند. محتوای محبوب در hot cache ذخیره می‌شود.

1 post IDs

- **Social Graph**[۱]: داده‌های ارتباطی مربوط به دوستان کاربرها[۲] را ذخیره می‌کند.
- **Action**: اطلاعاتی را در مورد اینکه آیا کاربر یک پست را پسندیده یا به یک پست پاسخ داده است یا اقدامات دیگری روی یک پست انجام داده است را ذخیره می‌کند.
- **Counters**[۳]: شمارنده‌هایی را برای لایک، پاسخ، دنبال‌کنندگان[۴]، دنبال‌شوندگان[۵] و غیره ذخیره می‌کند.

مرحله چهارم - جمع‌بندی

در این فصل یک سیستم فید خبری طراحی کردیم. طراحی ما شامل دو جریان است: انتشار فید و بازیابی فید خبری.

مانند هر سؤال مصاحبه طراحی سیستم، هیچ راه کاملی برای طراحی سیستم وجود ندارد. هر شرکتی محدودیت‌های منحصربه‌فرد خود را دارد و شما باید سیستمی را متناسب با این محدودیت‌ها طراحی کنید. درک صحیح انتخاب‌های طراحی و فناوری مورد استفاده شما مهم است. اگر چند دقیقه باقی‌مانده است، می‌توانید در مورد مسائل مقیاس‌پذیری[۶] صحبت کنید. برای جلوگیری از بحث تکراری، فقط نکات صحبت در سطح پیشرفته در زیر ذکر شده است.

مقیاس‌دهی پایگاه‌داده:

- مقیاس‌بندی عمودی در مقابل مقیاس‌بندی افقی[۷].
- SQL در برابر NoSQL.
- تکثیر[۸] Master-Slave.
- کپی/replica ها را بخوانید.

---

۱ گراف اجتماعی

2 user relationship

۳ شمارنده‌ها

4 follower
5 following
6 scalability
7 Vertical scaling vs Horizontal scaling
8 replication

- مدل‌های یکپارچگی[۱].
- پارتیشن‌بندی/شاردینگ پایگاه‌داده.

سایر نکات مورد صحبت:

- لایه وب را stateless نگه دارید.
- تا جایی که می‌توانید داده‌ها را در حافظه کَش ذخیره کنید.
- پشتیبانی از مراکز داده متعدد.
- کاهش وابستگی سرویس‌های مختلف به هم با استفاده از صف پیام[۲].
- معیارها و متریک کلیدی را نظارت[۳] کنید. به‌عنوان‌مثال، QPS در ساعات اوج مصرف و تأخیر[۴] به‌خصوص در زمانی که کاربران فید اخبار خود را به‌روزرسانی می‌کنند از جهت فاکتورهای نظارتی، موضوعات جالب هستند.

---

۱ در علوم کامپیوتر، یک مدل یکپارچگی (Consistency models) قراردادی را بین برنامه‌نویس و یک سیستم مشخص می‌کند که در آن سیستم تضمین می‌کند که اگر برنامه‌نویس از قوانین مربوط به عملیات روی حافظه پیروی کند، حافظه یکپارچه خواهد بود و نتایج آن خواندن، نوشتن یا به‌روزرسانی حافظه خواهد بود. مدل‌های یکپارچگی در سیستم‌های توزیع‌شده مانند سیستم‌های حافظه مشترک توزیع‌شده یا ذخیره‌های داده توزیع‌شده (مانند سیستم فایل، پایگاه‌های داده یا ذخیره‌سازی وب) استفاده می‌شوند.

2 message queues
3 Monitor
4 latency

منابع و مراجع فصل ۱۱

[1] How News Feed Works:
https://www.facebook.com/help/327131014036297/
[2] Friend of Friend recommendations Neo4j and SQL Sever:
http://geekswithblogs.net/brendonpage/archive/2015/10/26/friend-of-friendrecommendations-with-neo4j.aspx

فصل ۱۲

طراحی سیستم برنامه چت

در این فصل به بررسی طراحی یک سیستم چت می‌پردازیم. تقریباً همه از یک برنامه چت به‌خصوص استفاده می‌کنند. شکل ۱-۱۲ برخی از محبوب ترین برنامه‌ها را در بازار نشان می‌دهد.

شکل ۱-۱۲

یک برنامه چت عملکردهای مختلفی را برای افراد مختلف انجام می‌دهد. تعیین دقیق الزامات برای هر کاربرد بسیار مهم است. به‌عنوان‌مثال، زمانی که مصاحبه‌گر چت انفرادی را در ذهن دارد، نمی‌خواهد سیستمی طراحی کنید که بر چت گروهی تمرکز کند. پس بررسی الزامات ویژگی‌های موردنظر بسیار مهم است.

مرحله ۱ - درک مسئله و شروع به طراحی

توافق در مورد نوع برنامه چت برای طراحی بسیار مهم است. در بازار، برنامه‌های چت یک‌به‌یک مانند فیس‌بوک مسنجر، وی چت و واتس‌اپ، برنامه‌های چت اداری که بر چت گروهی مانند

Slack تمرکز می‌کنند یا برنامه‌های چت جهت بازی‌های رایانه‌ای مانند Discord وجود دارند که بر تعامل گروهی بزرگ و تأخیر زمانی کم جهت ارسال پیام‌های صوتی تمرکز دارند. اولین مجموعه سؤالات باید شفاف‌سازی جهت نحوه طراحی را مشخص کند که مصاحبه‌کننده دقیقاً زمانی که از شما می‌خواهد یک سیستم چت را طراحی کنید چه چیزی در ذهن دارد. حداقل، بفهمید که آیا باید روی یک برنامه چت یک‌به‌یک یا گروهی تمرکز کنید. برخی از سؤالاتی که ممکن است بپرسید به شرح زیر است:

کاندید: چه نوع برنامه چت طراحی کنیم؟ ۱ به ۱ یا بر اساس گروه؟

مصاحبه‌کننده: برنامه باید از چت ۱ به ۱ و گروهی پشتیبانی کند.

کاندید: آیا این یک برنامه تلفن همراه است؟ یا یک برنامه وب؟ یا هر دو؟

مصاحبه‌کننده: هر دو.

کاندید: مقیاس این برنامه چقدر است؟ یک برنامه استارتاپ یا برنامه‌ای با مقیاس بزرگ؟

مصاحبه‌کننده: باید از ۵۰ میلیون کاربر فعال روزانه[1] (DAU) پشتیبانی کند.

کاندید: برای چت گروهی، محدودیت اعضای گروه چقدر است؟

مصاحبه‌کننده: حداکثر ۱۰۰ نفر

کاندید: چه ویژگی‌هایی برای برنامه چت مهم است؟ آیا از پیوست[2] فایل پشتیبانی می‌کند؟

مصاحبه‌کننده: چت یک‌به‌یک، چت گروهی، نشانگر وضعیت آنلاین بودن کاربر. این سیستم فقط از پیام‌های متنی پشتیبانی می‌کند.

کاندید: آیا محدودیت اندازه پیام وجود دارد؟

مصاحبه‌کننده: بله، طول متن باید کمتر از ۱۰۰۰۰۰ کاراکتر باشد.

کاندید: آیا رمزگذاری سراسری لازم است؟

مصاحبه‌کننده: در حال حاضر لازم نیست، اما اگر زمان اجازه دهد، در مورد آن بحث خواهیم کرد.

---

1 (DAU) daily active users
2 attachment

کاندید: چه مدت باید تاریخچه چت را ذخیره کنیم؟

مصاحبه‌کننده: برای همیشه.

در این فصل، ما بر روی طراحی یک برنامه چت مانند پیام‌رسان فیس‌بوک با تأکید بر ویژگی‌های زیر تمرکز می‌کنیم:

- یک چت یک‌به‌یک با تأخیر تحویل کم
- گپ گروهی کوچک (حداکثر ۱۰۰ نفر)
- بررسی وضعیت آنلاین کاربر
- پشتیبانی از چند دستگاه. یک حساب کاربری را می‌توان به طور هم‌زمان به چندین حساب وارد کرد.
- Push notifications

توافق بر سر مقیاس طراحی نیز مهم است. ما سیستمی طراحی خواهیم کرد که ۵۰ میلیون کاربر فعال روزانه (DAU) را پشتیبانی کند.

مرحله ۲ – طراحی سطح پیشرفته

برای توسعه یک طراحی باکیفیت بالا، باید دانش اولیه‌ای از نحوه ارتباط کلاینت‌ها و سرورها داشته باشیم. در یک سیستم چت، کلاینت‌ها می‌توانند اپلیکیشن‌های موبایل یا اپلیکیشن‌های وب باشند. کلاینت‌ها مستقیماً با یکدیگر ارتباط برقرار نمی‌کنند. در عوض، هر کلاینت به یک سرویس چت متصل می‌شود که از تمام ویژگی‌های ذکر شده در بالا پشتیبانی می‌کند. بگذارید روی عملیات اساسی تمرکز کنیم. سرویس چت باید از عملکردهای زیر پشتیبانی کند:

• دریافت پیام از کاربرهای دیگر.

• گیرندگان مناسب برای هر پیام را بیابید و پیام را به گیرندگان منتقل کنید.

• اگر گیرنده‌ای آنلاین نیست، پیام‌های مربوط به آن گیرنده را تا زمانی که آنلاین نیست روی سرور نگه دارید.

شکل ۲-۱۲ روابط بین کاربرها (فرستنده و گیرنده) و سرویس چت را نشان می‌دهد.

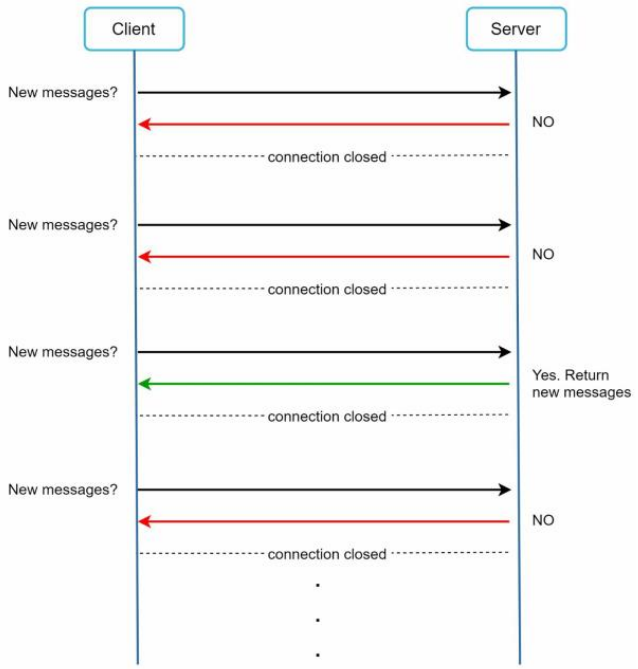
شکل ۲-۱۲

هنگامی که یک کاربر قصد دارد یک چت را شروع کند، سرویس چت را با استفاده از یک یا چند پروتکل شبکه متصل می‌کند. برای یک سرویس چت، انتخاب پروتکل‌های شبکه مهم است. بگذارید این موضوع را با مصاحبه‌کننده در میان بگذاریم. درخواست‌ها برای اکثر برنامه‌های سرویس گیرنده/سرور توسط کلاینت آغاز می‌شود. این برای طرف فرستنده یک برنامه چت نیز صادق است. در شکل ۲-۱۲، هنگامی که فرستنده از طریق سرویس چت پیامی را برای گیرنده ارسال می‌کند، از پروتکل HTTP تست شده با زمان[1] استفاده می‌کند که رایج‌ترین پروتکل وب است. در این سناریو، کلاینت یک اتصال HTTP را با سرویس چت باز می‌کند و پیامی را ارسال می‌کند و به سرویس اطلاع می‌دهد که پیام را به گیرنده ارسال کند. استفاده از هدر Keep-alive برای این کار مناسب است؛ زیرا هدر keep-alive به کلاینت اجازه می‌دهد تا ارتباط دائمی با سرویس چت را حفظ کند. همچنین تعداد handshakesهای مربوط به TCP را کاهش می‌دهد. HTTP یک گزینه خوب در سمت فرستنده است و بسیاری از برنامه‌های چت محبوب مانند Facebook [۱] در ابتدا از HTTP برای ارسال پیام استفاده می‌کردند. بااین‌حال، سمت گیرنده کمی پیچیده‌تر است. از آنجایی که HTTP توسط کلاینت آغاز می‌شود، ارسال پیام از سرور امری بی‌اهمیت نیست. در طول سال‌ها، تکنیک‌های زیادی برای شبیه‌سازی یک اتصال آغاز شده توسط سرور استفاده می‌شود: polling، long polling و WebSocket. اینها تکنیک‌های مهمی هستند که به طور گسترده در مصاحبه‌های طراحی سیستم مورد استفاده قرار می‌گیرند، بنابراین اجازه دهید هر یک از آنها را بررسی کنیم.

1 time-tested HTTP protocol

معرفی Polling

همان‌طور که در شکل ۳-۱۲ نشان‌داده‌شده است، polling[1] تکنیکی است که کلاینت به‌صورت دوره‌ای از سرور می‌پرسد که آیا پیام‌های موجود وجود دارد یا خیر. بسته به دفعات polling، استفاده از polling می‌تواند پرهزینه باشد. این می‌تواند منابع ارزشمند سرور را برای پاسخ دادن به سؤالی مصرف کند که اغلب اوقات پاسخ منفی ارائه می‌دهد.

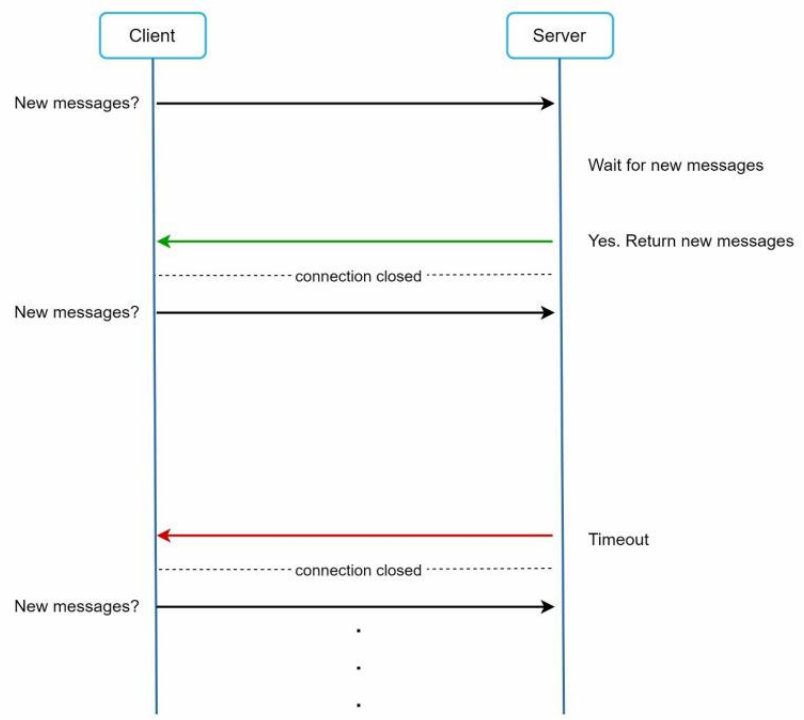


شکل ۳-۱۲

معرفی Long polling

ازآنجایی‌که polling می‌تواند ناکارآمد باشد، مرحله بعدی در معرفی long polling است. (شکل ۴-۱۲).

[1] در لغت به معنی نظرسنجی است.



شکل ۴-۱۲

در long polling، یک کلاینت اتصال را باز نگه می‌دارد تا زمانی که واقعاً پیام‌های جدیدی در دسترس باشد یا به آستانه مهلت مقرر رسیده باشد. هنگامی که کلاینت پیام‌های جدید دریافت می‌کند، بلافاصله درخواست دیگری را به سرور ارسال می‌کند و فرایند را دوباره راه‌اندازی می‌کند. long polling چند اشکال دارد:

- فرستنده و گیرنده ممکن است به یک سرور چت متصل نشوند. سرورهای مبتنی بر HTTP معمولاً stateless هستند. اگر از round robin[۱] برای متعادل‌کردن بار استفاده می‌کنید، سروری که پیام را دریافت می‌کند ممکن است ارتباط طولانی‌مدت با کلاینت دریافت‌کننده پیام نداشته باشد.

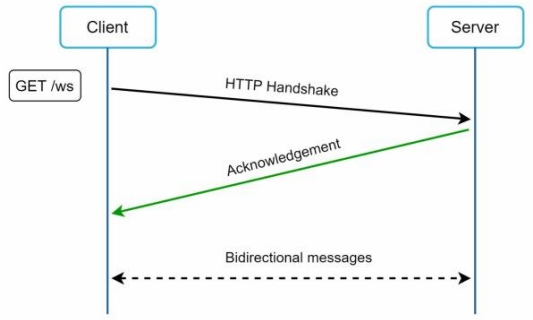
---

۱ راند رابین (به معنی نوبت گردشی) یک روش زمان‌بندی در علوم کامپیوتر است که در آن به هر فرآیند یا کاربر به طور متناوب و برای یک دوره زمانی مشخص، زمان پردازش اختصاص داده می‌شود. این روش معمولاً برای مدیریت صف‌ها، تخصیص منابع و برنامه‌ریزی وظایف استفاده می‌شود.

- یک سرور راه خوبی برای تشخیص قطع‌شدن ارتباط کلاینت ندارد.
- ناکارآمد و نامناسب است. اگر کاربر زیاد چت نمی‌کند، long polling همچنان پس از وقفه‌های زمانی، اتصالات دوره‌ای برقرار می‌کند.

معرفی WebSocket

وب سوکت رایج‌ترین راه‌حل برای ارسال به‌روزرسانی ناهم‌زمان[1] از سرور به کلاینت است. شکل ۵-۱۲ نحوه عملکرد آن را نشان می‌دهد.



شکل ۵-۱۲

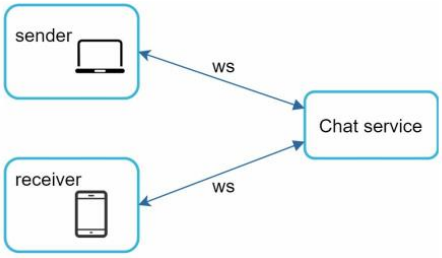
اتصال WebSocket توسط کلاینت آغاز می‌شود. دوجهته و پایدار است. حالت اولیه خود را به‌عنوان یک اتصال HTTP شروع می‌کند و می‌تواند از طریق چند handshake[2] کاملاً تعریف شده به یک اتصال WebSocket به طرز دقیقی متصل شود. از طریق این اتصال مداوم، یک سرور می‌تواند به‌روزرسانی‌ها را برای یک کلاینت ارسال کند. اتصالات WebSocket به‌طورکلی حتی اگر فایروال وجود داشته باشد کار می‌کنند. این به این دلیل است که آنها از پورت ۸۰ یا ۴۴۳ استفاده می‌کنند که توسط اتصالات HTTP/HTTPS نیز استفاده می‌شود. قبلاً گفتیم که در سمت فرستنده، HTTP پروتکل خوبی برای استفاده است، اما از آنجایی که WebSocket دوطرفه است، دلیل فنی قوی برای عدم استفاده از آن برای ارسال نیز وجود

---

1 asynchronous

۲ Handshaking در علوم کامپیوتر به فرایند شروع عملیات ارتباطی بین دو سیستم گفته می‌شود. هنگامی که یک سیستم پیامی را از طریق کانال ارتباطی به سیستم دیگر ارسال می‌کند و درخواست ایجاد یک ارتباط می‌کند، Handshaking آغاز می‌شود.

ندارد. شکل ۶-۱۲ نشان می‌دهد که چگونه WebSockets یا به‌اختصار (ws) برای هر دو طرف فرستنده و گیرنده استفاده می‌شود.



شکل ۶-۱۲

با استفاده از WebSocket هم برای ارسال و هم برای دریافت، طراحی و پیاده‌سازی را هم روی کلاینت و هم روی سرور ساده‌تر می‌کند. از آنجایی که اتصالات WebSocket پایدار هستند، مدیریت اتصال کارآمد در سمت سرور بسیار مهم است.

طراحی سطح بالا

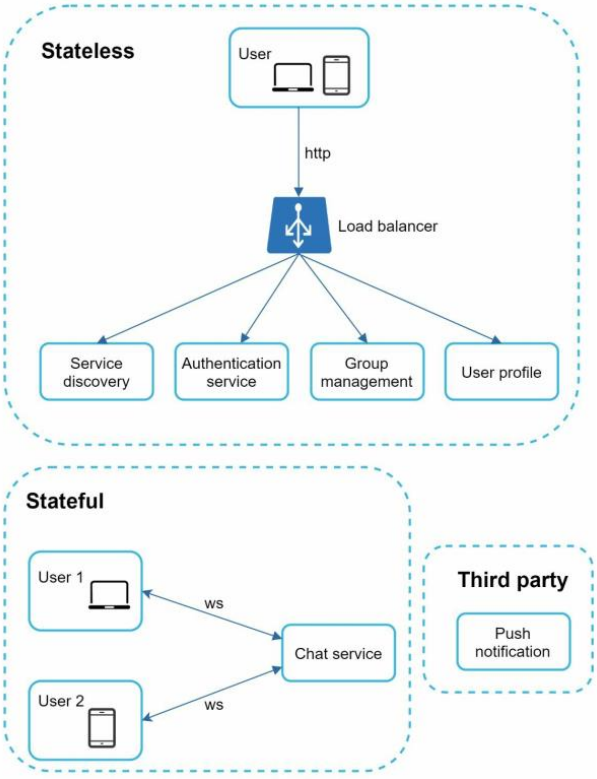
تا کنون اشاره کردیم که WebSocket به‌عنوان پروتکل ارتباطی اصلی بین کلاینت و سرور برای ارتباط دوطرفه آن انتخاب شده است، لازم به ذکر است که همه‌چیز دیگر نباید WebSocket باشد. در واقع، بیشتر ویژگی‌ها (ثبت‌نام، ورود به سیستم، پروفایل کاربر و غیره) یک برنامه چت می‌توانند از روش سنتی request/response روی HTTP استفاده کنند. اجازه دهید کمی کنجکاوی کنیم و به اجزای سطح بالای سیستم نگاه کنیم. همان‌طور که در شکل ۷-۱۲ نشان‌داده‌شده است، سیستم چت به سه دسته اصلی تقسیم می‌شود: stateless services و stateful services و یکپارچه سازی شخص ثالث[1].

Stateless Services

سرویس‌های Stateless، سرویس‌های معمولی request/response عمومی هستند که برای مدیریت ثبت‌نام، ورود به سیستم، پروفایل کاربر و غیره استفاده می‌شوند. اینها ویژگی‌های

1 third-party integration

رایج در میان بسیاری از وب‌سایت‌ها و برنامه‌ها هستند. سرویس‌های Stateless پشت یک متعادل‌کننده بار[1] قرار دارند که وظیفه آن هدایت درخواست‌ها به سمت سرویس‌های صحیح بر اساس مسیردهی درخواست‌ها[2] است. این سرویس‌ها می‌توانند یکپارچه[3] یا میکروسرویس‌های مستقل باشند. ما نیازی نداریم که بسیاری از این سرویس‌ها Stateless را خودمان بسازیم؛ زیرا سرویس‌هایی در بازار وجود دارد که به‌راحتی می‌توان آنها را ادغام کرد. یکی از سرویس‌هایی که در بررسی عمیق‌تر بیشتر به آن خواهیم پرداخت، service discovery است که وظیفه اصلی آن ارائه لیستی از نام‌های میزبان[4] برای DNS سرورهای چت است که کلاینت می‌تواند به آنها متصل شود.



شکل ۷-۱۲

1 load balancer
2 request
3 monolithic
4 host names

Stateful Service

این تنها سرویس stateful چت است، زیرا هر کلاینت یک اتصال شبکه دائمی به یک سرور چت را حفظ می‌کند. در این سرویس، یک کلاینت معمولاً تا زمانی که سرور هنوز در دسترس[1] است به سرور چت دیگری سوئیچ نمی‌کند. service discovery از نزدیک با سرویس چت هماهنگ می‌شود تا از بارگذاری بیش از حد سرور جلوگیری کند. ما به جزئیات بیشتر در بررسی عمیق‌تر خواهیم پرداخت.

Third-party integration – ادغام شخص ثالث

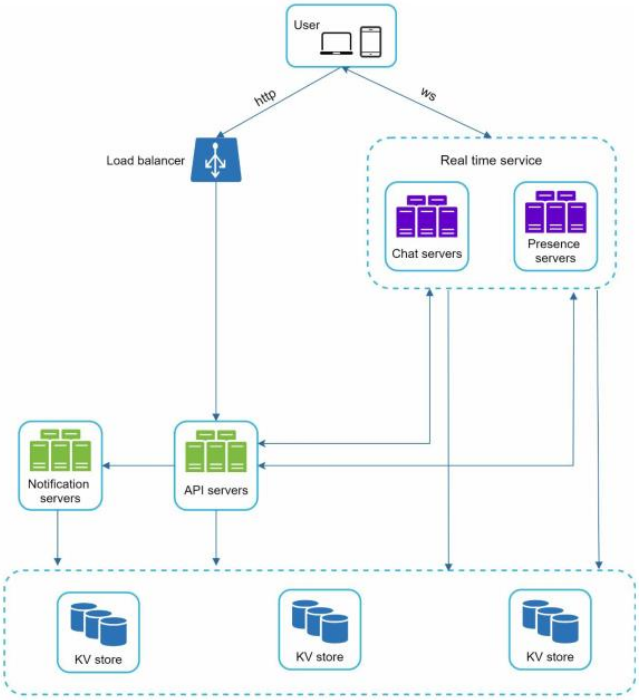
برای یک برنامه چت، push notification مهم‌ترین ادغام third-party است. این روشی است برای اطلاع کاربران از دریافت پیام‌های جدید، حتی زمانی که برنامه اجرا نمی‌شود. ادغام مناسب push notification بسیار مهم است. برای اطلاعات بیشتر به بخش ۱۰ مراجعه کنید.

مقیاس‌پذیری

در مقیاس کوچک، همه سرویس‌ها ذکر شده در بالا می‌توانند در یک سرور قرار بگیرند. حتی در مقیاسی که ما برای آن طراحی می‌کنیم، از نظر تئوری امکان جا دادن همه اتصالات کاربر در یک سرور ابری مدرن وجود دارد. تعداد اتصالات هم‌زمانی که یک سرور می‌تواند انجام دهد به‌احتمال زیاد عامل محدودکننده‌ای خواهد بود. در سناریوی ما، با ۱ میلیون کاربر هم‌زمان، با فرض اینکه هر اتصال کاربر به 10K حافظه روی سرور نیاز دارد (این رقم بسیار نادرست است و بسیار به انتخاب زبان‌بستگی برنامه نویسی دارد)، تنها به حدود ۱۰ گیگابایت حافظه نیاز دارد تا همه اتصالات را روی یک سرور نگه دارد. اگر طرحی را پیشنهاد کنیم که همه‌چیز در یک سرور قرار بگیرد، ممکن است یک پرچم قرمز[2] بزرگ در ذهن مصاحبه‌کننده ایجاد کند. هیچ مهندسی چنین مقیاسی را در یک سرور طراحی نمی‌کند. طراحی تک سرور به دلیل عوامل زیادی باعث ایجاد مشکل می‌شود. تنها نقطه شکست در میان آنها منجر به بزرگ‌ترین فاجعه‌ها می‌شود. بااین‌حال، شروع با طراحی یک سرور کاملاً خوب است. فقط مطمئن شوید که مصاحبه‌کننده می‌داند که این

1 available
2 red flag

یک نقطه شروع است. با کنار هم قراردادن همه‌چیزهایی که ذکر کردیم، شکل ۸-۱۲ طراحی سطح بالا تنظیم شده را نشان می‌دهد.



شکل ۸-۱۲

در شکل ۸-۱۲، کلاینت یک اتصال WebSocket دائمی به یک سرور چت برای پیام رسانی بلادرنگ حفظ می‌کند.

- سرورهای چت، ارسال/دریافت پیام را تسهیل می‌کنند.
- سرورهای حضوروغیاب[1] وضعیت آنلاین/آفلاین را مدیریت می‌کنند.
- سرورهای API همه‌چیز از جمله ورود کاربر، ثبت‌نام، تغییر پروفایل و غیره را مدیریت می‌کنند.
- سرورهای نوتیفیکیشن، push notificationها را ارسال می‌کنند.
- در نهایت، ذخیره‌ساز key-value برای ذخیره تاریخچه چت استفاده می‌شود. هنگامی که یک کاربر آفلاین یا آنلاین می‌شود، تمام سابقه چت قبلی خود را می‌بینند.

1 Presence server

ذخیره‌گاه (Storage)

در این مرحله، ما سرورهای آماده و سرویس‌ها در حال اجرا و ادغام‌های شخص ثالث را داریم. در اعماق تکنولوژی‌ها فنی[1]، لایه داده[2] وجود دارد. لایه داده‌ها معمولاً به تلاشی برای درست‌کردن و تصحیح آن نیاز دارد. تصمیم مهمی که باید بگیریم این است که در مورد نوع مناسب پایگاه‌داده برای استفاده تصمیم‌گیری کنیم: پایگاه‌های داده رابطه‌ای یا پایگاه‌های داده NoSQL؟ برای تصمیم‌گیری آگاهانه، انواع داده‌ها و الگوهای خواندن/نوشتن را بررسی می‌کنیم. دو نوع داده در یک سیستم چت معمولی وجود دارد. **اولین** مورد، داده‌های عمومی است، مانند پروفایل کاربر، تنظیمات، لیست دوستان کاربر. این داده‌ها در پایگاه‌داده‌های رابطه‌ای قوی و قابل‌اعتماد ذخیره می‌شوند. تکثیر[3] و چند قطعه‌کردن داده‌ها[4] تکنیک‌های رایج برای برآوردن نیازهای دردسترس‌بودن و مقیاس‌پذیری هستند. **دومین** مورد منحصر به سیستم‌های چت است: داده‌های تاریخچه چت و درک الگوی خواندن/نوشتن بسیار مهم است.

- مقدار داده برای سیستم‌های چت بسیار زیاد است. مطالعه قبلی [۲] نشان می‌دهد که پیام‌رسان فیس بوک و واتساپ روزانه ۶۰ میلیارد پیام را پردازش می‌کنند.
- فقط چت‌های اخیر اغلب قابل‌دسترسی هستند. کاربران معمولاً چت‌های قدیمی را جستجو نمی‌کنند.
- اگرچه در بیشتر موارد سابقه چت بسیار اخیر مشاهده می‌شود، کاربران ممکن است از ویژگی‌هایی استفاده کنند که به دسترسی تصادفی به داده‌ها نیاز دارد، مانند جستجو، مشاهده نام‌های شما، پرش به پیام‌های خاص، و غیره. این موارد باید توسط لایه دسترسی به داده پشتیبانی شوند.
- نسبت خواندن به نوشتن حدود ۱:۱ برای برنامه چت یک به یک است.

انتخاب سیستم ذخیره‌سازی صحیح که از همه موارد استفاده ما پشتیبانی می‌کند بسیار مهم است. ما ذخیره‌سازهایی از نوع Key-value را به دلایل زیر توصیه می‌کنیم:

---

1 stack
2 data layer
3 Replication
4 Sharding

- ذخیره‌سازهای از نوع Key-value امکان مقیاس‌بندی آسان افقی[1] را فراهم می‌کنند.
- ذخیره‌سازهای از نوع Key-value تأخیر بسیار کمی برای دسترسی به داده‌ها فراهم می‌کنند.
- پایگاه‌داده‌های رابطه‌ای معمولاً دنباله‌ی بلند [۳] از داده‌ها را به‌خوبی مدیریت نمی‌کنند. هنگامی که ایندکس‌ها بزرگ می‌شوند، دسترسی تصادفی پرهزینه است.
- ذخیره‌سازهای از نوع Key-value توسط دیگر برنامه‌های چت قابل‌اعتماد به کار گرفته می‌شوند. به‌عنوان‌مثال، پیام‌رسان فیس‌بوک و Discord از ذخیره‌سازهای از نوع Key-value استفاده می‌کنند. پیام‌رسان فیس‌بوک از HBase [۴] و Discord از Cassandra [۵] استفاده می‌کند.

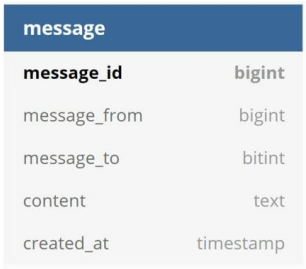
مدل داده‌ها (Data models)

در حال حاضر، ما در مورد استفاده از ذخیره‌سازهای از نوع Key-value به‌عنوان لایه ذخیره‌سازی صحبت کردیم. مهم‌ترین داده، داده‌های مربوط به پیام است. اجازه دهید نگاهی دقیق بیندازیم.

جدول پیام برای چت یک‌به‌یک

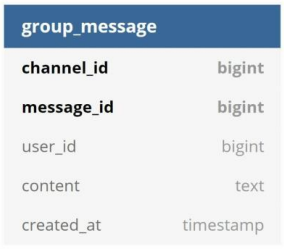
شکل ۹-۱۲ جدول پیام[2] را برای چت ۱ به ۱ نشان می‌دهد. کلید اصلی message_id است که به تعیین توالی پیام کمک می‌کند. ما نمی‌توانیم برای تصمیم‌گیری در مورد توالی پیام به create_at تکیه کنیم، زیرا می‌توان هم‌زمان دو پیام را ایجاد کند.

1 horizontal scaling
2 Message table



شکل ۹-۱۲

شکل ۱۰-۱۲ جدول پیام‌ها را برای چت گروهی نشان می‌دهد. کلید اولیه ترکیبی از (channel_id، message_id) است. کانال و گروه در اینجا معنای یکسانی را نشان می‌دهند. channel_id کلید پارتیشن است زیرا تمام کوئری‌ها در یک چت گروهی و در یک کانال عمل می‌کنند.



شکل ۱۰-۱۲

شناسه پیام (Message ID)

نحوه تولید message_id موضوع جالبی است که ارزش بررسی دارد. Message_id مسئولیت اطمینان از ترتیب پیام‌ها را بر عهده دارد. برای اطمینان از ترتیب پیام‌ها، message_id باید دو شرط زیر را برآورده کند:

- شناسه‌ها[1] باید منحصربه‌فرد باشند.
- شناسه‌ها باید بر اساس زمان قابل مرتب سازی باشند، به این معنی که ردیف‌های جدید شناسه‌های بالاتری نسبت به ردیف‌های قدیمی دارند.

1 IDs

چگونه می‌توانیم به این دو ضمانت دست یابیم؟ اولین ایده‌ای که به ذهن می‌رسد کلمه کلیدی "auto_increment" در MySQL است. بااین‌حال، پایگاه‌های داده NoSQL معمولاً چنین ویژگی را ارائه نمی‌دهند.

روش دوم استفاده از یک مولد اعداد توالی ۶۴ بیتی سراسری[1] مانند Snowflake [۶] است. این مورد در «فصل ۷: طراحی یک تولیدکننده شناسه منحصربه‌فرد در یک سیستم توزیع شده» بحث شده است. رویکرد نهایی استفاده از مولد شماره دنباله محلی[2] است. محلی به این معنی است که شناسه‌ها فقط در یک گروه منحصربه‌فرد هستند. دلیل کارکرد شناسه‌های محلی این است که حفظ توالی پیام در کانال یک به یک یا کانال گروهی کافی است. این رویکرد در مقایسه با پیاده‌سازی شناسه سراسری آسان‌تر است.

مرحله ۳ - بررسی عمیق‌تر

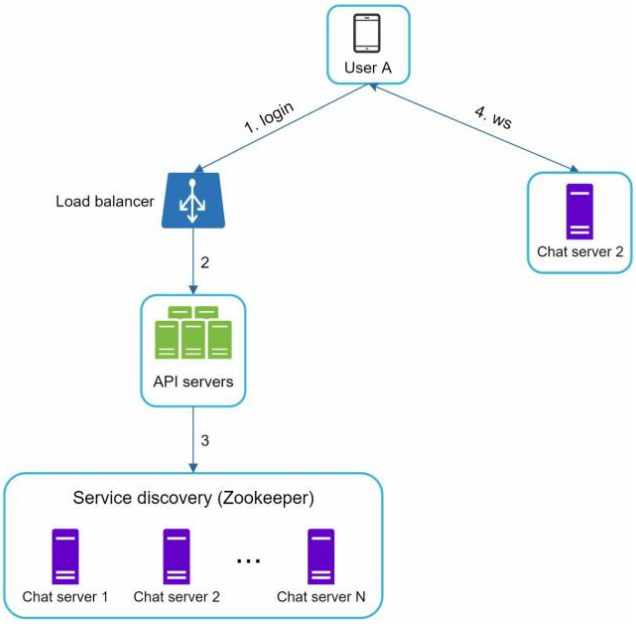
در یک مصاحبه طراحی سیستم، معمولاً از شما انتظار می‌رود که عمیقاً در برخی از اجزای طراحی سطح بالا غوطه‌ور شوید. برای سیستم چت، discovery سرویس‌ها، مسیرهای پیام‌رسانی و شاخص‌های آنلاین/آفلاین ارزش کاوش عمیق‌تر دارند.

Service discovery

نقش اصلی سرویس discovery این است که بهترین سرور چت را برای یک کلاینت بر اساس معیارهایی مانند موقعیت جغرافیایی، ظرفیت سرور و غیره توصیه کند. Apache Zookeeper [۷] یک راه‌حل منبع باز محبوب برای کشف سرویس است. تمام سرورهای چت موجود را ثبت می‌کند و بهترین سرور چت را برای یک کلاینت بر اساس معیارهای از پیش تعریف شده انتخاب می‌کند. شکل ۱۱-۱۲ نحوه عملکرد discovery سرویس Zookeeper را نشان می‌دهد.

---

1 global 64-bit sequence number generator
2 local sequence number generator



شکل ۱۱-۱۲

۱. کاربر A سعی می‌کند وارد برنامه شود.

۲. متعادل‌کننده بار یا load balancer درخواست ورود را به سرورهای API ارسال می‌کند.

۳. پس از احراز هویت کاربر توسط backend، سرویس Discovery بهترین سرور چت را برای کاربر A پیدا می‌کند. در این مثال، سرور ۲ انتخاب شده و اطلاعات سرور به کاربر A برگردانده می‌شود.

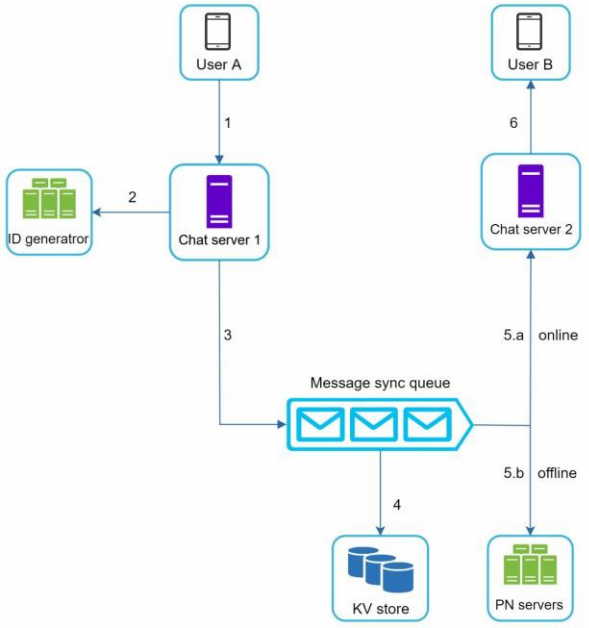
۴. کاربر A از طریق WebSocket به سرور چت ۲ متصل می‌شود.

مسیرهای پیام

درک مسئله end-to-end یک سیستم چت موضوعی جالب است. در این بخش، مسیرهای چت ۱ به ۱، همگام سازی پیام در چندین دستگاه و جریان چت گروهی را بررسی خواهیم کرد.

مسیرها در چت یک به یک

شکل ۱۲-۱۲ توضیح می‌دهد که وقتی کاربر A پیامی را برای کاربر B ارسال می‌کند چه اتفاقی می‌افتد.



شکل ۱۲-۱۲

۱. کاربر A یک پیام چت به سرور چت ۱ ارسال می‌کند.

۲. سرور چت ۱ یک شناسه پیام از تولیدکننده شناسه دریافت می‌کند.

۳. سرور چت ۱ پیام را به صف همگام سازی پیام[1] می‌فرستد.

۴. پیام در یک ذخیره‌ساز key-value ذخیره می‌شود.

(الف) اگر کاربر B آنلاین باشد، پیام به سرور چت ۲ که در آن کاربر B متصل است، ارسال می‌شود.

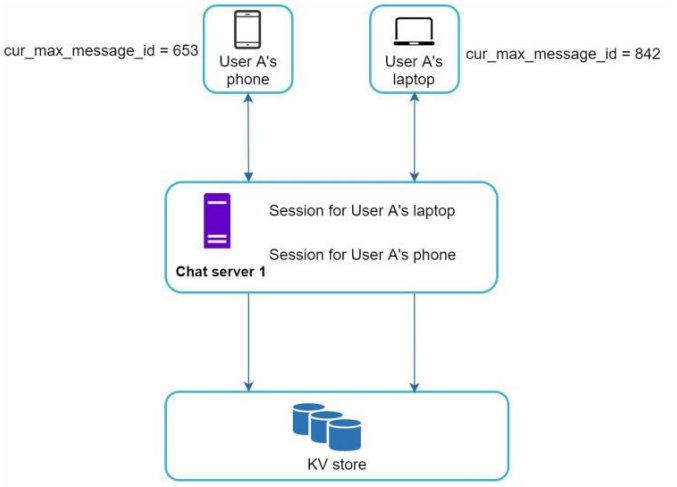
1 message sync queue

(ب) اگر کاربر B آفلاین باشد، یک push notification از سرورهای push notification یا به‌اختصار (PN) ارسال می‌شود.

۵. سرور چت ۲ پیام را به کاربر B ارسال می‌کند. یک اتصال WebSocket دائمی بین کاربر B و سرور چت ۲ وجود دارد.

همگام‌سازی پیام در چندین دستگاه

بسیاری از کاربران چندین دستگاه دارند. ما نحوه همگام‌سازی[1] پیام‌ها را در چندین دستگاه توضیح خواهیم داد. شکل ۱۳-۱۲ نمونه‌ای از همگام سازی پیام را نشان می‌دهد.



شکل ۱۳-۱۲

در شکل ۱۳-۱۲، کاربر A دو دستگاه دارد: یک تلفن و یک لپ تاپ. هنگامی که کاربر A با تلفن خود وارد برنامه چت می‌شود، یک اتصال WebSocket با سرور چت ۱ برقرار می‌کند. به طور مشابه، ارتباطی بین لپ‌تاپ و سرور چت ۱ وجود دارد. هر دستگاه متغیری به نام cur_max_message_id را نگه می‌دارد که ردیابی آخرین شناسه پیام روی دستگاه را بررسی می‌کند و پیام‌هایی که دارای دو شرط زیر باشند؛ پیام جدید محسوب می‌شوند:

- شناسه گیرنده برابر با شناسه کاربری وارد شده فعلی است.

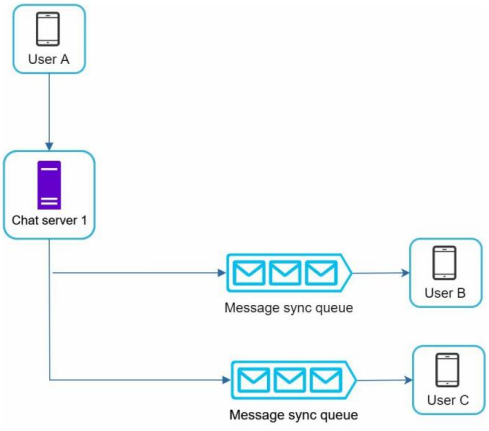
1 synchronization

- شناسه پیام در ذخیره‌ساز key-value بزرگتر از cur_max_message_id است.

با cur_max_message_id متمایز در هر دستگاه، همگام‌سازی پیام آسان است؛ زیرا هر دستگاه می‌تواند پیام‌های جدیدی را از ذخیره‌ساز KV دریافت کند.

مسیرها در یک چت گروهی کوچک

در مقایسه با چت یک به یک، منطق چت گروهی پیچیده‌تر است. شکل های ۱۴-۱۲ و ۱۵-۱۲ مسیرهای مربوطه را توضیح می دهند.



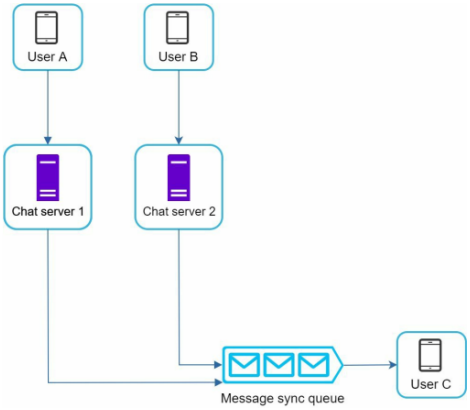
شکل ۱۴-۱۲

شکل ۱۴-۱۲ توضیح می‌دهد که وقتی کاربر A پیامی را در یک چت گروهی ارسال می‌کند چه اتفاقی می‌افتد. فرض کنید ۳ عضو در گروه (کاربر A، کاربر B و کاربر C) وجود دارد. ابتدا، پیام کاربر A در صف همگام‌سازی پیام[1] هر یک از اعضای گروه کپی می‌شود: یکی برای کاربر B و دومی برای کاربر C. می‌توانید صف همگام‌سازی پیام را به‌عنوان یک صندوق ورودی برای یک گیرنده در نظر بگیرید. این انتخاب طراحی برای چت گروهی کوچک خوب است زیرا:

- مسیر همگام‌سازی پیام را ساده می‌کند؛ زیرا هر کلاینت فقط باید صندوق ورودی خود را بررسی کند تا پیام‌های جدید دریافت کند.
- وقتی تعداد گروه کوچک است، ذخیره یک کپی در صندوق ورودی هر گیرنده خیلی پرهزینه نیست.

1 message sync queue

اپلیکیشن WeChat از رویکرد مشابهی استفاده می‌کند و یک گروه را به ۵۰۰ عضو محدود می‌کند [۸]. بااین‌حال، برای گروه‌هایی با تعداد زیادی کاربر، ذخیره یک کپی پیام برای هر عضو قابل قبول نیست. در سمت گیرنده، یک گیرنده می‌تواند پیام‌هایی از چندین کاربر دریافت کند. هر گیرنده یک صندوق ورودی (صف همگام سازی پیام) دارد که حاوی پیام‌هایی از فرستندگان مختلف است. شکل ۱۵-۱۲ این طرح را نشان می‌دهد.



شکل ۱۵-۱۲

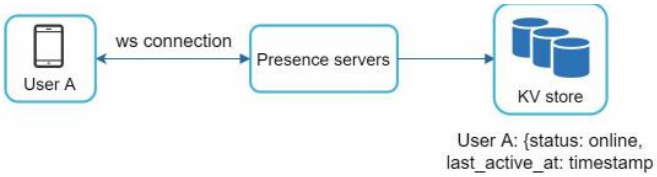
نشانگر وضعیت آنلاین

نشانگر وضعیت آنلاین[1] یکی از ویژگی‌های ضروری بسیاری از برنامه‌های چت است. معمولاً می‌توانید یک نقطه سبزرنگ در کنار تصویر پروفایل یا نام کاربری یک کاربر ببینید. در این بخش آنچه در پشت‌صحنه اتفاق می‌افتد توضیح می‌دهد. در طراحی سطح بالا، سرورهای نشانگر حضور مسئول مدیریت وضعیت آنلاین و ارتباط با کلاینت‌ها از طریق WebSocket هستند. چند مسئله وجود دارد که باعث تغییر وضعیت آنلاین[2] می‌شود. اجازه دهید هر یک از آنها را بررسی کنیم.

1 Online presence
2 online status

ورود کـــاربـــران

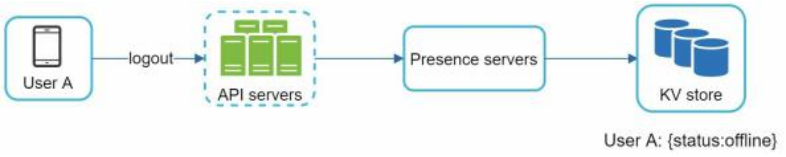
مسئله ورود کاربر[1] در بخش "Service Discovery" توضیح داده شده است. پس از ایجاد اتصال WebSocket بین کلاینت و سرویس بلادرنگ، وضعیت آنلاین کاربر A و timestamp یا برچسب زمانی با عنوان last_active_at در ذخیره‌ساز KV ذخیره می‌شود. درنتیجه نشانگر آنلاین کاربر، پس از ورود به سیستم آنلاین است.



شکل ۱۶-۱۲

خـــروج کـــاربـــران

هنگامی که یک کاربر از سیستم خارج می‌شود[2]، همان‌طور که در شکل ۱۷-۱۲ نشان‌داده‌شده است، از مسئله خروج کاربر عبور می‌کند. وضعیت آنلاین در ذخیره‌ساز KV به آفلاین تغییر کرده است. در نتیجه نشانگر آنلاین کاربر آفلاین است.



شکل ۱۷-۱۲

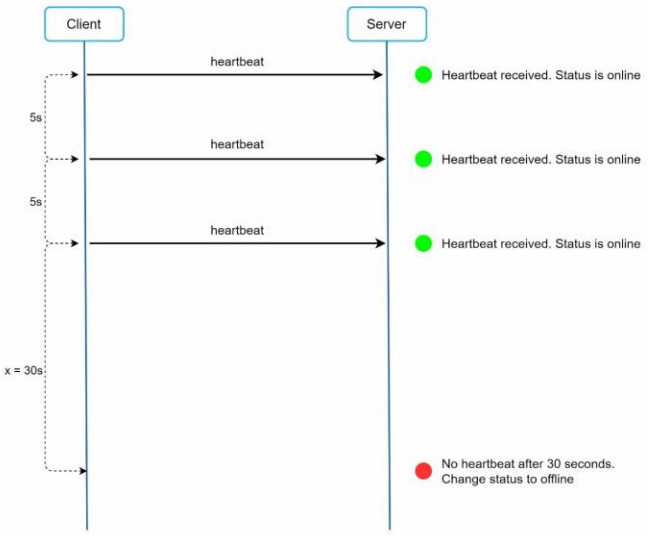
قطع ارتباط کاربر

همه ما آرزو می‌کنیم که اتصال اینترنتی ما ثابت و قابل‌اعتماد باشد. به‌هرحال این همیشه درست نیست؛ بنابراین، ما باید در طراحی خود به این موضوع بپردازیم. هنگامی که دسترسی

1 User login
2 User logout

کاربر از اینترنت قطع می‌شود، ارتباط دائمی بین کلاینت و سرور از بین می‌رود. یک روش ساده‌لوحانه برای مدیریت قطع ارتباط کاربر، علامت‌گذاری کاربر به‌عنوان آفلاین و تغییر وضعیت به آنلاین هنگام برقراری مجدد اتصال است. بااین‌حال، این رویکرد دارای یک نقص اساسی است. معمولاً کاربران در مدت‌زمان کوتاهی به طور مکرر اینترنت را قطع و دوباره به آن متصل می‌شوند. به‌عنوان‌مثال، اتصالات شبکه می‌تواند در زمانی که کاربر از طریق یک تونل می‌گذرد، خاموش و روشن باشد. به‌روزرسانی وضعیت آنلاین در هر قطع/وصل مجدد، نشانگر آنلاین[1] را اغلب تغییر می‌دهد و منجر به تجربه کاربری ضعیف می‌شود. ما مکانیسم ضربان قلب[2] را برای حل این مشکل معرفی می‌کنیم. به‌صورت دوره‌ای، یک کلاینت آنلاین یک رویداد heartbeat را به سرورهای آنلاین ارسال می‌کند. اگر سرورهای موجود یک رویداد ضربان قلب را در مدت‌زمان معینی، مثلاً x ثانیه از کلاینت دریافت کنند، کاربر به‌عنوان وضعیت آنلاین در نظر گرفته می‌شود. در غیر این صورت کاربر به‌عنوان وضعیت آفلاین است. در شکل ۱۸-۱۲، کلاینت هر ۵ ثانیه یک رویداد heartbeat را به سرور ارسال می‌کند. پس از ارسال ۳ رویداد ضربان قلب، کلاینت قطع می‌شود و در عرض $x = 30$ ثانیه دوباره وصل نمی‌شود (این عدد به‌صورت دلخواه برای نشان دادن منطق موردنظر انتخاب می‌شود). پس درنتیجه وضعیت آنلاین به آفلاین تغییر کرده است.

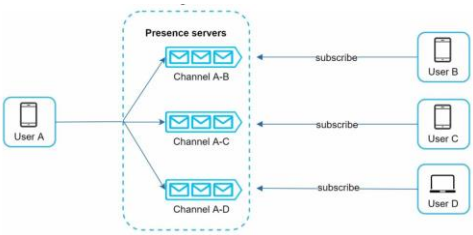
1 online status
2 heartbeat



شکل ۱۸-۱۲

وضعیت آنلاین دوستان

دوستان کاربر A چگونه از تغییرات وضعیت اطلاع دارند؟ شکل ۱۹-۱۲ نحوه عملکرد آن را توضیح می‌دهد. سرورهای بررسی نشانگر حضوری[1] از مدل انتشار-اشتراک[2] استفاده می‌کنند که در آن ارتباط هر جفت دوست در یک کانال نگهداری می‌شوند. هنگامی که وضعیت آنلاین کاربر A تغییر می‌کند، رویداد را در سه کانال از جمله کانال A-B، A-C و A-D منتشر می‌کند. این سه کانال به ترتیب توسط کاربر B، C و D مشترک هستند. بنابراین، دریافت به‌روزرسانی وضعیت آنلاین برای دوستان آسان است. ارتباط بین کلاینت‌ها و سرورها از طریق WebSocket بلادرنگ است.



شکل ۱۹-۱۲

1 Presence servers
2 publish-subscribe

طراحی فوق برای یک گروه کاربری کوچک مؤثر است. به‌عنوان‌مثال، WeChat از رویکرد مشابهی استفاده می‌کند؛ زیرا گروه کاربری آن تا ۵۰۰ عضو محدود است. برای گروه‌های بزرگ‌تر، اطلاع رسانی به همه اعضا در مورد وضعیت آنلاین، پرهزینه و زمان بر است. فرض کنید یک گروه ۱۰۰۰۰۰ عضو دارد. هر تغییر وضعیت ۱۰۰۰۰۰ رویداد ایجاد می‌کند. برای کاهش مشکل سازی این گلوگاه، یک راه‌حل ممکن این است که وضعیت آنلاین را تنها زمانی دریافت کنید که کاربر وارد یک گروه شود یا به‌صورت دستی لیست دوستان را بازخوانی کند.

مرحله ٤ – جمع‌بندی

در این فصل، ما یک معماری سیستم چت را ارائه کردیم که از چت یک‌به‌یک و چت گروهی کوچک پشتیبانی می‌کند. وب سوکت برای ارتباط بلادرنگ بین کلاینت و سرور استفاده می‌شود. سیستم چت شامل اجزای زیر است: سرورهای چت برای پیام‌رسانی بلادرنگ، سرورهای نشانگر حضوری برای مدیریت حضور آنلاین، سرورهای نوتیفیکیشن برای ارسال push notifications، ذخیره‌سازهای key-value برای نگهداری تاریخچه چت و سرورهای API برای سایر عملکردها. اگر در پایان مصاحبه وقت اضافی دارید، در اینجا نکات دیگری برای صحبت وجود دارد:

- برنامه چت را برای پشتیبانی از فایل‌های رسانه‌ای مانند عکس‌ها و فیلم‌ها گسترش دهید. اندازه فایل‌های رسانه‌ای به طور قابل‌توجهی بزرگ‌تر از متن است. فشرده‌سازی، ذخیره‌سازی ابری و تصاویر کوچک[1] موضوعات جالبی هستند که می‌توان در مورد آنها صحبت کرد.
- **رمزگذاری سراسری**. واتس‌اپ از رمزگذاری سراسری پیام‌ها پشتیبانی می‌کند. فقط فرستنده و گیرنده می‌توانند پیام‌ها را بخوانند. خوانندگان علاقه‌مند باید به مقاله در منابع مرجع [۹] مراجعه کنند.
- ذخیره پیام‌ها در سمت سرویس گیرنده برای کاهش انتقال داده بین کلاینت و سرور مؤثر است.

1 thumbnails

- **بهبود زمان بارگذاری**. اپلیکیشن Slack یک شبکه توزیع شده جغرافیایی برای ذخیره داده‌ها، کانال‌ها و موارد دیگر مربوط به کاربران برای زمان بارگذاری بهتر ایجاد کرده است [۱۰].
- **رسیدگی به خطاها**.
- **خطای سرور چت**. ممکن است صدها هزار یا حتی بیشتر اتصالات مداوم به یک سرور چت وجود داشته باشد. اگر سرور چت آفلاین شود، سرویس اکتشاف[1] (Zookeeper) یک سرور چت جدید را برای کلاینت‌ها فراهم می‌کند تا با آنها ارتباط جدیدی برقرار کنند.
- **مکانیسم ارسال مجدد پیام**. تلاش مجدد[2] و صف‌بندی تکنیک‌های رایج برای ارسال مجدد پیام‌ها هستند.

1 discovery
2 Retry

منابع و مراجع فصل ۱۲

[1] Erlang at Facebook: https://www.erlang-factory.com/upload/presentations/31/EugeneLetuchy-ErlangatFacebook.pdf
[2] Messenger and WhatsApp process 60 billion messages a day: https://www.theverge.com/2016/4/12/11415198/facebook-messenger-whatsapp-number- messages-vs-sms-f8-2016
[3] Long tail: https://en.wikipedia.org/wiki/Long_tail
[4] The Underlying Technology of Messages: https://www.facebook.com/notes/facebook- engineering/the-underlying-technology-of-messages/454991608919/
[5] How Discord Stores Billions of Messages: https://blog.discordapp.com/how-discord- stores-billions-of-messages-7fa6ec7ee4c7
[6] Announcing Snowflake: https://blog.twitter.com/engineering/en_us/a/2010/announcing- snowflake.html
[7] Apache ZooKeeper: https://zookeeper.apache.org/
[8] From nothing: the evolution of WeChat background system (Article in Chinese): https://www.infoq.cn/article/the-road-of-the-growth-weixin-background
[9] End-to-end encryption: https://faq.whatsapp.com/en/android/28030015/
[10] Flannel: An Application-Level Edge Cache to Make Slack Scale: https://slack.engineering/flannel-an-application-level-edge-cache-to-make-slack-scale- b8a6400e2f6b

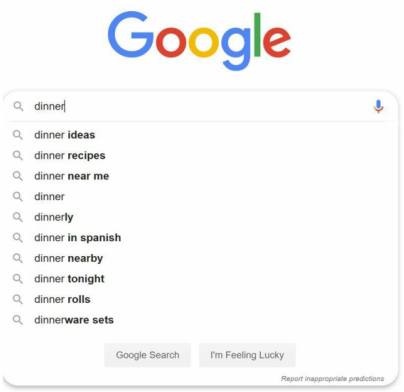
فصل ۱۳

طراحی یک سیستم جستجو مبتنی بر AUTOCOMPLETE

هنگام جستجو در گوگل یا خرید در آمازون، همان‌طور که در کادر جستجو تایپ می‌کنید، یک یا چند مورد مطابق با عبارت جستجو به شما ارائه می‌شود. این ویژگی به‌عنوان تکمیل خودکار یا autocomplete، پیش‌نویس[1]، جستجوی موارد مشابه[2] یا جستجوی افزایشی[3] نامیده می‌شود. شکل ۱-۱۳ نمونه‌ای از جستجوی گوگل را نشان می‌دهد که لیستی از نتایج تکمیل شده خودکار[4] را هنگام تایپ کلمه "dinner" در کادر جستجو نشان می‌دهد. تکمیل خودکار جستجو یکی از ویژگی‌های مهم بسیاری از محصولات است. این ما را به سؤال این مصاحبه نزدیک می‌کند: یک سیستم تکمیل خودکار جستجو طراحی کنید، که به آن «طراحی موارد برتر k[5]» یا «طراحی موارد برتر k در بیشترین کوئری‌ها[6]» نیز می‌گویند.

---

1 typeahead
2 search-as-you-type
3 incremental search
4 autocomplete
5 design top k
6 design top k most searched queries



شکل ۱-۱۳

مرحله ۱ - درک مسئله و شروع به طراحی

اولین قدم برای مقابله با هر سؤال مصاحبه طراحی سیستم، پرسیدن سؤالات کافی برای روشن‌کردن نیازمندی‌ها است. در اینجا نمونه‌ای از تعامل کاندید-مصاحبه‌کننده آورده شده است:

کاندید: آیا تطبیق فقط در ابتدای یک عبارت جستجو پشتیبانی می‌شود یا در وسط عبارت در حال جستجو؟

مصاحبه‌کننده: فقط در ابتدای یک عبارت در حال جستجو.

کاندید: سیستم چند پیشنهاد autocomplete را باید برگرداند؟

مصاحبه‌کننده: ۵ عدد.

کاندید: سیستم از کجا می‌داند که کدام یک از ۵ پیشنهاد برتر را برگرداند؟

مصاحبه‌کننده: این بر اساس محبوبیت تعیین می‌شود که با تعداد تکرار کوئری‌ها درگذشته تعیین می‌شود.

کاندید: آیا سیستم از چک‌کردن خطای تایپی و املایی پشتیبانی می‌کند؟

مصاحبه‌کننده: خیر، املا یا تصحیح خودکار پشتیبانی نمی‌شود.

کاندید: آیا سؤالات جستجو به زبان انگلیسی هستند؟

مصاحبه‌کننده: بله. اگر زمان در پایان اجازه دهد، می‌توانیم در مورد پشتیبانی چندزبانه صحبت کنیم.

کاندید: آیا اجازه حروف بزرگ و کاراکترهای خاص را می‌دهیم؟

مصاحبه‌کننده: نه، ما فرض می‌کنیم که همه عبارت‌های جستجو دارای حروف الفبای کوچک هستند.

کاندید: چند کاربر از محصول استفاده می‌کنند؟

مصاحبه‌کننده: ۱۰ میلیون DAU یا کاربر فعال روزانه.

الزامات طراحی

در اینجا خلاصه‌ای از الزامات مشخص شده است:

- **زمان پاسخ سریع**: زمانی که کاربر یک عبارت جستجو را تایپ می‌کند، پیشنهادها تکمیل خودکار باید به‌اندازه کافی سریع نمایش داده شوند. مقاله‌ای در مورد سیستم تکمیل خودکار فیس‌بوک [۱] نشان می‌دهد که این سیستم باید نتایج را در عرض ۱۰۰ میلی ثانیه برگرداند. در غیر این صورت باعث خطای stuttering[۱] می‌شود.
- **مرتبط**: پیشنهادها تکمیل خودکار باید با عبارت جستجو مرتبط باشد.
- **مرتب شده**: نتیجه‌های برگردانده شده توسط سیستم باید بر اساس محبوبیت یا سایر مدل‌های رتبه‌بندی مرتب شوند.

مقیاس‌پذیری: سیستم می‌تواند حجم ترافیک بالایی را مدیریت کند.

سطح دسترسی بالا: زمانی که بخشی از سیستم آفلاین است یا عملکرد آهسته‌ای دارد یا خطاهای شبکه غیرمنتظره‌ای را تجربه می‌کند، سیستم باید در دسترس باقی بماند.

تخمین طراحی

- فرض کنید این سیستم ۱۰ میلیون کاربر فعال روزانه[۲] (DAU) دارد.

---

۱ منظور نوعی تاخیر برای پیشنهاد دادن لغات در autocomplete است.

2 daily active users

- یک فرد به طور متوسط روزانه ۱۰ جستجو انجام می‌دهد.
- ۲۰ بایت داده در هر رشته کوئری:
  - فرض کنید از رمزگذاری کاراکتر ASCII استفاده می‌کنیم. ۱ کاراکتر = ۱ بایت
  - فرض کنید یک عبارت شامل ۴ کلمه و هر کلمه به طور متوسط شامل ۵ کاراکتر است.
  - یعنی ۲۰ = ۵ * ۴ ، پس ۲۰ بایت در هر کوئری موردنیاز است.
- برای هر کاراکتری که در کادر جستجو وارد می‌شود، یک کاربر درخواستی را برای پیشنهادها autocomplete را به backend ارسال می‌کند. به طور متوسط برای هر عبارت مورد جستجو، ۲۰ درخواست[1] ارسال می‌شود. به‌عنوان‌مثال، ۶ درخواست زیر تا زمانی که تایپ کردن "dinner" را به پایان برسانید به backend ارسال می‌شود.

search?q=d
search?q=di
search?q=din
search?q=dinn
search?q=dinne
search?q=dinner

- $\sim 24{,}000\ query\ per\ second\ (QPS) = 10{,}000{,}000\ users * 10\ queries\ /\ day * 20\ characters\ /24\ hours\ /\ 3600\ seconds.$
- مقدار اوج برابر است با $QPS * 2 = \sim 48000$
- فرض کنید که حدود ۲۰٪ از کوئری‌هایی در هر روز وارد می‌شود، جدید هستند. پس داریم:

۱۰ میلیون × ۱۰ کوئری / روز × ۲۰ بایت در هر کوئری × ۲۰٪ = ۰.۴ گیگابایت.

این بدان معناست که روزانه ۰.۴ گیگابایت داده جدید به فضای ذخیره‌سازی اضافه می‌شود.

1 requests

مرحله ۲ – طراحی سطح پیشرفته

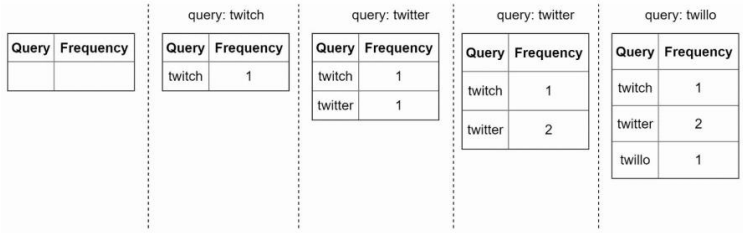
در سطح پیشرفته، سیستم به دو بخش تقسیم می‌شود:

• **سرویس جمع‌آوری داده‌ها**[1]: کوئری‌های ورودی کاربر را جمع‌آوری می‌کند و آنها را در زمان بلادرنگ جمع می‌کند. پردازش بلادرنگ برای مجموعه داده‌های بزرگ عملی نیست. بااین‌حال، نقطه شروع خوبی است. ما یک راه‌حل واقعی‌تر را در قسمت بررسی عمیق در این فصل شرح خواهیم داد.

• **سرویس کوئری**[2]: باتوجه‌به یک عبارت مورد جستجو یا یک پیشوند خاص، ۵ عبارتی یا کوئری که اغلب جستجو شده‌اند را برگردانید.

سرویس جمع‌آوری داده‌ها

اجازه دهید از یک مثال ساده استفاده کنیم تا ببینیم سرویس جمع‌آوری داده چگونه کار می‌کند. فرض کنید یک جدول فراوانی داریم که رشته کوئری و فراوانی آن را همان‌طور که در شکل ۲-۱۳ نشان‌داده‌شده است ذخیره می‌کند. در ابتدا جدول فراوانی خالی است. بعداً، کاربران عبارت‌های "twitch"، "twitter"، "twitter" و "twillo" را به ترتیب وارد می‌کنند. شکل ۲-۱۳ نحوه به‌روزرسانی جدول فراوانی را نشان می‌دهد



شکل ۲-۱۳

فرض کنید یک جدول فراوانی داریم که در جدول ۱-۱۳ نشان‌داده‌شده است و دو فیلد دارد.

• **کوئری**: رشته کوئری را ذخیره می‌کند.

---

1 Data gathering service
2 Query service

• **فراوانی**: نشان‌دهنده تعداد دفعاتی است که یک کوئری جستجو شده است.

Query	Frequency
Twitter	35
twitch	29
twilight	25
twin peak	21
twitch prime	18
twitter search	14
twillo	10
twin peak sf	8

جدول ۱-۱۳

هنگامی که کاربر عبارت "tw" را در کادر جستجو تایپ می‌کند، ۵ عبارت جستجو شده زیر نمایش داده می‌شود (شکل ۳-۱۳)، با فرض اینکه جدول فراوانی بر اساس جدول ۱-۱۳ باشد.

شکل ۳-۱۳

برای دریافت ۵ عبارت برتر که اغلب جستجو می‌شوند، کوئری SQL زیر را اجرا کنید:

```
SELECT * FROM frequency_table
WHERE query Like `prefix%`
ORDER BY frequency DESC
LIMIT 5
```

شکل ۴-۱۳

این یک راه‌حل قابل‌قبول است که مجموعه‌داده‌ها[1] کوچک باشد. وقتی مجموعه‌داده‌ها بزرگ باشد، دسترسی به پایگاه‌داده به یک گلوگاه تبدیل می‌شود. ما بهینه‌سازی‌ها را در قسمت عمیق‌شدن در طراحی شرح خواهیم کرد.

مرحله سوم - عمیق‌شدن در طراحی

در طراحی سطح پیشرفته، سرویس جمع‌آوری داده و سرویس کوئری را مورد بحث قرار دادیم. البته این طراحی در سطح پیشرفته بهینه نیست، اما به‌عنوان یک نقطه شروع خوب عمل می‌کند. در این بخش، ما به چند مؤلفه عمیق می‌پردازیم و بهینه‌سازی‌ها را به شرح زیر بررسی می‌کنیم:

- ساختار داده تِرای/درخت پیشوندی[2]
- سرویس جمع‌آوری داده‌ها
- سرویس کوئری
- مقیاس‌دهی[3] کردن storage
- عملیات trie

درخت پیشوندی ساختار داده

پایگاه‌داده‌های رابطه‌ای برای ذخیره‌سازی در طراحی سطح پیشرفته استفاده می‌شوند. بااین‌حال، واکشی ۵ عبارت جستجوی برتر از یک پایگاه‌داده رابطه‌ای ناکارآمد است. ساختار

---

1 data set

۲ در علم رایانه (trie) ترای یا درخت پیشوندی یک داده‌ساختار درختی است که برای آرایه‌های شرکت‌پذیری استفاده می‌شود که کلیدهای آن معمولاً از نوع رشته‌ای می‌باشند. برخلاف یک درخت دودویی جستجو در این درخت هیچ یک از گره‌ها، کلیدی را که توسط آن گره مشخص می‌شود را ذخیره نمی‌کند؛ در عوض، موقعیت آن در درخت نشان دهنده کلید مربوط به آن است. تمام فرزندان یک گره پیشوند مشترکی دارند که این پیشوند در گره مربوطه ذخیره می‌شود. گره ریشه نیز یک رشته خالی است. معمولاً همه گره‌ها مشخص‌کننده کلیدها نیستند. فقط برگ‌ها و بعضی از گره‌های داخلی با کلیدها مرتبط می‌شوند. گره‌های حاوی کلید به نحوی علامت‌گذاری می‌شوند تا تمام کلیدها مشخص شوند. البته یک تِرای الزاماً شامل رشته‌های کاراکتری نمی‌باشد، بلکه حتی برای جایگشتهای عددی و مواردی از این قبیل هم می‌توان از تِرای استفاده کرد.

3 Scale

داده trie[1] برای غلبه بر این مشکل استفاده می‌شود. از آنجایی که ساختار داده trie برای سیستم بسیار مهم است، ما زمان قابل توجهی را برای طراحی یک آزمایش سفارشی اختصاص خواهیم داد. لطفا توجه داشته باشید که برخی از ایده‌ها از مقالات [۲] و [۳] است. درک مقدمات ساختار داده‌های trie برای این سؤال مصاحبه ضروری است. بااین‌حال، این بیشتر یک سؤال ساختار داده است تا یک سؤال طراحی سیستم. علاوه بر این، بسیاری از مطالب آنلاین این مفهوم را توضیح می‌دهند. در این فصل، ما فقط یک نمای کلی از ساختار داده‌های trie را مورد بحث قرار می‌دهیم و بر چگونگی بهینه‌سازی یک trie مقدماتی برای بهبود زمان پاسخ تمرکز می‌کنیم.

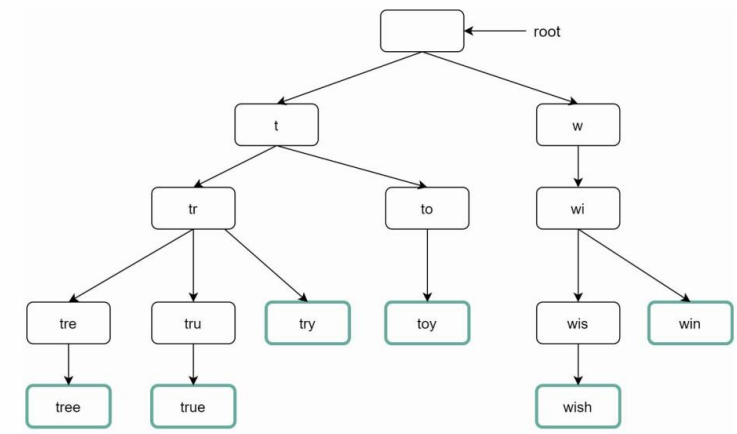
Trie (تلفظ "try") یک ساختار داده درخت‌مانند است که می‌تواند رشته‌ها را به طور فشرده ذخیره کند. این نام از کلمه retrieval[2] گرفته شده است که نشان می‌دهد برای عملیات بازیابی رشته طراحی شده است. ایده اصلی trie شامل موارد زیر است:

- trie یک ساختار داده درخت‌مانند است.
- ریشه؛ یک رشته خالی را نشان می‌دهد.
- هر گره یک کاراکتر را ذخیره می‌کند و دارای ۲۶ فرزند است، یک فرزند برای هر کاراکتر موجود. برای صرفه جویی در فضا، پیوندهای خالی را رسم نمی‌کنیم.
- هر گره درختی یک کلمه یا یک رشته پیشوند را نشان می‌دهد.

شکل ۵-۱۳ تلاشی را با عبارات جستجوی "tree"، "try"، "true"، "toy"، "wish"، "win" را نشان می‌دهد. عبارت‌های جستجو با حاشیه ضخیم‌تری مشخص می‌شوند.

---

1 درخت پیشوند
2 بازیابی



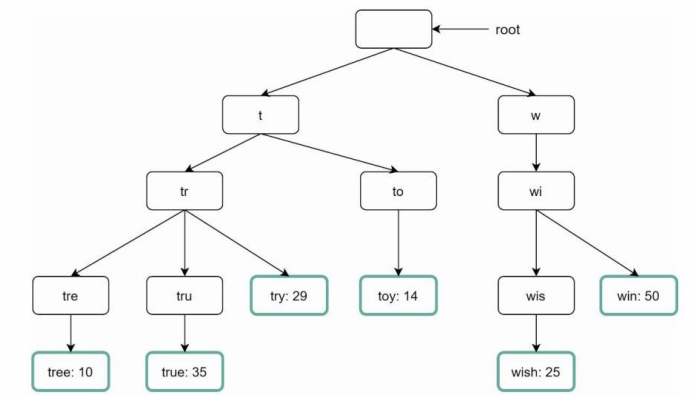
شکل ۵-۱۳

ساختار داده اولیه trie کاراکترها را در گره‌ها ذخیره می‌کند. برای پشتیبانی از مرتب‌سازی بر اساس فراوانی آن‌ها، اطلاعات فراوانی باید در گره‌ها گنجانده شود. فرض کنید جدول فراوانی زیر را داریم.

Query	Frequency
tree	10
try	29
true	35
toy	14
wish	25
win	50

جدول ۲-۱۳

پس از ذخیره اطلاعات فراوانی به گره‌ها، ساختار داده trie به‌روزرسانی شده در شکل۶-۱۳ نشان‌داده‌شده است.



شکل ۶-۱۳

چگونه تکمیل‌کننده خودکار[1] با trie کار می‌کند؟ قبل از بررسی این الگوریتم، اجازه دهید چند اصطلاح را تعریف کنیم.

- p: طول یک پیشوند[2]
- n: تعداد کل گره‌ها در یک trie
- c: تعداد فرزندان یک گره مشخص

مراحل به‌دست‌آوردن بیشترین عبارت‌های جستجو شده در زیر ذکر شده است:

۱. پیشوند را پیدا کنید. پیچیدگی زمانی برابر است با ***O(p)***.

۲. زیردرخت را از گره پیشوند عبور دهید تا همه فرزندان معتبر را دریافت کنید. یک فرزند در صورتی معتبر است که بتواند یک رشته کوئری معتبر تشکیل دهد. پیچیدگی زمانی در این حالت برابر است با: ***O(c)***

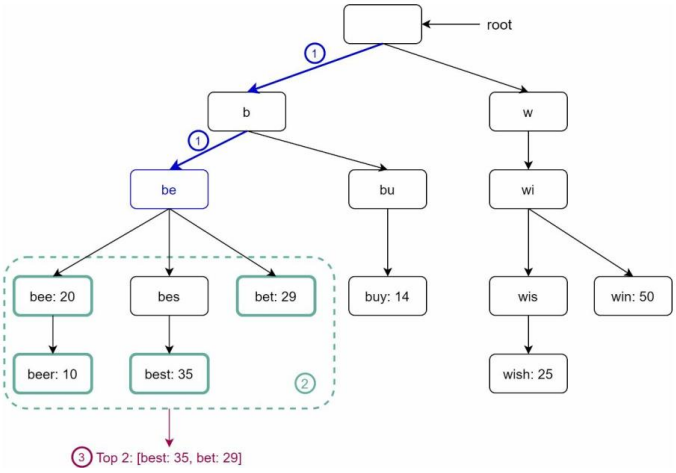
۳. فرزندان این درخت را مرتب[3] کنید و مقادیر k برتر را دریافت کنید. پیچیدگی زمانی برابر است با: ***O(clogc)***

---

1 autocomplete
2 prefix
3 Sort

اجازه دهید از مثالی همان‌طور که در شکل ۷-۱۳ نشان‌داده‌شده است برای توضیح الگوریتم استفاده کنیم. فرض کنید k برابر با ۲ باشد و کاربر عبارت"tr" را در کادر جستجو تایپ کند. الگوریتم به‌صورت زیر کار میکند:

- مرحله ۱: گره پیشوند "tr" را پیدا کنید.
- مرحله ۲: از زیر درخت عبور کنید تا همه فرزندان معتبر را بدست آورید. در این مورد، گره‌ها [tree: 10], [true:35], [try: 29] معتبر هستند.
- مرحله ۳: فرزندان این درخت را مرتب کنید و ۲ مورد برتر را دریافت کنید. [true: 35] و [try: 29] دو عبارت برتر با پیشوند "tr" هستند.



شکل ۷-۱۳

پیچیدگی زمانی این الگوریتم برابر با مجموع زمان صرف شده برای هر مرحله ذکر شده در بالا است، یعنی برابر مقدار زیر:

$$\boldsymbol{O(p) + O(c) + O(clogc)}$$

الگوریتم فوق، بسیار ساده است. بااین‌حال، سرعت آن بسیار آهسته است؛ زیرا ما باید همه درخت پیوندی/trie را پیمایش کنیم تا k نتایج برتر را در بدترین سناریو به دست بیاوریم. در زیر دو بهینه‌سازی وجود دارد:

۱. حداکثر طول یک پیشوند را محدود کنید.
۲. عبارت‌هایی که جستجوی بالایی دارند را در هر گره ذخیره کنید.

اجازه دهید این بهینه‌سازی‌ها را یکی‌یکی بررسی کنیم.

محدودکردن طول یک پیشوند

کاربران به‌ندرت یک عبارت جستجوی طولانی را در کادر جستجو تایپ می‌کنند؛ بنابراین، می‌توان گفت p یک عدد صحیح کوچک است، مثلاً ۵۰. اگر طول یک پیشوند را محدود کنیم، پیچیدگی زمانی «پیدا کردن پیشوند[1]» را می‌توان از O(p) به O(1) کاهش داد.

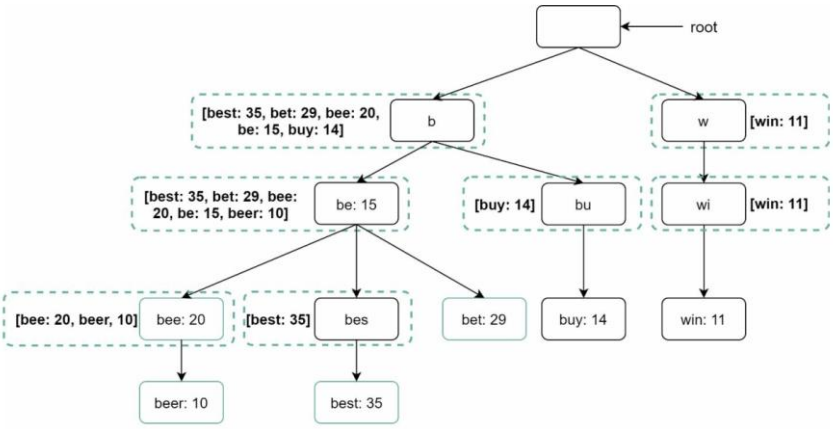
ذخیره عبارت‌هایی با جستجوی بالا در هر گره

برای جلوگیری از پیمایش کل درخت پیشوندی[2]، ما k کوئری پرکاربرد را در هر گره ذخیره می‌کنیم. از آنجایی که ۵ تا ۱۰ پیشنهاد تکمیل کننده خودکار[3] برای کاربران کافی است، در نتیجه k عدد نسبتا کمی است. در مورد خاص ما، فقط ۵ عبارت جستجوی برتر در حافظه موقت/کَش ذخیره می‌شوند.

با ذخیره عبارت‌های جستجوی برتر در هر گره، پیچیدگی زمانی برای بازیابی ۵ عبارت برتر را به میزان قابل توجهی کاهش می‌دهیم. بااین‌حال، این طراحی به فضای زیادی برای ذخیره کوئری‌ها و عبارت‌های برتر در هر گره نیاز دارد. وقتی سرعت پاسخگویی خیلی مهم باشد، صرفه‌جویی در وقت با کم کردن فضا، ارزش بیشتری را دارد.

شکل ۸-۱۳ ساختار داده درخت پیشوندی/trie به‌روزرسانی شده را نشان می‌دهد. که ۵ عبارت برتر در هر گره ذخیره می‌شود. به‌عنوان‌مثال، گره با پیشوند "be" موارد زیر را ذخیره می‌کند: [beer: 10 ،be: 15 ،bee: 20 ،bet: 29 ،best: 35].

1 Find the prefix
2 trie
3 autocomplete



شکل ۸-۱۳

اجازه دهید پس از اعمال این دو بهینه‌سازی، پیچیدگی زمانی الگوریتم را دوباره بررسی کنیم:

۱. گره پیشوند را پیدا کنید. پیچیدگی زمانی برابر O(1) است.

۲. بازگشت به موارد برتر k. از آنجایی که k کوئری/عبارت برتر در حافظه کَش ذخیره می‌شوند، پیچیدگی زمانی این مرحله O(1) است.

از آنجایی که پیچیدگی زمانی برای هر یک از مراحل به O(1) کاهش می‌یابد، الگوریتم ما فقط O(1) را برای واکشی k عبارت برتر نیاز دارد.

سرویس جمع‌آوری داده‌ها

در طراحی قبلی، هر زمان که کاربر یک عبارت جستجو[1] را تایپ می‌کند، داده‌ها در زمان واقعی به‌روزرسانی می‌شوند. این رویکرد به دو دلیل زیر عملی نیست:

- کاربران ممکن است میلیاردها عبارت و کوئری در روز وارد کنند. به‌روزرسانی درخت پیشوندی در هر کوئری به طور قابل‌توجهی سرعت سرویس کوئری را کاهش می‌دهد.
- پیشنهادها موارد برتر ممکن است پس از ساخته‌شدن درخت پیشوندی تغییر چندانی نکنند؛ بنابراین، به‌روزرسانی مکرر درخت پیشوندی ضروری نیست.

1 search query

برای طراحی یک سرویس جمع‌آوری داده مقیاس‌پذیر، بررسی می‌کنیم که داده‌ها از کجا می‌آیند و چگونه از داده‌ها استفاده می‌شود. یک اپلیکیشن بلادرنگ مانند توییتر به پیشنهادهای autocomplete به‌روزرسانی شده نیاز دارند. بااین‌حال، پیشنهادها autocomplete برای بسیاری از کلمات کلیدی گوگل ممکن است روزانه تغییر چندانی نداشته باشند.

به‌رغم تفاوت‌هایی که در موارد استفاده وجود دارد، زیربنای سرویس جمع‌آوری داده همچنان یکسان باقی می‌ماند، زیرا داده‌هایی که برای ساختن trie استفاده می‌شوند معمولاً از سرویس‌های آنالیز و تحلیل یا سرویس‌های log هستند.

شکل ۹-۱۳ سرویس جمع‌آوری داده‌هایی که بازطراحی شده‌اند را نشان می‌دهد. هر جزء یک به یک بررسی می‌شود.

شکل ۹-۱۳

آنالیز و تحلیل لاگ‌ها. اطلاعات خامی که از کوئری‌های جستجو به‌دست‌آمده را ذخیره می‌کند. لاگ‌ها فقط اضافه می‌شوند و ایندکس نمی‌شوند. جدول ۳-۱۳ نمونه‌ای از فایل لاگ را نشان می‌دهد..

query	time
tree	2019-10-01 22:01:01
try	2019-10-01 22:01:05
tree	2019-10-01 22:01:30
toy	2019-10-01 22:02:22
tree	2019-10-02 22:02:42
try	2019-10-03 22:03:03

جدول ۳-۱۳

تجمیع‌کننده‌ها[1]. اندازه لاگ‌های تحلیلی[2] معمولاً بسیار بزرگ است و داده‌ها در فرمت و قالب مناسبی نیستند. ما باید داده‌ها را جمع‌آوری کنیم تا بتوان به‌راحتی توسط سیستم پردازش شوند. بسته به مورد استفاده، ممکن است داده‌ها را به شیوه متفاوتی جمع‌آوری کنیم. برای برنامه‌های بلادرنگ[3] مانند توییتر، ما داده‌ها را در بازه زمانی کوتاه‌تری جمع‌آوری می‌کنیم؛ زیرا نتیجه‌های بلادرنگ مهم هستند. از سوی دیگر، جمع‌آوری داده‌ها به‌دفعات کمتر، مثلاً یک‌بار در هفته، ممکن است برای بسیاری از موارد استفاده کافی باشد. در طول جلسه مصاحبه، بررسی کنید که آیا نتایج بلادرنگ مهم هستند یا خیر. ما فرض می‌کنیم درخت پیشوندی یا trie به‌صورت هفتگی بازسازی می‌شود.

تجمیع داده‌ها[4]

جدول ۴-۱۳ نمونه‌ای از داده‌های تجمیع شده به‌صورت هفتگی را نشان می‌دهد. فیلد "time" زمان شروع یک هفته را نشان می‌دهد. فیلد "frequency" مجموع دفعات به وقوع پیوسته برای جستجوی مربوطه در آن هفته است.

query	Time	frequency
tree	2019-10-01	12000
tree	2019-10-08	15000
tree	2019-10-15	9000
toy	2019-10-01	8500
toy	2019-10-08	6256
toy	2019-10-15	8866

جدول ۴-۱۳

1 Aggregators
2 analytics logs
3 real-time
4 Aggregated Data

worker‌ها.

کارگرها[1] مجموعه‌ای از سرورها هستند که کارهای ناهم‌زمان را در فواصل زمانی معین انجام می‌دهند. آنها ساختار داده trie را می‌سازند و آن را در Trie DB ذخیره می‌کنند.

Trie Cache . در واقع Trie Cache یک سیستم کَش توزیع شده است که برای خواندن سریع داده‌ها، درخت پیشوندی یا Trie را در حافظه کَش نگه می‌دارد. این کار یک snapshot[2] هفتگی از DB می‌گیرد.

Trie DB. یک Trie DB نوعی ذخیره‌سازی دائمی[3] است. دو گزینه برای ذخیره داده‌ها در دسترس است:

۱. ذخیره اسناد[4]: از آنجایی که یک Trie جدید به‌صورت هفتگی ساخته می‌شود، می‌توانیم به‌صورت دوره‌ای از آن یک snapshot بگیریم، آن را سریال‌سازی کنیم و داده‌های سریال‌سازی شده را در پایگاه‌داده ذخیره کنیم. ذخیره‌سازهای اسناد مانند MongoDB [۴] برای داده‌های سریالی مناسب هستند.

۲. ذخیره کلید-مقدار[5]: با اعمال منطق زیر می‌توان یک Trie را به شکل جدول هش [۴] نشان داد:

- هر پیشوند در Trie به یک کلید در جدول هش نگاشت می‌شود.
- داده‌های هر گره آزمایشی به یک مقدار در جدول هش نگاشت می‌شوند.

شکل ۱۰-۱۳ نگاشت بین جدول Trie و هش را نشان می‌دهد.

---

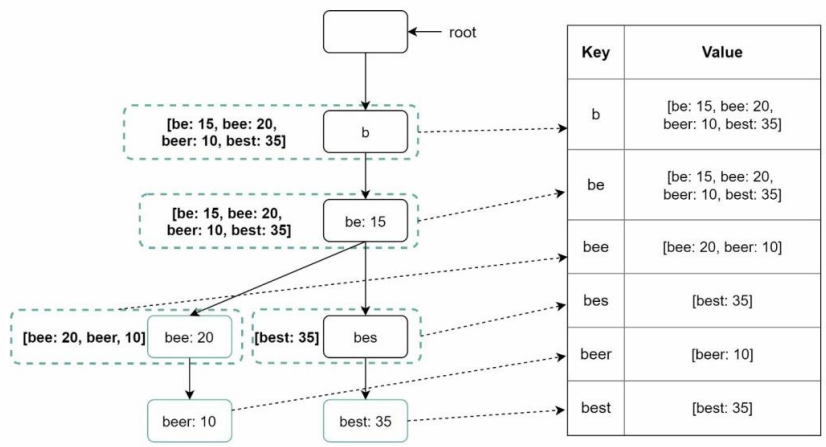
۱ worker‌ها

۲ یک snapshot نقطه‌ای در زمان است که مشخص می‌کند وضعیت دقیق فایل‌ها، پوشه‌ها و سیستم فایل کلی را در آن لحظه نگه می‌دارد.

3 persistent storage

4 Document store

5 Key-value

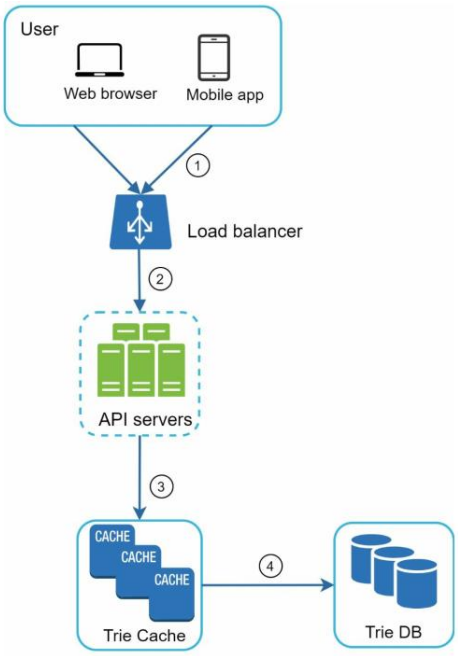


شکل ۱۰-۱۳

در شکل ۱۰-۱۳، هر گره Trie در سمت چپ به یک جفت <key, value> در سمت راست نگاشت شده است. اگر شما یا این موضوع آشنا نیستید که ذخیره‌سازهای با key-value چگونه کار می‌کنند، به فصل ۶ مراجعه کنید که طراحی یک key-value store نام دارد.

Query service

در طراحی سطح پیشرفته، سرویس کوئری مستقیماً پایگاه‌داده را فراخوانی می‌کند تا ۵ نتیجه برتر را دریافت کند. شکل ۱۱-۱۳ طراحی بهبود یافته‌ای را نشان می‌دهد زیرا طراحی قبلی ناکارآمد است.



شکل ۱۱-۱۳

۱. یک کوئری برای متعادل‌کننده بار ارسال می‌شود.

۲. متعادل‌کننده بار[1] درخواست را به سرورهای API هدایت می‌کند.

۳. سرورهای API داده‌های Trie را از Trie Cache دریافت می‌کنند و پیشنهادها تکمیل خودکار[2] را برای کلاینت ایجاد می‌کنند.

۴. اگر داده‌ای در حافظه Trie Cache پیدا نشود، آن داده را دوباره در حافظه کَش قرار می‌دهیم. به‌این‌ترتیب، تمام درخواست‌های بعدی برای همان پیشوند[3] از حافظه کش برگردانده می‌شوند. فقدان کش[4] زمانی اتفاق می‌افتد که سرور کش حافظه کافی نداشته باشد یا به‌صورت آفلاین باشد. در چنین شرایطی، سیستم مجبور است تا داده‌ها را از منبع اصلی که احتمالاً کندتر است (مانند پایگاه‌داده تِرای[5])، بازیابی کند.

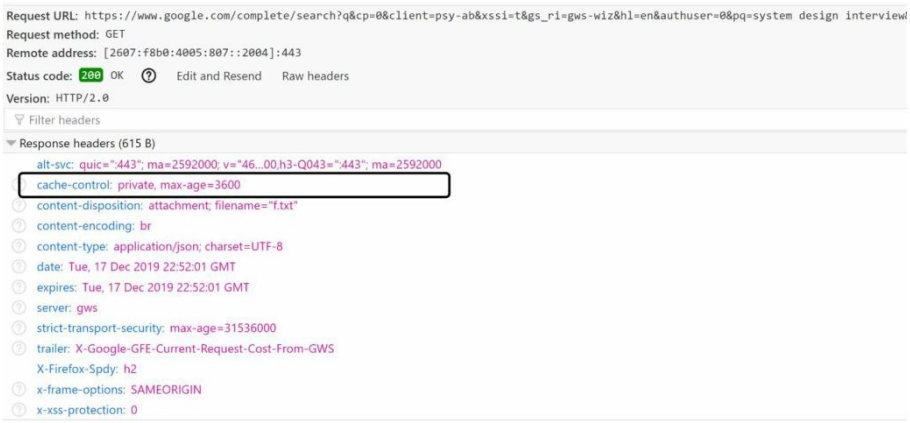
---

1 load balancer
2 autocomplete
3 prefix
4 cache miss
5 Trie DB

سرویس کوئری نیازمندی سرعتی مانند سرعت نور است. درنتیجه ما بهینه‌سازی‌های زیر را پیشنهاد می‌کنیم:

- **درخواست AJAX**. برای برنامه‌های تحت وب، مرورگرها معمولاً درخواست‌های AJAX را برای واکشی نتایج autocomplete ارسال می‌کنند. مزیت اصلی AJAX این است که ارسال/دریافت[1] و درخواست/پاسخ[2] ، تمامی یک صفحه وب در مرورگر را تازه‌سازی[3] نمی‌کند.
- **کَش مرورگر**[4]. برای بسیاری از برنامه‌ها، پیشنهادهای جستجوی عبارت برای autocomplete ممکن است در مدت کوتاهی تغییر چندانی نکنند؛ بنابراین، autocomplete را می‌توان در حافظه کَش مرورگر ذخیره کرد تا به درخواست‌های بعدی اجازه دهد مستقیماً از حافظه کَش نتیجه بگیرند. موتور جستجوی گوگل از همان مکانیزم کَش استفاده می‌کند. شکل ۱۲-۱۳ هدر پاسخ را هنگام تایپ «مصاحبه طراحی سیستم» در موتور جستجوی گوگل نشان می‌دهد. همان‌طور که می‌بینید، گوگل نتایج را به مدت ۱ ساعت در مرورگر و در حافظه کَش آن، ذخیره می‌کند. لطفاً توجه داشته باشید: "private" در کنترل کش به این معنی است که نتایج برای یک کاربر در نظر گرفته شده است و نباید توسط یک کَش مشترک ذخیره شوند. "maxage=3600" به این معنی است که حافظه کَش برای ۳۶۰۰ ثانیه که معادل یک ساعت است، معتبر است.

1 sending/receiving
2 request/response
3 refresh
4 Browser caching



شکل ۱۲-۱۳

نمونه‌برداری از داده‌ها: برای یک سیستمی که مقیاس بزرگی[1] دارد، ثبت و لاگ[2] گرفتن از هر درخواست یا عبارت مورد جستجو به قدرت پردازشی و ذخیره‌سازی زیادی نیاز دارد. در نتیجه نمونه‌برداری از داده‌ها مهم است. به‌عنوان‌مثال، از هر N درخواست فقط ۱ مورد توسط سیستم ثبت و لاگ می‌شود.

عـمـلـیـات Trie

Trie جزء اصلی سیستم autocomplete است. اجازه دهید به نحوه عملکرد عملیات Trie (ایجاد، به‌روزرسانی و حذف)[3] نگاه کنیم.

ایجاد – Create

Trie توسط workerها و با استفاده از داده‌های تجمیع شده[4] ایجاد می‌شود. منبع داده‌ها از تحلیل و آنالیز مربوط به Log/DB است.

به‌روزرسانی – Update

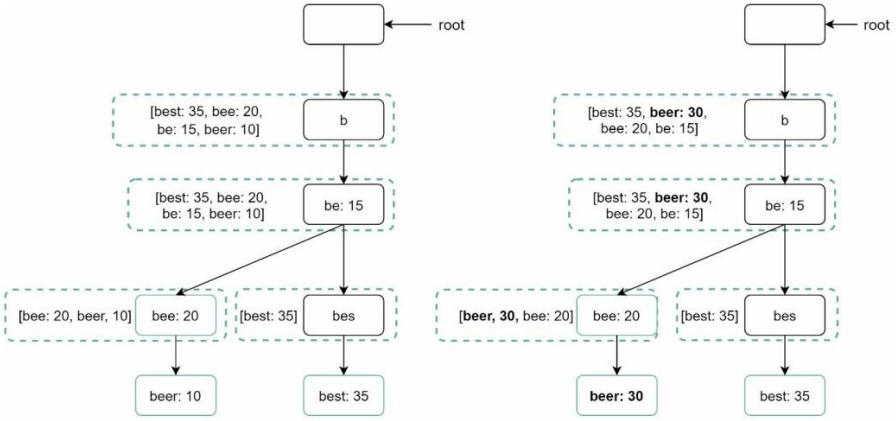
دوراه برای به‌روزرسانی Trie وجود دارد.

---

1 large-scale
2 logging
3 create, update, delete
4 aggregated data

گزینه ۱: درخت پیوندی/Trie را به‌صورت هفتگی به‌روزرسانی کنید. هنگامی که یک Trie جدید ایجاد می‌شود، Trie جدید جایگزین Trie قدیمی می‌شود.

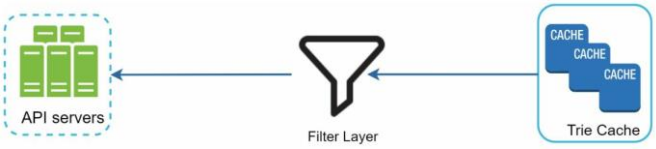
گزینه ۲: گره Trie جداگانه را مستقیماً به‌روزرسانی کنید. ما سعی می‌کنیم از این عملیات اجتناب کنیم زیرا تا حدودی آهسته است. بااین‌حال، اگر اندازه Trie کوچک باشد، راه‌حل قابل قبولی است. وقتی یک گره trie را به‌روزرسانی می‌کنیم، والدین آن تا ریشه باید به‌روزرسانی شوند، زیرا والدین، کوئری‌های برتر و اصلی فرزندان خود را ذخیره می‌کنند. شکل ۱۳-۱۳ نمونه‌ای از نحوه عملکرد عملیات به‌روزرسانی را نشان می‌دهد. در سمت چپ، عبارت جستجوی "Beer" دارای مقدار اصلی ۱۰ است. در سمت راست، به ۳۰ به‌روزرسانی می‌شود. همان‌طور که می‌بینید، گره و والدین آن دارای مقدار "Beer" به ۳۰ به‌روزرسانی شده‌اند.



شکل ۱۳-۱۳

حذف یا Delete

ما باید پیشنهادهای autocomplete تنفرآمیز،خشونت‌آمیز، خطرناک را حذف کنیم. یک لایه فیلتر مانند (شکل ۱۴-۱۳) در جلوی Trie Cache اضافه می‌کنیم تا پیشنهادهای ناخواسته را فیلتر کنیم. داشتن یک لایه فیلتر به ما انعطاف‌پذیری حذف نتایج را بر اساس قوانین فیلتر مختلف می‌دهد. پیشنهادهای ناخواسته به طور فیزیکی از پایگاه‌داده به‌صورت ناهم‌زمان حذف می‌شوند، بنابراین مجموعه داده‌های صحیح برای ساخت Trie در چرخه به‌روزرسانی بعدی استفاده می‌شود.



شکل ۱۴-۱۳

مقیاس‌دهی کردن storage

اکنون که سیستمی را برای ارائه عبارت‌ها و کوئری‌های autocomplete به کاربران، توسعه داده‌ایم. زمان آن رسیده است که مشکل مقیاس‌پذیری[1] را حل کنیم، زمانی که trie آن‌قدر بزرگ می‌شود که در یک سرور قرار نمی‌گیرد.

از آنجایی که انگلیسی تنها زبان پشتیبانی شده است، یک راه ساده برای شارد کردن[2] بر اساس کاراکتر اول است. در اینجا چند نمونه آورده شده است.

- اگر به دو سرور برای ذخیره‌سازی نیاز داشته باشیم، می‌توانیم عبارت‌هایی را که با حرف "a" شروع می‌شود تا حرف "m" در سرور اول و از حرف "n" تا حرف "z" را در سرور دوم ذخیره کنیم.
- اگر به سه سرور نیاز داریم، می‌توانیم عبارت‌ها را از حرف "a" تا حرف "i" در سرور اول و از حرف "j" تا حرف "r" را در سرور دوم و از حرف "s" تا حرف "z" در را سرور سوم تقسیم کنیم.

با پیروی از این منطق، می‌توانیم کوئری و عبارت‌ها را تا ۲۶ سرور تقسیم کنیم، زیرا در زبان انگلیسی ۲۶ کاراکتر الفبایی وجود دارد. اجازه دهید شاردینگ را بر اساس کاراکتر اول به‌عنوان تقسیم‌بندی سطح اول تعریف کنیم. برای ذخیره داده‌های فراتر از ۲۶ سرور، می‌توانیم در سطح دوم یا حتی در سطح سوم شارد کنیم. برای مثال، کوئری‌های داده‌ای که با "a" شروع می‌شوند، می‌توانند به ۴ سرور تقسیم شوند، به‌عنوان‌مثال: "aa-ag","ah-an","ao-au", "av-az".

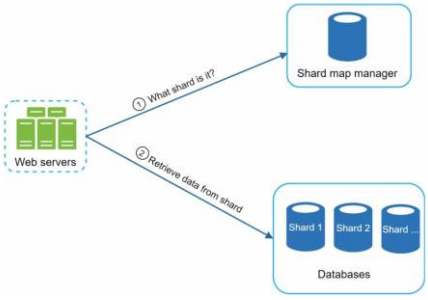
---

1 scalability

۲. Shard – نوعی تقسیم بدی داده‌ها در ذخیره‌سازهای توزیع شده است و در فصل ۱ بیشتر توضیح داده شده است.

در نگاه اول این رویکرد معقول به نظر می‌رسد، تا زمانی که متوجه شوید که تعداد کلماتی که با حرف "c" شروع می‌شوند بسیار بیشتر از "x" هستند. این باعث ایجاد توزیع نابرابر بار روی سرورها می‌شود.

ما الگوی توزیع تاریخچه داده‌ها[1] را آنالیز می‌کنیم و منطق shard بندی هوشمندتری را اعمال می‌کنیم، همان‌طور که در شکل ۱۵-۱۳ نشان‌داده‌شده است. این مدیر نگاشت شارد[2] از یک پایگاه‌داده برای شناسایی اینکه کدام سطرهای داده باید در کدام قسمت نگهداری می‌کند، استفاده می‌کند. برای مثال، اگر تعداد مشابه‌ای از تاریخچه کوئری‌ها و عبارت‌ها برای "s" و برای "z" "y" "x", "w", "v", "u" ترکیب شوند، حالا می‌توانیم دو شارد مختلف را نگهداری کنیم: یک shard برای "s" و دیگری برای "u" تا "z".



شکل ۱۵-۱۳

مرحله ٤ - جمع‌بندی

پس از پایان قسمت بررسی عمیق، مصاحبه‌کننده ممکن است چند سؤال دیگر از شما بپرسد.

مصاحبه‌کننده: چگونه طراحی خود را برای پشتیبانی از چندین زبان گسترش می‌دهید؟

برای پشتیبانی از دیگر عبارت‌ها و کوئری‌های غیرانگلیسی، ما کاراکترهای یونی‌کد را در گره‌های trie ذخیره می‌کنیم. اگر با یونی‌کد[3] آشنا نیستید، یک تعریف ساده اما دقیق از یونی‌کد برابر است با: «استاندارد رمزگذاری همه نویسه‌های تمام سیستم‌های نوشتاری جهان که حتی خط‌های مدرن و باستانی را پوشش می‌دهد» [۵].

---

1 historical data distribution pattern
2 shard map manager
3 Unicode

مصاحبه‌کننده: اگر جستجوهای پربیننده و برتر در یک کشور با کشورهای دیگر متفاوت باشد چه اتفاقی میفتد؟ در این مورد، ممکن است ساختار داده درخت پیشوندی یا trieهای مختلفی برای کشورهای مختلف بسازیم. برای بهبود زمان پاسخ، می‌توانیم trieها را در CDN ذخیره کنیم.

مصاحبه‌کننده: چگونه می‌توانیم از جستجوی عبارات اخیراً مطرح و محبوب شده[1] به صورت بلادرنگ پشتیبانی کنیم؟

با فرض وقوع یک رویداد خبری، یک عبارت جستجو ناگهان محبوب می‌شود. درنتیجه طرح اصلی ما کار نخواهد کرد زیرا:

- Workerهای آفلاین هنوز برای به‌روزرسانی trie برنامه‌ریزی و زمان‌بندی نشده‌اند؛ زیرا این کار به‌صورت هفتگی انجام می‌شود.
- حتی اگر برنامه‌ریزی شده باشد، ساخت trie زمان زیادی می‌برد.

ساختن یک تکمیل خودکار جستجوی بلادرنگ[2] پیچیده است و خارج از محدوده این کتاب است، بنابراین ما فقط چند ایده ارائه می‌دهیم:

- با استفاده از **شاردبندی**، حجم مجموعه داده‌های کاری را کاهش دهید.
- مدل رتبه‌بندی را تغییر دهید و وزن بیشتری را به عبارت‌های مورد جستجوی اخیر اختصاص دهید.
- از آنجایی که داده‌ها ممکن است به‌صورت جریانی[3] وارد شوند، بنابراین ما به همه داده‌ها به طور هم‌زمان دسترسی نداریم. جریان داده به این معنی است که داده‌ها به طور مداوم تولید می‌شوند. پردازش جریان به مجموعه‌ای از سیستم‌های متفاوت نیاز دارد: Apache Hadoop MapReduce [6]، Apache Spark Streaming [7]، Apache Storm [8]، Apache Kafka [۹]، و غیره. از آنجایی که همه آن موضوعات به دانش خاصی نیاز دارند، ما قصد نداریم به جزئیات در اینجا بپردازیم.

---

1 trending
2 real-time search autocomplete
3 stream

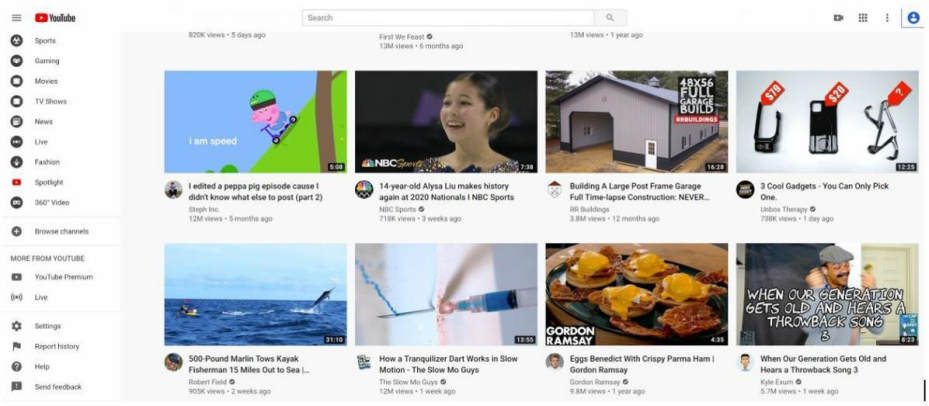
منابع و مراجع فصل ۱۳

[1] The Life of a Typeahead Query: https://www.facebook.com/notes/facebookengineering/the-life-of-a-typeahead-query/389105248919/
[2] How We Built Prefixy: A Scalable Prefix Search Service for Powering Autocomplete: https://medium.com/@prefixyteam/how-we-built-prefixy-a-scalable-prefix-search-servicefor-powering-autocomplete-c20f98e2eff1
[3] Prefix Hash Tree An Indexing Data Structure over Distributed Hash Tables: https://people.eecs.berkeley.edu/~sylvia/papers/pht.pdf
[4] MongoDB wikipedia: https://en.wikipedia.org/wiki/MongoDB
[5] Unicode frequently asked questions: https://www.unicode.org/faq/basic_q.html
[6] Apache hadoop: https://hadoop.apache.org/
[7] Spark streaming: https://spark.apache.org/streaming/
[8] Apache storm: https://storm.apache.org/
[9] Apache kafka: https://kafka.apache.org/documentation/

فصل ۱۴

طراحی یوتیوب

در این فصل از شما خواسته می‌شود تا یوتیوب را طراحی کنید. راه‌حل این سؤال را می‌توان برای سایر سؤالات مصاحبه مانند طراحی یک پلتفرم اشتراک‌گذاری ویدئو مانند aparat و Netflix و Hulu اعمال کرد. شکل ۱-۱۴ صفحه اصلی YouTube را نشان می‌دهد.



شکل ۱-۱۴

یوتیوب ساده به نظر می‌رسد: سازندگان محتوا ویدئوها را آپلود می‌کنند و بینندگان روی پخش کلیک می‌کنند. آیا واقعاً به همین سادگی است؟ نه واقعاً. بسیاری از فن آوری‌های پیچیده در زیر سادگی وجود دارد. اجازه دهید به چند آمار چشمگیر، جمعیت‌شناسی و حقایق سرگرم‌کننده یوتیوب در سال ۲۰۲۰ نگاه کنیم [۱], [۲].

- تعداد کل کاربران فعال ماهانه: ۲ میلیارد.
- تعداد ویدئوهای تماشا شده در روز: ۵ میلیارد.
- ۷۳ درصد از بزرگسالان ایالات متحده از YouTube استفاده می‌کنند.
- ۵۰ میلیون سازنده ویدئو در یوتیوب.
- درآمد تبلیغات YouTube برای کل سال ۲۰۱۹ حدود ۱۵.۱ میلیارد دلار بود که ۳۶ درصد نسبت به سال ۲۰۱۸ افزایش داشت.
- YouTube مسئول ۳۷٪ از کل ترافیک اینترنت تلفن همراه است.
- YouTube به ۸۰ زبان مختلف در دسترس است.

از این آمار، ما می‌دانیم که یوتیوب بسیار بزرگ، جهانی است و پول زیادی به دست می‌آورد.

مرحله ۱ – درک مسئله و شروع به طراحی

همان‌طور که در شکل ۱-۱۴ نشان‌داده‌شده است، علاوه بر تماشای یک ویدئو، می‌توانید کارهای بیشتری را در YouTube انجام دهید. به‌عنوان‌مثال، نظر دهید، به اشتراک بگذارید، یا یک ویدئو را لایک کنید، یک ویدئو را در لیست‌های پخش ذخیره کنید، در یک کانال مشترک شوید و غیره. طراحی همه‌چیز در یک مصاحبه ۴۵ یا ۶۰ دقیقه‌ای غیرممکن است. بنابراین، پرسیدن سؤالات برای محدود کردن دامنه مهم است.

کاندید: چه ویژگی‌هایی مهم هستند؟

مصاحبه‌کننده: امکان آپلود ویدئو و تماشای ویدئو.

کاندید: چه نوع کاربرانی و در چه پلتفرم‌هایی را باید پوشش‌دهی کنیم؟

مصاحبه‌کننده: برنامه‌های موبایل، مرورگرهای وب و تلویزیون هوشمند.

کاندید: روزانه چند کاربر فعال داریم؟

مصاحبه‌کننده: ۵ میلیون.

کاندید: میانگین زمان صرف شده روزانه هر کاربر برای این برنامه چقدر است؟

مصاحبه‌کننده: حدود ۳۰ دقیقه در هر روز.

کاندید: آیا ما نیاز به حمایت از کاربران بین‌المللی داریم؟

مصاحبه‌کننده: بله، درصد زیادی از کاربران، کاربران بین‌المللی هستند.

کاندید: وضوح تصویر پشتیبانی شده چیست؟

مصاحبه‌کننده: سیستم موردنظر باید اکثر وضوح و فرمت‌های ویدئویی را قبول کند.

کاندید: آیا تبدیل فرمت ویدئو لازم است؟

مصاحبه‌کننده: بله

کاندید: آیا اندازه فایل موردنیاز برای ویدئوها وجود دارد؟

مصاحبه‌کننده: پلتفرم ما روی ویدئوهای کوچک و متوسط تمرکز دارد. حداکثر اندازه مجاز ویدئو ۱ گیگابایت است.

کاندید: آیا می‌توانیم برخی از زیرساخت‌های ابری موجود توسط آمازون، گوگل یا مایکروسافت استفاده کنیم؟

مصاحبه‌کننده: این یک سؤال عالی است. ساختن همه‌چیز از ابتدا برای اکثر شرکت‌ها غیرواقعی است، توصیه می‌شود از برخی از سرویس‌های ابری موجود استفاده کنید.

در این فصل، ما بر طراحی یک سرویس پخش ویدئو با ویژگی‌های زیر تمرکز می‌کنیم:

- امکان آپلود سریع فیلم‌ها
- پخش ویدئویی روان
- امکان تغییر کیفیت ویدئو
- هزینه زیرساخت پایین
- الزامات دردسترس‌بودن، مقیاس‌پذیری و قابلیت اطمینان بالا
- کاربران پشتیبانی شده: برنامه‌های تلفن همراه، مرورگر وب و تلویزیون هوشمند.

تخمین و برآورد

برآوردهای زیر بر اساس بسیاری از مفروضات است، بنابراین مهم است که با مصاحبه‌کننده ارتباط برقرار کنید تا مطمئن شوید که او در همان مسیر مورد نظر است.

فرض کنید محصول ۵ میلیون کاربر فعال روزانه (DAU) دارد.

کاربران روزانه ۵ ویدئو تماشا می‌کنند.

- ۱۰ درصد از کاربران روزانه ۱ ویدئو آپلود می‌کنند.
- فرض کنید حجم متوسط ویدئو ۳۰۰ مگابایت است.
- کل فضای ذخیره‌سازی روزانه مورد نیاز:

ترابایت ۱۵۰ = مگابایت ۳۰۰ × ٪۱۰ × میلیون ۵

- هزینه CDN.
- هنگامی که CDN ابری یک ویدئو را ارائه می‌دهد، برای داده‌های منتقل شده به خارج از CDN هزینه دریافت می‌کنید.
- اجازه دهید از CDN CloudFront آمازون برای تخمین هزینه استفاده کنیم (شکل ۲-۱۴) [۳]. فرض کنید ٪۱۰۰ ترافیک از ایالات متحده ارائه می‌شود. میانگین هزینه هر گیگابایت ۰.۰۲ دلار است. برای سادگی، ما فقط هزینه پخش ویدئو را محاسبه می‌کنیم.
- ۱۵۰۰۰۰ دلار در روز = ۵ میلیون × ۵ ویدئو × ۰.۳ گیگابایت × ۰.۰۲ دلار.

از برآورد هزینه تقریبی، می‌دانیم که ارائه ویدئوها از CDN هزینه زیادی دارد. حتی اگر ارائه‌دهندگان ابری تمایل داشته باشند هزینه‌های CDN را به میزان قابل‌توجهی برای مشتریان بزرگ کاهش دهند، این هزینه همچنان قابل‌توجه است. ما راه‌هایی برای کاهش هزینه‌های CDN در بررسی عمیق را مورد بحث قرار خواهیم داد.

Per Month	United States & Canada	Europe & Israel	South Africa & Middle East	South America	Japan	Australia	Singapore, South Korea, Taiwan, Hong Kong, & Philippines	India
First 10TB	$0.085	$0.085	$0.110	$0.110	$0.114	$0.114	$0.140	$0.170
Next 40TB	$0.080	$0.080	$0.105	$0.105	$0.089	$0.098	$0.135	$0.130
Next 100TB	$0.060	$0.060	$0.090	$0.090	$0.086	$0.094	$0.120	$0.110
Next 350TB	$0.040	$0.040	$0.080	$0.080	$0.084	$0.092	$0.100	$0.100
Next 524TB	$0.030	$0.030	$0.060	$0.060	$0.080	$0.090	$0.080	$0.100
Next 4PB	$0.025	$0.025	$0.050	$0.050	$0.070	$0.085	$0.070	$0.100
Over 5PB	$0.020	$0.020	$0.040	$0.040	$0.060	$0.080	$0.060	$0.100

شکل ۲-۱۴- قیمت گذاری بر اساس تقاضا برای انتقال داده به اینترنت (در هر گیگابایت).

مرحله ۲ – طراحی سطح پیشرفته

همان‌طور که قبلاً بحث شد، مصاحبه‌کننده به‌جای ساختن همه‌چیز از ابتدا، استفاده از سرویس‌های ابری موجود را توصیه کرد. CDN و ذخیره‌ساز حبابی[۱]، سرویس‌های ابری ویژه‌ای هستند که ما از آنها استفاده خواهیم کرد. برخی از خوانندگان ممکن است بپرسند چرا همه‌چیز را خودمان نسازیم؟ دلایل در زیر ذکر شده است:

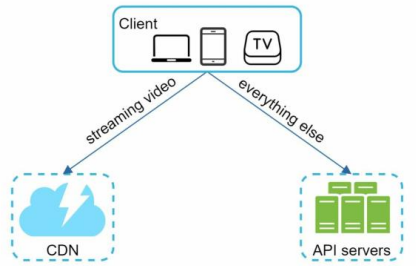
- مصاحبه‌های طراحی سیستم در مورد ساختن همه‌چیز از ابتدا نیست. در چارچوب زمانی محدود، انتخاب فناوری مناسب برای انجام یک کار به‌درستی مهم‌تر از توضیح نحوه عملکرد این فناوری با جزئیات است. به‌عنوان‌مثال، ذکر فضای ذخیره‌ساز حبابی برای ذخیره ویدئوهای منبع برای مصاحبه کافی است. صحبت در مورد طراحی دقیق برای ذخیره‌ساز حبابی می‌تواند بیش از حد باشد.

---

۱ Blob Storage یک راه‌حل ذخیره‌سازی برای داده‌های بدون ساختار (بلاک‌ها) است. اصطلاح "Blob" مخفف Binary Large Object است و به مجموعه‌ای از داده‌های دودویی اشاره دارد که به هیچ فرمت فایل خاصی تطابق ندارند Blob Storage این مجموعه‌های داده را در مناطق ذخیره‌سازی غیر سلسله‌مراتبی به نام "دیتا لیک" نگه می‌دارد. در واقع، Blob Storage داده‌های بلاک را به عنوان اشیاء در یک مخزن داده مانند دریاچه‌ای مسطح نگه می‌دارد. این نوع ذخیره‌سازی مناسب برای داده‌های بدون ساختار است که در راه‌های ذخیره‌سازی سنتی مانند ذخیره‌سازی فایل یا ذخیره‌سازی بلاک جا نمی‌شوند. با استفاده از Blob Storage، می‌توانیم حجم داده‌های تقریباً نامحدودی را ذخیره کنیم. این بلاک‌ها در یک ابر اجرا می‌شوند که توسط شرکت‌های ارائه‌دهنده خدمات ابری مدیریت می‌شود. به عنوان مثال، Azure Blob Storage یکی از معروف‌ترین سرویس‌های ابری در این زمینه است.

- ایجاد ذخیره‌سازی حباب مقیاس‌پذیر یا CDN بسیار پیچیده و پرهزینه است. حتی شرکت‌های بزرگی مانند نتفلیکس یا فیس‌بوک همه‌چیز را خودشان نمی‌سازند. نتفلیکس از خدمات ابری آمازون [۴] استفاده می‌کند و فیس بوک از CDN Akamai [۵] استفاده می‌کند.

در طراحی سطح بالا، سیستم شامل سه جزء است (شکل ۳-۱۴).



شکل ۳-۱۴

کلاینت: می‌توانید YouTube را در رایانه، تلفن همراه و تلویزیون هوشمند خود تماشا کنید.

CDN: فیلم‌ها در CDN ذخیره می‌شوند. وقتی دکمه پخش را فشار می‌دهید، یک ویدئو از CDN پخش می‌شود.

سرورهای API: همه‌چیز به جز پخش ویدئو از طریق سرورهای API انجام می‌شود. این شامل پیشنهادهای ارائه شده در timeline و feed، ایجاد URL آپلود ویدئو، به‌روزرسانی metadata پایگاه‌داده و حافظه کَش، ثبت‌نام کاربر و غیره است.

در جلسه پرسش/پاسخ، مصاحبه‌گر در دو جریان علاقه نشان داد:

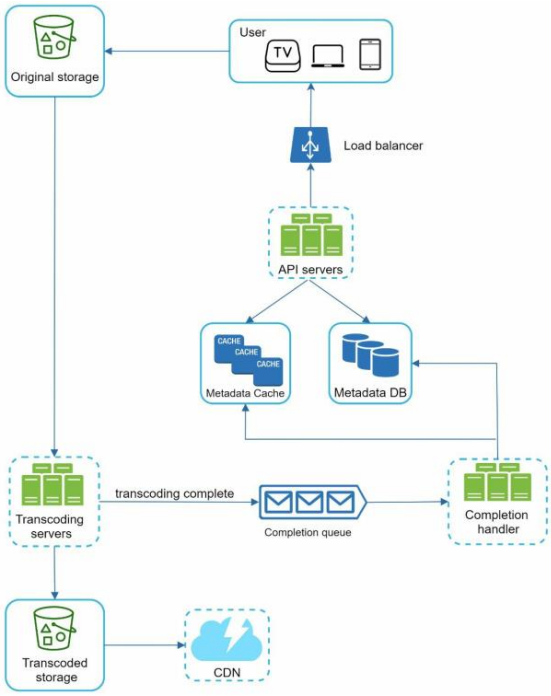
- مسئله آپلود ویدئو
- مسئله پخش[1] ویدئو

ما طراحی سطح بالا برای هر یک از آنها را بررسی خواهیم کرد.

1 streaming

مسیر آپلود ویدئو

شکل ۴-۱۴ طراحی سطح بالا برای آپلود ویدئو را نشان می‌دهد.



شکل ۴-۱۴

این مورد از اجزای زیر تشکیل شده است:

- **کاربر**: کاربر YouTube را در دستگاه‌هایی مانند رایانه، تلفن همراه یا تلویزیون هوشمند تماشا می‌کند.
- **متعادل‌کننده بار**[1]: یک متعادل‌کننده بار درخواست‌ها را به طور مساوی بین سرورهای API توزیع می‌کند.
- **سرورهای API**: تمام درخواست‌های کاربر از طریق سرورهای API به جز پخش ویدئو انجام می‌شود.

1 Load balancer

- **Metadata D** : ابرداده‌های ویدئویی در Metadata DB ذخیره می‌شوند. برای برآورده‌کردن عملکرد و الزامات دردسترس‌بودن بالا[1]، داده‌ها چندتکه یا در اصطلاح shard و تکثیر[2] شده است.
- **کَش Metadata:** برای عملکرد بهتر، ابرداده یا Metadataهای ویدئویی و user object ها در حافظه کَش[3] ذخیره می‌شوند.
- **ذخیره‌سازی اصلی**: یک سیستم ذخیره‌سازی حبابی برای ذخیره ویدئوهای اصلی استفاده می‌شود. نقل‌قول در ویکی‌پدیا در مورد ذخیره‌ساز حبابی نشان می‌دهد که: «Binary Large Object به‌اختصار (BLOB) که یک شیء بزرگ باینری شامل مجموعه‌ای از داده‌های باینری است که به‌عنوان یک موجودیت واحد در یک سیستم مدیریت پایگاه‌داده ذخیره می‌شود» [۶].
- **سرورهای Transcoding**: تبدیل فرمت ویدئو را transcoding ویدئو نیز می‌گویند. این فرایند تبدیل فرمت ویدئویی به فرمت‌های دیگر (MPEG، HLS و غیره) است که بهترین پخش‌های ویدئویی[4] ممکن را برای دستگاه‌های مختلف و قابلیت‌های پهنای باند ارائه می‌دهد.
- **ذخیره‌سازی Transcoded**: یک ذخیره‌ساز حبابی است که فایل‌های ویدئویی تبدیل فرمت شده را ذخیره می‌کند.
- **CDN** : ویدئوها در CDN ذخیره می‌شوند. وقتی روی دکمه پخش کلیک می‌کنید، یک ویدئو از CDN پخش می‌شود.
- **صف تکمیل**[5]: یک صف پیام[6] است که اطلاعات مربوط به رویدادهای تکمیل تبدیل فرمت ویدئو را ذخیره می‌کند.

---

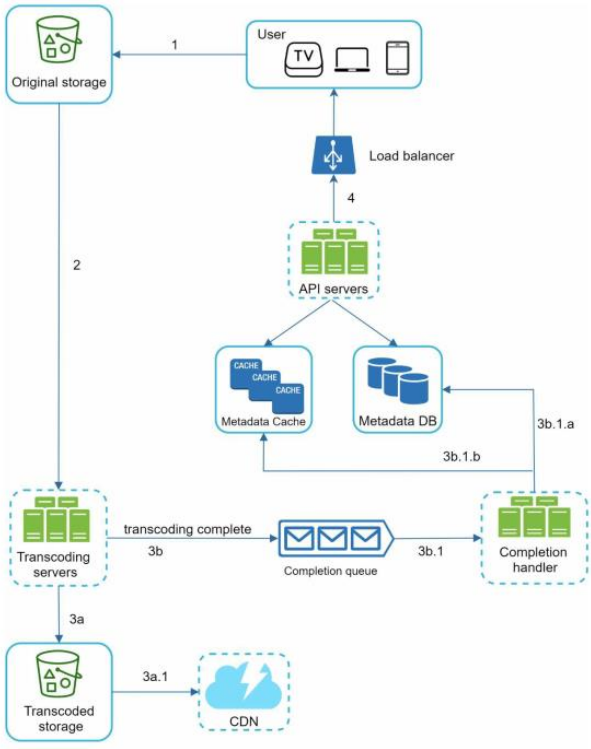
1 high availability
2 replicated
3 cache
4 video streams
5 Completion queue
6 message queue

- **کنترلر تکمیل**[1]: این شامل لیستی از workerهاست که داده‌های رویداد را از «صف تکمیل» می‌کِشند و کَش metadata و پایگاه‌داده را به‌روزرسانی می‌کنند.

اکنون که هر جزء را به‌صورت جداگانه درک می‌کنیم، اجازه دهید نحوه عملکرد مسیر آپلود ویدئو را بررسی کنیم. این مسئله به دو فرایند در حال اجرا به‌صورت موازی تقسیم می‌شود.

- الف. ویدئوی واقعی را آپلود کنید.
- ب. به‌روزرسانی متادیتای ویدئو؛ متادیتا حاوی اطلاعاتی درباره URL ویدئو، اندازه، وضوح، قالب، اطلاعات کاربر و غیره است.

مسیر a: آپلود ویدئوی اصلی



شکل ۵-۱۴

1 Completion handler

شکل ۵-۱۴ نحوه آپلود ویدئوی واقعی را نشان می‌دهد. توضیح در زیر نشان‌داده‌شده است:

۱. فیلم‌ها در حافظه اصلی آپلود می‌شوند.

۲. سرورهای Transcoding، تبدیل فرمت ویدئوها را از حافظه اصلی واکشی می‌کنند و تبدیل فرمت را شروع می‌کنند.

۳. پس از تکمیل تبدیل فرمت، دو مرحله زیر به‌صورت موازی اجرا می‌شوند:

۳ الف. ویدئوهای تبدیل فرمت شده به فضای ذخیره‌سازی مخصوص تبدیل فرمت[1] ارسال می‌شوند.

۳. ب. رویدادهای تکمیل فرایند «تبدیل فرمت» در صف تکمیل[2] قرار می‌گیرند.

۳. الف.۱. ویدئوهای تغییر فرمت داده شده در CDN توزیع می‌شوند.

۳. ب.۱. کنترلر تکمیل[3] شامل دسته‌ای از workerها است که به طور مداوم داده‌های هر رویداد را از صف بیرون می‌کِشند.

۳. ب.۱.الف. و ۳.ب.۱.ب؛ پس از تکمیل تبدیل فرمت ویدئو، کنترلر تکمیل، پایگاه‌داده متادیتا و حافظه cache را به‌روزرسانی می‌کند.

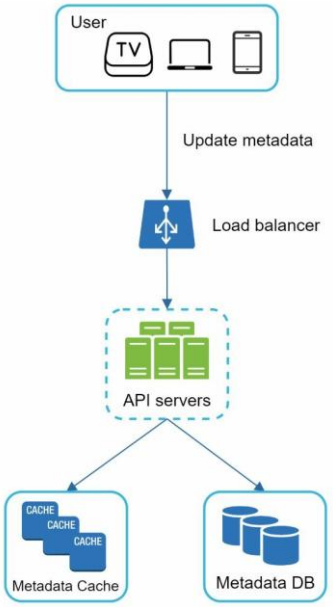
۴. سرورهای API به کاربر اطلاع می‌دهند که ویدئو با موفقیت آپلود شده و آماده پخش است.

مسیر b: به‌روزرسانی متادیتا

درحالی‌که یک فایل در حافظه اصلی آپلود می‌شود، سرویس گیرنده به طور موازی درخواستی برای به‌روزرسانی metadata فایل ویدئویی را همان‌طور که در شکل ۶-۱۴ نشان‌داده‌شده است، ارسال می‌کند. این درخواست حاوی metadataهای فایل ویدئویی، از جمله نام فایل، اندازه، قالب و غیره است. سرورهای API حافظه cache و پایگاه‌داده metadata را به‌روزرسانی می‌کنند.

---

1 transcoded storage
2 Completion queue
3 Completion handler



شکل ۶-۱۴

مسیر پخش ویدئو

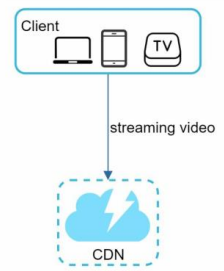
هر وقت ویدئویی را در یوتیوب تماشا می‌کنید، معمولاً بلافاصله پخش ویدئو[1] را شروع می‌کند و منتظر نمی‌مانید تا کل ویدئو دانلود شود. دانلود به این معنی است که کل ویدئو در دستگاه شما کپی می‌شود، درحالی‌که streaming به این معنی است که دستگاه شما به طور مداوم جریان‌های (streams) ویدئویی را از ویدئوهای منبع دریافت می‌کند. هنگامی که ویدئوهای در حال پخش را تماشا می‌کنید، کاربر شما در هر زمان کمی داده بارگیری می‌کند تا بتوانید بلافاصله و به طور مداوم ویدئوها را تماشا کنید. قبل از بحث درباره streaming ویدئو، اجازه دهید به یک مفهوم مهم نگاه کنیم: پروتکل streaming. این یک روش استاندارد برای کنترل انتقال داده برای پخش ویدئو است. پروتکل‌های پخش محبوب عبارت‌اند از:

- به‌عنوان‌مثال MPEG–DASH و MPEG[2] و DASH[3] است.

1 streaming
2 Moving Picture Experts Group
3 Dynamic Adaptive Streaming over HTTP

- اپل HLS[1] است.
- مایکروسافت Smooth Streaming.
- Adobe HTTP Dynamic Streaming (HDS) .

شما نیازی به درک کامل یا حتی به‌خاطر سپردن نام‌های پروتکل streaming ندارید؛ زیرا جزئیات سطح پایینی هستند که به دانش و پیش‌زمینه خاصی نیاز دارند. نکته مهم در اینجا این است که درک کنیم که پروتکل‌های پخش مختلف از کدگذاری‌های ویدئویی و پخش‌کننده‌های مختلف پشتیبانی می‌کنند. هنگامی که ما یک سرویس پخش ویدئو را طراحی می‌کنیم، باید پروتکل پخش مناسب را برای پشتیبانی از موارد استفاده خود انتخاب کنیم. برای کسب اطلاعات بیشتر در مورد پروتکل‌های streaming، در اینجا یک مقاله عالی موجود است [۷]. ویدئوها مستقیماً از CDN پخش می‌شوند. سرور لبه نزدیک به شما ویدئو را تحویل می‌دهد. بنابراین، تأخیر بسیار کمی وجود دارد. شکل ۷-۱۴ سطح بالایی از طراحی برای پخش ویدئو را نشان می‌دهد.



شکل ۷-۱۴

مرحله ۳ - عمیق‌شدن در طراحی

در طراحی سطح بالا، کل سیستم به دو بخش تقسیم می‌شود: مسیر آپلود ویدئو و مسیر پخش ویدئو. در این بخش، هر دو مسئله را با بهینه‌سازی‌های مهم اصلاح می‌کنیم و مکانیسم‌های رسیدگی به خطا را معرفی می‌کنیم.

1 HTTP Live Streaming

تبدیل فرمت ویدئو (Video transcoding)

هنگامی که یک ویدئو ضبط می‌کنید، دستگاه (معمولاً یک تلفن یا دوربین) فرمت خاصی را به فایل ویدئویی می‌دهد. اگر می‌خواهید ویدئو به‌راحتی در دستگاه‌های دیگر پخش شود، ویدئو باید با نرخ بیت و فرمت‌های سازگار کدگذاری شود. نرخ بیت سرعتی است که بیت‌ها در طول زمان پردازش می‌شوند. نرخ بیت بالاتر به‌طورکلی به معنای کیفیت ویدئو بالاتر است. پخش ویدئو با نرخ بیت بالا به قدرت پردازش بیشتر و سرعت اینترنت سریع نیاز دارند.

تبدیل فرمت ویدئو به فرمت دیگر به دلایل زیر مهم است:

- ویدئوی خام حجم زیادی از فضای ذخیره‌سازی را مصرف می‌کند. یک ویدئوی یک‌ساعته باکیفیت بالا ضبط شده با سرعت ۶۰ فریم در ثانیه می‌تواند چند صد گیگابایت فضا را اشغال کند.
- بسیاری از دستگاه‌ها و مرورگرها فقط از انواع خاصی از فرمت‌های ویدئویی پشتیبانی می‌کنند؛ بنابراین تبدیل فرمت یک ویدئو به فرمت‌های مختلف به دلایل سازگاری مهم است.
- برای اطمینان از تماشای ویدئوهای باکیفیت بالا و درعین‌حال پخش روان آن، ایده خوبی است که ویدئوهای با وضوح بالاتر را به کاربرانی که پهنای باند شبکه بالایی دارند و ویدئویی با وضوح پایین‌تر را به کاربرانی که پهنای باند پایینی دارند ارائه دهید.
- شرایط شبکه می‌تواند تغییر کند، به‌خصوص در دستگاه‌های تلفن همراه. برای اطمینان از پخش مداوم یک ویدئو، تغییر کیفیت ویدئو به‌صورت خودکار یا دستی بر اساس شرایط شبکه برای تجربه کاربری روان ضروری است.

بسیاری از انواع فرمت‌های قابل‌تبدیل ویدئو در دسترس هستند. بااین‌حال، اکثر آنها شامل دو بخش هستند:

- کانتینر[1]: مانند سبدی است که حاوی فایل ویدئویی، صدا و metadata است. می‌توانید قالب کانتینر را با پسوند فایل، مانند avi، mov یا mp4 تشخیص دهید.

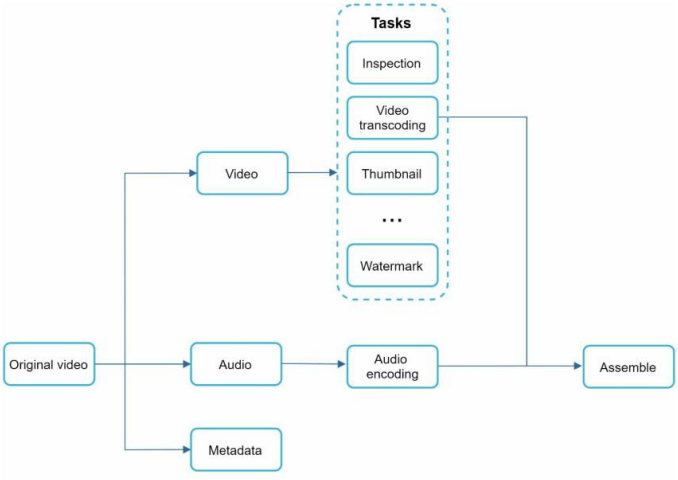
1 Container

- کدک‌ها[1]: اینها الگوریتم‌های فشرده‌سازی و رفع فشرده‌سازی هستند که با هدف کاهش اندازه ویدئو در عین حفظ کیفیت ویدئو انجام می‌شوند. کدک‌های ویدئویی پرکاربرد H.264، VP9 و HEVC هستند.

مدل گراف غیر چرخه‌ای جهت‌دار (DAG).

رمزگذاری یک ویدئو از نظر محاسباتی پرهزینه و زمان‌بر است. علاوه بر این، سازندگان محتوای مختلف ممکن است نیازمندی‌های متفاوتی برای پردازش ویدئو داشته باشند. به‌عنوان‌مثال، برخی از تولیدکنندگان محتوا به واترمارک در بالای ویدئوهای خود نیاز دارند، برخی خودشان تصاویر کوچک ارائه می‌کنند و برخی ویدئوهای باکیفیت بالا را آپلود می‌کنند، درحالی‌که برخی دیگر این کار را نمی‌کنند. برای پشتیبانی از pipeline پردازش ویدئوی مختلف و حفظ موازی‌کاری[2] بالا جهت پردازش آن‌ها، اضافه‌کردن سطحی از انتزاع و اجازه‌دادن به کاربرهای برنامه‌نویس برای تعیین کارهایی که باید اجرا کنند بسیار مهم است. به‌عنوان‌مثال، موتور پخش ویدئوی فیس‌بوک از یک مدل برنامه‌نویسی گراف غیر چرخه‌ای جهت‌دار که DAG[3] نام نامیده می‌شود استفاده می‌کند که taskها را در مراحلی تعریف می‌کند تا بتوان آنها را به‌صورت متوالی یا موازی اجرا کرد [۸]. در طراحی خود، ما یک مدل DAG مشابه را برای دستیابی به انعطاف‌پذیری و موازی پذیری اتخاذ می‌کنیم. شکل ۸-۱۴ یک DAG را برای رمزگذاری ویدئو نشان می‌دهد.

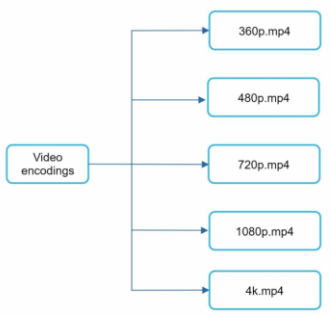
1 Codecs
2 parallelism
3 directed acyclic graph



شکل ۸-۱۴

در شکل ۸-۱۴، ویدئوی اصلی به ویدئو، صدا و metadata تقسیم شده است. در اینجا برخی از وظایفی که می‌توان بر روی یک فایل ویدئویی اعمال کرد آورده شده است:

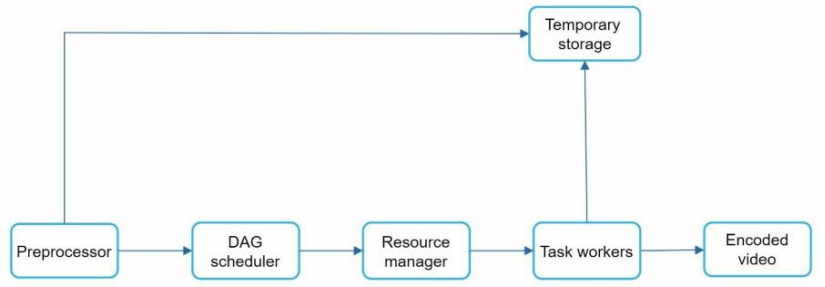
- **بازرسی**: مطمئن شوید که ویدئوها کیفیت خوبی دارند و بدشکل نیستند.
- **تبدیل فرمت‌های ویدئویی**: فیلم‌ها برای پشتیبانی از رزولوشن‌های مختلف، codec، نرخ بیت و غیره تبدیل می‌شوند. شکل ۹-۱۴ نمونه‌ای از فایل‌های کدگذاری شده ویدئویی را نشان می‌دهد.
- **Thumbnail یا بندانگشتی**: Thumbnail می‌توانند توسط کاربر آپلود شوند یا به‌صورت خودکار توسط سیستم تولید شوند.
- **واترمارک**: یک پوشش تصویر در بالای ویدئوی شما حاوی اطلاعات شناسایی درباره ویدئوی شما است.



شکل ۹-۱۴

معماری تبدیل فرمت ویدئو

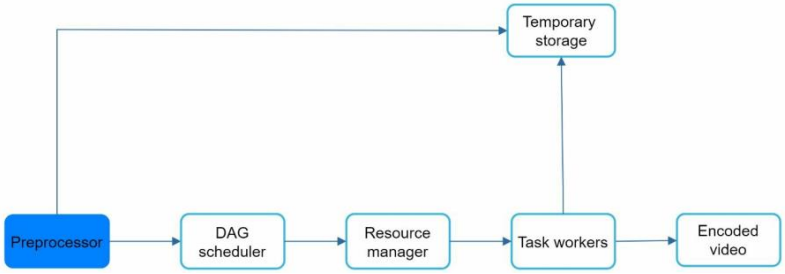
معماری تبدیل فرمت ویدئوی پیشنهادی که از سرویس‌های ابری استفاده می‌کند، در شکل ۱۰-۱۴ نشان‌داده‌شده است.



شکل ۱۰-۱۴

این معماری دارای شش جزء اصلی است: پیش‌پردازنده، زمان‌بندی DAG، مدیر منابع، task workerها، ذخیره‌سازی موقت و ویدئوی کدگذاری شده به‌عنوان خروجی. اجازه دهید نگاهی دقیق به هر جزء داشته باشیم.

پیش‌پردازش (Preprocessor)



شکل ۱۴-۱۱

۱. **تقسیم ویدئو**. پخش ویدئو به گروه کوچک‌تر تصاویر که GOP[1] نامیده می‌شود تقسیم شده یا در آینده تقسیم خواهد شد. GOP یک گروه/تکه از فریم‌ها است که به ترتیب خاصی چیده شده‌اند. هر قطعه یک واحد مستقل قابل‌پخش است که معمولاً چند ثانیه طول دارد.

۲. برخی از دستگاه‌های تلفن همراه یا مرورگرهای قدیمی ممکن است از تقسیم ویدئو پشتیبانی نکنند. پیش‌پردازنده[2] ویدئوها را با تراز[3] GOP برای کاربرهای قدیمی تقسیم می‌کند.

۳. **تولید DAG**. پردازنده DAG را بر اساس فایل‌های پیکربندی که کاربرها می‌نویسند، تولید می‌کند. شکل ۱۲-۱۴ یک نمایش ساده شده DAG است که دارای ۲ گره و ۱ لبه است:

شکل ۱۲-۱۴

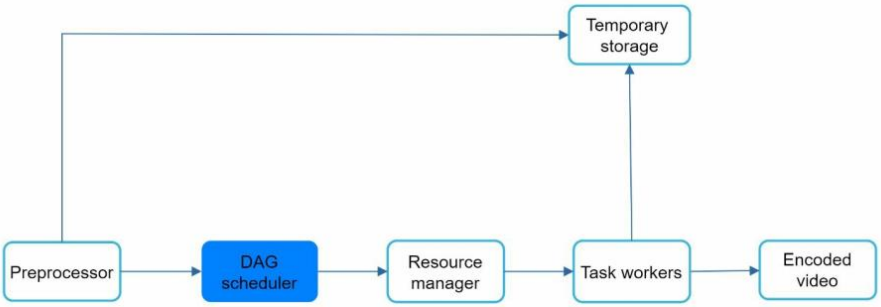
این DAG نشان‌داده‌شده از دو فایل پیکربندی زیر تولید می‌شود (شکل ۱۳-۱۴):

1 smaller Group of Pictures
2 Preprocessor
3 GOP alignment

شکل ۱۳-۱۴ – به منبع شماره[۹] رجوع شود

۴. **داده‌های کَش**. پیش‌پردازنده دارای یک حافظه کَش برای ویدئوهای بخش‌بندی[1] شده است. برای قابلیت اطمینان بهتر، پیش پردازشگر GOPها و متادیتاها را در ذخیره‌ساز موقت ذخیره می‌کند. اگر رویه تبدیل فرمت ویدئو ناموفق باشد، سیستم می‌تواند از داده‌های باقی‌مانده برای ادامه و تکرار عملیات استفاده کند.

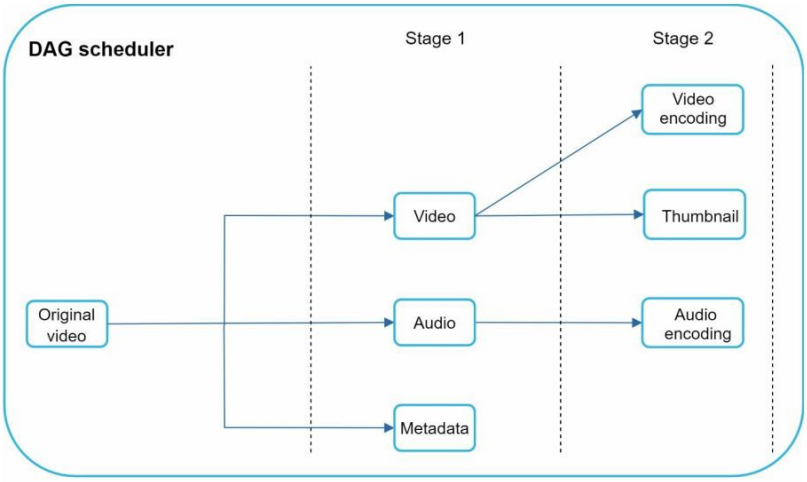
DAG scheduler



شکل ۱۴-۱۴

زمان‌بند DAG یا DAG scheduler یک گراف DAG را به مراحلی از تسک‌ها تقسیم می‌کند و آنها را در صف تسک‌ها در مدیر منابع[2] قرار می‌دهد. شکل ۱۵-۱۴ نمونه‌ای از نحوه عملکرد زمانبند DAG را نشان می‌دهد.

1 segmented
2 resource manager



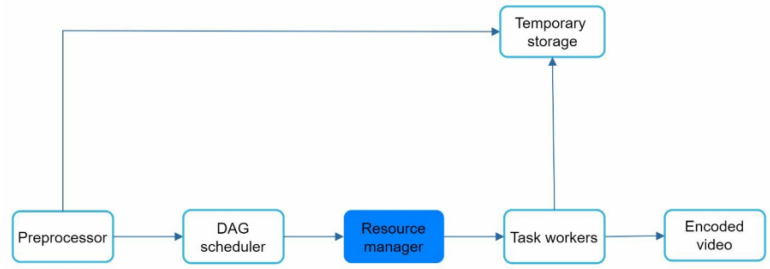
شکل ۱۵-۱۴

همان‌طور که در شکل ۱۵-۱۴ نشان‌داده‌شده است، ویدئوی اصلی به سه مرحله تقسیم می‌شود:

مرحله ۱: ویدئو، صدا و متادیتا.

فایل ویدئویی در مرحله ۲ به دو تسک تقسیم می‌شود: رمزگذاری ویدئو و کوچک سازی تصویر[1]. فایل صوتی به کدکذاری[2] صدا به‌عنوان بخشی از تسک مرحله ۲ نیاز دارد.

مدیریت منابع (Preprocessor)

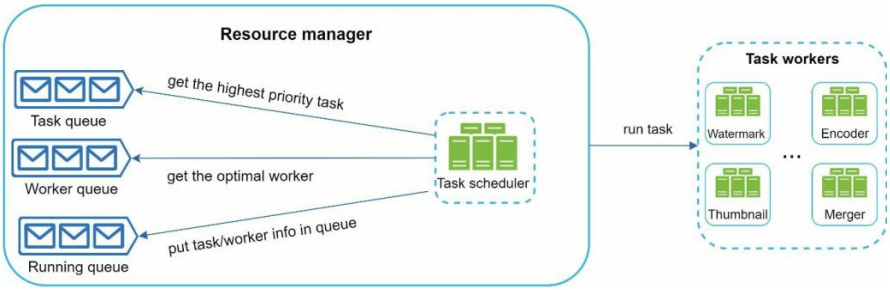


شکل ۱۶-۱۴

1 thumbnail
2 encoding

مدیر منابع[۱] مسئول مدیریت بسیار مؤثر و دقیق جهت تخصیص منابع است. این شامل ۳ صف و یک زمانبندی تسک[۲] همان‌طور که در شکل ۱۷-۱۴ نشان‌داده‌شده است.

- **صف تسک‌ها**[۳]: یک صف اولویت‌دار[۴] است که شامل تسک‌هایی است که باید اجرا شوند.
- **صف Worker**: یک صف اولویت‌دار است که حاوی اطلاعات استفاده از worker است.
- **صف در حال اجرا**: حاوی اطلاعاتی در مورد تسک‌های در حال اجرا و workerهایی که تسک‌ها را انجام می‌دهند.
- **زمان‌بندی تسک**[۵]: که تسک یا worker بهینه را انتخاب می‌کند و به worker انتخاب شده دستور می‌دهد تا تسک را اجرا کند.



شکل ۱۷-۱۴

مدیر منابع به شرح زیر عمل می‌کند:

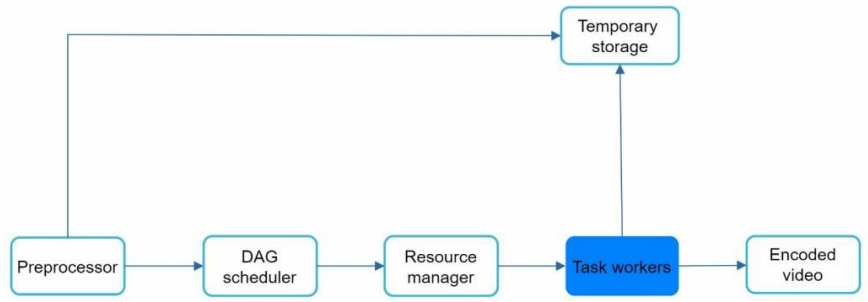
- زمان‌بندی تسک بیشترین اولویت را از صف تسک[۶] دریافت می‌کند.
- زمان‌بندی تسک، تسک بهینه را برای اجرای کار از صف کارگر[۷] دریافت می‌کند.
- زمان‌بندی تسک به worker انتخاب شده دستور می‌دهد تا تسک را اجرا کند.

---

1 resource manager
2 task scheduler
3 task queue
4 priority queue
5 task scheduler
6 task queue
7 worker queue

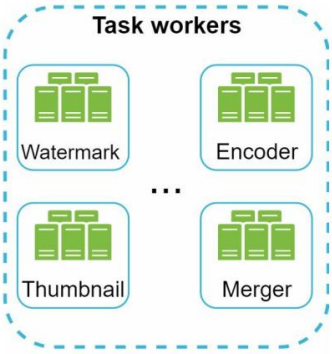
- زمان‌بندی تسک اطلاعات تسک یا worker را به هم متصل می‌کند و آن را در صف در حال اجرا قرار می‌دهد.
- زمان‌بندی تسک پس از اتمام کار، تسک را از صف در حال اجرا[1] حذف می‌کند.

Task workers



شکل ۱۸-۱۴

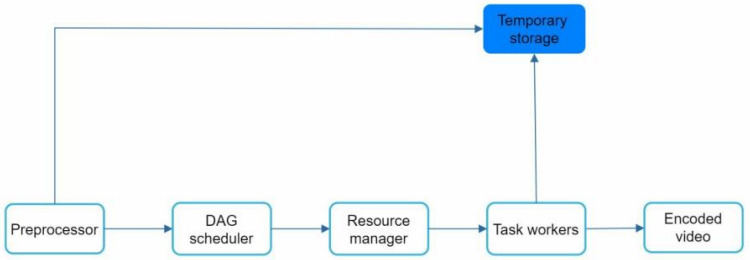
این Task workerها وظایفی را که در DAG تعریف شده‌اند را اجرا می‌کنند. workerهای مختلف ممکن است وظایف مختلفی را همان‌طور که در شکل ۱۹-۱۴ نشان‌داده‌شده است انجام دهند.



شکل ۱۹-۱۴

1 running queue

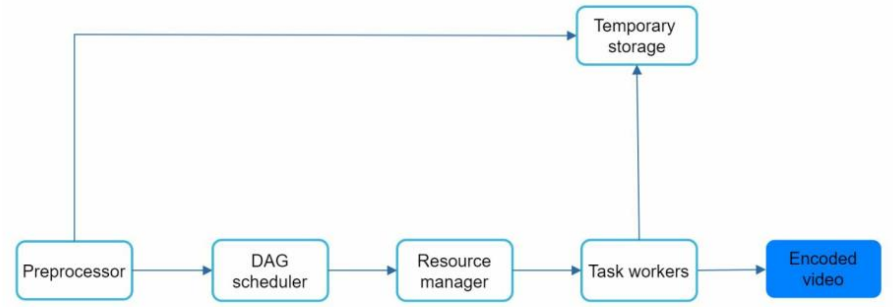
Temporary storage



شکل ۲۰-۱۴

در اینجا از چندین سیستم ذخیره‌ساز[1] مختلف استفاده می‌شود. انتخاب سیستم ذخیره‌سازی به عواملی مانند نوع داده، اندازه داده، دفعات دسترسی، طول عمر داده و غیره بستگی دارد. به‌عنوان‌مثال، متادیتا اغلب توسط workerها قابل‌دسترسی است و اندازه داده آن معمولاً کوچک است؛ بنابراین ذخیره متادیتا در حافظه ایده خوبی است. برای داده‌های ویدئویی یا صوتی، آنها را در ذخیره‌ساز حبابی[2] قرار می‌دهیم. پس از تکمیل پردازش ویدئوی مربوطه، داده‌های موجود در فضای ذخیره‌سازی موقت آزاد می‌شوند.

کدگذاری ویدئو (Encoded video)



شکل ۲۱-۱۴

1 storage
2 blob storage

ویدئوی کدگذاری[1] شده خروجی نهایی pipeline کدگذاری است. در اینجا یک نمونه از خروجی موجود است:

.funny_720p.mp4

سیستم بهینه‌سازی

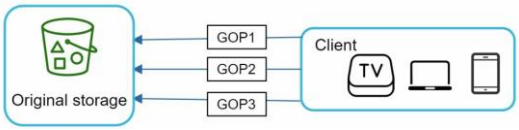
در این مرحله، شما باید درک خوبی در مورد مسائلی مانند آپلود ویدئو، پخش ویدئو و فرمت‌بندی ویدئو داشته باشید. در مرحله بعد، سیستم را با بهینه‌سازی‌هایی از جمله افزایش سرعت، ایمنی و صرفه‌جویی در هزینه اصلاح خواهیم کرد.

سیستم بهینه‌سازی: موازی‌سازی آپلود ویدئو

آپلود یک ویدئو به‌صورت یک جا و به‌عنوان یک واحد یک‌پارچه، روشی نامناسب و ناکارآمد است. همان‌طور که در شکل ۲۲-۱۴ نشان‌داده‌شده است، می‌توانیم یک ویدئو را با تراز GOP به قطعات کوچکتر تقسیم کنیم.

شکل ۲۲-۱۴

هنگامی که آپلود قبلی ناموفق بود، این کار امکان آپلود سریع و قابل ازسرگیری را فراهم می‌کند. کار تقسیم یک فایل ویدئویی توسط GOP می‌تواند توسط کلاینت برای بهبود سرعت آپلود همان‌طور که در شکل ۲۳-۱۴ نشان‌داده‌شده است پیاده‌سازی شود.

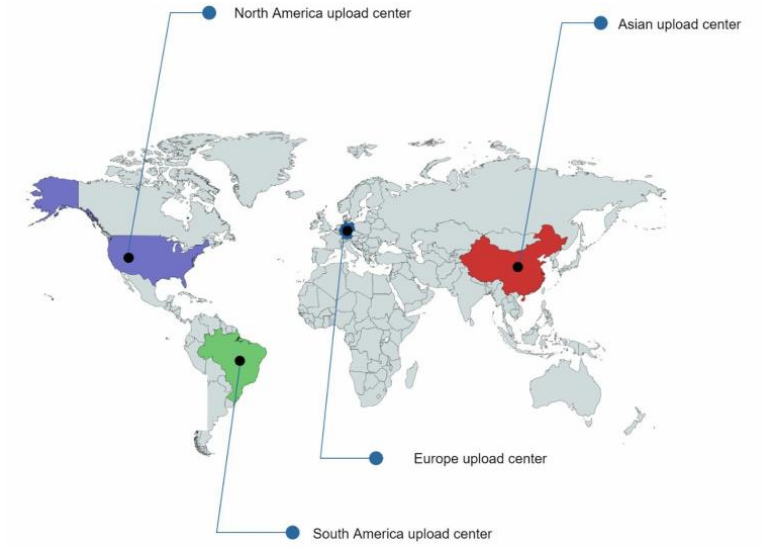


شکل ۲۳-۱۴

1 Encoded

سیستم بهینه‌سازی: قراردادن دیتاسنتر به کاربر تا جایی که امکان دارد

راه دیگر برای بهبود سرعت آپلود، راه‌اندازی چندین مرکز آپلود در سراسر جهان است (شکل ۲۴-۱۴). مردم ایالات متحده می‌توانند ویدئوها را در مرکز بارگذاری[1] درآمریکای شمالی بارگذاری کنند و مردم چین می‌توانند ویدئوها را در مرکز آپلود آسیایی، بارگذاری کنند. برای رسیدن به این هدف، از CDN به‌عنوان مراکز بارگذاری استفاده می‌کنیم.



شکل ۲۴-۱۴

سیستم بهینه‌سازی: موازی‌سازی همه‌ی چیزها

دستیابی به تأخیر زمانیِ کم‌نیاز به تلاش جدی دارد. یکی دیگر از بهینه‌سازی‌ها، ساختن سیستم‌هایی با پیوستگی ضعیف[2] و فعال‌کردن مکانیزمی با قابلیت پردازش موازی بالا[3] است. طراحی ما برای دستیابی به موازی‌سازی سطح بالا نیاز به تغییراتی مهمی دارد. اجازه دهید

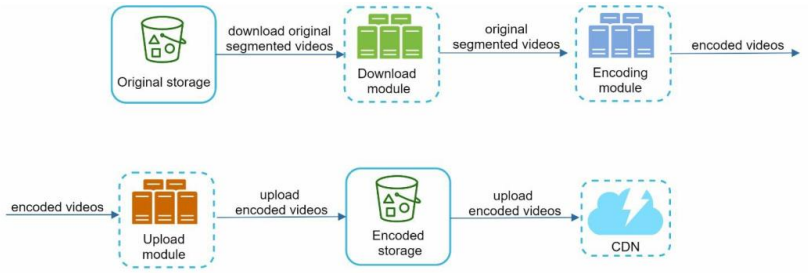
---

1 upload center

۲ Loosely Coupled به نوعی معماری در توسعه سیستم‌های نرم‌افزاری گفته می‌شود که در آن، اجزا یا بخش‌های مختلف اپلیکیشن تا حد ممکن مستقل از یکدیگر خواهند بود. به عبارت دیگر، این اصطلاح به میزان ارتباط مستقیم ماژول‌های مختلف اپلیکیشن با یکدیگر اطلاق می‌گردد.

3 high parallelism

در جریان نحوه انتقال یک ویدئو از حافظه اصلی به CDN بزرگ‌نمایی کنیم. مسئله در شکل ۱۴-۲۵ نشان‌داده‌شده است و نشان می‌دهد که خروجی به ورودی مرحله قبل وابستگی دارد. این وابستگی، موازی‌سازی را دشوار می‌کند.

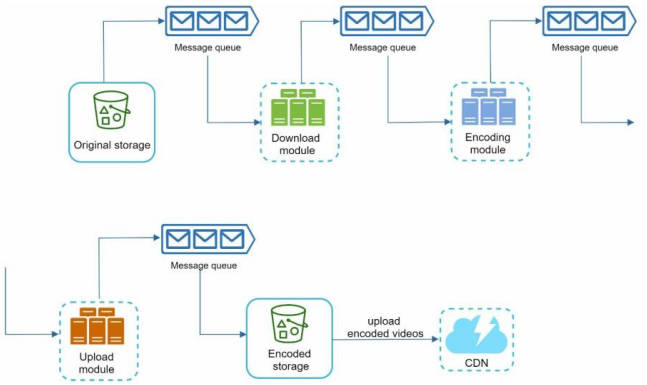


شکل ۱۴-۲۵

برای اینکه سیستم به‌صورت بهتری loosely coupled شود، صف‌های پیام[1] را همان‌طور که در
شکل ۱۴-۲۶ نشان‌داده‌شده است، معرفی کردیم. اجازه دهید از یک مثال استفاده کنیم تا توضیح دهیم که چگونه صف‌های پیام باعث می‌شود که اجزای سیستم گسسته‌تر یا میزان وابستگی اجزا به هم دیگر کمتر شود.

- قبل از معرفی صف پیام، ماژول کدگذاری[2] باید منتظر خروجی ماژول دانلود باشد.
- پس از معرفی صف پیام، ماژول کدگذاری دیگر نیازی به انتظار برای خروجی ماژول دانلود ندارد. اگر رویدادهایی در صف پیام وجود داشته باشد، ماژول کدگذاری می‌تواند آن کارها را به‌صورت موازی اجرا کند.

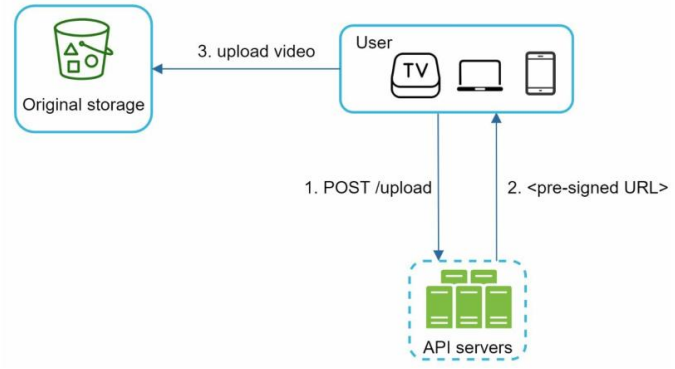
1 message queues
2 encoding



شکل ۲۶-۱۴

سیستم بهینه‌سازی: pre-signed upload URL

ایمنی یکی از مهم‌ترین جنبه‌های هر محصولی است. برای اطمینان از اینکه فقط کاربران مجاز ویدئوها را در مکان مناسب آپلود می‌کنند، URLهای پیش‌امضا شده[۱] را همان‌طور که در شکل ۲۷-۱۴ نشان‌داده‌شده است معرفی می‌کنیم.



شکل ۲۷-۱۲

---

۱ یک URL بارگذاری از پیش امضا شده (pre-signed upload URL)، یک URL موقت است که به کاربران اجازه می‌دهد تا فایل‌ها را به طور مستقیم به یک سرویس ذخیره‌سازی ابری مانند Amazon S3 بارگذاری کنند، بدون اینکه نیاز به اعتبارنامه‌های امنیتی AWS داشته باشند .این URL ها توسط API پشتیبانی سیستم ایجاد می‌شوند و می‌توانند برای مدت زمان محدودی استفاده شوند.

مسیرهای آپلود به شرح زیر به‌روزرسانی می‌شود:

۱. کلاینت یک درخواست HTTP به سرورهای API ارسال می‌کند تا URL پیش‌امضا شده را واکشی کند که این کار اجازه دسترسی به شیء مشخص شده در URL را می‌دهد. مورد URL پیش‌امضا شده[1] با آپلود فایل‌ها در Amazon S3 استفاده می‌شود. سایر ارائه‌دهندگان خدمات ابری ممکن است از نام دیگری استفاده کنند. به‌عنوان‌مثال، ذخیره‌ساز حبابی مایکروسافت Azure از همین ویژگی پشتیبانی می‌کند، اما آن را "Shared Access Signature" می‌نامند [۱۰].

۲. سرورهای API با یک URL پیش‌امضا شده پاسخ می‌دهند.

۳. هنگامی که کاربر پاسخ را دریافت کرد، ویدئو را با استفاده از URL از قبل امضا شده آپلود می‌کند.

سیستم بهینه‌سازی: محافظت از ویدئوهای بارگذاری شده

بسیاری از تولیدکنندگان محتوا تمایلی به ارسال ویدئوهای آنلاین ندارند زیرا می‌ترسند ویدئوهای اصلی آنها به سرقت برود. برای محافظت از ویدئوهای دارای حق نسخه‌برداری، می‌توانیم یکی از سه گزینه محافظتی زیر را اتخاذ کنیم:

- **سیستم‌های مدیریت حقوق دیجیتال[2] یا به‌اختصار (DRM)**: سه سیستم اصلی DRM عبارت‌اند از Apple FairPlay، Google Widevine و Microsoft PlayReady.
- **رمزگذاری AES**: می‌توانید یک ویدئو را رمزگذاری کنید و یک خط‌مشی مجوز انتشار را تنظیم کنید. ویدئوی رمزگذاری شده پس از پخش رمزگشایی می‌شود. این کار تضمین می‌کند که فقط کاربران مجاز می‌توانند یک ویدئوی رمزگذاری شده را تماشا کنند.

1 pre-signed upload URL
2 Digital rights management

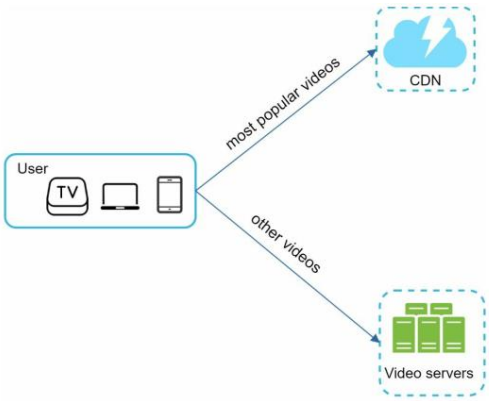
- **واترمارک**[1]: این یک پوشش تصویری در بالای ویدئوی شما است که حاوی اطلاعات شناسایی ویدئوی شما است. این می‌تواند لوگوی شرکت یا نام شرکت شما باشد.

بهینه‌سازی‌های مربوط به هزینه‌های مالی

در واقع CDN جزء حیاتی از سیستم ما است. این گزینه تحویل سریع ویدئو را در مقیاس جهانی تضمین می‌کند. بااین‌حال، باتوجه به حجم محاسباتی، می‌دانیم که CDN گران است، به‌خصوص زمانی که اندازه داده بزرگ است. چگونه می‌توانیم هزینه را کاهش دهیم؟

تحقیقات قبلی نشان می‌دهد که پخش‌های ویدئویی YouTube از توزیع دنباله بلند[2] پیروی می‌کنند [۱۱] [۱۲]. این بدان معنی است که چند ویدئوی محبوب به طور مکرر در دسترس هستند، اما بسیاری دیگر تعداد کمی بیننده دارند یا اصلاً بیننده ندارند. بر اساس این مشاهدات، ما چند بهینه سازی را پیاده‌سازی می‌کنیم:

۱. فقط محبوب‌ترین ویدئوها را از CDN دریافت کنید و سایر ویدئوها را از سرورهای ویدئوی ذخیره‌سازی با ظرفیت بالا ارائه دهید (شکل ۲۸-۱۴).



شکل ۲۸-۱۴

1 watermarking
2 long-tail

۲. برای محتوای با محبوبیت کمتر، ممکن است نیازی به ذخیره بسیاری از نسخه‌های ویدئویی کدگذاری شده نداشته باشیم. ویدئوهای کوتاه را می‌توان در صورت درخواست کدگذاری کرد.
۳. برخی ویدئوها فقط در مناطق خاصی محبوب هستند. نیازی به توزیع این ویدئوها در مناطق دیگر نیست.
۴. شاید بهتر باشد CDN خود را مانند Netflix بسازید و با ارائه‌دهندگان خدمات اینترنت (ISP) شریک شوید. ساخت CDN شما یک پروژه غول‌پیکر است. بااین‌حال، این کار می‌تواند برای شرکت‌های پخش بزرگ منطقی باشد. یک ISP می‌تواند Comcast، AT&T، Verizon یا سایر ارائه‌دهندگان اینترنت باشد. ISPها در سرتاسر جهان قرار دارند و به کاربران نزدیک هستند. با مشارکت با ISPها، می‌توانید تجربه مشاهده را بهبود بخشید و هزینه‌های پهنای باند را کاهش دهید.

همه آن بهینه‌سازی‌ها بر اساس محبوبیت محتوا، الگوی دسترسی کاربر، اندازه ویدئو و غیره است. تجزیه‌وتحلیل الگوهای مشاهده تاریخی قبل از انجام هر گونه بهینه‌سازی مهم است. در اینجا برخی از مقالات جالب در این زمینه آورده شده است: [۱۲] [۱۳] .

رسیدگی به خطاها

برای یک سیستم در مقیاس بزرگ[1]، خطاهای سیستمی اجتناب‌ناپذیر است. برای ساختن یک سیستم بسیار مقاوم در برابر خطا، باید خطاها را باظرافت مدیریت کنیم و خیلی سریع آنها بازیابی کنیم. دو نوع خطا وجود دارد:

- **خطای قابل بازیابی**. برای خطاهای قابل بازیابی؛ مانند خطای قطعه‌بندی[2] ویدئویی که معمولاً در قسمت در تبدیل فرمت اتفاق می‌افتد، ایده کلی این است که این عملیات را چند بار دوباره امتحان کنید. اگر کار همچنان با شکست مواجه شود و سیستم معتقد باشد که قابل بازیابی نیست، یک کد خطای مناسب را به کاربر بر می‌گرداند.

1 large-scale system
2 segment

- **خطای غیرقابل‌جبران**. برای خطاهای غیر قابل بازیابی؛ مانند فرمت نادرست ویدئو، سیستم باید تسک‌های در حال اجرا مرتبط با ویدئو را متوقف می‌کند و کد خطای مناسب را به کاربر بر می‌گرداند.

خطاهای معمولی برای هر جزء سیستم توسط راهنمای زیرپوشش داده شده است:

- **خطای آپلود**: چند لحظه دیگر دوباره امتحان یا retry کنید.
- **خطای تقسیم ویدئو**: اگر نسخه‌های قدیمی‌تر کلاینت‌ها نتوانند ویدئوها را با تراز GOP تقسیم کنند، کل ویدئو به سرور ارسال می‌شود. کار تقسیم فیلم‌ها در سمت سرور انجام می‌شود.
- **خطای تبدیل فرمت**[1]: دوباره امتحان مجدد کنید.
- **خطای پیش‌پردازنده**: نمودار DAG را بازسازی کنید.
- **خطای زمان‌بندی DAG**: یک تسک را دوباره برنامه‌ریزی کنید.
- **صف مدیر منابع**: از یک کپی‌ساز[2] استفاده کنید.
- **Task worker ازکارافتاده است**: دوباره تسک را روی یک worker جدید امتحان کنید.
- **API server ازکارافتاده است**: سرورهای API معمولاً stateless هستند، بنابراین درخواست‌ها به سرور API دیگری هدایت می‌شوند.
- **سرور کَش متادیتا [3] ازکارافتاده است**: داده‌ها چندین مرتبه تکثیر[4] می‌شوند. اگر یک گره[5]، پایین بیاید، همچنان می‌توانید به سایر گره‌ها برای واکشی داده‌ها دسترسی داشته باشید. ما می‌توانیم یک سرور cache جدید برای جایگزینی سرور ازکارافتاده ایجاد کنیم.
- **سرور پایگاه‌داده متادیتا ازکارافتاده است:**

---

1 Transcoding
2 replica
3 Metadata cache server
4 replicate

۵ - گره یا node مفهومی در محاسبات ابری است.

- سرور Master ازکارافتاده است. اگر سرور Master ازکارافتاده است، یکی از Slave را به‌عنوان Master جدید ارتقا دهید.
- سرور Slave ازکارافتاده است. اگر یک Slave از کار بیفتد، می‌توانید از Slave دیگری برای خواندن استفاده کنید و یک سرور پایگاه‌داده دیگری را برای جایگزینی سرور ازکارافتاده راه‌اندازی کنید.

مرحله ٤ – جمع‌بندی

در این فصل، طراحی معماری سرویس‌های پخش ویدئو مانند یوتیوب را ارائه کردیم. اگر در پایان مصاحبه زمان اضافی وجود داشت، پس در اینجا چند نکته اضافی دیگر وجود دارد:

- **مقیاس‌پذیری لایه API**: از آنجایی که سرورهای API به‌صورت stateless هستند، مقیاس‌پذیری یا scale لایه API به‌صورت مقیاس‌پذیری افقی آسان است.
- **مقیاس‌پذیری پایگاه‌داده**: می‌توانید در مورد replication و sharding پایگاه‌داده صحبت کنید.
- **پخش زنده**: به روند نحوه ضبط و پخش ویدئو در زمان بلادرنگ[1] اشاره دارد. اگرچه سیستم ما به طور خاص برای پخش زنده طراحی نشده است، پخش زنده و پخش غیرزنده شباهت‌هایی دارند: هر دو به آپلود، رمزگذاری و پخش نیاز دارند. پس تفاوت‌های قابل‌توجه عبارت‌اند از:
  - پخش زنده نیاز به تأخیر زمانی پر اهمیت‌تری دارد، بنابراین ممکن است به پروتکل پخش متفاوتی نیاز داشته باشد.
  - پخش زنده نیاز کمتری به موازی‌سازی دارد؛ زیرا تکه‌های کوچکی از داده‌ها در حین زمان بلادرنگ پردازش می‌شوند.
  - پخش زنده نیاز به مجموعه‌های مختلفی از خطایابی و رسیدگی به خطاها دارد. هر گونه رسیدگی به خطا که زمان زیادی می‌برد قابل‌قبول نیست.

---

1 real-time

- **حذف ویدئو**: ویدئوهایی که حق نسخه‌برداری، پورنوگرافی یا سایر اعمال غیرقانونی را نقض می‌کنند باید حذف شوند. برخی را می‌توان در طول فرایند آپلود توسط سیستم کشف کرد، درحالی‌که برخی دیگر ممکن است از طریق پرچم گذاری[1] سایر کاربرها کشف شوند.

1 real-time

منابع و مراجع فصل ۱۴

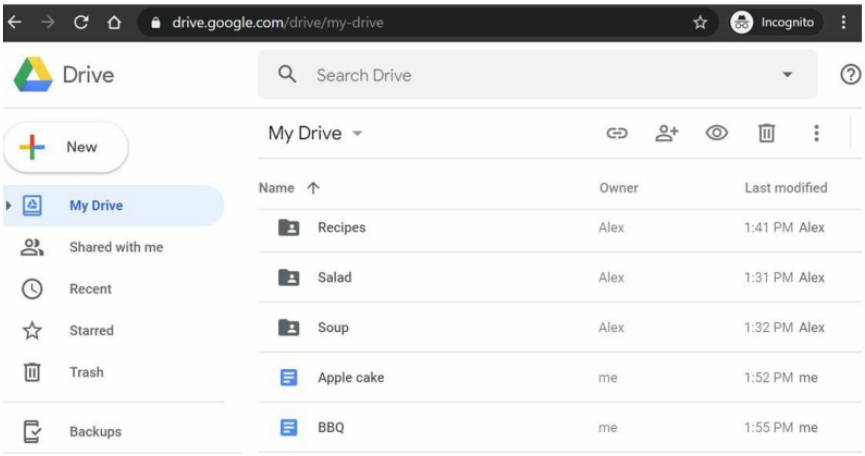
[1] YouTube by the numbers: https://www.omnicoreagency.com/youtube-statistics/
[2] 2019 YouTube Demographics: https://blog.hubspot.com/marketing/youtube-demographics
[3] Cloudfront Pricing: https://aws.amazon.com/cloudfront/pricing/
[4] Netflix on AWS: https://aws.amazon.com/solutions/case-studies/netflix/
[5] Akamai homepage: https://www.akamai.com/
[6] Binary large object: https://en.wikipedia.org/wiki/Binary_large_object
[7] Here's What You Need to Know About Streaming Protocols: https://www.dacast.com/blog/streaming-protocols/
[8] SVE: Distributed Video Processing at Facebook Scale: https://www.cs.princeton.edu/~wlloyd/papers/sve-sosp17.pdf
[9] Weibo video processing architecture (in Chinese): https://www.upyun.com/opentalk/399.html
[10] Delegate access with a shared access signature: https://docs.microsoft.com/en-us/rest/api/storageservices/delegate-access-with-shared-accesssignature
[11] YouTube scalability talk by early YouTube employee: https://www.youtube.com/watch? v=w5WVu624fY8
[12] Understanding the characteristics of internet short video sharing: A youtube-based measurement study. https://arxiv.org/pdf/0707.3670.pdf
[13] Content Popularity for Open Connect: https://netflixtechblog.com/content-popularity-for-open-connect-b86d56f613b

فصل ۱۵

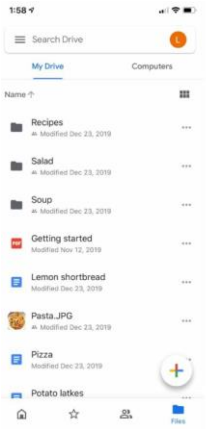
طراحی GOOGLE DRIVE

در سال‌های اخیر، سرویس‌های ذخیره‌سازی ابری مانند Google Drive، Dropbox، Microsoft OneDrive و Apple iCloud بسیار محبوب شده‌اند. در این فصل از شما خواسته می‌شود تا گوگل درایو را طراحی کنید.

اجازه دهید قبل از اینکه وارد طراحی شویم، کمی وقت بگذاریم تا Google Drive را درک کنیم. Google Drive یک سرویس ذخیره‌سازی و همگام‌سازی فایل است که به شما کمک می‌کند اسناد، عکس‌ها، ویدئوها و سایر فایل‌ها را در فضای ابری ذخیره کنید. شما می‌توانید از هر کامپیوتر، گوشی هوشمند و تبلتی به فایل‌های خود دسترسی داشته باشید. شما به‌راحتی می‌توانید آن فایل‌ها را با دوستان، خانواده و همکاران به اشتراک بگذارید [۱]. شکل ۱-۱۵ و ۲-۱۵ به ترتیب نشان می‌دهد که گوگل درایو در مرورگر و برنامه تلفن همراه چگونه است.



شکل ۱-۱۵



شکل ۲-۱۵

مرحله ۱ - درک مسئله و شروع به طراحی

طراحی گوگل درایو یک پروژه بزرگ است، بنابراین مهم است که سؤالاتی را برای شفاف‌سازی مسئله بپرسید.

کاندید: مهم‌ترین ویژگی‌های این برنامه چیست؟

مصاحبه‌کننده: آپلود و دانلود فایل‌ها، همگام‌سازی[1] فایل‌ها و اعلان‌ها[2].

کاندید: آیا این یک برنامه تلفن همراه است، یک برنامه وب یا هر دو؟

مصاحبه‌کننده: هر دو.

کاندید: فرمت‌های فایل پشتیبانی شده چیست؟

مصاحبه‌کننده: هر نوع فایل.

کاندید: آیا فایل‌ها نیاز به رمزگذاری دارند؟

مصاحبه‌کننده: بله، فایل‌های موجود در حافظه باید رمزگذاری شوند.

کاندید: آیا محدودیت حجم فایل وجود دارد؟

مصاحبه: بله، فایل‌ها باید ۱۰ گیگابایت یا کمتر باشند.

کاندید: محصول چند کاربر دارد؟

مصاحبه‌کننده: ۱۰ میلیون کاربران فعال روزانه (DAU).

در این فصل، ما بر روی ویژگی‌های زیر تمرکز می‌کنیم:

- افزودن فایل‌ها. ساده‌ترین راه برای افزودن فایل، کشیدن و رهاکردن فایل در گوگل درایو است.
- دانلود فایل‌ها.
- همگام‌سازی فایل‌ها در چندین دستگاه. هنگامی که یک فایل به یک دستگاه اضافه می‌شود، به طور خودکار با دستگاه‌های دیگر همگام‌سازی می‌شود.
- نسخه‌های مختلف از فایل‌ها را ببینید و آن‌ها را ویرایش کنید.
- فایل‌ها را با دوستان، خانواده و همکاران خود به اشتراک بگذارید.
- هنگامی که یک فایل ویرایش، حذف یا با شما به اشتراک گذاشته می‌شود، نوتیفیکیشنی را ارسال کنید.

ویژگی‌هایی که در این فصل مورد بحث قرار نگرفته‌اند عبارت‌اند از:

---

1 sync
2 notifications

- ویرایش سند گوگل و همکاری در آن. Google doc به چندین نفر اجازه می‌دهد تا یک سند را به طور هم‌زمان ویرایش کنند. این خارج از محدوده طراحی ما است.

به‌غیراز روشن‌کردن الزامات و نیازمندی‌ها، درک الزامات غیرعملکردی نیز مهم است:

- قابلیت اطمینان[1]. قابلیت اطمینان برای یک سیستم ذخیره‌سازی بسیار مهم است. ازدست‌دادن داده‌ها غیرقابل‌قبول است.
- سرعت همگام‌سازی سریع. اگر همگام‌سازی فایل زمان زیادی را صرف کند، کاربران بی‌حوصله می‌شوند و محصول را رها می‌کنند.
- استفاده از پهنای باند. اگر یک محصول از پهنای باند غیرضروری شبکه زیادی استفاده کند، کاربران ناراضی خواهند بود، به‌خصوص زمانی که در برنامه در حال استفاده از حالت داده شبکه تلفن همراه هستند.
- مقیاس‌پذیری[2]. سیستم باید بتواند حجم بالایی از ترافیک را مدیریت کند.
- دردسترس‌بودن بالا[3]. هنگامی که برخی از سرورها آفلاین هستند، سرعت پایینی دارند یا خطاهای شبکه غیرمنتظره دارند، کاربران همچنان باید بتوانند از سیستم استفاده کنند.

محاسبات و تخمین

- فرض کنید برنامه دارای ۵۰ میلیون کاربر ثبت نام شده و ۱۰ میلیون کاربران فعال روزانه (DAU) است.
- کاربران ۱۰ گیگابایت فضای رایگان دریافت می‌کنند.
- فرض کنید کاربران ۲ فایل در روز آپلود می‌کنند. حجم متوسط فایل ۵۰۰ کیلوبایت است.
- نسبت خواندن به نوشتن ۱:۱.
- کل فضای اختصاص‌داده‌شده برابر است با:

---

1 Reliability
2 Scalability
3 High availability

پتابایت ۵۰۰ = گیگابایت ۱۰ × میلیون ۵۰

- QPS برای آپلود API:

۲۴۰ ~ = ثانیه ۳۶۰۰ / ساعت ۲۴ / آپلود ۲ * میلیون ۱۰

- حداکثر مقدار QPS برابر است با: QPS * 2 = 480

مرحله ۲ - طراحی سطح پیشرفته.

به‌جای نشان‌دادن نمودار طراحی سطح پیشرفته در ابتدا، از رویکرد کمی متفاوت استفاده خواهیم کرد. ما با چیز ساده شروع خواهیم کرد: ساختن همه‌چیز در یک سرور واحد. سپس، به‌تدریج آن را برای پشتیبانی از میلیون‌ها کاربر افزایش دهید. با انجام این تمرین، حافظه شما را در مورد برخی از موضوعات مهم مطرح شده در کتاب تازه خواهد شد.

اجازه دهید با یک راه‌اندازی سرور واحد به شرح زیر شروع کنیم:

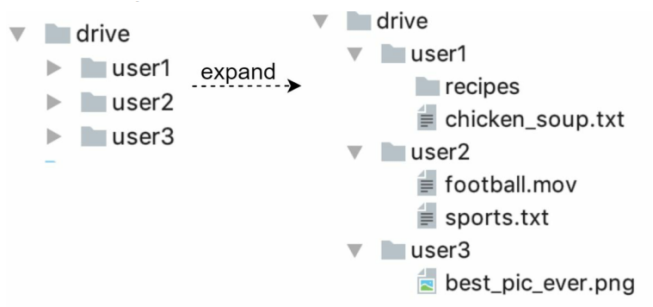
- یک وب سرور برای آپلود و دانلود فایل‌ها.
- یک پایگاه‌داده برای پیگیری فراداده‌ها[1] مانند داده‌های کاربر، اطلاعات ورود به سیستم، اطلاعات فایل‌ها و غیره.
- یک سیستم ذخیره‌سازی برای ذخیره فایل‌ها. ما ۱ ترابایت فضای ذخیره‌سازی را برای ذخیره فایل‌ها اختصاص می‌دهیم.

ما چند ساعت را صرف راه‌اندازی یک وب سرور آپاچی، یک پایگاه‌داده MySQL و یک دایرکتوری به نام /drive به‌عنوان دایرکتوری اصلی برای ذخیره فایل‌های آپلود شده می‌کنیم. در زیر پوشه /drive فهرستی از دایرکتوری‌ها وجود دارد که به‌عنوان فضاهای نام[2] شناخته می‌شوند. هر فضای نام شامل تمام فایل‌های آپلود شده برای آن کاربر است. نام فایل روی سرور مانند نام فایل اصلی باقی می‌ماند. هر فایل یا پوشه را می‌توان با پیوستن به فضای نام و مسیر نسبی به طور منحصربه‌فرد شناسایی کرد.

---

1 metadata
2 namespaces

شکل ۳-۱۵ مثالی از نحوه ظاهر دایرکتوری drive/ در سمت چپ و نمای گسترش یافته آن در سمت راست را نشان می‌دهد.



شکل ۳-۱۵

بـــررســـــی APIها

APIها چه شکلی هستند؟ ما در ابتدا به ۳ عدد API نیاز داریم: یکی برای آپلودکردن فایل، یکی برای دانلودکردن فایل و یکی دیگر هم برای ویرایش‌کردن فایل.

آپلودکردن فایل برای گوگل درایو

دو نوع آپلود پشتیبانی می‌شود:

- **بارگذاری ساده**. وقتی حجم فایل کوچک است از این نوع آپلود استفاده کنید.
- **بارگذاری قابل ازسرگیری**. از این نوع آپلود زمانی استفاده کنید که حجم فایل بزرگ و احتمال قطع‌شدن شبکه زیاد باشد.

در اینجا یک نمونه از API آپلود مجدد وجود دارد:

https://api.example.com/files/upload?uploadType=resumable

پارامترها:

• نوع آپلود = قابل ازسرگیری.

• داده‌ها: فایل روی سیستم شخصی باید آپلود شود.

یک بارگذاری قابل ازسرگیری با ۳ مرحله زیر حاصل می‌شود [۲]:

- درخواست اولیه را برای بازیابی URL قابل ازسرگیری ارسال کنید.
- آپلود داده‌ها و نظارت بر وضعیت آپلود.
- اگر آپلود با اختلال مواجه شد، آپلود را از سر بگیرید.

دانلودکردن فایل از گوگل درایو

نمونه API: https://api.example.com/files/download

پارامترها:

• مسیر[1]: مسیر فایل دانلود.

نمونه‌ای از پارامترها:

```
{
    "path": "/recipes/soup/best_soup.txt"
}
```

گرفتن نسخه‌های مختلف و ادیت شده‌ی فایل

نمونه API: https://api.example.com/files/list_revisions

پارامترها:

• مسیر: مسیر فایلی که می‌خواهید تاریخچه ویرایش آن را دریافت کنید.

• محدودیت[2]: حداکثر تعداد بازبینی‌ها باز می‌گرداند.

نمونه‌ای از پارامترها:

```
{
    "path": "/recipes/soup/best_soup.txt,"
    "limit": 20
}
```

1 path
2 limit

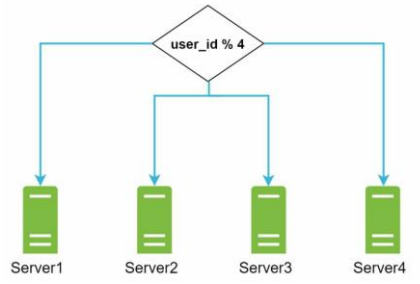
همه APIها نیاز به احراز هویت[1] کاربر دارند و از HTTPS استفاده می‌کنند. لایه SSL از انتقال داده بین سرویس گیرنده و سرورهای backend محافظت می‌کند.

حرکت به سمت استفاده از چند سرور

همان‌طور که فایل‌های بیشتری آپلود می‌شوند، در نهایت همان‌طور که در شکل ۴-۱۵ نشان‌داده‌شده است، هشدار «**فضا به طور کامل پر شده است**» را دریافت می‌کنید.

شکل ۴-۱۵

فقط ۱۰ مگابایت فضای ذخیره‌سازی باقی مانده است! این یک وضعیت اضطراری است زیرا کاربران دیگر نمی‌توانند فایل‌ها را آپلود کنند. اولین راه حلی که به ذهن می‌رسد این است که داده‌ها را shard کنید، بنابراین داده‌ها در چندین سرور ذخیره‌سازی[2] ذخیره می‌شود. شکل ۵-۱۵ نمونه‌ای از شاردبندی[3] بر اساس user_id را نشان می‌دهد.

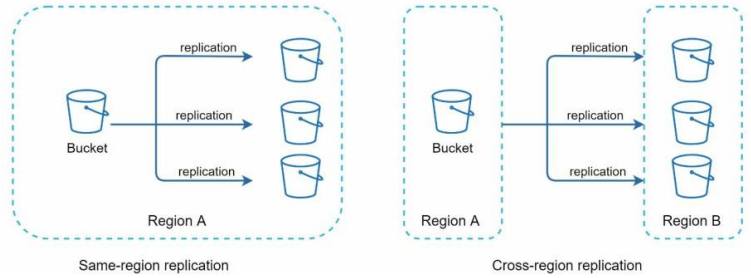


شکل ۵-۱۵

شما شبانه‌روز کار می‌کنید تا شاردبندی پایگاه‌داده را راه‌اندازی کنید و از نزدیک آن را مانیتور کنید. بعد از آن همه‌چیز به طور روان کار کرد. پس شما شاخ غول را شکسته‌اید، اما همچنان نگران ازدست‌دادن داده‌های احتمالی در صورت قطع‌شدن سرور ذخیره‌سازی هستید. شما

1 authentication
2 storage
3 sharding

سراغ اطرافیانتان رفتید و دوست متخصص‌تان در زمینه backend، آقای فرانک، به شما گفته است که بسیاری از شرکت‌های پیشرو مانند Netflix و Airbnb از Amazon S3 برای ذخیره‌سازی[1] استفاده می‌کنند. «سرویس ذخیره‌سازی ساده آمازون (Amazon S3) یک سرویس ذخیره‌سازی اشیا/object storage است که برای مواردی مقیاس‌پذیری[2]، دردسترس‌بودن[3] داده‌ها، امنیت و کارایی بالا را ارائه می‌دهد و پیشرو در صنعت ابری است» [۳]. شما تصمیم می‌گیرید کمی تحقیق کنید تا ببینید آیا این پیشنهاد مناسب است یا خیر. پس از مطالعه زیاد، درک خوبی از سیستم ذخیره‌سازی S3 به دست می‌آورید و تصمیم می‌گیرید فایل‌ها را در S3 ذخیره کنید. آمازون S3 از تکثیر[4] هم-ناحیه‌ای[5] و بین-ناحیه‌ای[6] پشتیبانی می‌کند. یک ناحیه در واقع یک منطقه جغرافیایی است که سرویس‌های وب آمازون (AWS) مراکز داده مختلفی در آنجا دارد. همان‌طور که در شکل ۶-۱۵ نشان‌داده‌شده است، داده‌ها را می‌توان در همان ناحیه (سمت چپ) و بین‌ناحیه‌ای (سمت راست) تکرار و تکثیرکرد. فایل‌های اضافی در چندین منطقه ذخیره می‌شوند تا ازدست‌رفتن داده‌ها محافظت کنند و از دردسترس‌بودن آن اطمینان حاصل کنند. یک پیمانه یا سطل[7] مانند یک‌پوشه در سیستم فایل است.



شکل ۶-۱۵

1 Storage
2 scalability
3 availability
4 replication
5 same-region
6 cross-region
7 bucket

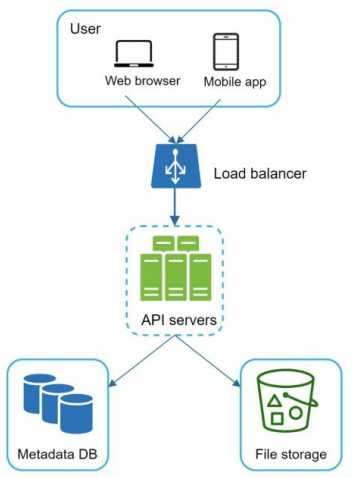
پس از قراردادن فایل‌ها در S3، در نهایت می‌توانید بدون نگرانی ازدست‌رفتن اطلاعات، خوابی راحت داشته باشید. برای جلوگیری از بروز مشکلات مشابه در آینده، تصمیم می‌گیرید تحقیقات بیشتری در زمینه‌هایی که می‌توانید بهبود بخشید انجام دهید. در اینجا چند موضوع موردتوجه وجود دارد:

- **متعادل‌کننده بار**[1]: برای توزیع ترافیک شبکه یک متعادل‌کننده بار اضافه کنید. یک متعادل‌کننده بار توزیع ترافیک شبکه یکنواخت را تضمین می‌کند و اگر وب سرور از کار بیفتد، ترافیک را مجدداً توزیع می‌کند.
- **وب سرورها**[2]: پس از اضافه‌شدن یک متعادل‌کننده بار، بسته به بار ترافیکی، وب سرورهای بیشتری را می‌توان به‌راحتی اضافه/حذف کرد.
- **پایگاه‌داده فراداده**[3]: پایگاه‌داده را به خارج از سرور منتقل کنید تا از یک نقطه‌ضعف و خرابی جلوگیری شود. در همین حال، تکثیر و به شاردبندی[4] داده‌ها را برای برآورده‌کردن الزامات دردسترس‌بودن و مقیاس‌پذیری آن‌ها تنظیم کنید.
- **ذخیره‌ساز فایل**[5]: آمازون S3 برای ذخیره‌سازی فایل استفاده می‌شود. برای اطمینان از دردسترس‌بودن و ماندگاری، فایل‌ها را در دو منطقه جغرافیایی جداگانه تکثیر[6] می‌شوند.

پس از اعمال بهبودهای فوق، وب سرورها، پایگاه‌داده متادیتا و ذخیره‌سازی فایل را با موفقیت از یک سرور جدا کرده‌اید. پس طرح به‌روزرسانی شده در شکل ۷-۱۵ نشان‌داده‌شده است.

---

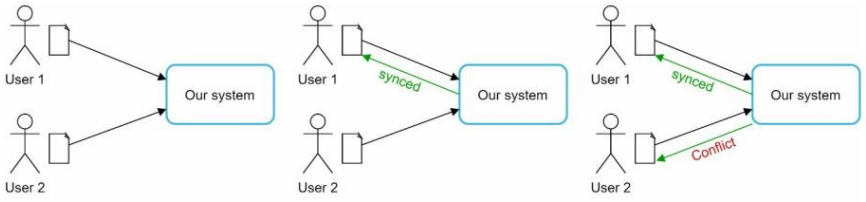
1 Load balancer
2 Web servers
3 Metadata database
4 sharding
5 File storage
6 replication



شکل ۷-۱۵

تداخل هم‌زمانی[1]

برای یک سیستم ذخیره‌سازی بزرگ مانند Google Drive، تداخل‌های همگام‌سازی گاهی اوقات اتفاق می‌افتد. هنگامی که دو کاربر یک فایل یا پوشه را به طور هم‌زمان تغییر می‌دهند، یک تضاد یا تداخل اتفاق می‌افتد. چگونه می‌توانیم مشکل تداخل را حل کنیم؟ استراتژی ما این است: اولین نسخه‌ای که پردازش می‌شود برنده می‌شود و نسخه‌ای که بعداً پردازش می‌شود یک تداخل را دریافت می‌کند. شکل ۸-۱۵ نمونه‌ای از تداخل در همگام سازی را نشان می‌دهد.

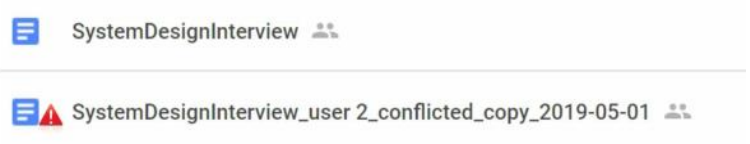


شکل ۸-۱۵

در شکل ۸-۱۵، کاربر ۱ و کاربر ۲ سعی می‌کنند یک فایل را به طور هم‌زمان به‌روزرسانی کنند، اما فایل کاربر ۱ ابتدا توسط سیستم ما پردازش می‌شود. عملیات به‌روزرسانی کاربر ۱

1 Sync conflicts

انجام می‌شود، اما کاربر ۲ با یک تضاد همگام‌سازی[1] مواجه می‌شود. چگونه می‌توانیم تداخل مربوط به کاربر ۲ را حل کنیم؟ سیستم ما هر دو نسخه از یک فایل را ارائه می‌دهد: نسخه محلی کاربر ۲ و آخرین نسخه از سرور (شکل ۹-۱۵). کاربر ۲ این گزینه را دارد که هر دو فایل را ادغام کند یا یک نسخه روی نسخه دیگر بازنویسی کند.

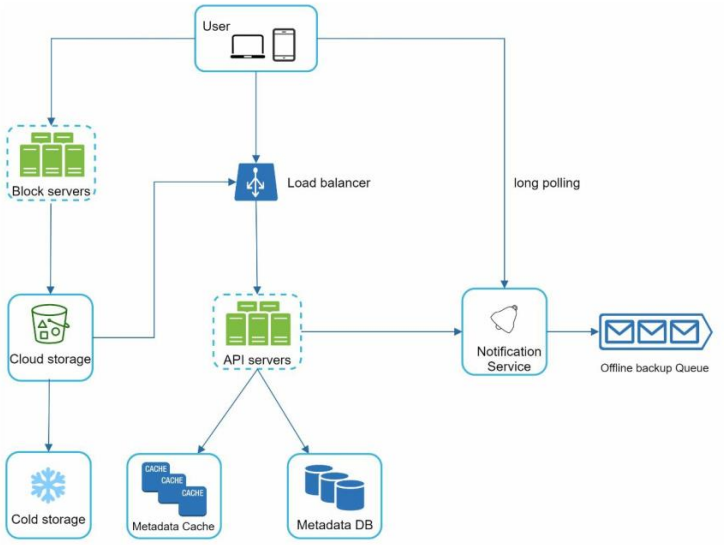


شکل ۹-۱۵

درحالی‌که چندین کاربر هم‌زمان یک سند را ویرایش می‌کنند، همگام نگه‌داشتن یک سند چالش‌برانگیز است. خوانندگان علاقه‌مند باید به مطالب مرجع [۴] [۵] مراجعه کنند.

ادامه طراحی در سطح پیشرفته

شکل ۱۰-۱۵ طرح پیشنهادی سطح پیشرفته‌ای را نشان می‌دهد. اجازه دهید هر یک از اجزای سیستم را بررسی کنیم.



شکل ۱۰-۱۵

1 sync conflict

کاربر-User: کاربر از طریق مرورگر یا اپلیکیشن موبایل از این اپلیکیشن استفاده می‌کند.

سرورهای بلوک (Block servers): سرورهای بلوکی[۱]، بلوک‌ها را به فضای ذخیره‌سازی ابری آپلود می‌کنند. ذخیره‌سازی بلوکی[۲] که به آن ذخیره‌سازی در سطح بلوک گفته می‌شود، یک فناوری برای ذخیره فایل‌های داده در محیط‌های مبتنی برابر است. یک فایل را می‌توان به چندین بلوک تقسیم کرد که هر کدام دارای یک مقدار هش منحصربه‌فرد هستند و در پایگاه‌داده ابرداده[۳] ما ذخیره می‌شوند. هر بلوک به‌عنوان یک شیء[۴] مستقل در نظر گرفته می‌شود و در سیستم ذخیره‌سازی ما (S3) ذخیره می‌شود. برای بازسازی یک فایل، بلوک‌ها به ترتیب خاصی به هم متصل می‌شوند. در مورد اندازه بلوک، ما از Dropbox به‌عنوان مرجع استفاده می‌کنیم: حداکثر اندازه یک بلوک را ۴ مگابایت تعیین می‌کند [۶].

فضای ذخیره‌سازی ابری (Cloud storage): یک فایل به بلوک‌های کوچک‌تر تقسیم شده و در فضای ذخیره‌سازی ابری ذخیره می‌شود.

ذخیره‌سازی سرد (Cold storage):

ذخیره‌سازی سرد یک سیستم کامپیوتری است که برای ذخیره داده‌های غیرفعال طراحی شده است، به این معنی که فایل‌ها برای مدت طولانی قابل‌دسترسی نیستند.

متعادل‌کننده بار: یک متعادل‌کننده بار به طور مساوی درخواست‌ها را بین سرورهای API توزیع می‌کند.

---

۱ منظور از «بلوک» (block) بخش‌های مجزا از یک فایل بزرگ است که برای ذخیره‌سازی بهینه‌تر و کارآمدتر به این صورت تقسیم‌بندی می‌شوند. فضای ذخیره‌سازی ابری (cloud storage) محلی برای ذخیره‌سازی این بلوک‌ها است. سرورهای بلوکی وظیفه دارند تا این بلوک‌ها را از فضای ذخیره‌سازی ابری دانلود کرده و سپس پردازش‌های لازم را روی آن‌ها انجام دهند.

2 Block storage
3 metadata databas
4 object

API servers: اینها تقریباً مسئول همه‌چیز غیر از جریان بارگذاری[1] هستند. سرورهای API برای احراز هویت کاربر، مدیریت پروفایل کاربر، به‌روزرسانی ابرداده فایل[2] و غیره استفاده می‌شوند.

Metadata database :

متادیتای کاربران، فایل‌ها، بلوک‌ها، نسخه‌ها و غیره را ذخیره می‌کند. لطفاً توجه داشته باشید که فایل‌ها در فضای ابری ذخیره می‌شوند و پایگاه‌داده ابرداده فقط حاوی ابرداده است.

Metadata cache : برخی از ابرداده‌ها برای بازیابی سریع ذخیره می‌شوند.

سرویس اطلاع‌رسانی: این یک سیستم انتشار/مشترکین[3] است که اجازه می‌دهد تا داده‌ها از سرویس اطلاع‌رسانی[4] به کلاینت‌ها در صورت وقوع رویدادهای خاص منتقل شوند. در مورد خاص ما، سرویس اعلان به کلاینت‌های مربوطه هنگامی که فایلی در جای دیگری اضافه/ویرایش/حذف می‌شود اطلاع می‌دهد تا بتوانند آخرین تغییرات را انجام دهند.

صف پشتیبان آفلاین (Offline backup queue):

اگر کلاینت آفلاین باشد و نتواند آخرین تغییرات را روی فایل را انجام دهد، صف پشتیبان آفلاین اطلاعات را ذخیره می‌کند تا وقتی کلاینت آنلاین است، تغییرات همگام‌سازی شود. ما در مورد طراحی Google Drive در سطح پیشرفته بحث کرده‌ایم. برخی از اجزاء پیچیده هستند و ارزش بررسی دقیق را دارند. ما اینها را به‌تفصیل در قسمت بررسی عمیق مورد بحث قرار خواهیم داد.

مرحله ۳ - عمیق‌شدن در طراحی

در این بخش، موارد زیر را به‌دقت بررسی خواهیم کرد: سرورهای بلوک، پایگاه‌داده ابرداده، جریان آپلود، جریان دانلود، سرویس اطلاع‌رسانی، ذخیره فضای ذخیره‌سازی و مدیریت خرابی.

---

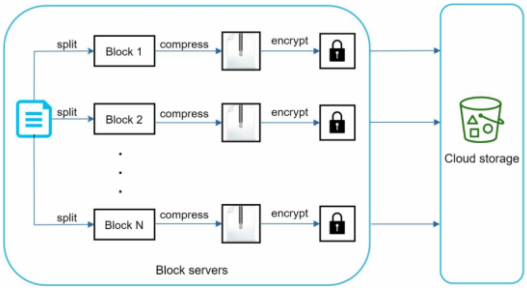
1 uploading flow
2 file metadata
3 publisher/subscriber
4 notification

Block servers

برای فایل‌های حجیمی که به طور مرتب آپدیت می‌شوند، ارسال کل فایل در هر آپدیت پهنای باند زیادی مصرف می‌کند. دو بهینه‌سازی برای به‌حداقل‌رساندن میزان ترافیک شبکه در حال انتقال پیشنهاد شده است:

- **همگام‌سازی تغییرات**[1]. هنگامی که یک فایل اصلاح می‌شود، تنها بلوک‌های اصلاح شده به‌جای کل فایل با استفاده از الگوریتم همگام‌سازی، [۷] [۸] همگام‌سازی می‌شوند.
- **فشرده‌سازی**. اعمال فشرده‌سازی بر روی بلوک‌ها می‌تواند اندازه داده‌ها را به میزان قابل‌توجهی کاهش دهد؛ بنابراین بلوک‌ها با استفاده از الگوریتم‌های فشرده‌سازی بسته به نوع فایل فشرده می‌شوند. به‌عنوان‌مثال، gzip و bzip2 برای فشرده‌سازی فایل‌های متنی استفاده می‌شوند. الگوریتم‌های فشرده‌سازی مختلفی برای فشرده‌سازی تصاویر و فیلم‌ها موردنیاز است.

در سیستم ما، سرورهای بلوکی[2] کار سنگین بارگذاری فایل‌ها را انجام می‌دهند. سرورهای بلوکی، فایل‌های ارسال شده از کلاینت‌ها را با تقسیم یک فایل به بلوک و فشرده‌سازی هر بلوک و رمزگذاری آنها پردازش می‌کنند. به جای آپلود کل فایل در سیستم ذخیره‌سازی، فقط بلوک‌های اصلاح شده منتقل می‌شوند. شکل ۱۱-۱۵ نشان می‌دهد که چگونه یک سرور بلوکی هنگام اضافه شدن یک فایل جدید کار می‌کند.

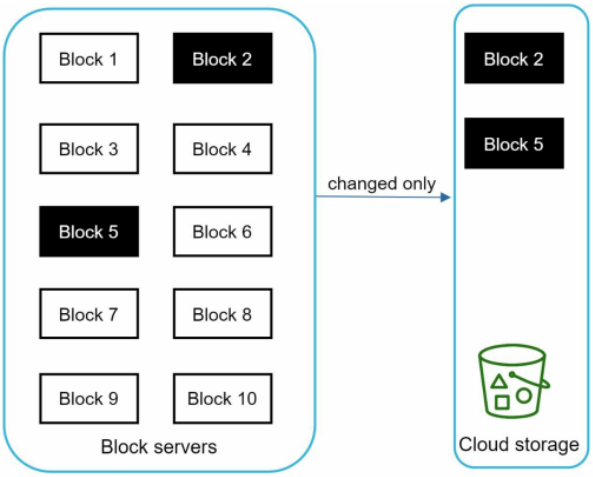


شکل ۱۱-۱۵

1 Delta sync
2 Block servers

- یک فایل به بلوک‌های کوچک‌تر تقسیم می‌شود.
- هر بلوک با استفاده از الگوریتم‌های فشرده‌سازی، فشرده می‌شود.
- برای اطمینان از نظر امنیت، هر بلوک قبل از ارسال به فضای ذخیره‌سازی ابری رمزگذاری می‌شود.
- بلوک‌ها در فضای ذخیره‌سازی ابری آپلود می‌شوند.

شکل ۱۲-۱۵ همگام سازی دلتا را نشان می‌دهد، به این معنی که فقط بلوک‌های اصلاح شده به فضای ذخیره‌سازی ابری منتقل می‌شوند. بلوک‌های برجسته "بلوک ۲" و "بلوک ۵" نشان دهنده بلوک‌های تغییر یافته است. با استفاده از همگام سازی دلتا، فقط آن دو بلوک در فضای ذخیره‌سازی ابری آپلود می‌شوند.



شکل ۱۲-۱۵

سرورهای بلوکی به ما این امکان را می‌دهند که با ارائه همگام‌سازی و فشرده‌سازی دلتا در ترافیک شبکه صرفه‌جویی کنیم.

نیازمندی‌های یکپارچگی بالا

سیستم ما به‌صورت پیش‌فرض، به **یکپارچگی قوی**[1] نیاز دارد. این قابل‌قبول نیست که یک فایل در یک‌زمان مشخص برای کاربران مختلف به‌صورت متفاوتی نمایش داده شود. سیستم برای لایه‌های کش متادیتا و پایگاه‌داده نیازمند است تا یکپارچگی قوی را فراهم کند.

حافظه موقت/کَش به طور پیش‌فرض یک مدل یکپارچگی تدریجی[2] را اتخاذ می‌کند، به این معنی که کپی‌های[3] مختلف ممکن است داده‌های متفاوتی داشته باشند. برای دستیابی به یکپارچگی قوی، باید موارد زیر را تضمین کنیم:

- داده‌های ذخیره شده در نسخه‌های مختلف کش باید دقیقاً با نسخه اصلی[4] که در پایگاه‌داده قرار دارد، یکسان باشند. به‌عبارت‌دیگر، در هر لحظه اطلاعات موجود در تمام کپی‌های کش باید انعکاسی به‌روز و همگام از نسخه اصلی باشد.
- هر زمان تغییری در نسخه اصلی در پایگاه‌داده صورت می‌گیرد، نسخه‌های کَش شده موجود در حافظه نیز باید بلافاصله به‌روز شوند (یا اصطلاحاً "Invalid" شوند) تا مطمئن شویم که داده‌های موجود در کش و پایگاه‌داده کاملاً یکسان هستند.

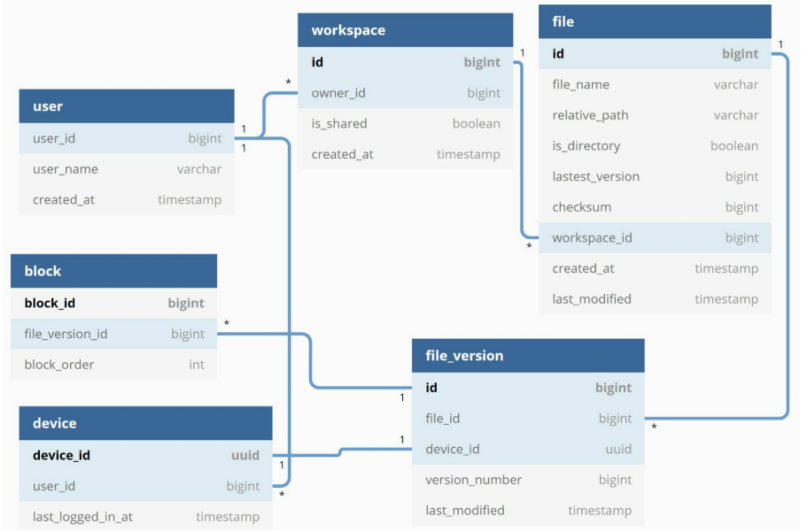
دستیابی به یکپارچگی قوی در یک پایگاه‌داده رابطه‌ای آسان است؛ زیرا ویژگی‌های ACID (اتمی، یکپارچگی، جداسازی، دوام)[5] را حفظ می‌کند [۹]. بااین‌حال، پایگاه‌های داده NoSQL به طور پیش‌فرض از ویژگی‌های ACID پشتیبانی نمی‌کنند. ویژگی‌های ACID باید به‌صورت برنامه‌ریزی‌شده در منطق همگام‌سازی گنجانده شوند. ما در طراحی خود از پایگاه‌داده‌های رابطه‌ای را انتخاب می‌کنیم زیرا ACID به‌صورت پیش‌فرض پشتیبانی می‌شود.

Metadata database

شکل ۱۳-۱۵ طراحی شماتیک پایگاه‌داده را نشان می‌دهد. لطفاً توجه داشته باشید که این یک نسخه بسیار ساده شده است زیرا فقط مهم‌ترین جداول و فیلدهای جالب را شامل می‌شود.

---

1 strong consistency
2 eventual consistency
3 replicas
4 master
5 Atomicity, Consistency, Isolation, Durability



شکل ۱۳-۱۵

کاربر-User: جدول کاربر حاوی اطلاعات اولیه در مورد کاربر مانند نام کاربری، ایمیل، عکس پروفایل و غیره است.

دستگاه-Device: جدول دستگاه اطلاعات دستگاه را ذخیره می‌کند. Push_id برای ارسال و دریافت push notifications موبایل استفاده می‌شود. لطفاً توجه داشته باشید که یک کاربر می‌تواند چندین دستگاه داشته باشد.

Namespace: فضای نام دایرکتوری ریشه یک کاربر است.

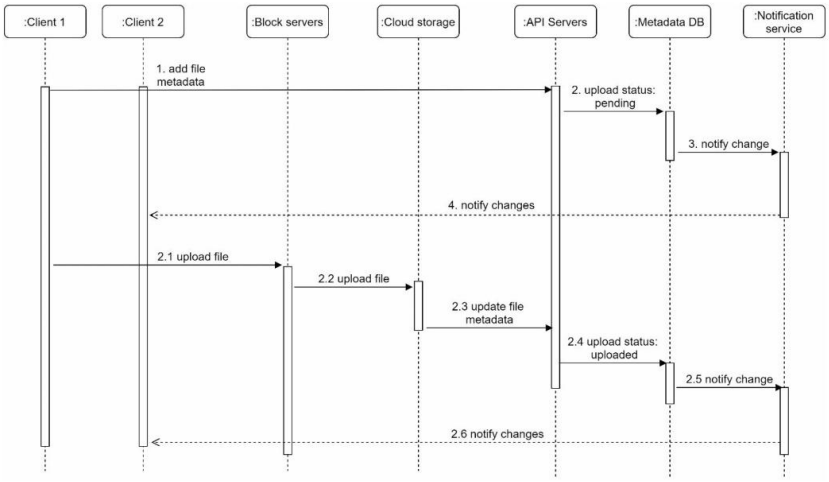
File: جدول فایل همه‌چیز مربوط به آخرین فایل را ذخیره می‌کند.

File_version: تاریخچه نسخه یک فایل را ذخیره می‌کند. ردیف‌های موجود فقط خواندنی هستند تا یکپارچگی تاریخچه ویرایش فایل حفظ شود.

Block: همه‌چیز مربوط به یک بلوک فایل را ذخیره می‌کند. یک فایل از هر نسخه‌ای را می‌توان با پیوستن همه بلوک‌ها به ترتیب صحیح بازسازی کرد.

مسیر به‌روزرسانی

اجازه دهید در مورد اینکه چه اتفاقی می‌افتد زمانی که یک کلاینت یک فایل را آپلود می‌کند صحبت کنیم. برای درک بهتر این جریان، نمودار توالی[1] را همان‌طور که در شکل ۱۴-۱۵ نشان‌داده‌شده است ترسیم می‌کنیم.



شکل ۱۴-۱۵

در شکل ۱۴-۱۵، دو درخواست به‌صورت موازی ارسال می‌شود: که این دو مورد، افزودن ابرداده فایل و آپلود فایل در فضای ذخیره‌سازی ابری هستند و هر دو درخواست از کلاینت ۱ سرچشمه می‌گیرند.

- اضافه‌کردن ابرداده فایل.

۱- کلاینت شماره ۱ درخواستی برای افزودن ابرداده فایل جدید ارسال می‌کند.

۲- فراداده فایل جدید را در DB ابرداده ذخیره کنید و وضعیت آپلود فایل را به «در انتظار[2]» تغییر دهید.

۳- به سرویس اطلاع‌رسانی/ نوتیفیکیشن اطلاع دهید که فایل جدیدی در حال اضافه‌شدن است.

---

1 sequence diagram
2 pending

۴- سرویس نوتیفیکیشن به کلاینت‌ها مربوطه (کلاینت شماره ۲) اطلاع می‌دهد که یک فایل در حال آپلود است.

- آپلود فایل‌ها در فضای ذخیره‌سازی ابری.
  - ۲.۱- کلاینت ۱ محتوای فایل را برای مسدود کردن سرورها آپلود می‌کند.
  - ۲.۲- سرورهای بلوکی فایل‌ها را به بلوک‌ها یا تِکه‌هایی تقسیم می‌کنند، آنها را فشرده کرده و سپس رمزگذاری می‌کنند و در فضای ذخیره‌سازی ابری آپلود می‌کنند.
  - ۲.۳- پس از آپلود فایل، فضای ذخیره‌سازی ابری[1] یک تابع بازگشت[2] برای تکمیل آپلود را فراخوانی می‌کند. در این لحظه، درخواستی به سرورهای API ارسال می‌شود.
  - ۲.۴- وضعیت فایل به «**آپلود شده**[3]» در Metadata DB تغییر کرد.
  - ۲.۵- در نتیجه به سرویس نوتیفیکیشن اطلاع دهید که وضعیت یک فایل به «آپلود شده» تغییر کرده است.
  - ۲.۶- سرویس نوتیفیکیشن [4] به کلاینت‌های مربوطه (نوتیفیکیشن ۲) اطلاع می‌دهد که یک فایل به طور کامل آپلود شده است.

هنگامی که یک فایل ویرایش می‌شود، این مسیر مشابه است، بنابراین ما آن را تکرار نمی‌کنیم.

مسیر دانلود

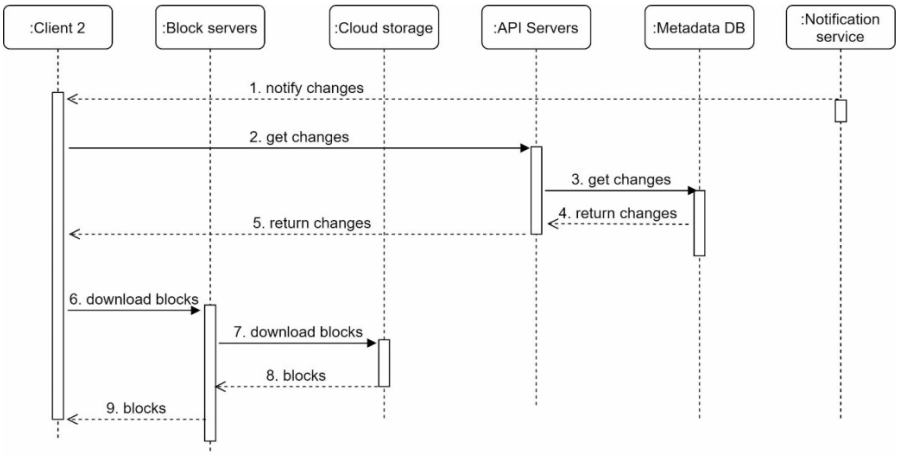
مسیر دانلود زمانی فعال می‌شود که یک فایل در جای دیگری اضافه یا ویرایش شود. چگونه یک کلاینت متوجه می‌شود که یک فایل توسط کلاینت دیگری اضافه یا ویرایش شده است؟ دوراه وجود دارد که کلاینت می‌تواند این مورد را بداند:

---

1 cloud storage
2 callback
3 uploaded
4 notification service

- درصورتی‌که کاربر A در حال آنلاین بودن باشد و فایلی توسط کاربر دیگری تغییر کند، سرویس نوتیفیکیشن به کاربر A اطلاع می‌دهد که تغییراتی صورت‌گرفته است و نیاز است تا آخرین نسخه فایل را دریافت کند.
- درصورتی‌که کاربر A در زمان تغییر فایل توسط کاربر دیگر، آفلاین باشد، داده‌ها در حافظه موقت/کَش ذخیره می‌شوند. هنگامی که کاربر آفلاین دوباره آنلاین می‌شود، می‌تواند آخرین تغییرات را دریافت کند.

هنگامی که یک کاربر می‌داند که یک فایل تغییر کرده است، ابتدا متادیتا را از طریق سرورهای API درخواست می‌کند، سپس بلوک‌ها را برای ساخت فایل را دانلود می‌کند. شکل ۱۵-۱۵ مسیر دقیق را نشان می‌دهد. توجه داشته باشید که به دلیل محدودیت فضا تنها مهم‌ترین اجزا در نمودار نشان‌داده‌شده است.



شکل ۱۵-۱۵

۱. سرویس نوتیفیکیشن به کاربر شماره ۲ اطلاع می‌دهد که یک فایل در جایی تغییر کرده است.

۲. هنگامی که کاربر شماره ۲ می‌داند که به‌روزرسانی‌های جدید در دسترس است، درخواستی برای واکشی متادیتا ارسال می‌کند.

۳. سرورهای API برای دریافت اطلاعات (متادیتا) مربوط به تغییرات، پایگاه‌داده حاوی متادیتا[1] را فراخوانی می‌کنند.
۴. متادیتا به سرورهای API بازگردانده می‌شود.
۵. کاربر شماره ۲ متادیتا را دریافت می‌کند.
۶. هنگامی که کاربری متادیتا را دریافت کرد، درخواست‌هایی برای سرورهای بلوکی جهت دانلود بلوک‌ها ارسال می‌کند.
۷. سرورهای بلوکی ابتدا بلوک‌ها را از فضای ذخیره‌سازی ابری دانلود می‌کنند.
۸. فضای ذخیره‌سازی ابری بلوک‌ها را به سرورهای بلوکی بر می‌گرداند.
۹. کاربر شماره ۲ تمام بلوک‌های جدید را برای بازسازی فایل را دانلود می‌کند.

سرویس اطلاع‌رسانی

برای حفظ یکپارچگی فایل[2]، هر گونه تغییری در یک فایل که به‌صورت محلی انجام می‌شود، باید به سایر کاربران اطلاع داده شود تا از بروز تداخل[3] جلوگیری شود. سرویس نوتیفیکیشن برای همین منظور ساخته شده است. این سرویس به‌طورکلی، امکان انتقال داده‌ها به کاربران را در زمان وقوع رویدادها فراهم می‌کند. در اینجا چند گزینه برای پیاده‌سازی این سرویس وجود دارد:

- Long polling. شرکت Dropbox از Long polling استفاده می‌کند [۱۰].
- WebSocket . وب سوکت یک ارتباط دائمی بین کلاینت و سرور فراهم می‌کند که این ارتباط دوطرفه است.

اگرچه هر دو گزینه به‌خوبی کار می‌کنند، ما به دو دلیل زیر Long polling را انتخاب می‌کنیم:

- ارتباط برای سرویس نوتیفیکیشن دوطرفه نیست. سرور، اطلاعات مربوط به تغییرات فایل را برای کلاینت ارسال می‌کند، اما نه برعکس.

---

1 metadata DB
2 file consistency
3 conflicts

- WebSocket برای ارتباطات دوطرفه بلادرنگ مانند برنامه چت مناسب است. برای Google Drive، نوتیفیکیشن‌ها به‌ندرت به‌صورت انبوه ارسال می‌شوند.

با Long polling، هر کلاینت یک ارتباط Long polling با سرویس نوتیفیکیشن برقرار می‌کند. اگر تغییرات در یک فایل شناسایی شود، کلاینت اتصال Long polling را می‌بندد. بستن اتصال به این معنی است که یک کلاینت برای دانلود آخرین تغییرات باید به سرور متادیتا متصل شود. پس از دریافت پاسخ یا پایان‌یافتن زمان اتصال[1]، کلاینت بلافاصله درخواست جدیدی برای باز نگه‌داشتن اتصال ارسال می‌کند.

صرفه‌جویی در فضای ذخیره‌سازی

برای پشتیبانی از تاریخچه نسخه‌های مختلف فایل‌ها و قابل‌اطمینان بودن برنامه، چندین نسخه از یک فایل در چندین مرکز داده[2] ذخیره می‌شود. فضای ذخیره‌سازی را می‌توان با پشتیبان‌گیری مکرر از تمام ویرایش‌های فایل به‌سرعت پر کرد. سه تکنیک برای کاهش هزینه‌های ذخیره‌سازی پیشنهاد شده است:

- حذف بلوک‌های تکراری داده. حذف بلوک‌های اضافی در سطح حساب کاربری یک راه آسان برای صرفه‌جویی در فضا است. دو بلوک زمانی یکسان هستند اگر مقدار هش یکسانی داشته باشند.
- یک استراتژی پشتیبان‌گیری هوشمند از داده‌ها اتخاذ کنید. دو استراتژی بهینه‌سازی را می‌توان اعمال کرد:
  - یک محدودیت اعمال کنید: می‌توانیم محدودیتی برای تعداد نسخه‌های ذخیره‌سازی تعیین کنیم. در صورت رسیدن به حد مجاز، قدیمی‌ترین نسخه با نسخه جدید جایگزین می‌شود.
  - فقط نسخه‌های ارزشمند را نگه دارید: برخی از فایل‌ها ممکن است مرتباً ویرایش شوند. به‌عنوان‌مثال، ذخیره هر نسخه ویرایش شده برای یک سند به‌شدت اصلاح شده

1 timeout
2 data centers

می‌تواند به این معنی باشد که فایل در مدت کوتاهی بیش از ۱۰۰۰ بار ذخیره می‌شود. برای جلوگیری از کپی‌های غیر ضروری، می‌توانیم تعداد نسخه‌های ذخیره شده را محدود کنیم. ما به نسخه‌های اخیر وزن بیشتری می‌دهیم. انجام آزمایش‌های تجربی برای تعیین تعداد بهینه نسخه‌های ذخیره‌سازی شده مفید است.

- انتقال داده‌های کمتر مورد استفاده به cold storage. داده‌های سرد داده‌هایی هستند که ماه‌ها یا سال‌ها فعال نبوده‌اند. cold storage مانند Amazon S3 glacier [۱۱] بسیار ارزان‌تر از S3 است.

رسیدگی به خطاها

شکست‌ها و خطاها می‌توانند در یک سیستم در مقیاس بزرگ رخ دهند و ما باید استراتژی‌های طراحی را برای رفع این شکست‌ها اتخاذ کنیم. ممکن است مصاحبه‌کننده شما علاقه‌مند به شنیدن نحوه مدیریت خرابی‌های سیستم زیر باشد:

- **خرابی توزیع‌کننده بار**[1]: اگر یک توزیع‌کننده بار خراب شود، توزیع‌کننده ثانویه فعال شده و ترافیک را برعهده می‌گیرد. توزیع‌کننده‌های بار معمولاً با استفاده از ضربان قلب[2] که یک سیگنال دوره‌ای است که بین بار توزیع‌کننده‌ها ارسال می‌شود، یکدیگر را کنترل می‌کنند. اگر یک توزیع‌کننده بار برای مدتی ضربان قلب را ارسال نکرده باشد، ازکارافتاده و معیوب در نظر گرفته می‌شود.
- **خرابی یک بلاک سرور**: اگر سرور بلوکی از کار بیفتد، سایر سرورها کارها را ناتمام شده یا در انتظار در نظر می‌گیرند.
- **خرابی ذخیره‌سازی ابری**: سطل‌های S3 چندین بار در مناطق مختلف تکثیر[3] می‌شوند. اگر فایل‌ها در یک منطقه در دسترس نباشند، می‌توان آن‌ها را از مناطق مختلف واکشی کرد.

---

1 Load balancer
2 heartbeat
3 replicated

- **خرابی سرور API**: این مورد یک سرویس stateless است. اگر سرور API از کار بیفتد، ترافیک توسط یک توزیع‌کننده بار به سرورهای دیگر API هدایت می‌شود.
- **خرابی کَش متادیتا**: سرورهای کش متادیتا چندین بار تکثیر[1] می‌شوند. اگر یک گره[2] پایین بیاید، همچنان می‌توانید به سایر گره‌ها برای واکشی داده‌ها دسترسی داشته باشید. ما یک سرور کش جدید برای جایگزینی سرور ناموفق ایجاد می‌کنیم.
- **خرابی پایگاه‌داده متادیتا (Metadata DB)**.
  - ازکارافتادن سرور Master: اگر سرور Master از کار بی افتد، یکی از Slaveها را ارتقا دهید تا به‌عنوان یک master جدید عمل کند و یک Slave Node جدید ایجاد کند.
  - ازکارافتادن سرور Slave: اگر یک Slave خراب است، می‌توانید از Slave دیگری برای عملیات خواندن استفاده کنید و سرور پایگاه‌داده دیگری را جایگزین سرور خراب کنید.
- **خرابی سرویس نوتیفیکیشن**: هر کاربر آنلاین یک ارتباط long poll با سرور نوتیفیکیشن برقرار می‌کند؛ بنابراین، هر سرور نوتیفیکیشن به بسیاری از کاربران متصل است. طبق مقاله Dropbox در سال ۲۰۱۲ [۶]، بیش از ۱ میلیون اتصال در هر دستگاه/ماشین باز است. اگر یک سرور از کار بیفتد، تمام اتصالات long poll از بین می‌رود، بنابراین کلاینت‌ها باید دوباره به سرور دیگری متصل شوند. حتی اگر یک سرور می‌تواند بسیاری از اتصالات باز را حفظ کند، نمی‌تواند همه اتصالات از دست رفته را یک مرتبه، به‌صورت مجدد وصل کند. برقراری ارتباط مجدد با همه کلاینت‌های از دست رفته فرایند نسبتاً آهسته‌ای است.

---

1 replicated
2 node

- خرابی در صف پشتیبان گیری آفلاین: صف‌ها چندین بار تکثیر می‌شوند. اگر یک صف با شکست و خرابی مواجه شود، مصرف‌کنندگان[1] صف ممکن است نیاز به اشتراک مجدد[2] در صف پشتیبان داشته باشند.

مرحله ٤ – جمع‌بندی

در این فصل، ما یک طراحی سیستم برای پشتیبانی Google Drive پیشنهاد کردیم. ترکیبی از یکپارچگی قوی[3]، پهنای باند کم شبکه و همگام‌سازی سریع طراحی را جالب می‌کند. طراحی ما شامل دو مسیر است: مدیریت متادیتا فایل و همگام‌سازی فایل. سرویس نوتیفیکیشن یکی دیگر از اجزای مهم سیستم است. از long poll برای به‌روز نگه‌داشتن کاربران با تغییرات فایل استفاده می‌کند.

مانند هر سؤال مصاحبه طراحی سیستم، هیچ راه‌حل کاملی وجود ندارد. هر شرکتی محدودیت‌های منحصربه‌فرد خود را دارد و شما باید سیستمی را متناسب با این محدودیت‌ها طراحی کنید. دانستن گزینه‌های مورد انتخاب جهت طراحی و فناوری مورد استفاده توسط شما، مهم است. اگر چند دقیقه باقی‌مانده است، می‌توانید در مورد انتخاب‌های مختلف طراحی صحبت کنید.

برای مثال، می‌توانیم فایل‌ها را به‌جای رفتن از سرورهای بلوکی، مستقیماً از کلاینت در فضای ذخیره‌سازی ابری آپلود کنیم. مزیت این روش این است که آپلود فایل را سریع‌تر می‌کند؛ زیرا یک فایل فقط باید یک‌بار به فضای ذخیره‌سازی ابری منتقل شود. در طراحی ما، یک فایل ابتدا به سرورهای بلوکی و سپس به فضای ذخیره‌سازی ابری منتقل می‌شود. بااین‌حال، رویکرد جدید دارای چند اشکال است:

- نخست، منطق شارد شدن[4]، فشرده‌سازی و رمزگذاری یکسان باید در پلتفرم‌های مختلف (iOS، Android، Web) اجرا شود که این مسئله مستعد خطا است و به

---

1 consumers
2 re-subscribe
3 strong consistency
4 chunking

تلاش زیادی نیاز دارد. در این طراحی، تمام آن منطق‌ها در یک مکان متمرکز پیاده‌سازی می‌شوند که «سرورهای بلوک» نام دارد.

- دوم، از آنجایی که یک کلاینت می‌تواند به‌راحتی هک یا دست‌کاری شود، پیاده‌سازی منطق رمزگذاری در سمت کلاینت ایده‌آل نیست.

یکی دیگر از تحولات جالب سیستم انتقال منطق آنلاین/آفلاین به یک سرویس جداگانه است. بگذارید آن را سرویس حضوروغیاب[1] بنامیم. با انتقال سرویس حضوروغیاب به خارج از سرورهای نوتیفیکیشن، عملکرد آنلاین/آفلاین می‌تواند به‌راحتی توسط سایر سرویس‌ها یکپارچه شود.

1 presence service

منابع و مراجع فصل ۱۵

[1] Google Drive: https://www.google.com/drive/
[2] Upload file data: https://developers.google.com/drive/api/v2/manage-uploads
[3] Amazon S3: https://aws.amazon.com/s3
[4] Differential Synchronization https://neil.fraser.name/writing/sync/
[5] Differential Synchronization youtube talk https://www.youtube.com/watch?v=S2Hp_1jqpY8
[6] How We’ve Scaled Dropbox: https://youtu.be/PE4gwstWhmc
[7] Tridgell, A., & Mackerras, P. (1996). The rsync algorithm.
[8] Librsync. (n.d.). Retrieved April 18, 2015, from https://github.com/librsync/librsync
[9] ACID: https://en.wikipedia.org/wiki/ACID
[10] Dropbox security white paper: https://www.dropbox.com/static/business/resources/Security_Whitepaper.pdf
[11] Amazon S3 Glacier: https://aws.amazon.com/glacier/faqs/

فصل ۱۶

ادامه یادگیری

طراحی سیستم‌های خوب نیازمند سال‌ها انباشت دانش است. یک میان‌بر این است که به معماری سیستم‌های دنیای واقعی بپردازید. در زیر مجموعه‌ای از مطالب مفید برای خواندن آورده شده است. ما به شما توصیه می‌کنیم که هم به اصول مشترک و هم به فناوری‌های اساسی توجه کنید. تحقیق در مورد هر فناوری و درک این که چه مشکلاتی را حل می‌کند، راهی عالی برای تقویت پایه دانش شما و اصلاح فرایند طراحی است.

مثال‌هایی از دنیای واقعی

1. Facebook Timeline: Brought To You By The Power Of Denormalization: https://goo.gl/FCNrbm
2. Scale at Facebook: https://goo.gl/NGTdCs
3. Building Timeline: Scaling up to hold your life story: https://goo.gl/8p5wDV
4. Erlang at Facebook (Facebook chat): https://goo.gl/zSLHrj
5. Facebook Chat: https://goo.gl/qzSiWC
6. Finding a needle in Haystack: Facebook’s photo storage: https://goo.gl/edj4FL
7. Serving Facebook Multifeed: Efficiency, performance gains through redesign:
8. https://goo.gl/adFVMQ

9. Scaling Memcache at Facebook: https://goo.gl/rZiAhX
10. TAO: Facebook's Distributed Data Store for the Social Graph: https://goo.gl/Tk1DyH
11. Amazon Architecture: https://goo.gl/k4feoW
12. Dynamo: Amazon's Highly Available Key-value Store: https://goo.gl/C7zxDL
13. A 360 Degree View Of The Entire Netflix Stack: https://goo.gl/rYSDTz
14. It's All A/Bout Testing: The Netflix Experimentation Platform: https://goo.gl/agbA4K
15. Netflix Recommendations: Beyond the 5 stars (Part 1): https://goo.gl/A4FkYi
16. Netflix Recommendations: Beyond the 5 stars (Part 2): https://goo.gl/XNPMXm
17. Google Architecture: https://goo.gl/dvkDiY
18. The Google File System (Google Docs): https://goo.gl/xj5n9R
19. Differential Synchronization (Google Docs): https://goo.gl/9zqG7x
20. YouTube Architecture: https://goo.gl/mCPRUF
21. Seattle Conference on Scalability: YouTube Scalability: https://goo.gl/dH3zYq
22. Bigtable: A Distributed Storage System for Structured Data: https://goo.gl/6NaZca
23. Instagram Architecture: 14 Million Users, Terabytes Of Photos, 100s Of Instances, Dozens
24. Of Technologies: https://goo.gl/s1VcW5
25. The Architecture Twitter Uses To Deal With 150M Active Users: https://goo.gl/EwvfRd
26. Scaling Twitter: Making Twitter 10000 Percent Faster: https://goo.gl/nYGC1k
27. Announcing Snowflake (Snowflake is a network service for generating unique ID numbers at
28. high scale with some simple guarantees): https://goo.gl/GzVWYm
29. Timelines at Scale: https://goo.gl/8KbqTy
30. How Uber Scales Their Real-Time Market Platform: https://goo.gl/kGZuVy
31. Scaling Pinterest: https://goo.gl/KtmjW3
32. Pinterest Architecture Update: https://goo.gl/w6rRsf
33. A Brief History of Scaling LinkedIn: https://goo.gl/8A1Pi8
34. Flickr Architecture: https://goo.gl/dWtgYa

35. How We've Scaled Dropbox: https://goo.gl/NjBDtC
36. The WhatsApp Architecture Facebook Bought For $19 Billion: https://bit.ly/2AHJnFn

وبلاگ‌های شرکت‌های فعال در حوزه تکنولوژی

اگر قصد دارید با یک شرکت مصاحبه کنید، خواندن وبلاگ‌های مهندسی آن‌ها و آشنایی با فناوری‌ها و سیستم‌هایی که در آنجا به کار گرفته شده و اجرا شده‌اند، ایده خوبی است. علاوه بر این، وبلاگ‌های مهندسی بینش ارزشمندی در مورد زمینه‌های خاص ارائه می‌دهند. خواندن مرتب آنها می‌تواند به ما کمک کند تا مهندس بهتری شویم.

در اینجا لیستی از وبلاگ‌های مهندسی شرکت‌ها و استارت‌آپ‌های بزرگ شناخته شده است.

1. Airbnb: https://medium.com/airbnb-engineering
2. Amazon: https://developer.amazon.com/blogs
3. Asana: https://blog.asana.com/category/eng
4. Atlassian: https://developer.atlassian.com/blog
5. Bittorrent: http://engineering.bittorrent.com
6. Cloudera: https://blog.cloudera.com
7. Docker: https://blog.docker.com
8. Dropbox: https://blogs.dropbox.com/tech
9. eBay: http://www.ebaytechblog.com
10. Facebook: https://code.facebook.com/posts
11. GitHub: https://githubengineering.com
12. Google: https://developers.googleblog.com
13. Groupon: https://engineering.groupon.com
14. Highscalability: http://highscalability.com
15. Instacart: https://tech.instacart.com
16. Instagram: https://engineering.instagram.com
17. Linkedin: https://engineering.linkedin.com/blog
18. Mixpanel: https://mixpanel.com/blog
19. Netflix: https://medium.com/netflix-techblog
20. Nextdoor: https://engblog.nextdoor.com
21. PayPal: https://www.paypal-engineering.com
22. Pinterest: https://engineering.pinterest.com
23. Quora: https://engineering.quora.com

24. Reddit: https://redditblog.com
25. Salesforce: https://developer.salesforce.com/blogs/engineering
26. Shopify: https://engineering.shopify.com
27. Slack: https://slack.engineering
28. Soundcloud: https://developers.soundcloud.com/blog
29. Spotify: https://labs.spotify.com
30. Stripe: https://stripe.com/blog/engineering
31. System design primer: https://github.com/donnemartin/system-design-primer
32. Twitter: https://blog.twitter.com/engineering/en_us.html
33. Thumbtack: https://www.thumbtack.com/engineering
34. Uber: http://eng.uber.com
35. Yahoo: https://yahooeng.tumblr.com
36. Yelp: https://engineeringblog.yelp.com
37. Zoom: https://medium.com/zoom-developer-blog

سخن پایانی

تبریک می‌گویم! شما در انتهای این راهنمای مصاحبه هستید. شما مهارت‌ها و دانش را برای طراحی سیستم‌ها کسب کرده‌اید. همه این نظم و انضباط را ندارند که آموخته‌های شما را یاد بگیرند. یک‌لحظه وقت بگذارید و به مسیر پشت سر گذاشته نگاه کنید. زحمات شما نتیجه خواهد داد. رسیدن به یک شغل رؤیایی یک سفر طولانی است و به زمان و تلاش زیادی نیاز دارد. تمرین، معجزه می‌کند. موفق باشید!

ممنون از خرید و خواندن این کتاب. بدون خوانندگانی مثل شما، کار ما وجود نداشت. امیدواریم از کتاب لذت برده باشید! اگر مشکلی ندارید، لطفاً این کتاب را در آمازون مرور کنید: https://tinyurl.com/y7d3ltbc این به من کمک می‌کند خوانندگان شگفت‌انگیز بیشتری مانند شما را جذب کنم. اگر می‌خواهید در صورت دردسترس‌بودن فصل‌های جدید مطلع شوید، لطفاً در فهرست ایمیل ما مشترک شوید: https://bit.ly/3dtIcsE

ISBN : 978-622-302-684-3

نشر گنجور

9 786223 026843