دانشگاه آزاد اسلامی

واحد تهران جنوب

دانشکده فنی مهندسی

پایان نامه کارشناسی

مهندسی(کامپیوتر-نرم افزار)

عنوان:

پیاده سازی یک روش indexing بر پایه ی Neo4j

استاد راهنما:
مهندس امیرحسین روحانی

نام و نام خانوادگی دانشجو:
زهرا نمینی میانجی

شماره دانشجویی:
۸۸۱۱۲۱۲۶۸۷

۱۳۹۲/۱۰

با سپاس ازسه وجود مقدس:

آنان که ناتوان شدند تا ما به توانايي برسيم ...

موهایشان سپید شد تا ماروسفید شویم...

و عاشقانه سوختند تا گرمابخش وجود ما و روشنگر راهمان باشند...

پدرانمان

مادرانمان

استادانمان

حالا من یک مهندسم!

فهرست مطالب

عنوان مطلب | شماره صفحه

منابع و ماخذ

فهرست منابع لاتین

سایت های اطلاع رسانی

فهرست اشکال

عنوان تصویر شماره صفحه

چکیده

در این پروژه ابتدا به معرفی دیتابیس های nosql می پردازم. در مورد موارد استفاده ی صحیح و انواع آن ها سخن خواهم گفت و سپس در مورد دیتابیس neo4j که یکی از انواع این نوع دیتابیس ها می باشد صحبت می کنم تا در ادامه بعد از معرفی ساختار search engine ها، از این دیتابیس برای معرفی یک روش جدید indexing استفاده خواهم کرد.

مقدمه

nosql به پایگاه داده هایی گفته می شود که relational نیستند و ساختارهای ارتباطی آنها با پایگاه داده های ارتباطی متفاوت است. اصطلاح nosql نامی عمومی است که به مجموعه‌ای از پایگاه‌های داده اطلاق می‌شود که از زبان پرس‌وجوی ساخت‌یافته sql یا مدل داده رابطه‌ای استفاده نمی‌کنند. گاهی این اصطلاح را مخفف not only sql می‌دانند تا تأکید کنند که طرفداران انواع پایگاه‌های داده غیررابطه‌ای معتقدند که پایگاه‌های داده رابطه‌ای سنتی تنها راه موجود برای ذخیره‌سازی داده نیستند، اما این به آن معنا نیست که به خودی خود انتخاب نادرستی باشند. این دیتابیس ها به ٤ دسته تقسیم می شوند که در ادامه با آن ها آشنا خواهیم شد، اما در این پروژه graph database ها مدنظر ما هستند زیرا که مدل پیشنهادی indexing بر پایه ی یکی از انواع این دسته از دیتابیس های nosql پیاده سازی شده است.

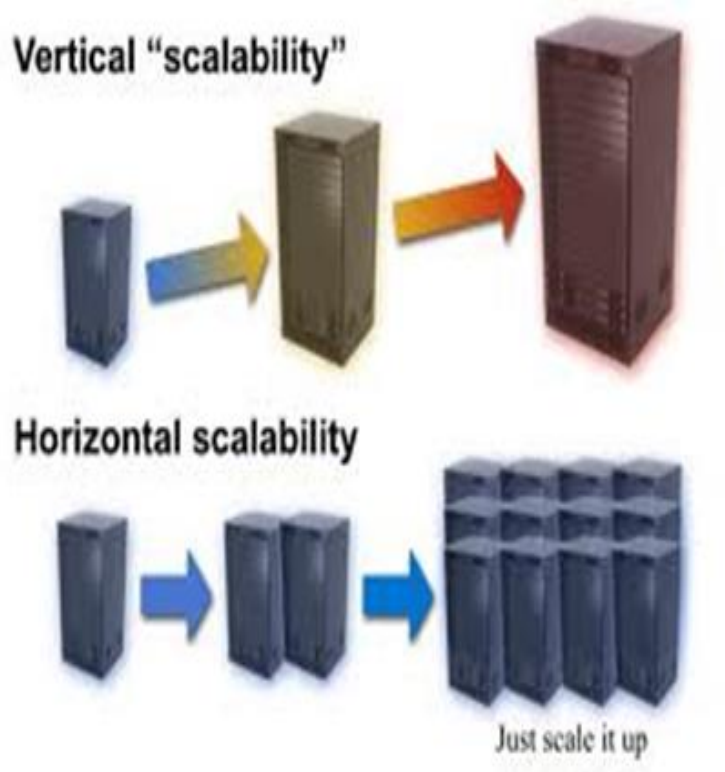
فصل اول:

۱ معرفی پایگاه داده های nosql

۱-۱ دیتابیس های nosql

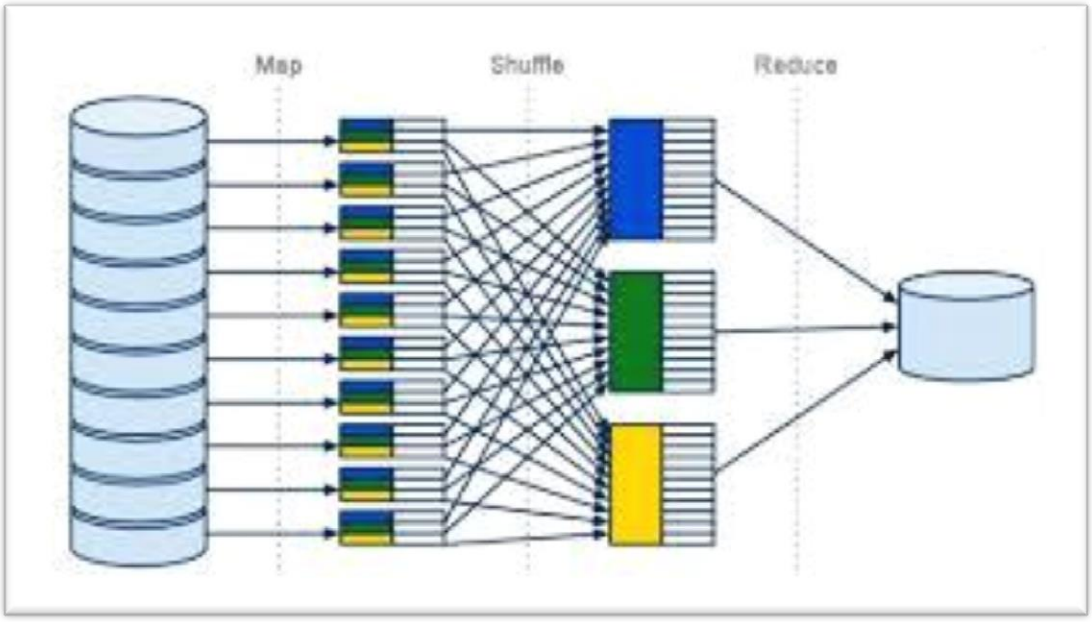
با این دیتابیس ها در مقدمه آشنا شدیم. از ویژگی های خاص این نوع دیتابیس ها می توان به طراحی بدون شما و گسترش پذیری اشاره کرد. در واقع این دو ویژگی مهم باعث توانایی این نوع دیتابیس ها برای مواجه با حجم اطلاعات زیاد یا به اصطلاح big data ها، می شود.

گسترش پذیری دو نوع دارد؛ گسترش پذیری عمودی و افقی. در گسترش پذیری افقی به ارتقا سخت افزار می پردازیم، مانند استفاده از cpu های قوی تر، حجم حافظه ی بیش تر و ... که هزینه را بیش تر می-کند. اما در گسترش پذیری عمودی از مدل map-reduce استفاده می کنیم.



تصویر ۱-۱: گسترش پذیری افقی و عمودی

در مدل map-reduce ، گره اصلی پروسس ها را بین گره های کارگر پش می کند. گره های کارگر نیز ممکن است این روش را ادامه دهند و تشکیل یک نمودار درختی را دهند. سپس هر گره بعد از حل مساله جواب را به گره بالاتر برمی گرداند و در آخر گره اصلی با join پاسخ ها، خروجی مساله را میسازد. مبحث map-reduce ، زیر مبحثی از grid computing محسوب می شود.



تصویر ۱-۲: مدل map-reduce

۲-۱ انواع پایگاه داده های nosql

۴ رده و گروه عمده بانک‌های اطلاعاتی NoSQL وجود دارند؛ شامل:

الف) key-value stores که پایه بانک‌های اطلاعاتی nosql را تشکیل داده و اهدافی عمومی را دنبال می‌کنند.

ب) wide column stores که در شرکت‌های بزرگ اینترنتی بیشتر مورد استفاده قرار گرفته‌اند.

ج) document stores یا بانک‌های اطلاعاتی nosql سندگرا.

د) graph databases که بیشتر برای ردیابی ارتباطات بین موجودیت‌ها بکار می‌روند.

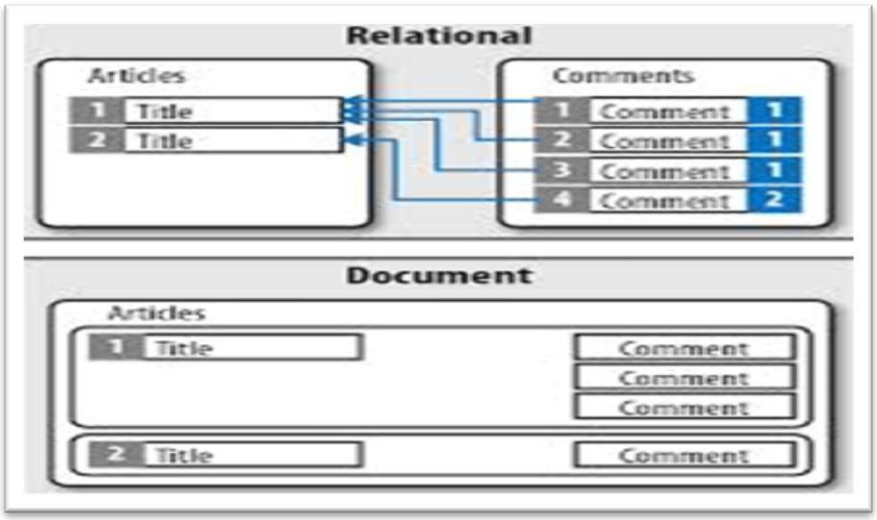
در زیر مثال هایی از این ۴ دسته مشاهده می کنید:

تصویر ۱-۳: مثالی از key-value store ها

این شکل معرف نمونه ای از دیتابیس های key-value است که هر key شامل یکسری اطلاعات مربوط به آن کلید خاص می باشد. این نوع دیتابیس ها به دلیل خاصیت ذخیره سازی اطلاعات به همراه کلید مربوطه، دارای بالاترین سرعت جست و جو می باشند. از معروفترین انواع این دیتابیس ها، redis را می توان نام برد.

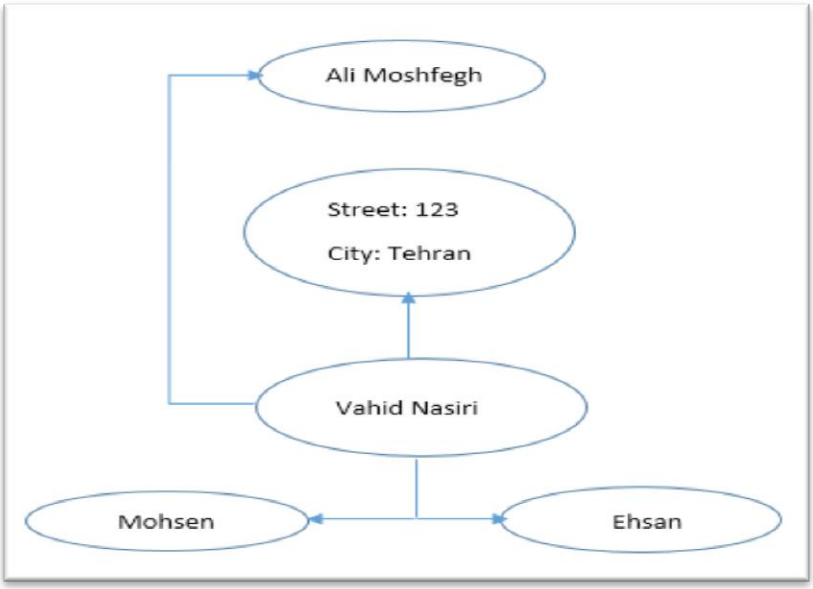
تصویر ۱-۴: مثالی از wide column store ها

شکل بالا نمونه ای از wide column store ها می باشد. همان طور که در شکل مشاهده می کنید جدولی که اطلاعات را بر اساس سطر جدا کرده، اگر بخواهیم به این نوع از دیتابیس های nosql تبدیل کنیم، جدول به ستون های تشکیل دهنده اش تجزیه شده و هر داده با id مخصوص خود ذخیره می شود. از این نوع دیتابیس ها بیش تر برای مواجه با big data ها استفاده می کنند و معروفترین آن ها cassandra است.



تصویر ۱-۵: مثالی از document store ها

طبق شکل مشاهده می شود که دو جدول موجود در حالت relational که به یکدیگر refrence داده شده اند، در این نوع دیتابیس ها اطلاعات مربوط به یکدیگر از دو جدول با هم و در قالب یک سند ذخیره می شوند. از معروفترین انواع این دیتابیس ها می توان mongo db را نام برد.



تصویر ۱-۶: مثالی از graph database ها

این شکل یک graph database را نشان می دهد. graph databases نـوع خاصـی از بانـک‌هـای اطلاعاتی nosql هستند که جهت ردیابی ارتباطات بین اطلاعات طراحی شده‌اند و برای برنامه‌های شبکه‌هـای اجتماعی بسیار مفید هستند. در واژه نامه این بانک‌های اطلاعاتی nodes و edges اتصال دهنـده‌هـای نودهـا تعریف شده‌اند. در اینجا نودها می‌توانند دارای خاصیت‌ها و مقادیر متناظر با آن‌ها باشند. یکی از معروفترین graph databasesمورد استفاده، neo4j نام دارد.

در این شکل یک شخص را که دارای رابطه آدرس با شیء آدرس ذکر شده است را مشاهده مـی‌کنیـد. همچنین این شخص دارای رابطه دوستی با سه شخص دیگر است.

۳-۱ Acidity

acidیا atomicity, consistency, isolation وdurability یک روش اسـاندارد بـرای قضـاوت درباره data integrity یا جامعیت داده بین DBMSها است، این مجموعه از خصوصیات تضمین می دهنـد که تراکنش های پایگاه داده با اطمینان انجام می شوند.

I. atomicity به معنی این است که هر تراکنش یا باید کامل اجرا شود یا اصلا اجرا نشود.

II. consistency تضمین می دهد که هر تراکنش، پایگاه داده را از یک وضعیت سالم و معتبر به یک وضعیت سالم و معتبر دیگر می برد.

III. isolation تضمین می دهد که اجرای همزمان ترکنش ها مشکلی در سلامت پایگاه داده ایجاد نکند.

IV. durability به معنی ذخیره کردن تغییرات داده ای اعمال شده توسط هر تراکنش بمنظور بازگردادن پایگاه داده به وضعیت قبل از تراکنش در زمان های crashe یا رخ دادن خطا.

به دلیل اطلاق واژه ی nosql به طیف عظیمی از پایگاه داده ها، رویکرد هر یک نسبت به قضیه ی acidity متفاوت است. در این پروژه چون به طور خاص از پایگاه داده ی neo4j استفاده می شود، acidity را درباره ی این پایگاه داده ی خاص تحقیق کردم. در وبسایت رسمی این پایگاه داده full acid معرفی شده است.

البته neo4j به دلیل ساختار گره و لبه قابلیت گسترش پذیری افقی را ندارد؛ اما در مدل پیشنهادی برای indexing ، به دلیل تقسیم گراف اصلی بر اساس تقسیم بندی سایت منبع به چند زیر گراف، این مشکل حل می شود.

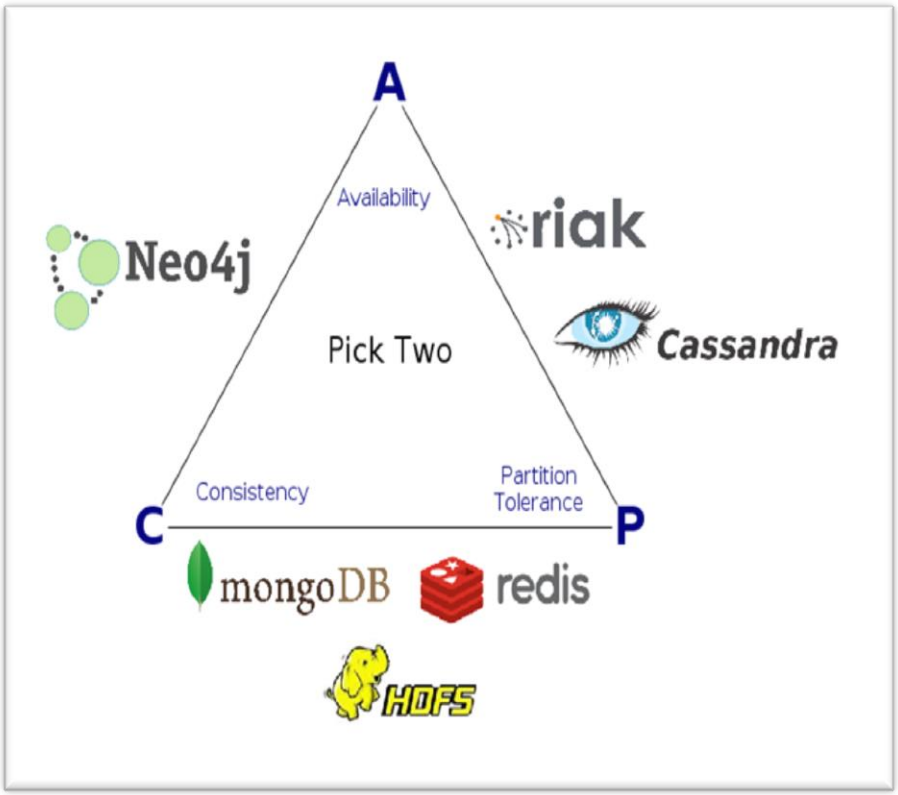
CAP Theorem ۴-۱

تصویر ۱-۷: قضیه ی cap

در این دیاگرام سه گوشه اصلی نشانگر ثبات (consistency)، در دسترس بودن (availability)و قابلیت بخش بخش سازی (partition tolerance)هستند. ثبات در اینجا یعني همه کلاینت‌ها همواره به داده‌هاي مشابه دسترسی داشته باشند، در دسترس بودن یعني همه کلاینت‌ها امکان خواندن و نوشتن را داشته باشند و قابلیت بخش بخش سازی نیز به معنای این است که سیستم کلی بتواند در تمام بخش‌هاي شبکه فیزیکی کارکند. بر اساس نظریه cap، تنها دو عنصر از این سه عنصر در سیستم‌هاي واقعی قابل انتخاب هستند و بر همین اساس، برای داشتن هر جفت مشخصه، مي‌توان راه‌حلی را که روي ضلع مشترک آن‌ها آورده شده است، انتخاب کرد.

به دلیل اطلاق واژه ی nosql به طیف عظیمی از دیتابیس ها، رویکرد هر یک در مقابل نظریه ی cap متفاوت است؛ اما همان طور که قبلا گفتم در این پروژه درباره ی دیتابیس neo4j صحبت می کنم. neo4j دو

ویژگی availablity و consistency را داراست. دلیل لین که partition tolerance را ندارد، این است که partition tolerance به این معنی است که برخلاف وجود تقسیم بندی های فیزیکی، سیستم به خوبی کار کند و در neo4j به دلیل ساختار گره و لبه و relationship بین گره ها این تقسیم بندی امکان پذیر نیست؛ اما در این مدل indexing، همراه با تقسیم بندی های وب سایت، گراف نیز به چند زیر گراف شکسته می شود و مشکل partition tolerance به این صورت حل می شود.



تصویر ۱-۸: بررسی قضیه ی cap برای پایگاه داده یneo4j

در ادامه، به دلیل توانایی دیتابیس های nosql در مواجه با big data ها و سرعت بالاتر عملیاتی آن ها، برای استفاده ی کاربردی از این دیتابیس ها، یک موتور جست و جو برای وب سایت را ارائه خواهم داد.

فصل دوم:

۲ معرفی ساختار موتورهای جست وجو و پایگاه داده ی neo4j

۲-۱ موتورهای جست و جو

اعضای کلی یک موتور جست و جو عبارتند از :

I. spider ها: کدهای html سایت ها را بررسی کرده و دسته بندی می کنند. برای این کار از الگوریتم- های هوش مصنوعی استفاده می کنند.

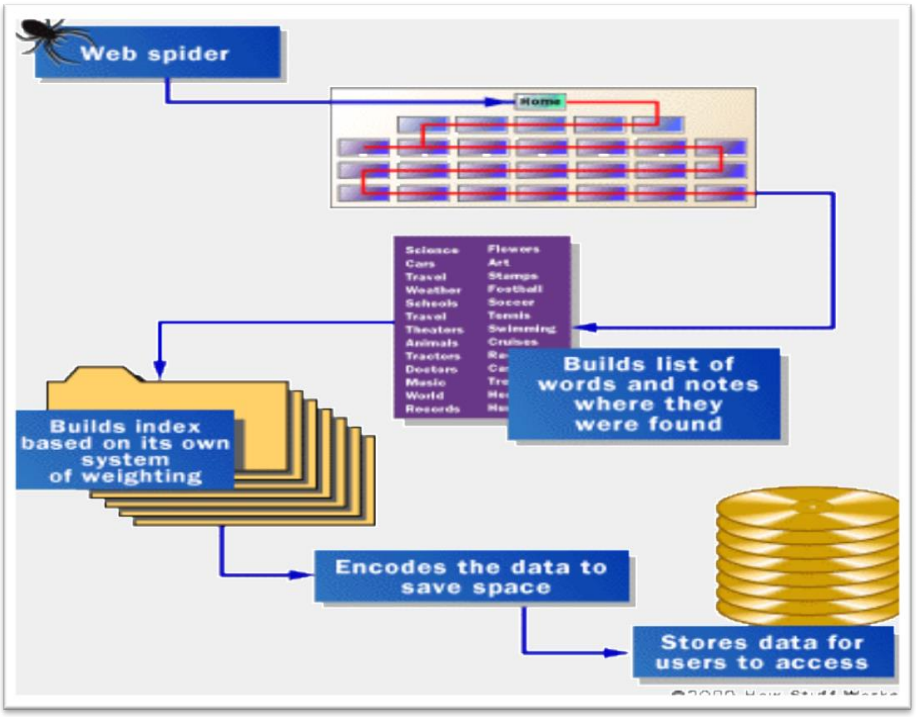
II. crawler ها: بر کار spider ها نظارت دارند و مکانیزم دسته بندی و نوع اطلاعاتی که باید دسته بندی شوند را مشخص می کنند.

III. indexer ها: روابط منطقی بین اطلاعات را مشخص می کنند.

IV. database ها: برای ذخیره اطلاعات استفاده می شوند.

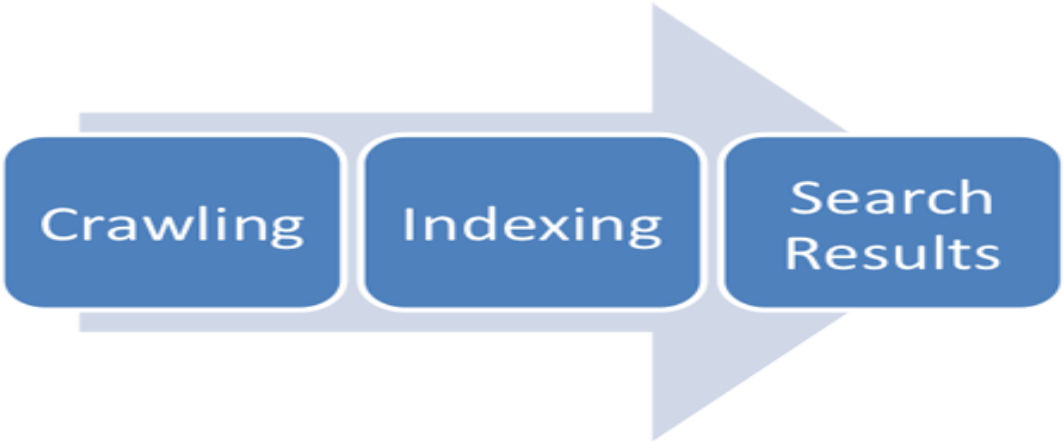
V. ranker ها: معیارهای اولویت بندی نمایش اطلاعات به کاربر را مشخص می کنند.

VI. برای مثال lucene یک کتابخانه ی آماده برای موتورهای جست وجو است و nutch یک crawler و solr یک indexer آماده است.



تصویر ۲-۱: ساختار یک موتور جست و جو

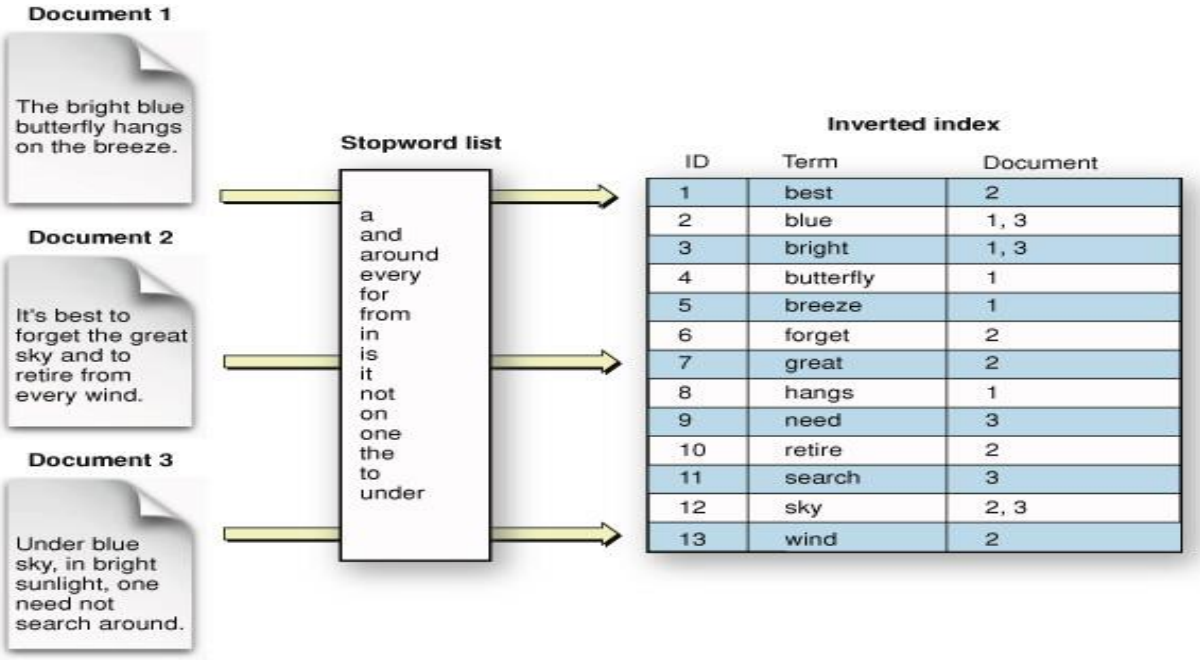
برای ساخت موتور جست و جو، برای قسمت web crawler ، من از کتابخانه ی آماده ی jsoup استفاده کردم.



تصویر ۲-۲: پس از مرحله ی crawling ،indexing را خواهیم داشت.

برای بخش indexing ، پس از بررسی الگوریتم های معروف indexing که شامل، b-tree ، hashmap ، inverse indexing (معروفترین و پرکاربردترین الگوریتم که در lucene و wikipedia استفاده می شود.) و secondry index cassandra (cassandra یک nosql دیتابیس است که یک سیستم indexing آماده دارد که طرز کاری مشابه با inverse indexing دارد.) هستند، inverse indexing را به دلیل وسعت کاربرد انتخاب کردم. در ادامه نحوه ی کار آن را بررسی می کنیم.

Inverse Indexing ۲-۲



تصویر ۲-۳: مراحل inverse indexing

همان طور که در شکل مشاهده می کنید، نحوه ی index کردن یک document به این صورت است که کلمات کلیدی برای جست و جو انتخاب شده و document های مربوط به آن کلمات، همراه آن کلمه ذخیره می شوند؛ و برای جست و جوی یک رشته در یک document دو روش موجود است؛ روش اول این که کلمه ی اول جست و جو و document های مربوط به آن پیدا و در آن به جست و جوی کلمه ی دوم بپردازیم و این رویه را تا آخر ادامه دهیم. در روش دوم رشته را به کلمات تشکیل دهنده ی آن شکسته و document های هر یک را جست و جو و در آخر اشتراک بگیریم. روش دوم سریع تر و بهینه تر است؛ پس با این روش ادامه می دهیم. شکل زیر مثالی از این روش است.

```
T[0] = "it is what it is"
T[1] = "what is it"
T[2] = "it is a banana"
```

we have the following inverted file index (where the integers in the set notation brackets refer to the indexes (or keys) of the text symbols, `T[0]`, `T[1]` etc.):

```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

A term search for the terms `"what"`, `"is"` and `"it"` would give the set $\{0,1\} \cap \{0,1,2\} \cap \{0,1,2\} = \{0,1\}$.

تصویر ۲-۴: مثالی از جست و جوی یک رشته به وسیله ی الگوریتم inverse indexing

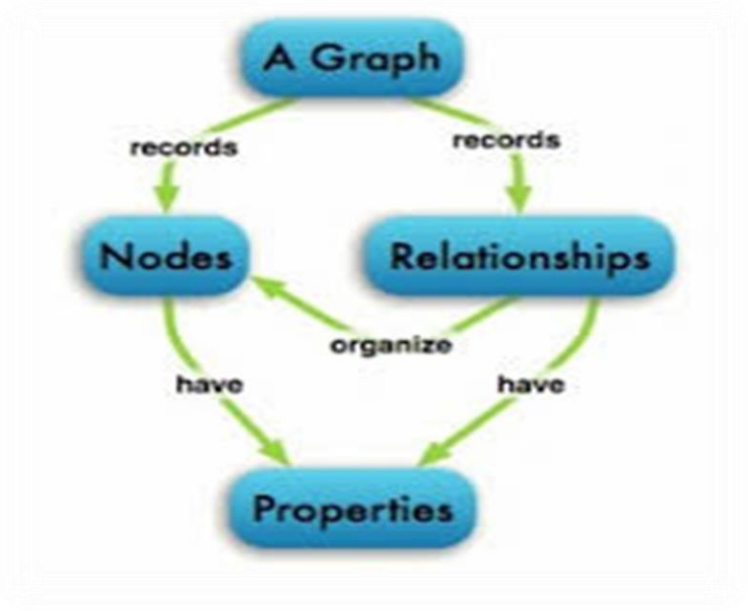
در شکل بالا ۳ رشته داریم که کلمات کلیدی مورد جست و جو index شده اند، و document هـای مربوطه مشخص شده اند. حال برای جست و جوی رشته ی "what is it" ، document های هر سه را با هم اشتراک می گیریم.

پس از بررسی این الگوریتم متوجه یک انتقاد شدم که به طور مثال اگر یک رشته ی ٥ کلمه ای داشته باشیم و ٤ تا از این کلمات جز کلمات مورد جست و جو باشند، حتی اگر ۳ کلمه با هـم document مشـترک داشته باشند، به دلیل عدم وجود document مشترک برای کلمـه ی چهـارم، نتیجـه ی جسـت و جـو تهـی برگردانده می شود.این جست و جوی هوشمندی نیست؛ به همین دلیل با تحقیق بیش تر به یک مـدل index کردن اطلاعات دست یافتم که به وسیله ی گراف پیاده سازی می شود و توانایی برطرف کـردن ایـن مشـکل را دارد. برای پیاده سازی این مدل نیاز به یک گراف دیتابیس داشتم که به همین منظور از neo4j استفاده کردم.

Neo4j ۳-۲

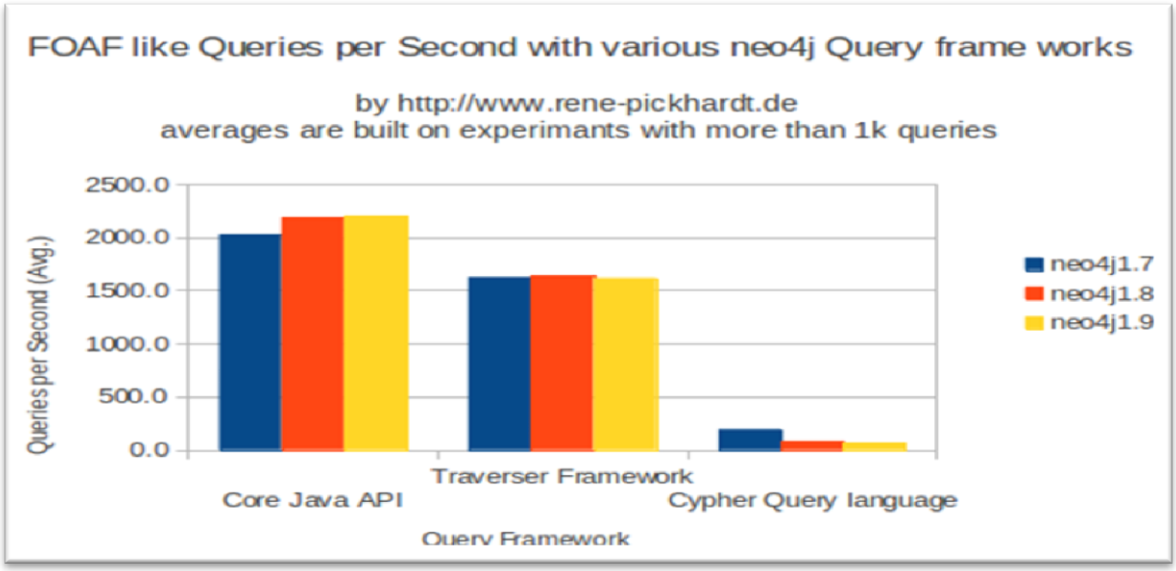
یک گراف دیتابیس است که می توان ۳۲ بیلیون node در آن قرار داد. node ها هر یک خاصیت های مربوط به خود را دارند و محل ذخیره ی اطلاعات می باشند. node ها با relation ها با یکدیگر ارتباط دارنـد. این پایگاه داده به طور خودکار گسترش پذیری را پشتیبانی نمی کند، اما در این مدل وقتی قرار اسـت کـه هـر تب، یک گراف جداگانه تشکیل دهد، پس می توان هر گراف را جداگانه جست و جو و در آخـر نتـایج را بـا هـم

مقایسه کرد.



تصویر ۲-۵: ساختار پایگاه داده ی neo4j

در کار با neo4j ، چند روش موجود است که در زیر مشاهده می کنید که مقایسه شده اند.



تصویر ۲-۶: مقایسه ی روش های کار با پایگاه داده ی neo4j

فصل سوم:

۳ ارائه ی یک مدل indexing بر پایه ی پایگاه داده ی neo4j

۳-۱ نحوه ی Indexing

ابتدا کلمات قابل جست و جو برای هرتب از منو در سایت به طور جدا انتخاب و برای هر یک از تب ها و زیر تب هایش و کلمات کلیدی صفحه ی تب مورد نظر یک گراف تشکیل می دهیم که کلمات موردنظر، گره های گراف و لبه ها نیز براساس ارتباط بین کلمات مشخص می شوند؛ به این صورت که تب اصلی گره مرتبه اول و زیر تب ها به ترتیب گره های مرتبه های پایین تر را تشکیل می دهند و هر گره مرتبه ی بالاتر با گره های مرتبه های پایین تر ارتباط دارد. برای جست و جو نیز دو روش برای شناسایی رشته در گراف وجود دارد؛ ایجاد subgraph و برچسب گذاری relation ها وجود داشت؛ راه حل ایجاد subgraph به این صورت بود که با مشخص شدن کلمات، گره ها مشخص و زیر گراف ایجاد میشد و با پیمایش زیرگراف، نتیجه ی جست و جو مشخص می شد؛ اما در این روش مشکل اشغال حافظه توسط زیرگراف وجود داشت.

اما در روش برچسب گذاری، پس از مشخص شدن گره های موردنظر، برچسب وضعیت relation بین آن گره ها به حالت فعال تبدیل شده و فقط آن لبه ها پیمایش می شوند.

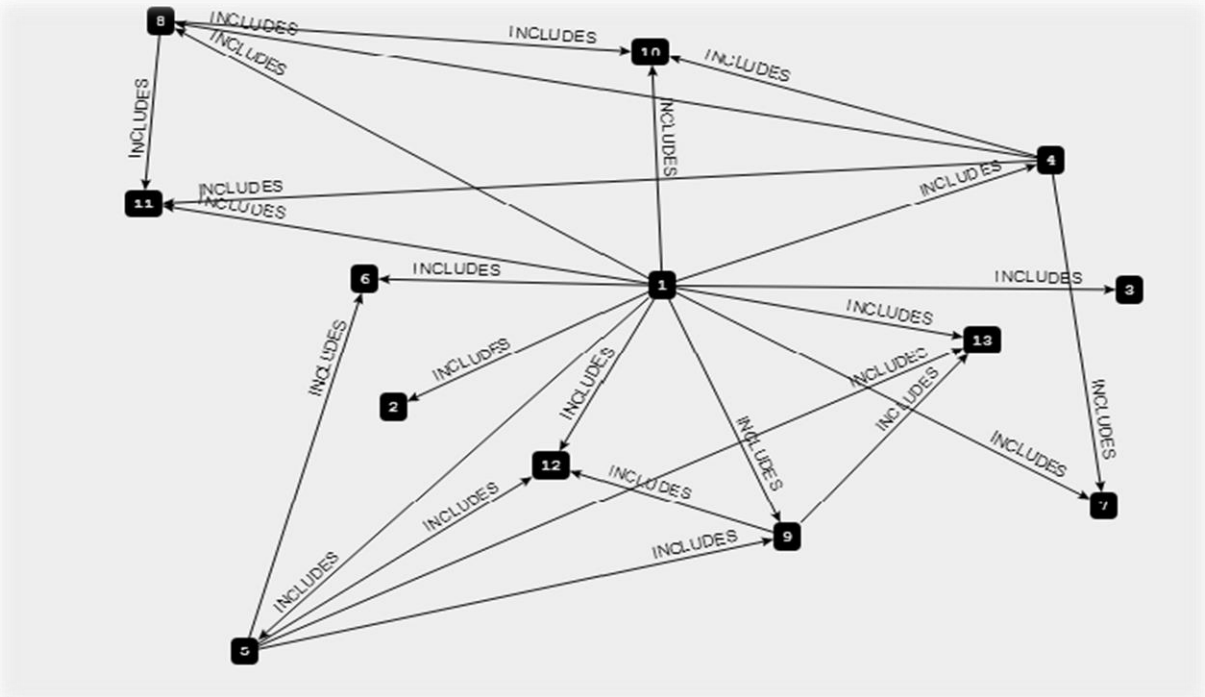
۳-۲ DAG Graph

در گراف های ایجاد شده، برای جلوگیری از ایجاد loop وdeadlock ، گراف های ایجاد شده تبدیل به گراف dag می شوند. این گراف، گرافی است که در آن هیچ دوری وجود ندارد.

۳-۳ ترسیم گراف

در ترسیم گراف ابتدا وجود گره ی تکراری ممنوع بود؛ اما با جست و جوی بیش تر به این نتیجه رسیدم که نمایش گره های مشابه به عنوان نتیجه ی جست و جو نیز می تواند سود مند باشد؛ پس وجود گره های تکراری مجاز شد. در اشکال زیر نمودار و گراف مربوطه را مشاهده می کنید.

تصویر ۳-۱: ساختار درختی مدل آزمایشی



تصویر ۳-۲: گراف ایجاد شده برای مدل آزمایشی در پایگاه داده ی neo4j

نتیجه گیری

به طور کل بهترین حالت استفاده از پایگاه داده های nosql ، همراه کردن آن ها بـا پایگـاه داده هـای relational است. بسیاری از شرکت های بزرگ نیز از این امر تبعیت می کنند؛ مانند facebook .

استفاده از nosql+sql به این معنی نیست که یک داده را اول در nosql و بعد در sql دائمی کنـیم، بلکه یعنی داده به چند بخش تقسیم شود و هر بخش در جای خود قرار گیرد.

فهرست منابع لاتین

Professional NoSql (written by Tiwari)

سایت های اطلاع رسانی

1. docs.neo4j.org
2. stackoverflow.com
3. github.com
4. groups.google.com
5. gogseo.com
6. dzone.com
7. rosettacode.org
8. datastax.com
9. wiki.apache.org
10. wikipedia.org
11. blog.jelastic.com