دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش کارآموزی

محل کارآموزی: شرکت سامانه گستر سحاب‌پرداز

نام استاد کارآموزی

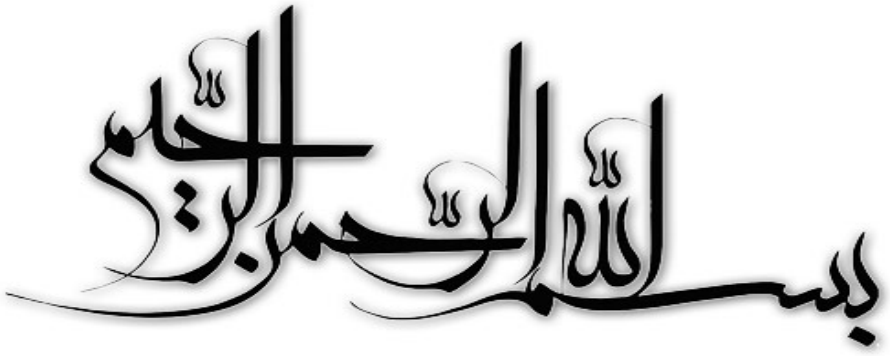
دکتر مسعود صبائی

نام دانشجو

امیرمحمد پیرحسینلو

۹۵۳۱۰۱۴

تابستان ۱۳۹۸


بسم الله الرحمن الرحيم

چکیده

این دوره‌ی کارآموزی حول محور داده‌های حجیم برگزار شد و در آن با مفاهیم و ابزارهای پردازش داده‌ها آشنا شدیم، برنامه نویسی سیستم‌های توزیع شده را فراگرفتیم و با انجام دو پروژه بر مهارت‌های فنی خود افزودیم. جدا از نکات فنی، اجرای یک کار گروهی ۶ نفره از نکات برجسته‌ی این دوره بود که باعث شد کار در یک محیط واقعی با یک پروژه واقعی را تجربه کنیم. هدف از این گزارش، شرح فعالیت‌های انجام شده و مطالب فراگرفته در این دوره است که در هفت فصل به آن‌ها خواهم پرداخت.

واژه‌های کلیدی:

داده‌های حجیم، معماری توزیع شده، جاوا، پایگاه داده، موتور جستجو

فهرست مطالب

فهرست اشکال

فهرست اختصارات

مخفف	فرم کامل
$HDFS$	**Hadoop Distributed File System**
$HTML$	**HyperText Markup Language**
$HTTP$	**HyperText Transfer Protocol**
$JSON$	**JavaScript Object Notation**
JVM	**Java Virtual Machine**
RTM	**Real Time Messaging**
URL	**Uniform Resource Locator**
$YARN$	**Yet Another Resource Negotiator**

فصل اول

مقدمه

در این دوره‌ی کارآموزی، با ابزارها و مفاهیم جدیدی در حوزه‌ی داده‌های حجیم آشنا شدیم. هدف از این دوره یادگیری کار با ابزارهای برنامه نویسی، پایگاه داده‌های توزیع شده و سیستم عامل لینوکس بود که تا حد خیلی خوبی با آن‌ها آشنا شدیم. دو پروژه تعریف شد و در قالب تیم‌های ۶ نفره پیاده‌سازی شد. در هر کدام از پروژه‌ها ابتدا با ابزارهای مورد نیاز برای انجام پروژه آشنا شدیم، سپس طراحی معماری نرم‌افزار صورت گرفت و بعد از آن به پیاده سازی و برنامه نویسی پرداختیم.

در فصل دوم به معرفی شرکت سحاب پرداز و زمینه‌ی کاری آن می‌پردازیم، سپس به سراغ روند کارآموزی در این شرکت می‌رویم. در فصل سوم، سراغ پروژه اول و روند پیاده سازی آن می‌رویم.

در فصل‌های چهارم و پنجم، ابزارهای مورد نیاز برای ساخت پروژه دوم که یک موتور جستجو است، معماری نرم‌افزار آن و نحوه پیاده سازی و برنامه نویسی آن شرح داده خواهد شد.

در فصل ششم ایده‌هایی برای بهبود عملکرد موتور جستجوی ساخته شده در پروژه دوم مطرح می‌شود و نحوه پیاده‌سازی یکی از ایده‌ها توضیح داده می‌شد.

فصل آخر هم به نتیجه‌گیری اختصاص یافته است.

فصل دوم
معرفی شرکت سحاب پرداز و روند کارآموزی در این شرکت

۲-۱ مقدمه

در این فصل ابتدا به معرفی شرکت سحاب پرداز و زمینه‌ی کاری آن می‌پردازیم، سپس به سراغ روند کارآموزی در این شرکت می‌رویم و در انتها مطالبی که در هفته‌ی اول یادگرفتیم را بازگو می‌کنیم.

۲-۲ معرفی شرکت سحاب پرداز

این شرکت خصوصی در سال ۱۳۹۳ تاسیس شده است و هم اکنون در زمینه‌های مختلفی فعالیت می‌کند که مهم‌ترین آن **Big Data**[۱] ست و در حال ایجاد محصولات خاص و تجربیات جدید در این زمینه است. ارزشمندترین سرمایه سحاب، اعضای خانواده و فرهنگی است که با آن زندگی می‌کنیم. اعضای این شرکت را ۷۵ نفر تشکیل می‌دهند که در بخش‌های منابع انسانی، زیرساخت، مدیریت و ... فعالیت می‌کنند.

آدرس: خیابان سهروردی، خیابان خرمشهر، پ ۲۴

شماره تماس: ۸۸۷۶۲۶۷۲ (۰۲۱)

ایمیل: **info@sahab.ir**

۲-۳ روند کارآموزی

۲-۳-۱ نیمبو

نیمبو نام دوره کارآموزی فشرده است که تابستان امسال توسط شرکت سحاب پرداز برگزار می‌شود و مسیری است برای توسعه مهارت‌های فنی و غیر فنی که با آن می توان به دنیای حرفه‌ای‌ها وارد شد. در این دوره اهداف یادگیری متنوعی مبتنی بر پروژه در نظر گرفته شده است که در حوزه **Big Data** و **Data Science**[۲] است. از همه مهم‌تر مربی‌هایی در کنار ما هستند که در یادگیری به ما و تیم‌مان کمک می‌کنند. این مربی‌ها از با تجربه‌ترین همکاران سحاب انتخاب شده‌اند. آموزش و مهارت‌های دوره نیمبو طوری طراحی شده است که در آینده بتوانیم کاربردش را در مسیر شغلی و تحصیلی‌مان ببینیم.

۲-۳-۲ مدت زمان کارآموزی

کارآموزی در ۶ هفته برگزار می‌شود که در هر هفته از شنبه تا چهارشنبه از ساعت ۹ صبح تا ۶ بعد از ظهر باید در شرکت حضور داشته باشیم.

۲-۳-۳ گروه بندی

در این دوره ۱۸ نفر کارآموز حضور دارند که برای پروژه اول به ۹ تیم دو نفره و برای پروژه دوم به ۳ تیم ۶ نفره تقسیم می‌شوند.

۲-۳-۴ پروژه اول

پروژه اول در رابطه با آشنایی با **real time messaging (rtm)** است. در این پروژه قصد داریم به عنوان **Subscriber** از کانال **github-events** داده‌ها را دریافت کنیم و با پردازش داده‌ها افرادی که بیشترین فعالیت را در **github** داشته‌اند را بیابیم. بیشترین فعالیت بر اساس تعداد **commit** ها و **push** ها و چند متغیر دیگر تعریف می‌شود. جزئیات بیشتر درمورد روند کار با یک پروتکل **rtm**

[۱] داده‌ی حجیم
[۲] علم داده

در سایت **satori** [۳] موجود است. زبان برنامه نویسی جاوا به همراه ابزار مدیریت پکیج **maven** [۴] برای پیاده سازی این پروژه انتخاب شده است.

۲-۳-۵ پروژه دوم

تعریف پروژه دوم پس از پایان پروژه اول در اختیار ما قرار می‌گیرد.

۲-۴ C++ vs Java

اولین کارگاه آموزشی در این هفته به مدت یک ساعت و نیم برگزار شد و محور آن بحث درمورد تفاوت‌های زبان‌های برنامه نویسی **Java** و **C++** بود. این کارگاه به این دلیل برگزار شد که تعدادی از کارآموزها در درس برنامه نویسی پیشرفته زبان **C++** را یاد گرفته بودند و با **Java** آشنایی نداشتند. مهم‌ترین تفاوت‌ها به شرح زیر است:

۲-۴-۱ نبود **garbage collector** در **C++**

زبان **C++** فاقد **garbage collector** است به همین دلیل برنامه نویس باید دقت کند هر جا که نیاز به یک **object** [۵] از بین می‌رود باید **destructor** [۶] آن را فراخوانی کند و هر جا نیاز به یک آرایه ندارد آن را مستقیما **delete** [۷] کند در غیر این صورت فضای حافظه اختصاص داده شده به این موارد در حافظه سیستم آزاد نشده و در اختیار سیستم عامل قرار نمی‌گیرد.

۲-۴-۲ ارث بری چندگانه

یکی از ویژگی‌های زبان برنامه نویسی **C++** ، ارث بری چندگانه [۸] است که در جاوا وجود ندارد. این ویژگی مزایا و معایب خود را دارد که از آن‌ها صرف نظر می‌کنیم اما برنامه نویس باید دقت کند که در صورت استفاده از ارث بری چندگانه ممکن است به مشکل **diamond** [۹] برخورد کند.

۲-۴-۳ جاوا مستقل از سکو است

یکی از مزایای عالی زبان **Java** ، مستقل بودن آن از سیستم عامل [۱۰] یا سکو است. برنامه‌های جاوا در سیستم عاملی که **Java Virtual Machine(JVM)** در آن نصب است اجرا می‌شود و برنامه نویس را از نوشتن چند برنامه برای سیستم عامل‌های مختلف رها می‌سازد. اما زبان **C++** این گونه نیست و به دلیل وجود مترجم‌های مختلف مانند **g++**، **visual studio compiler** و ... برای هر سیستم عامل، ممکن است کدی که برای یک سیستم عامل نوشته شده باشد برای سیستم عامل دیگر قابل اجرا نباشد.

تفاوت‌های دیگری هم وجود دارد که در گزارش به موارد بالا اکتفا می‌کنیم.

---

[۳] https://www.satori.com/docs/rtm-sdks/tutorials/java-sdk-quickstart

[۴] https://maven.apache.org

[۵] شیء

[۶] نابود کننده

[۷] حذف کردن

[۸] **multiple inheritance**

[۹] https://www.freecodecamp.org/news/multiple-inheritance-in-c-and-the-diamond-problem-7c12a9ddbbec/

[۱۰] **platform independency**

فصل سوم
پروژه اول

۳-۱ مقدمه

در هفته‌ی دوم، یک پروژه تعریف شد که باید در قالب تیم‌های دو نفره انجام می‌شد. تعریف پروژه در ادامه آمده است. هدف این پروژه آشنایی با رابط‌های برنامه نویسی و تمرین کار گروهی است. در انتها کدهای تیم‌ها مورد ارزیابی قرار گرفت و نواقص رفع شد.
در این فصل ابتدا به تعریف پروژه می‌پردازیم سپس سراغ معرفی ابزارهای پیاده‌سازی پروژه می‌رویم. در ادامه، مفاهیم و اصطلاحاتی را که در حوزه پروژه است، تعریف و بیان می‌کنیم و در قسمت آخر نحوه‌ی اجرای کد پروژه اول را نشان می‌دهیم.

۳-۲ تعریف پروژه اول

در این پروژه قصد داریم با استفاده از رابط برنامه نویسی سایت `satori.com` به کانال **github**[۱] دسترسی پیدا کرده و اطلاعاتی را از آن استخراج کنیم. این اطلاعات شامل نام کاربری افراد با بیشترین تعداد **commit** و **push** در سایت `github.com` در یک ساعت اخیر است که به طور تناوبی هر یک دقیقه یک‌بار این اطلاعات استخراج می‌شود.

۳-۳ ابزارهای پیاده‌سازی پروژه

برای پیاده‌سازی پروژه از ابزارهای زیر استفاده شده‌است:

۳-۳-۱ زبان برنامه نویسی جاوا

زبان برنامه نویسی جاوا یک زبان سطح بالا است و یکی از قدرتمندترین زبان‌های برنامه نویسی به شمار می‌رود که در اکثر زمینه‌ها کاربرد دارد. از ویژگی‌های آن می‌توان به موارد زیر اشاره کرد:

- **platform independency**[۲]
- **garbage collector**[۳]
- فراهم بودن تعداد بسیار زیادی از **library**[۴] ها و **API (Application Programming Interface)**[۵] ها
- مستندات عالی برای زبان
- تعداد بسیار زیادی ابزار مدیریت **package** و...

۳-۳-۲ git

git یک ابزار مدیریت نسخه است که در توسعه نرم‌افزارها از آن استفاده می‌شود. از کاربردهای آن می‌توان به نگه‌داری کدها، آسان‌سازی برنامه نویسی برای تیم‌ها و موارد بسیار زیاد دیگر اشاره کرد.

[۱] `https://www.github.com`

[۲] مستقل از سکو است، یعنی در هر سیستم عاملی می‌توان از آن استفاده کرد.

[۳] `https://www.geeksforgeeks.org/garbage-collection-java`

[۴] کتابخانه

[۵] رابط برنامه نویسی

maven ۳-۳-۳

maven یک ابزار مدیریت **package** است که برای ساخت نرم‌افزارهای جاوایی از آن استفاده می‌شود. هنگامی که تعداد کتابخانه‌های استفاده شده در پروژه زیاد می‌شود، به شدت روند ایجاد فایل **jar** را آسان می‌کند.

۴-۳ مفاهیم و اصطلاحات

پیش از شرح معماری پروژه بهتر است بعضی از اصطلاحات و مفاهیم توضیح داده شود تا ابهامات رفع شود و ابعاد پروژه روشن گردد.

channel ۱-۴-۳

channel[۶] یک **queue**[۷] است که همزمان چندین **agent**[۸] می‌توانند در آن داده **push** کنند و از طرف دیگر **agent** های دیگر می‌توانند داده‌ها را دریافت کنند. به دلیل اینکه چندین **agent** در حال کار با این صف هستند، این صف باید **thread-safe**[۹] باشد تا مشکل **race condition**[۱۰] پیش نیاید.

یک نکته مهم: عامل‌هایی که از صف داده‌ها را می‌خوانند می‌توانند به همه‌ی داده‌های قرار داده شده در کانال دسترسی داشته باشند.

publisher ۲-۴-۳

عامل‌هایی هستند که در صف (کانال) داده قرار می‌دهند.

برای مثال یک برنامه می‌تواند **log** های خروجی خود را داخل یک کانال قرار دهد. در اینجا **log** همان داده است که در کانال قرار می‌گیرد و برنامه هم همان عامل است

subscriber ۳-۴-۳

عامل‌هایی که داده‌ها را از صف (کانال) برداشته و مورد استفاده قرار می‌دهند. برای مثال یک برنامه ناظر که خروجی برنامه‌های دیگر را از کانال دریافت کرده و عمل نظارت را انجام می‌دهد. در صورتی که خطایی در یک برنامه رخ داده باشد یا میزان **load**[۱۱] در یک برنامه بالا رفته باشد، هشدار می‌دهد. در اینجا عامل همان برنامه‌ی ناظر است و داده‌های کانال خروجی برنامه‌ها هستند.

real time messaging (rtm) ۴-۴-۳

امروزه با پیشرفت تکنولوژی‌ها تعداد پیام‌رسان‌ها و شبکه‌های اجتماعی زیاد شده است. برای پیاده‌سازی این نرم‌افزارها معماری‌های زیادی وجود دارد. یکی از این معماری‌ها، معماری **publisher-subscriber**[۱۲] است.

در این معماری تعدادی از کاربرها (عامل‌ها) عضو یک کانال هستند پیام‌های آن کانال را دریافت می‌کنند و مورد استفاده قرار می‌دهند. به این کاربران **subscriber** گفته می‌شود. تعدادی از کاربران (عامل‌ها) هم در کانال داده (پیام) قرار می‌دهند که به آن‌ها **publisher** گفته می‌شود.

برای مثال در پیام‌رسان تلگرام کانال سایت `varzesh3.com` وجود دارد که تعداد زیادی کاربر دارد که هر روزه از اخبار جدید ورزشی

---

[۶] کانال
[۷] صف
[۸] عامل
[۹] https://www.baeldung.com/java-thread-safety
[۱۰] https://stackoverflow.com/questions/34510/what-is-a-race-condition
[۱۱] بارگیری
[۱۲] https://en.wikipedia.org/wiki/Publish-subscribe_pattern

بهره‌مند می‌شوند. این کاربران نقش **subscriber** را دارند. ادمین‌های این کانال نقش **publisher** را ایفا می‌کنند چون آن‌ها مطالب را در کانال قرار می‌دهند.

فرایند ذکر شده به این صورت است که بلافاصله بعد از اینکه یک ادمین مطلبی را در کانال قرار داد، این مطلب در اختیار کاربران قرار می‌گیرد. به همین دلیل این فرایند یک پیام‌رسانی برخط **(rtm)** است.

۵-۴-۳ satori rtm

وبسایت www.satori.com این امکان را فراهم می‌کند که با استفاده از یک رابط کاربری خوب بدون درگیر شدن با جزئیات ایجاد کانال به صورت نرم‌افزاری، با چند کلیک یک کانال ایجاد کنیم (برای استفاده‌های غیرتجاری ایجاد کانال رایگان است). این کانال بر روی سرورهای این شرکت ایجاد شده و می‌توانیم با استفاده از رابط برنامه نویسی از آن استفاده کنیم.

در ابتدا باید در وبسایت ثبت نام کنیم. بعد از ساخت کانال یک **endpoint** و یک **appkey** در اختیار ما قرار می‌گیرد که برای وصل شدن به کانال به آن‌ها نیاز داریم. پیکربندی‌های مختلفی برای یک کانال وجود دارد که به دو مورد آن در این‌جا می‌پردازیم.

- یک کانال می‌تواند **public**[۱۳] باشد یعنی هر کسی با دانستن نام کانال و **appkey** و **endpoint** می‌تواند به داده‌های آن دسترسی پیدا کند.
- یک کانال می‌تواند **private**[۱۴] باشد که در این صورت برای دسترسی به داده‌های آن باید احراز هویت به وسیله نام کاربری و رمز عبور صورت گیرد.

برای کانال‌ها پیکربندی‌های زیادی وجود دارد. برای مثال می‌توان تعیین کرد که چه داده‌هایی وارد کانال شود و جلوی ورود چه نوع داده‌هایی گرفته شود.

اعمال زیادی نیز می‌توان بر روی داده‌های کانال‌ها انجام داد. برای مثال:

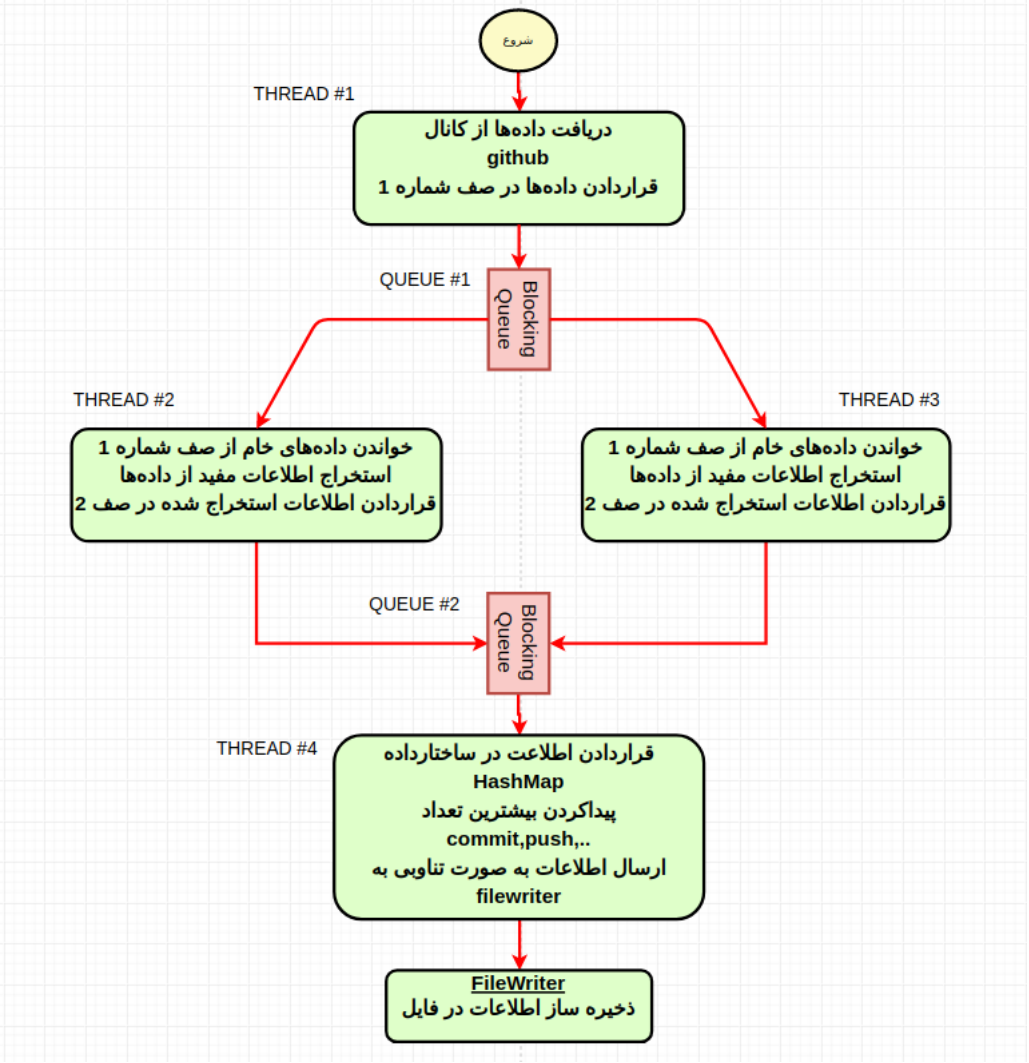
- مشاهده **history**[۱۵] کانال که شامل داده‌های پیشین کانال است.
- فیلترکردن پیام‌های دریافتی بر اساس پارامترها
- دریافت پیام‌ها در یک بازه زمانی خاص به صورت **stream**[۱۶]

۵-۳ github channel

کانال **github** یک کانال عمومی است که رویدادهای اخیر سایت github.com در آن ثبت می‌شود. هر کاربری با داشتن **appkey** و **endpoint** آن می‌تواند به داده‌های این کانال دسترسی پیدا کند.

داده‌های این کانال شامل رویدادهای رخداده مانند ایجاد یک **commit**،عملیات **push** ، **tag** و ... است. از این کانال برای پیاده‌سازی پروژه استفاده می‌کنیم.

[۱۳] عمومی
[۱۴] خصوصی
[۱۵] تاریخچه
[۱۶] جریان



شکل ۳-۱: معماری نرم‌افزار پروژه اول

۳-۶ satori API

سایت satori.com یک رابط برنامه نویسی در اختیار ما قرار می‌دهد که با استفاده از کلاس‌ها و فراخوانی توابع آن و در دست داشتن نام کانال، **appkey** و **endpoint** می‌توانیم به یک کانال وصل شده و از آن داده دریافت کنیم. برای ارسال داده هم باید علاوه بر موارد ذکر شده احراز هویت نیز انجام دهیم. در این پروژه به کانال **github** متصل شده و از آن داده دریافت می‌کنیم و از رویدادها مطلع می‌شویم.

۳-۷ معماری نرم‌افزار

شمای کلی نرم‌افزار در شکل ۳-۱ ترسیم شده است.

به صورت صریح چهار نخ در برنامه حضور دارند و از دو صف استفاده شده است.

ابتدا توسط نخ شماره یک پیام‌ها از کانال **github** توسط رابط برنامه نویسی سایت **satori** دریافت شده و داخل یک **blocking**

queue[۱۷] قرار داده می‌شود. حال سوالی که مطرح می‌شود این است که **blocking queue** چه فرقی با صف معمولی دارد؟
blocking queue صفی است که **thread-safe** است یعنی همزمان می‌توان در آن داده قرار داد و همزمان داده خواند. چون در این‌جا یک نخ در حال قرار دادن داده‌های کانال در صف و دو نخ در حال خواندن داده‌ها هستند از این ساختار داده استفاده کردیم. این ساختار داده در زبان جاوا پیاده‌سازی شده است.
از آن‌جا که سرور با سرعت زیادی داده در کانال قرار می‌دهد (تعداد کاربران **github** خیلی زیاد است به همین دلیل تعداد رخدادها زیاد است.)، با یک نخ داده‌ها را در صف یک قرار داده و با دو نخ داده‌ها را از آن صف می‌خوانیم و پردازش می کنیم تا صف اشباع نشود. این تعداد نخ به صورت تجربی به دست آمده است.
پردازش به این شکل است که تنها داده‌هایی که شامل **commit** و **push** است را فیلتر کرده و سایر داده‌ها را دور می‌ریزیم. حال داده‌های به دست آمده را در صف شماره دو قرار می‌دهیم تا یک نخ دیگر عملیات نهایی را انجام دهد.
نخ شماره چهار داده‌های فیلتر شده را دریافت کرده و آن‌ها را در یک **HashMap**[۱۸] ذخیره می‌کند. در این ساختار داده، **Key** برابر نام کاربری و **value** شامل تعداد **commit** ها و **push** های شخص در یک ساعت اخیر است. هر یک دقیقه یک بار نام کاربری شخصی که بیشترین تعداد **commit** و **push** را دارد از **HashMap** استخراج شده و به ماژول **FileWriter** فرستاده می‌شود تا در یک فایل ذخیره شود.

در انتها این فایل خروجی برنامه است که هر یک دقیقه یک‌بار نام شخصی که بیشترین فعالیت را در یک ساعت اخیر در **github** داشته است در انتهای آن ثبت می‌شود.

۳-۸ نتیجه‌گیری

برنامه‌ی مورد نظر در یک هفته آماده شد و نحوه‌ی **build**[۱۹] و اجرای آن در ادامه آمده است. با اجرای این پروژه، یک کار تیمی برنامه نویسی دو نفره را تجربه کردیم و با مفاهیم زیادی در حوزه معماری **publisher-subscriber** آشنا شدیم.

۱-۸-۳ code

کد در آدرس زیر موجود است:

https://github.com/Su6lime/GitHubTrends

۲-۸-۳ ساخت فایل **jar** و اجرای آن

mvn clean install

java -jar GitHubTrends-1.0-SNAPSHOT-jar-with-dependencies.jar

java -jar GitHubTrendsGUI.jar

[۱۷] https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html
[۱۸] https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
[۱۹] فرایند ساختن فایل **jar** را **build** می‌گویند.

فصل چهارم
پروژه دوم - آشنایی با ابزارها و مفاهیم جدید در حوزه‌ی داده‌های حجیم

۴-۱ مقدمه

در هفته سوم، تعریف پروژه دوم در اختیار ما قرار گرفت که هدف آن طراحی یک موتور جستجو است. برای طراحی چنین نرم‌افزاری، نیاز است با تعدادی از ابزارها و همچنین با مفاهیمی در حوزه‌ی داده‌های حجیم **(big data)** آشنا شویم. این ابزارها شامل موارد زیر می‌شوند:

- فریم ورک **Hadoop** [۱]
- پایگاه داده‌ی **HBase** [۲]
- موتور جستجوی **ElasticSearch** [۳]
- فریم ورک **Spark** [۴]
- پلتفرم **Kafka** [۵]

در هفته سوم با پایگاه داده‌های **ACID** و تئوری **CAP** آشنا شدیم. همچنین نحوه نصب و کار با ابزارها و فریم‌ورک‌های یاد شده را در سطح مقدماتی فراگرفتیم. در این فصل ویژگی‌های این ابزارها را توضیح خواهیم داد.
طراحی معماری نرم‌افزار و برنامه نویسی در فصل‌های بعد بررسی می‌شوند.

۴-۲ تعریف پروژه دوم

هدف طراحی یک موتور جستجو است که باید حدود ۱۰۰ میلیون صفحه **(web page)** را بازدید و محتوای آن شامل لینک‌های خروجی و متن نوشته شده در صفحه را **index**[۶] کند. برای ذخیره‌سازی و پردازش داده‌ها دو سرور در اختیار ما قرار گرفته است. هر سرور ۸ ترابایت فضای ذخیره‌سازی از نوع دیسک سخت **(Hard Disk - HDD)** ، ۱۶ گیگابایت حافظه **RAM** و یک **CPU** ۸ هسته دارد.
فایل تعریف پروژه در کنار فایل گزارش قرار دارد.

۴-۳ معرفی ابزارها

در این بخش به معرفی جزئی چند تا از مهم‌ترین ابزارهای مورد استفاده در این پروژه می‌پردازیم.

۴-۳-۱ فریم‌ورک Hadoop

Hadoop یک فریم‌ورک برای ذخیره‌سازی داده‌های انبوه و پردازش آن‌ها است که بر بستر کامپیوترهای معمولی نصب و اجرا می‌شود. **Hadoop** با این فرض ساخته شده است که احتمال خراب شدن **(failure)** حافظه ذخیره‌سازی جانبی [۷] (معمولا از نوع **HDD**) کم نیست، به همین دلیل فریم‌ورک طوری ساخته شده است که این خرابی را به صورت اتوماتیک تشخیص داده و با انجام عملیات‌هایی نظیر ایجاد افزونگی **(replication)** مانع از دست‌رفتن داده **(data loss)** می‌شود.

[۱] https://en.wikipedia.org/wiki/Apache_Hadoop
[۲] https://en.wikipedia.org/wiki/Apache_HBase
[۳] https://en.wikipedia.org/wiki/Elasticsearch
[۴] https://en.wikipedia.org/wiki/Apache_Spark
[۵] https://en.wikipedia.org/wiki/Apache_Kafka
[۶] فهرست کردن
[۷] https://www.komprise.com/glossary_terms/secondary-storage

مدل پردازش داده‌ها در این فریم‌ورک، **Map/Reduce** است و یک موتور مخصوص این کار در دل **Hadoop** قرار دارد. از نسخه ۲ به بعد این موتور جای خود را به فریم‌ورک **YARN (Yet Another Resource Negotiator)** داده است، البته خود **YARN** از این موتور استفاده می‌کند.

این فریم‌ورک برای ذخیره‌سازی داده‌ها از یک سیستم مدیریت فایل به‌نام **HDFS (Hadoop Distributed File System)** استفاده می‌کند. داده‌ها به صورت بلوکی همراه با افزونگی توسط این **file system** در سرورها قرار می‌گیرند. این فریم‌ورک از پروسه‌های زیر تشکیل شده است:

NameNode

مدیریت فایل سیستم **Hadoop** یا همان **HDFS** برعهده‌ی این پروسه است. متادیتا **(metadata)** ی فایل‌های موجود در خوشه **(cluster)** در اختیار این پروسه است. مثلا اطلاعاتی مانند این‌که هر فایل به چند بلوک تقسیم شده‌است، هر بلوک در کدام سرور قرار دارد و برای هر بلوک چند افزونگی ایجاد شده و آن‌ها در کجا قرار دارند. تنها یک پروسه **NameNode** برای کل خوشه اجرا می‌شود. برای جلوگیری از **SPOF (Single Point Of Failure)** می‌توان یک **SecondaryNameNode** اجرا کرد که از **NameNode** اصلی به صورت دوره‌ای **snapshot**[۸] می‌گیرد و در صورت از کار افتادن **NameNode** اصلی، مانع از دست رفتن اطلاعات حیاتی و متادیتاها می‌شود.

DataNode

داده‌های اصلی توسط پروسه‌های **DataNode** در سرورهای مختلف که در خوشه هستند نگه‌داری می‌شود. معمولا به ازای هر سرور، یک پروسه **DataNode** اجرا می‌شود. این پروسه‌ها به صورت تناوبی در هر ۳ ثانیه صحت عملکرد خود را به **NameNode** گزارش می‌دهند.

TaskTracker

داده‌ی مورد نظر برای انجام عملیات‌های **Map**[۹] و **Reduce**[۱۰] را از **DataNode** گرفته و پس از انجام عملیات، نتیجه را به **JobTracker** باز می‌گرداند. معمولا در کنار هر **DataNode** ، یک **TaskTracker** حضور دارد.

JobTracker

برنامه‌ی **Map/Reduce** که توسط کاربر نوشته می‌شود به این پروسه تحویل داده می‌شود تا اجرا شود و نتایج توسط این پروسه به کاربر برگردانده می‌شود. برای انجام عملیات، این پروسه کارها را بین **TaskTracker** ها پخش می‌کند، نتیجه کار را از آن‌ها گرفته، ادغام کرده و نتیجه نهایی را به کاربر برمی‌گرداند. این پروسه در صورتی که یک **TaskTracker** با مشکل روبه‌رو شود یا نتواند وظایفش را انجام دهد، وظیفه را به یک **TaskTracker** دیگر سپرده و خطاهای پیش‌آمده را تا جایی که بتواند رفع می‌کند. اطلاعات مربوط به محل فایل‌ها در خوشه را از **NameNode** دریافت کرده و در اختیار **TaskTracker** ها قرار می‌دهد.

این‌طور می‌توان گفت که در این معماری **Master-Slave**[۱۱] ، **NameNode** و **JobTracker** نقش ارباب **(master)** و **DataNode** ها و **TaskTracker** ها نقش برده **(slave)** را ایفا می‌کنند.

[۸] نمونه برداری
[۹] نگاشت
[۱۰] کاهش

[۱۱] https://en.wikipedia.org/wiki/Master/slave_(technology)

برای ارتباط با **Hadoop** می‌توان از **Java API** استفاده کرد. یک برنامه ساده که عملیات **MapReduce** را بر روی یک فایل کوچک انجام می‌دهد در کنار فایل گزارش قرار دارد. وظیفه این برنامه پیدا کردن فرکانس کلمات در فایل ورودی است. نحوه نصب فریم‌ورک و اجرای برنامه در ادامه آمده است.

۲-۳-۴ پایگاه داده **HBase**

یک پایگاه‌داده‌ی توزیع‌شده [۱۲]، غیررابطه‌ای [۱۳]، ستون‌گرا [۱۴] و کلید-مقدار [۱۵] است که بر بستر **HDFS** اجرا می‌شود. برای ذخیره‌سازی مقدار بسیار زیادی از داده‌های تنک **(sparse)** همراه با ویژگی **fault tolerant** ساخته شده است. برای عملیات‌های خواندن و نوشتن سریع بر روی دیتاست **(dataset)** های بزرگ همراه با گذردهی **(throughput)** بالا و تاخیر **(latency)** کم بسیار مناسب است. به‌طور خلاصه از ویژگی‌های **HBase** می‌توان به موارد زیر اشاره کرد:

- فشرده‌سازی **(compression)**
- عملیات‌های درون حافظه‌ای [۱۶]
- سازگاری **(consistency)**
- تا حدودی قابلیت تحمل خطا [۱۷]
- و...

راه‌های دسترسی و فرمان‌دادن به **HBase** عبارتند از:

- **Java API**
- **REST API** [۱۸]
- **Apache Avro** [۱۹]
- **Apache Thrift** [۲۰]

در این پروژه از **Java API** استفاده می‌کنیم به همین دلیل از توضیح موارد دیگری که در بالا ذکر شد اجتناب می‌کنیم.
یک نمونه کد ساده برای اتصال به **HBase** همراه گزارش ضمیمه شده است. در این کد چند جدول در داخل پایگاه‌داده ساخته شده و در آن داده قرار می‌دهیم و داده‌ها را می‌خوانیم و به روز رسانی می‌کنیم. نحوه نصب **HBase** و اجرای کد در ادامه آمده است.

[۱۲] **distributed**
[۱۳] **non relational**
[۱۴] **column oriented**
[۱۵] **key-value store**
[۱۶] **in-memory operations**
[۱۷] **partially tolerable against failure**
[۱۸] https://en.wikipedia.org/wiki/Representational_state_transfer
[۱۹] https://en.wikipedia.org/wiki/Apache_Avro
[۲۰] https://en.wikipedia.org/wiki/Apache_Thrift

۴-۳-۳ موتور جستجوی ElasticSearch

یک موتور جستجوی متن توزیع‌شده، بلادرنگ **(real time)** ، مقیاس‌پذیر، مستقل از سکو [۲۱] و کاراست که برپایه **Apache Lucene** [۲۲] ساخته شده است. امکان جستجوی بلادرنگ را فراهم می‌کند و مانند پایگاه داده‌ی **Mongodb** ، سندگرا **(document-based)** است. انواع داده‌های ساخت‌یافته **(structured)** و بدون ساختار **(unstructured)** را پشتیبانی می‌کند.

ElasticSearch موارد زیر را پشتیبانی می‌کند:

- توزیع‌پذیری **(distribution)**
- **sharding** [۲۳]
- تکثیر **(replication)**
- خوشه بندی **(clustering)**
- معماری چندگره‌ای [۲۴]
- عملیات‌های فله‌ای [۲۵]
- و...

اما موارد زیر را پشتیبانی نمی‌کند:

- عملیات‌های نگاشت-کاهش [۲۶]
- معاملات توزیع‌شده [۲۷]

برای برقراری ارتباط با **ElasticSearch** باید از پروتکل **HTTP** استفاده کرد. داده‌های ارسالی یا دریافتی در قالب ساختارداده‌ی **JSON** در قسمت **body** درخواست **(request)** یا پاسخ **(response)** قرار می‌گیرند.

برای مثال دستور زیر

```
curl --header "Content-Type: application/json" --request POST

--data '{"title": "Java 8 In Depth", "category":"Java"}'

http://localhost:9200/bookstore/books/1001
```

داده‌ی موجود در قالب **JSON** را در ایندکس **bookstore** با تایپ **book** و ایدی **1001** قرار می‌دهد.

ساختار **URL (Uniform Resource Locator)** ارسالی به **ElasticSearch** به صورت زیر است:

---

[۲۱] **cross-platform**
[۲۲] https://en.wikipedia.org/wiki/Apache_Lucene
[۲۳] https://en.wikipedia.org/wiki/Shard_(database_architecture)
[۲۴] **multinode architecture**
[۲۵] **bulk operations**
[۲۶] **Map-Reduce operations**
[۲۷] **distributed transactions**

http://server:port/index(lowercase)/Type/

یک برنامه ساده برای ارسال داده به **ElasticSearch** و دریافت داده از آن همراه گزارش ضمیمه شده است که شامل کد جاوا و کد **curl**[۲۸] می‌شود. نحوه نصب **ElasticSearch** و اجرای برنامه در ادامه آمده است.

۴-۴ حالات نصب

ابزارهای نام برده شده در قسمت‌های قبل را می‌توان به دو صورت نصب کرد:

۴-۴-۱ Standalone mode

در این حالت، یک نمونه **(instance)** از هر کدام از ابزارها را بر روی تنها یک سرور اجرا می‌کنیم. نکته قابل توجه این است که این ابزارها برای سیستم‌های توزیع‌شده و خوشه‌ای ساخته شده‌اند و محبوبیت و کارایی بالای آن‌ها به خاطر استفاده از معماری توزیع‌شده است. با محدود کردن آن‌ها تنها به یک سرور نمی‌توانیم کارایی مطلوب را به دست آوریم. به همین دلیل شرکت دو سرور در اختیار ما قرار داده است تا آن‌ها را به صورتی که در ادامه توضیح می‌دهیم نصب کنیم.

۴-۴-۲ Distributed mode

در این حالت در هر سرور در خوشه یک پروسه از ابزار مورد نظر اجرا کرده و پیکربندی مربوط به ارباب **(master)** و برده‌ها **(slaves)** را به صورت مجزا انجام می‌دهیم.

۴-۵ نصب ابزارها و اجرای کدهای قسمت‌های قبل

کد کار با ابزارها در پوشه‌های مختلف ضمیمه شده است، منتهی قبل اجرای آن‌ها باید ابزارها نصب باشند. برای نصب آن‌ها به آدرس‌های زیر مراجعه کنید.

- نصب **Hadoop**

https://www.edureka.co/blog/install-hadoop-single-node-hadoop-cluster

- نصب **HBase**

https://computingforgeeks.com/how-to-install-apache-hadoop-hbase-on-ubuntu

- نصب **ElasticSearch**

https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html

نحوه اجرای کدها در سیستم عامل لینوکس (باید جاوا از قبل نصب باشد.):

[۲۸] https://curl.haxx.se

- اجرای نمونه کد کار با **Hadoop**

java -jar test-1.0-SNAPSHOT-jar-with-dependencies.jar input.txt output

برای مشاهده خروجی:

cat output/part-r-00000

- اجرای نمونه کد کار با **HBase**

ابتدا باید **Hadoop** و **HBase** اجرا شده باشند:

start-all.sh

start-hbase.sh

حال کد را اجرا می‌کنیم:

java -jar 1-1.0-SNAPSHOT-jar-with-dependencies.jar

- اجرای نمونه کد کار با **ElasticSearch**

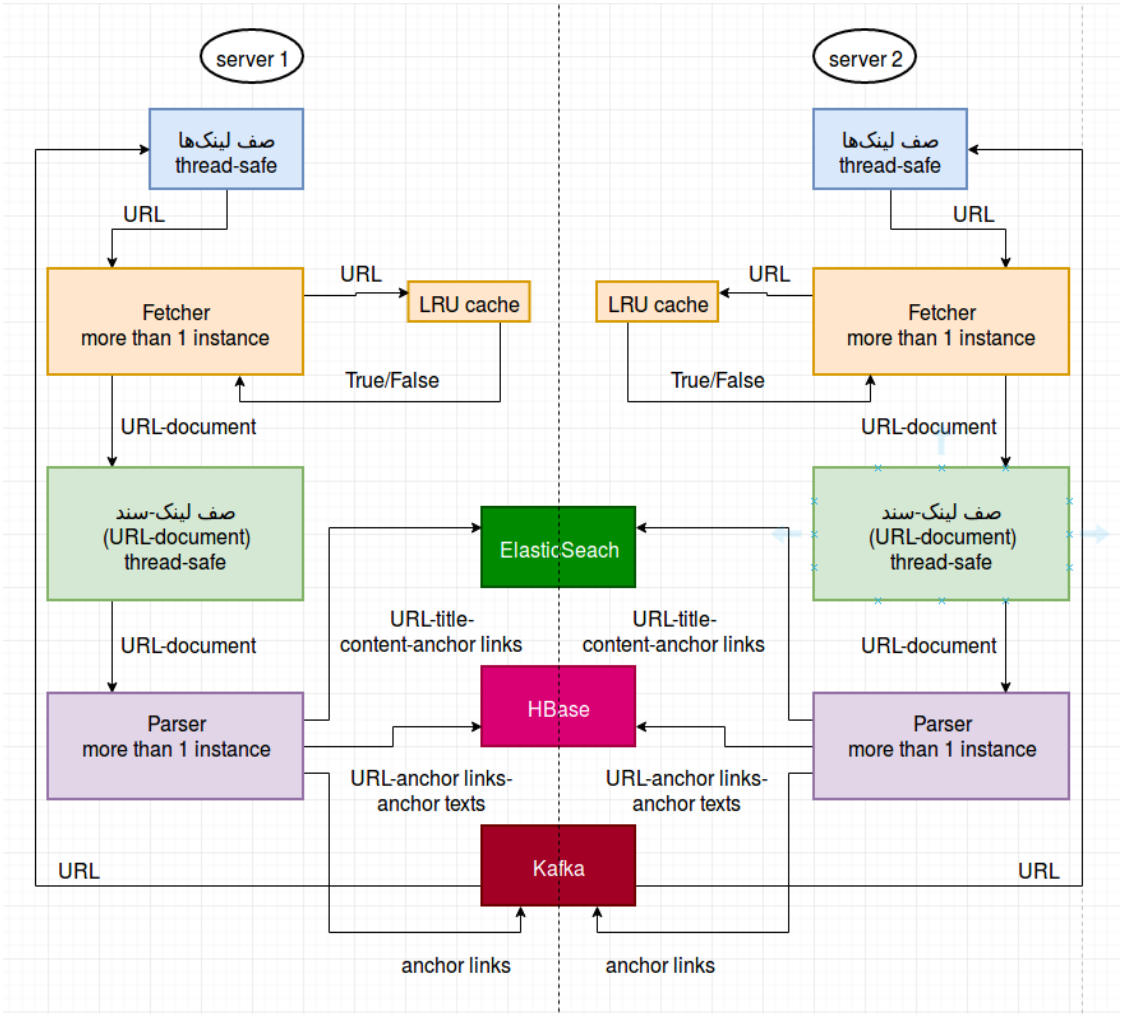
java -jar 1-1.0-SNAPSHOT-jar-with-dependencies.jar 2> output.txt

دستورات **curl** نیز در فایل **commands.txt** قرار دارد.

۶-۴ نتیجه گیری

در هفته سوم با ابزارها و مفاهیم جدیدی در حوزه داده‌های حجیم آشنا شدیم که از میان آن‌ها می‌توان به ابزارهای **Hadoop** ، **HBase** و **ElasticSearch** اشاره کرد. نصب ابزارها و کار با آن‌ها را به صورت مقدماتی فراگرفتیم و معماری نرم‌افزار را در قدم بعدی انجام خواهیم داد.

فصل پنجم
معماری پروژه دوم (موتور جستجو)



شکل ۵-۱: معماری نرم‌افزار موتور جستجو

۵-۱ مقدمه

در هفته چهارم به طراحی معماری موتور جستجو پرداختیم. همچنین برنامه‌نویسی و پیاده‌سازی بخش‌های مستقل از معماری مانند بخش ارتباط با پایگاه‌داده‌ها انجام شد.

ابتدا هر کدام از افراد تیم معماری مدنظر خود را ارائه داد و در نهایت با ادغام طرح‌های موجود یک معماری جامع انتخاب شد. هر کدام از افراد تیم برنامه‌نویسی و پیاده‌سازی یک بخش از این معماری را بر عهده گرفتند. پیاده‌سازی بخش **HBase** به من واگذار شد.

در ادامه اجزای این معماری معرفی و توضیح داده می‌شود.

۵-۲ اجزای موتور جستجو

شکل ۵-۱ نمای کلی از معماری موتور جستجو و یک معماری توزیع‌شده **(distributed)** را نشان می‌دهد. دو سرور داریم که دقیقا عین هم هستند و پایگاه داده‌های آن‌ها به هم متصل است. طبیعتا پایگاه داده‌ها مثل **Hbase** و موتور جستجوی **ElasticSearch** و پلتفرم **Kafka** را در حالت توزیع شده نصب و اجرا کرده‌ایم.

حال به شرح اجزا در شکل ۵-۱ می‌پردازیم:

۵-۲-۱ صف لینک‌ها

این صف شامل لینک‌ها **(URL)** هایی است که باید بازدید شده و محتوای آن‌ها مورد پردازش قرار گیرند. این صف در ابتدا با **URL** های زیر به عنوان مقادیر اولیه (**seed** اولیه) پر می‌شود:

- https://en.wikipedia.org/wiki/Main_Page
- https://us.yahoo.com
- https://www.nytimes.com/
- https://www.msn.com/en-us/news
- http://www.telegraph.co.uk/news/
- http://www.alexa.com
- http://www.apache.org
- https://en.wikipedia.org/wiki/Main_Page/World_war_II
- http://www.news.google.com
- https://www.geeksforgeeks.org
- https://mvnrepository.com/

به دلیل این‌که **object** های زیادی از این صف داده می‌خوانند، این صف باید بین نخ ها امن [۱] باشد.

۵-۲-۲ Fetcher

object ای است که به طور متناوب از صف لینک‌ها (در قسمت قبل توضیح داده شد.) **URL** دریافت می‌کند سپس محتوای آن را دانلود می‌کند و **URL** و محتوا را در صف سندها قرار می‌دهد.

پیش از دانلود محتوای یک لینک بررسی می‌شود که آیا درخواست به **host** این لینک مجاز است یا خیر. در صورت مجاز نبودن درخواست، لینک در انتهای صف لینک‌ها قرار می‌گیرد. حال سوالی که مطرح می‌شود این است که معیار مجاز بودن چیست؟

اگر در ۳۰ ثانیه اخیر به یک **host** درخواست داده باشیم، دیگر مجاز نیستیم که به آن درخواست دهیم زیرا ممکن است آدرس **IP** ما را مسدود **(block)** کند (ممکن است فکر کند برنامه ما یک ربات مولد درخواست برای ایجاد حمله **DOS (Denial Of Servise)** است.). برای بررسی این وضعیت از یک حافظه از نوع **LRU (Least Recently Used)** استفاده می‌کنیم.

چندین نمونه از **Fetcher** در برنامه اجرا می‌شوند تا سرعت پردازش و دریافت لینک‌ها افزایش یابد. البته تعداد آن‌ها به مواردی مانند منابع سیستم (میزان **RAM** ، قدرت **CPU** ، میزان حجم دیسک ذخیره سازی جانبی (**(HDD (Hard Disk Drive)**)، تعداد سایر نخ‌ها (**thread** ها) و ... بستگی دارد.

[۱] **Thread-safe**

۵-۲-۳ صف اسناد

این صف شامل جفت **(pair)** های لینک-سند **(url-document)** است که باید محتوای بخش سند این جفت‌ها تجزیه **(parse)** شود و اطلاعات زیر از آن‌ها استخراج شود:

- عنوان سند **(title)**
- محتوای پاراگراف‌ها یا برچسب **(tag)** های **<p>**
- **anchor link** ها و عنوان‌های هر کدام

به دلیل این‌که **object** های زیادی از این صف داده می‌خوانند، این صف باید امن [۲] باشد.

۵-۲-۴ Parser

object های **Parser** وظیفه دارند به صورت تناوبی از صف اسناد جفت‌های لینک-سند را خوانده و اطلاعاتی که در بخش صف اسناد توضیح داده شد را استخراج کنند. پس از استخراج عملیات‌های زیر انجام می‌شود:

- قرار دادن لینک سند، محتوای پاراگراف‌های سند و عنوان سند **(title)** در **ElasticSearch**
- قرار دادن لینک سند، **anchor link** ها و عنوان‌های هر کدام در **HBase**
- تحویل **anchor link** ها به **Kafka**

چندین پروسه **Parser** در برنامه داریم تا سرعت استخراج اطلاعات افزایش یابد. البته تعداد آن‌ها به مواردی مانند منابع سیستم (میزان **RAM** ، قدرت **CPU** ، میزان حجم دیسک ذخیره سازی جانبی **(HDD)**)، تعداد سایر نخ‌ها (**thread** ها) و ... بستگی دارد.

۵-۲-۵ Kafka

Kafka مدیریت صف لینک‌ها را بر عهده دارد. **anchor link** ها را از **Parser** گرفته و آن‌هایی که تاکنون مشاهده نشده‌اند را در انتهای صف لینک‌ها قرار می‌دهد.
این نوع معماری پردازش **BFS (Breadth First Search)** برای گراف وب را فراهم می‌کند.

۵-۳ نتیجه گیری

در هفته چهارم معماری نرم‌افزار طراحی شد و بخشی از کار برنامه‌نویسی و پیاده‌سازی انجام شد.

[۲] **Thread-safe**

فصل ششم
پیاده سازی پروژه دوم (موتور جستجو)

۶-۱ مقدمه

در هفته پنجم به پیاده سازی و برنامه نویسی موتور جستجو پرداختیم. اعضای تیم به برنامه نویسی بخش‌های مختلف پروژه پرداختند و در انتها با ترکیب کدها که در بستر **git** ذخیره شده بودند، برنامه‌ی واحدی که همان موتور جستجو است ساخته شد. این برنامه از دو بخش تشکیل شده است:

- خزنده: یک برنامه که با زبان جاوا نوشته شده است و وظیفه‌ی آن خزش **(crawl)** کردن صفحات وب و ذخیره اطلاعات آن است و
- **search.sh**: یک اسکریپت است که از کاربر **query** گرفته و آن‌را در بین اسناد جستجو می‌کند و نتیجه را برمی‌گرداند. نتیجه شامل اسنادی است که شامل **query** هستند یا عبارات **query** در آن‌ها حضور دارد. نتایج با توجه به امتیازشان به صورت نزولی مرتب شده و نمایش داده می‌شوند (بهترین سند یافت شده اول نمایش داده می‌شود.).

در ادامه به توضیح پکیج‌ها و کلاس‌های پروژه می‌پردازیم.

۶-۲ پکیج‌ها

پروژه از پکیج‌های زیر تشکیل شده است:

۶-۲-۱ crawler

این پکیج شامل تابع **main** است. در این تابع صف لینک‌ها با **URL** های اولیه پر می‌شود و نخ **(thread)** آمارگیری (کلاس **Statistics**) اجرا می‌شود. همچنین به تعداد نیاز از کلاس‌های **Fetcher** و **Parser** که خود نخ هستند نمونه ساخته شده و اجرا می‌شوند.
این پکیج شامل کلاس‌های زیر است: (کلاس‌ها در ادامه توضیح داده خواهند شد.)

- **Crawler**
- **Fetcher**
- **Parser**
- **LruCache**

۶-۲-۲ index

این پکیج شامل کلاس **Elastic** است که وظیفه برقراری ارتباط با **ElasticSearch** و تبادل داده با آن را برعهده دارد.

۶-۲-۳ storage

این پکیج شامل کلاس **HBase** است که وظیفه برقراری ارتباط با **HBase** و تبادل داده با آن را برعهده دارد.

۶-۲-۴ test

شامل کلاس‌هایی است که برای تست سایر کلاس‌های نرم‌افزار به کار می‌روند.

utils ۵-۲-۶

کلاس‌های زیر در این پکیج قرار می‌گیرند:

- **Constant**
- **Pair**
- **Prints**
- **Statistics**

این کلاس‌ها در ادامه توضیح داده می‌شوند.

language ۶-۲-۶

شامل کلاس‌های مورد نیاز برای تشخیص زبان یک متن است.

۳-۶ کلاس‌ها

برای پیاده سازی پروژه از کلاس‌های مختلفی استفاده شده است که در ادامه به تشریح آن‌ها می‌پردازیم:

Crawler ۱-۳-۶

تابع **main** در این کلاس قرار دارد. عملیات انجام شده در این تابع در قسمت پکیج **crawler** توضیح داده شد.

Fetcher ۲-۳-۶

این کلاس خود یک نخ است و وظیفه دارد که به طور متناوب از صف لینک‌ها **(URLs queue)** ، لینک دریافت کرده و محتوای آن را دانلود کند سپس لینک و محتوایش **(document)** را در صف لینک-سند **(url-document)** قرار دهد. برای انجام این عملیات از توابع زیر استفاده شده است:

fetchURL

یک لینک از سر صف لینک‌ها برداشته و برمی‌گرداند.

getDomainIfLruAllowed

بررسی می‌کند که آیا به **host** لینک مورد نظر در ۳۰ ثانیه اخیر درخواستی داده شده است یا خیر. در صورت مثبت بودن، این لینک مجددا در انتهای صف لینک‌ها قرار می‌گیرد و لینک دیگری اخذ می‌شود.

fetch

محتوای یک لینک را دانلود کرده و برمی‌گرداند.

putFetchedData

لینک و محتوای آن را در صف لینک-سند قرار می‌دهد.

LruCache ۳-۳-۶

در این کلاس پیاده سازی یک **LruCache** انجام شده است.

Parser ۴-۳-۶

این کلاس خود یک نخ است که به طور متناوب از صف لینک-سند داده (لینک و سند مربوط به آن) ر ا استخراج می‌کند. محتوای لینک را **parse** کرده و موارد زیر را از آن استخراج می‌کند:

- عنوان لینک **(title)**
- **anchor link** ها و **anchor text** ها
- تگ‌های **<p>** (پاراگراف‌ها)

پس از استخراج اطلاعات بالا، عملیات‌های زیر انجام می‌شود:

- قرار دادن لینک و عنوان آن و پاراگراف‌ها در **ElasticSearch**
- قرار دادن لینک، **anchor link** ها و **anchor text** ها در **HBase**
- تحویل **anchor link** ها به **Kafka**

برای انجام عملیات‌های بالا از توابع زیر استفاده شده است:

takeFetchedData

از سر صف لینک-سند داده برداشته و برمی‌گرداند.

extractLinkAnchors

anchor link های یک سند را استخراج می‌کند و برمی‌گرداند.

putToElastic

لینک و عنوان آن و پاراگراف‌های سند مربوط به لینک را در **ElasticSearch** قرار می‌دهد.

putAnchorsToHBase

لینک، **anchor link** های درون سند مربوط به لینک و **anchor text** ها را در **HBase** قرار می‌دهد.

putToKafka

anchor link های مربوط به یک سند را به **Kafka** تحویل می‌دهد.

Elastic ۵-۳-۶

رابط برنامه و موتور جستجوی **ElasticSearch** است. مهم‌ترین تابع آن **indexData** است که لینک و عنوان آن و پاراگراف‌های سند مربوط به آن را دریافت کرده و داخل **ElasticSearch** قرار می‌دهد.

HBase ۶-۳-۶

رابط برنامه و پایگاه داده‌ی **HBase** است. مهم‌ترین تابع آن **insertLinks** است که لینک، **anchor link** های درون سند مربوط به لینک و **anchor text** ها را گرفته و داخل **HBase** قرار می‌دهد.

Constants ۷-۳-۶

مقادیر ثابت و پارامترهای برنامه در این کلاس قرار دارند.

Statistics ۸-۳-۶

کلاسی است که خود یک نخ است و وظیفه دارد به صورت تناوبی وضعیت سایر نخها (**Parser** ها و **Fetcher** ها) را گزارش دهد.

Pair ۹-۳-۶

پیاده سازی یک ساختار داده برای نگه‌داری یک کلید و مقدار متناظر آن است.

۴-۶ استثناها

موتور جستجوی ساخته شده تا حد بسیار زیادی مطابق با اطلاعات داده شده در این گزارش و گزارش‌های قبلی است و فقط در چند مورد زیر تفاوت دارد که علت آن نیز ذکر شده است:

- عدم استفاده از پکیج تشخیص زبان: به دلیل خطای بالای تشخیص زبان و زمان زیادی هم که می‌گرفت، از کلاس‌های این پکیج استفاده نشد. در اسرع وقت راه‌حلی برای مشکل پیش آمده پیدا می‌کنیم.
- عدم استفاده از **Kafka**: به جای **Kafka** ، **anchor link** ها را مستقیما در صف لینک‌ها قرار دادیم و عدم مشاهده آن‌ها تاکنون را به یک صف دیگر که مدیریت آن بر عهده **HBase** است، سپردیم. دلیل عدم استفاده از **Kafka** ، عدم پیکربندی مناسب و از کار افتادن بی مورد آن بود. در اسرع وقت سعی می‌کنیم این مشکل را حل کنیم.

۵-۶ اجرای برنامه

برنامه توسط ابزار مدیریت پکیج **maven**[۱] ساخته شده است. نسخه‌ای که در کنار گزارش قرار دارد نسخه‌ای است که بر روی یک سرور اجرا می‌شود. برای مشاهده نسخه اصلی کد که در دو سرور اجرا می‌شود به آدرس زیر مراجعه کنید:

https://github.com/Ahmad535353/NimboSearchEngine

برای اجرای بخش خزنده برنامه کافی است روند زیر طی شود:

- اجرای **Hadoop** ، **HBase** و **ElasticSearch**
- ایجاد یک جدول به‌نام **CrawlerTable** در **HBase** با فامیلی ستون **data**[۲]
- **java -jar Crawler.jar**

[۱] https://maven.apache.org
[۲] **Column Familiy**

برای ارسال **query** و دریافت پاسخ، کافی است فایل اجرایی **search.sh** را اجرا کنید. با اجرای فایل از شما درخواست می‌شود تا **query** را وارد کنید. پس از قرار دادن **query** و زدن دکمه **Enter** ، پاسخ دریافت و نمایش داده می‌شود.

۶-۶ موارد باقی مانده

به دلیل ضیق وقت، فرصت نشد تا موارد زیر پیاده سازی شوند. سعی می‌شود در اسرع وقت این موارد پیاده سازی شوند:

- پشتیبانی گرفتن از صف‌های لینک، لینک-سند و صف کلاس **HBase** و جلوگیری از دست رفتن داده‌ها (**data loss**) هنگام قطع شدن برق.
- فیلتر کردن برخی پروتکل‌ها مانند **FTP** و ...
- برطرف کردن مشکل **Kafka**
- پیاده سازی و یا پیدا کردن یک ماژول تشخیص دهنده زبان یک متن
- جایگزین کردن کلاس‌های منسوخ شده **(deprecated)** برنامه مانند کلاس‌های استفاده شده در کلاس **Elastic** و ...

۷-۶ نتیجه گیری

با اتمام پیاده سازی، یک موتور جستجو در اختیار ماست که فقط «کار» می‌کند. نتیجه **query** ها اصلا خوب نیستند. تیم شرکت به ما توصیه کردند که از الگوریتم **Page Rank**[۳] و مدل برنامه نویسی **Map/Reduce** برای امتیازدهی سایت‌ها و رتبه بندی آن‌ها استفاده کنیم و همچنین نواقص قبلی را برطرف کنیم. برای پیاده سازی **Page Rank** از فریم‌ورک **Spark**[۴] استفاده می‌کنیم. درباره‌ی **Spark** و **Page Rank** در فصل بعد توضیح داده می‌شود.

[۳] https://en.wikipedia.org/wiki/PageRank
[۴] https://spark.apache.org/

فصل هفتم
بهبود عملکرد موتور جستجو

۷-۱ مقدمه

همان‌طور که در قسمت نتیجه‌گیری فصل قبل ذکر شد، نتایج جستجو کیفیت خوبی ندارند. به همین دلیل مدیران تیم و اعضای شرکت پیشنهاداتی برای بهبود نتایج مطرح کردند که شامل موارد زیر می‌شود:

- بهبود جستجو با محاسبه تعداد لینک‌های ورودی
- بهبود جستجو با اعمال متن **anchor link** ها
- بهبود جستجو با **Page Rank**

به دلیل کمبود وقت و صرف وقت برای ارائه نهایی، تنها فرصت شد تا مورد اول را پیاده سازی کنیم. به همین دلیل در موارد دیگر تنها به توضیح بخشی از روند کاری که باید انجام شود می‌پردازیم.

۷-۲ بهبود جستجو با محاسبه تعداد لینک‌های ورودی

برای این‌که نتایج جستجو کیفیت بیشتری داشته باشند، باید صفحات مهم‌تر، شانس بیشتری برای ظاهر شدن در نتایج جستجو داشته باشند. ساده‌ترین راه، پیدا کردن تعداد لینک‌ها به یک صفحه است. اگر به یک صفحه، تعداد ارجاعات بیشتری باشد، طبعا مهم‌تر است. تعداد ارجاعات را با **Map/Reduce** پیدا می‌کنیم، آن‌گاه آن‌ها را به یک امتیاز تبدیل کرده و در کنار امتیازی که **ElasticSearch** برای **query** ها در نظر می‌گیرد به کار برده و در نتیجه‌ی نهایی تاثیر می‌دهیم.
برای پیاده‌سازی موارد بالا، دو برنامه نوشته شده است:

- برنامه اول برنامه‌ای است که تعداد لینک‌ها به یک صفحه را محاسبه کرده و در یک جدول در **HBase** قرار می‌دهد. این برنامه همراه گزارش ضمیمه شده است. برای اجرای آن کافی است مراحل زیر طی شوند:
  یک **instance** از **HBase** با دستور **start-hbase.sh** اجرا شود.
  یک جدول با نام **InnerlinksTable** با ستون فامیلی [۱] **NumOfLinks** ساخته شود.
  java -jar innerlinks-calculator.jar
- برنامه‌ی دوم برنامه‌ای است که از کاربر **query** می‌گیرد و با در نظر گرفتن تعداد لینک‌های ورودی به هر صفحه و تطابق محتوای هر صفحه با کلمات **query** ورودی، بهترین نتایج را به صورت نزولی نمایش می‌دهد.
  برای اجرای این برنامه باید مراحل زیر طی شود:
  اجرای **HBase** با دستور **start-hbase.sh**
  اجرای **ElasticSearch** با اجرای دستور **sudo systemctl start elasticsearch.service**
  java -jar query-processor-with-innerlinks.jar

۷-۳ بهبود جستجو با اعمال متن **anchor link**

اگر صفحات عنوان خوبی داشتند، عالی بود ولی خیلی از صفحات یا عنوان خوبی ندارند یا در عنوان‌شان فقط به یکی از جنبه‌های اطلاعاتی که می‌توان در آن صفحه یافت اشاره می‌کنند. حال چگونه می‌توان عناوین خوب تولید کرد؟ پاسخ، متن‌هایی است که صفحات دیگر

[۱] **Column Family**

برای آن صفحه انتخاب کرده‌اند یا همان متن **anchor link** ها.

حال باید با **Map/Reduce** ، **anchor text** های پر تکرار برای هر صفحه را بیابیم (البته نه عبارات بی اثری مانند «این‌جا» یا «لینک») و آن‌ها را در کوئری‌ها تاثیر دهیم تا جستجوهای بهتری داشته باشیم.

۴-۷ بهبود جستجو با Page Rank

برای محاسبه‌ی **Page Rank** ، به سراغ فریم‌ورک **Spark** می‌رویم. اما این دو چه هستند؟

- **Page Rank** الگوریتمی است که شرکت گوگل برای امتیازدهی به صفحات وب در موتور جستجویش از آن استفاده می‌کند. این الگوریتم با محاسبه‌ی تعداد لینک‌ها به یک صفحه و کیفیت آن‌ها میزان اهمیت آن صفحه را پیدا کرده و به آن یک عدد نسبت می‌دهد. فرض بر این است که سایت مهم‌تر تعداد ارجاعات بیشتری از طرف سایت‌های دیگر دارد. اطلاعات بیشتر در آدرس زیر:

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

- **Spark**[۲] یک فریم‌ورک برای انجام محاسبات در یک خوشه از کامپیوترها **(cluster)** است. پردازش موازی داده‌ها در خوشه را امکان‌پذیر می‌سازد و به طور ضمنی **fault tolerant** است. کتابخانه‌های زیر از **Spark** استفاده می‌کنند:
  - **MLIB**
  - **GraphX**
  - **Spark SQL**
  - **Spark Streaming**

به دلیل این‌که **Spark** از عملیات‌های درون حافظه‌ای [۳] استفاده می‌کند، تا ۱۰ برابر سریع تر از **YARN** عمل می‌کند و این یکی از دلایلی است که به سراغ آن می‌رویم.

۵-۷ نتیجه‌گیری

علاوه بر ایده‌های مطرح شده برای بهبود عملکرد موتور جستجو، همچنان ایده‌ها و راه‌های زیادی برای بهتر کردن کیفیت **query** ها وجود دارد. موفق به پیاده سازی تنها یکی از این ایده‌ها شدیم و امید است بتوانیم خارج از تایم کارآموزی به بهبود عملکرد این موتور جستجو بپردازیم.

[۲] https://en.wikipedia.org/wiki/Apache_Spark
[۳] **in-memory operatios**

فصل هشتم
نتیجه‌گیری

در این دوره با ابزارهای جدیدی مانند **Hadoop** ، **HBase** ، **ElasticSearch** ، **Kafka** و **Spark** آشنا شدیم. همچنین با ابزار مدیریت پکیج **maven** ، ابزار مدیریت نسخه **git** و سیستم عامل لینوکس پروژه را پیاده سازی کردیم و با کتابخانه‌های زیادی در زبان جاوا کار کردیم. از میان معماری‌های موجود برای طراحی نرم‌افزارهاُ، با معماری **client-server** و **publisher-subscriber** آشنا شدیم و دید خوبی نسبت به این معماری ها پیدا کردیم. ترکیب مدل برنامه نویسی **Map/Reduce** و فریم‌ورک **Spark** و پردازش داده‌هایی که بر روی سرورهای مختلف قرار داشتند بخش هیجان انگیز کار بود که قبلا با چنین مسئله‌ای مواجه نشده بودیم.

حاصل این دوره کارآموزی، یک موتور جستجو بود که هر چند به قدرتمندی موتور جستجوهای امروزی نیست اما با توجه به میزان زمانی که برای ساخت آن صرف شد، سطح دانش توسعه دهندگان آن، قدرت سخت افزاری سرورها و موارد دیگر، نتایج مطلوبی با جستجوی **query** ها حاصل شد.

همچنین در کنار یادگیری نکات فنی، کار گروهی خوبی را تجربه کردیم و با نمونه کار واقعی آشنا شدیم. همچنان می‌توان بهبودهای زیادی در موتور جستجوی ساخته شده ایجاد کرد که چند مورد در فصل قبل بیان شدند.

امیدوارم این گزارش مثمر ثمر واقع شود.

واژه‌نامه‌ی انگلیسی به فارسی

A

agent . . . . . عامل نرم‌افزاری یا انسانی

architecture . . . . . معماری

B

big data . . . . . داده‌ی حجیم

blocking queue . . . . . نوع خاصی از صف که همزمان چندین نفر می‌توانند در آن داده بگذارند و چندین نفر داده‌ها را دریافت کنند بدون این که مشکلی پیش آید.

block . . . . . مسدود کردن

body . . . . . بدنه

build . . . . . ساختن. در این‌جا منظور ساخت فایل **jar** است.

bulk . . . . . توده، فله

C

channel . . . . . کانال

cluster . . . . . خوشه

clustring . . . . . خوشه بندی

compression . . . . . فشرده سازی

consistency . . . . . سازگاری

crawl . . . . . خزیدن

D

data loss . . . . . از دست دادن داده

data science . . . . . علم داده

dataset . . . . . مجموعه‌ای از داده‌ها

deprecated . . . . . منسوخ شده

distributed . . . . . توزیع شده

distribution . . . . . توزیع

document-based . . . . . سندگرا

document . . . . . سند

F

failure . . . . . نقصان،خطا

fault tolerant . . . . . مقاوم در برابر خطا

fetcher . . . . . دریافت کننده

G

garbage collector . . . . . موجودیتی در جاوا است که حافظه اشیایی که دسترسی به آنها از بین رفته است را آزاد می کند

H

hard disk . . . . . دیسک سخت

history . . . . . تاریخچه

host . . . . . میزبان

I

index . . . . . فهرست کردن

instance . . . . . نمونه

L

latency . . . . . تاخیر

library . . . . . مجموعه‌ای از کلاس‌ها و توابع (کدها) که کاربرد خاصی دارند و در کنار هم جمع شده‌اند.

load . . . . . بارگیری

M

map . . . . . نگاشتن

master . . . . . ارباب

O

object . . . . . شیء

operation . . . . . عملیات

P

pair . . . . . جفت، دوتایی

parser . . . . . تجزیه کننده

platform . . . سکو، در اینجا منظور سیستم عامل است.

private . . . . . خصوصی

public . . . . . عمومی

publisher . . . . . انتشاردهنده داده (پیام) در کانال

Q

queue . . . . . صف

R

real time . . . . . بلادرنگ

reduce . . . . . کاهش دادن

replication . . . . . تکثیر

request . . . . . درخواست

response . . . . . واکنش

rtm (real time messaging) . پیام رسانی لحظه ای

S

slave . . . . . برده

snapshot . . . . . کپی

sparse . . . . . تنک

stream جریانی از داده‌ها که با سرعت بالا ارسال می‌شوند

structured . . . . . ساخت‌یافته

subscriber . . . . .در اینجا به معنای مشتری یک کانال که به تمام پیام های ارسال شده به کانال دسترسی دارد

T

tag . در اینجا منظور برچسب های فایل **HTML** است

thread . . . . . نخ

throughput . . . . . گذردهی

title . . . . . عنوان

transaction . . . . . معامله

U

unstructured . . . . . بدون ساختار

U

web page . . . . . صفحه وب