مقایسه ی عملکرد پرس و جو های KNN بر روی پایگاه داده های مکانی

مهدی سامانی[1]، مهدی فولادگر[2] و مژگان چاکری[3]
[1]دانشگاه صنعتی اصفهان، m.samani@ec.iut.ac.ir
[2]دانشگاه صنعتی اصفهان، m.fooladgar@ec.iut.ac.ir
[3]دانشگاه صنعتی اصفهان، m.chakeriyazdi@ec.iut.ac.ir

چکیده – امروزه پایگاه داده های مکانی به منظور انجام پرس وجو های مکانی نظیر یافتن فاصله، یافتن *K* نزدیک ترین همسایه(*KNN*) و یافتن محیط استفاده می شوند. پرس و جوی *KNN* یک پرس و جوی کاربردی است که زیربنای بسیاری از نرم افزار های مبتنی بر داده های مکانی را تشکیل می دهد. این پرس وجو ها زمانی داده ها در حجم بالا مورد استفاده قرار می گیرد بسیار زمانبر خواهد شد. در این مقاله هدف مقایسه ای میان عملکرد پایگاه داده های مکانی *GeoSpark* (*Non Relational , In-memory*)، *MemSQL* (*Relational , In-memory*) و *PostGIS* (*Relational , Disk database*) برای انجام پرس و جوی *KNN* می باشد. همچنین به معرفی پایگاه داده های مختلف و بررسی ساختار ایندکس ها مکانی مانند *R-Tree* برای تسریع در سرعت اینگونه پرس و جو ها در پایگاه داده های مختلف انجام شده است.

کلید واژه- KNN، GeoSpark، MemSQL، PostGIS، R-Tree

۱- مقدمه

امروزه حجم داده های مکانی به سرعت رو به افزایش است و در نتیجه آن، پرس و جو های مکانی نیز در حال توسعه و گسترش می‌باشند. پرس وجوی یافتن k نزدیک ترین همسایه(KNN) پرس وجویی است که شاکله ی اصلی بسیاری از برنامه های کاربردی و پرس و جو های مختلف، بر مبنای آن قرار گرفته است. به عنوان نمونه می توان برنامه هایی را در نظر گرفت که نیاز به یافتن k نزدیک ترین خودرو در اطراف یک نقطه ی مشخص دارد. در این صورت زمانی که با حجم انبوهی از داده سروکار داریم علاوه بر یافتن صحیح نقاط مورد نظر زمان یافتن این نقاط نیز بسیار اهمیت خواهد داشت.

امروزه پایگاه داده های مکانی مختلفی برای انجام چنین پرس وجوهایی به کمک کاربران آمده اند و با ارائه ی الگوریتم های متنوع و استفاده از ساختارهای ایندکس گوناگون تلاش به سرعت بخشیدن به این پرس و جوها کرده اند. از جمله این پایگاه داده ها می توان به GeoSpark، AsterixDB، Neo4j، MemSQL و PostGIS اشاره کرد.

GeoSpark یک سیستم محاسباتی خوشه ای برای پردازش داده های مکانی در مقیاس بزرگ است. GeoSpark یک توسعه ای از Apache Spark است که با بهره گیری از Spatial Resilient Distributed Datasets (SRDDs) بارگذاری، پردازش و تجزیه و تحلیل داده های مکانی در مقیاس بزرگ را در سراسر ماشین ها به طور موثر انجام می دهد.

AsterixDB پایگاه‌داده‌ای است که توسط شرکت Apache توسعه داده شده و پشتیبانی می‌شود. این پایگاه‌داده به کاربران اجازه می‌دهد از بسترهای فایل‌سیستم‌های توزیع‌شده[1] استفاده کرده و پایگاه‌داده‌ای نیم-ساختاریافته[2] را با زبانی مشابه با SQL مدیریت کرده و به اجرای جست‌وجوهای خود بپردازند. این پایگاه‌داده به صورت پیش فرض مفهوم نقطه و ایندکس‌های مکانی را پشتیبانی می‌کند.

Neo4j یک کتابخانه و پلتفرم مبتنی بر جاوا برای پیاده‌سازی و مدل‌سازی گراف‌هاست. Neo4JSpatial یک افزونه

[1] Distributed

[2] Semi-Structured

برای Neo4j است که توابع مورد نیاز برای پرس‌وجوهای مکـانی را فراهم آورده است.

MemSQL یک پایگاه‌داده مقیم در حافظـه[۳] رابطـه‌ای اسـت این پایگاه‌داده می‌تواند داده‌ها را به صـورت سـطری و یـا سـتونی ذخیره نماید. این پایگاه‌داده با پیاده‌سازی توابـع مکـانی بـه بسـتر مناسبی برای اجرای پرس‌وجوهای مکانی تبدیل شده است.

GeoMesa یک پایگاه داده ی توزیع شـده ی مکـانی-زمـانی متن باز است که بر روی سیستم های ذخیره سـازی ابـری ماننـد HBase، Accumulo، Cassandra کـار مـی کنـد. در واقـع GeoMesa بـا اسـتفاده از پـردازش مـوازی و اسـتراتژی ایندکس گذاری همانند عملکرد PostGIS بر روی Postgresql، پرس و جـو های مکانی-زمانی را بر روی این سیستم های توزیع شـده فـراهم می سازد.(متاسفانه مشکلاتی کـه در اجـرای ایـن پایگـاه داده بـر روی این سیستم های نام برده شده به وجود آمـد باعـث شـد کـه آن را از آزمایشات و مقایسه ها کنار گذاریم. این مشـکلات بیشـتر در زمینه ی راه اندازی و اجرای پرس و جو هـای مکـانی بـر روی این بستر همچنین عدم وجود داکیومنت مناسب می باشد.)

PostGIS نرم افزاری منبع باز است که به منظور بهره گیـری از داده های مکانی بـه پایگـاه داده رابطـه ایPostgreSQL اضـافه می‌شود. درواقع postgis با پشتیبانی از داده های جغرافیایی، ایـن امکان را برای کاربر فراهم می‌کند تا پرس و جوهـای مکـانی را در SQL اجرا کند .

در این مقاله پس از معرفی ابزار هـا و مـدل هـای لازم بـرای بررسی قـدرت پایگاه‌داده‌هـای مختلـف در اجـرای پرس‌وجوهـای مکانی، به معرفـی و بررسـی الگـوریتم هـای KNN و Similarity Path پرداخته می‌شود و در نهایت در پایگاه داده های مختلـف بـا توجه به خصوصیات آن ها و بر اساس پارامتر های متفـاوت مـورد مقایسه و ارزیابی قرار می‌گیرند.

۲- معرفی ابزار و مدل ها

پس از بررسی‌های اولیه پایگاه‌داده‌های متنوعی نظیر Spark، POSTGis، AsterixDB، Neo4JSpatial و ... به عنوان کاندیدهای اولیه برای اجرای پرس‌وجوهای KNN مطرح شدند. که هر یک از این پایگاه‌های داده ویژگی‌های منحصر به فرد برای نوع خاصی از داده‌ها و پرس‌جوها را تامین می‌کنند. در نهایت پس از بررسی از میان این پایگاه‌داده‌ها سه پایگاه‌داده به عنوان کاندیدا مقایسه انتخاب شدند.

در این قسمت ما ابتدا به معرفی دیتاستی انتخابی خود برای انجام مقایسات و سپس به بررسی هر یک از این پایگاه‌های داده و مراحل پیاده‌سازی و اجرای پرس‌وجوهای KNN در آن‌ها خواهیم پرداخت.

۲-۱- Spark

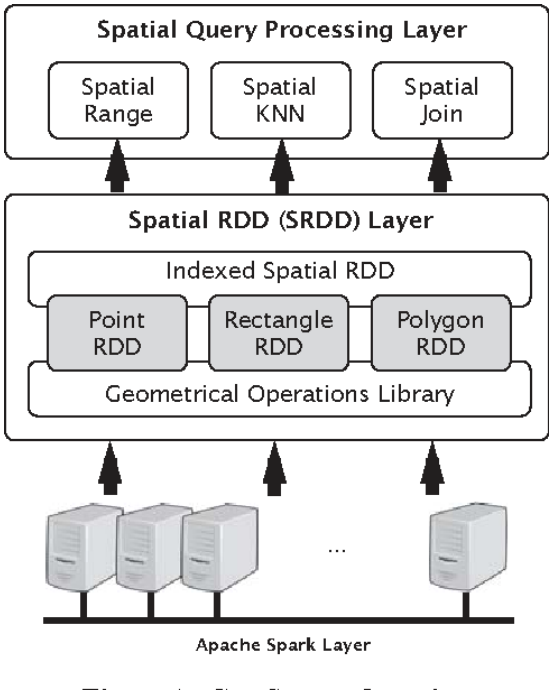
Apache Spark یک سیستم محاسباتی خوشه ای در حافظه است. اسپارک یـک سـطح انتـزاع از داده را فـراهم مـی کنـد کـه resilient distributed datasets نام دارد و داده ها بر روی تمـامی ماشین ها توزیع شده انـد. RDD هـای موجـود در حافظـه باعـث شده که اسپارک نسبت به رقیب های خود در امـر پـردازش هـای توزیع شـده ماننـد MapReduce بهتـر عمـل کنـد[1] . متاسـفانه spark خود به تنهایی داده های مکـانی را پشـتیبانی نمـی کنـد و برای کار با داده های مکانی نیاز به یـک افزونـه ماننـد GeoSpark است.

۲-۲- GeoSpark

GeoSpark یک فریم ورک برای پردازش داده های مکانی بـا حجم بسیار زیاد است که داده ها را در حافظـه ی اصـلی ذخیـره می کند و این باعث تسریع در امر محاسبات و پردازش ها خواهـد شـد. GeoSpark عـلاوه بـر پشـتیبانی از داده هـای مکـانی از ایندکس و سایر عملیات بر روی داده های مکانی ماننـد KNN و... پشـتیبانی مـی کنـد. اسـپارک بـا پشـتیبانی از سـاختار Spatial Resilient Distributed Dataset (SRDD) باعث می شود ذخیـره و پردازش داده های مکانی بر روی بستر اسپارک به راحتـی انجـام شود. همچنین GeoSpark از سـاختار هـای R-tree و Quad-tree برای ایندکس گذاری بر روی پارتیشن های داده استفاده می کنـد که باعث سرعت بخشیدن به پردازش های مکانی خواهـد شـد. در شکل ۱ لایه های تشکیل دهنده ی GeoSpark نشـان داده شـده اسـت. لایـه ی اول Apache Spark را نشـان مـی دهـد کـه در قسمت Spark توضیح داده شد و عملیـات ذخیـره و بازیـابی داده

[۳] In Memory

ها در این لایـه انجـام مـی شـود. لایـه ی دوم (SRDD) در واقـع همان توسعه ی اسپارک برای ذخیره سازی داده های مکـانی مـی باشـد. PointRDD، RectangleRDD و PolygonRDD از جملـه ساختارهای قابل تعریف در این فریم ورک می باشد. در نهایـت در لایـه ی آخـر کـه Spatial Query Processing نـام دارد پـرس وجوهای مکانی مانند range، join و KNN را در بر میگیرد. بـرای پـردازش پـرس و جـو KNN مکـانی، GeoSpark بـا اسـتفاده از الگوریتم top-k مبتنی بر heap که شامل دو مرحلـه اسـت عمـل میکند: انتخـاب[۴] و ادغـام[۵]. بـرای محاسـبه نزدیـک تـرین اجسـام اطراف نقطـه P در مرحلـه انتخـاب، بـرای هـر پارتیشـن SRDD، GeoSpark فاصله ی بین هر شی بـا نقطـه داده شـده را محاسـبه می کند، سپس با افزودن یا حذف عناصر بـر اسـاس فاصـله، یـک هیپ[۶] محلی به وجـود مـی آورد. در ایـن هیـپ k نزدیـک تـرین اجسام به نقطه ی P وجود خواهند داشت[1] .



شکل ۱لایه های تشکیل دهنده یGeoSpark

PostGIS -۲-۳

همانگونه که قبلا ذکر شد، postgis، پرس و جو هـای مکـانی روی پایگاه داده های رابطه ای را ممکن مـی‌سـازد بـدین منظـور کـــافی اســـت ابتـــدا در پایگـــاه داده PostgreSQL، postgis extension را نصب نمود و جـداول، شـماها، توابـع، ایندکس هـا و tigger های مکانی را ساخت و پـرس و جـو هـای مکـانی مـورد نظر را اعمال نمود. پس از ساخت جداول و توابـع و ... مـی‌تـوان از فایل هایی با فرمـت Text، Binary و CSV داده هـا را بـا تعیـین پارامتر های مختلف چـون Delimiter، OID و ... بـه جـداول وارد نمود. برای کار با داده های مکانی عمومـا نیـاز اسـت کـه در ایـن داده های وارد شده تغییراتی اعمال شود تـا بـرای اعمـال پـرس و

[۴] selection

[۵] merge

[۶] heap

جو های مکانی آماده سازی شوند. برای مثال در فایل وارد شده، مکان در هر رکورد به صورت نقطه ذخیره نشده است بلکه طول و عرض جغرافیایی آن به صورت جداگانه قرار داده شده است. حال برای آماده سازی این نوع فایل برای اعمال پرس و جو های مکانی کافیست با استفاده از توابع موجود در PostGIS دو attribute طول و عرض جغرافیایی را به یک attribute مکانی[۷] نقطه جغرافیایی تبدیل کرد و بدین ترتیب فایل داده ها را برای اعمال پرس و جو های مکانی آماده می‌شود. به طور کلی، در PostGIS می‌توان attribute های مکانی مختلف از جمله Point، Polygon، MultiLineString و ... را ساخت و استفاده نمود.

برای درک بیشتر از مفاهیم PostGIS، ساختار آن به طور خلاصه در ادامه بیان می‌شود. همانطور که گفته شد، PostGIS در واقع پلاگینی است بر روی پایگاه داده PostgreSQL که کتابخانه های مختلفی دارد که عبارتست از :

- GEOS: شامل توابع پیش فرض، نوع داده ها و عملگرهای مکانی چون Intersects، Union، Distance، Point و....
- Proj4: شامل فیلتر های استاندارد UNIX که برای تبدیل مختصات جغرافیایی به مختصات دکارتی مورد استفاده قرار می‌گیرد.
- LibXML2: کتابخانه ای که می‌تواند برنامه به زبان های مختلف C++، python و ... را به یکدیگر مرتبط نماید.

در این گزارش، از کتابخانه GEOS این پلاگین استفاده شده است و از توابع و عملگر های پیش فرض آن استفاده شده است.

در ادامه این بخش، به معرفی چند تابع و نوع داده و عملگر پیش فرضی که در این گزارش از آن استفاده شده است پرداخته می‌شود.

ST_MakePoint: تابعی که با گرفتن دو مقدار اعشاری، نقطه ای به مختصات آن می‌سازد. اما در PostGIS باید به هر نقطه ای که ساخته می‌شود یک شناسه خاص یا SRID نسبت داد که این کار با تابع ST_SetSRID و با در نظر گرفتن یک ID مثلا ۴۳۲۶ برای نقطه خاصی ممکن می‌شود. اما برای تبدیل یک نقطه که به صورت متنی از کاربر گرفته می‌شود، شناسه SRID با استفاده از تابع ST_GeomFromText ممکن است.

ST_Distance: با گرفتن دو مقدار Geometry، فاصله آن دو را بر می‌گرداند. که این فاصله بر حسب متر می‌باشد.

ST_Makeline: این تابع با گرفتن چندین نقطه یک خط که می‌تواند باز یا بسته باشد (مثل چندضلعی) را بسازد. این تابع برای ساخت مسیر در این گزارش استفاده شده است.

ST_HausdorffDistance: برای برگرداندن فاصله Hausdorff دو داده geometry استفاده می‌شود که برای اندازه گیری شباهت یا عدم شباهت دو داده مقداری اعشاری را بر می‌گرداند.

توابع پیش فرض زیادی در این پلاگین پیاده سازی شده است اما در اینجا چند تا از توابع معروف و استفاده شده در این گزارش بیان شد. [3]

AsterixDB -۲-۴

همانگونه که قبلا ذکر شد، AsterixDB، به ما اجازه می‌دهد تا پرس‌وجوهای مکانی را در محیط‌های توزیع شده و بر روی داده‌های نیم-ساختاریافته اجرا کرده و نتیجه را بیابیم. علاوه بر آن CloudBerry نیز با برقراری ارتباط با این پایگاه‌داده به ما در نمایش[۸] در آوردن داده ها کمک خواهد کرد.

این پایگاه‌داده برای اجرای پرس‌وجوهای پیشرفته‌ای نظیر KNN راه‌حل آماده ای ندارد لذا یکی از ساده‌ترین راه‌حل‌ها تعریف تابع فاصله و اجرای پرس‌وجوی Top-K بر روی آن‌هاست.

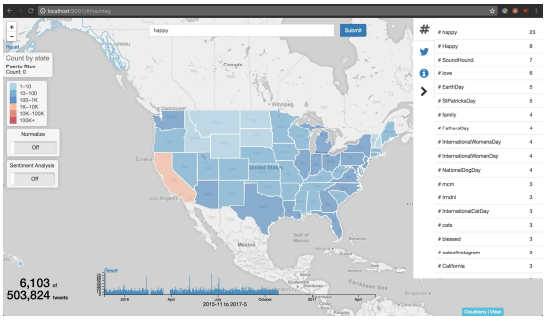
[۷] Geography type

[۸] Visualize

بدیهی است که این راهکار برای این حجم از داده راهکاری معقول و کارا نیست.

در این پایگاه‌داده می‌توان با یکی از دو زبان SQL++ و یا AQL پرس‌وجوهای مورد نیاز را نوشت. هر دو زبان توسط سازندگان AsterixDB گسترش یافته اند، لذا مستندات این زبان‌ها تنها در سایت رسمی AsterixDB موجود است و همچنین هنوز به بلوغ خود نرسیده است. به طور مثال در این زبان ابزار لازم برای تجزیه[۹] زمان و نقطه از فایل‌های CSV هنوز پیاده‌سازی نشده است.

یکی از نقاط مثبت این پایگاه‌داده وجود پلتفرم‌های نمایش داده‌ها با نام CloudBerry است. این پلتفرم محیطی ساده و شکیل برای نمایش داده‌ها در اختیار کاربران قرار می‌دهد. از طرف دیگر وجود ابزارهایی برای ورود استریم داده‌هایی نظیر توئیت‌های توئیتر نیز از دیگر نقاط مثبت این پایگاه‌داده است.



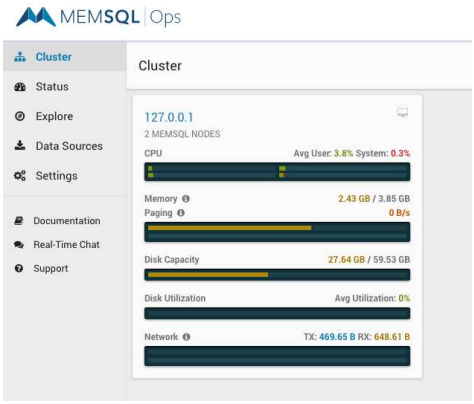
شکل ۲ نمونه ای از محیط اجرای AsterixDB

در مورد ایندکس‌های پیاده‌سازی شده در این پایگاه‌داده می‌توان روش اندیس‌گذاری R-Tree را نام برد.

۵-۲- MemSQL

MemSQL یک پایگاه‌داده مقیم در حافظه است. این پایگاه‌داده با قابلیت ذخیره‌سازی داده‌ها به دو شکل سطری و ستونی و پیاده‌سازی توابع مکانی به گزینه خوبی برای اجرا پرس‌وجوهای مکانی تبدیل شده است.

از طرف دیگر در اختیار داشتن محیط مناسبی برای مانیتور کردن منابع مصرفی این پایگاه‌داده را به ابزاری منحصر به فرد تبدیل می‌کند.



شکل ۳ محیط اجرای MemSQL

اما متاسفانه روش‌های اندیس‌گذاری در این پایگاه‌داده تنها محدود به B-Tree و Hash هستند که این موضوع کار را برای اندیس‌گذاری داده‌های مکانی سخت می‌کند. توابع مکانی پیاده‌سازی شده در این پایگاه‌داده عبارت‌اند از:

- GEOGRAPHY_AREA
- GEOGRAPHY_POINT
- GEOGRAPHY_DISTANCE
- و ...

برای ورود اطلاعات دیتاست به پایگاه‌داده نیازمند ابزاری برای خواندن فایل‌های CSV و ورود آن‌ها به پایگاه‌داده ایم. خوشبختانه MemSQL به طور پیش‌فرض این ابزار را فراهم کرده است. پس ابتدا باید جدول مورد نیاز برای داده‌ها را بسازیم برای این کار پرس‌وجوی زیر کاراست:

```
CREATE TABLE IF NOT EXISTS ADB (
        id BIGINT AUTO_INCREMENT,
        tid BIGINT,
        time CHAR(15),
        lat FLOAT,
        lon FLOAT,

        key(id)
);
```

شکل ۴ ساخت جدول

[۹] Parse

پس از آن با اجرای دستوری داده‌ها را داخل پایگاه‌داده وارد کرده و با ویرایش، آن‌ها را تبدیل به داده‌های از نوع مکانی می‌کنیم:

```
LOAD DATA INFILE '<path-to-data>'
INTO TABLE ADB
COLUMNS TERMINATED BY ',';

ALTER TABLE ADB ADD COLUMN loc GEOGRAPHYPOINT;

UPDATE ADB SET loc=GEOGRAPHY_POINT(lat, lon);
```

شکل ۵ وارد کردن داده در MemSQL

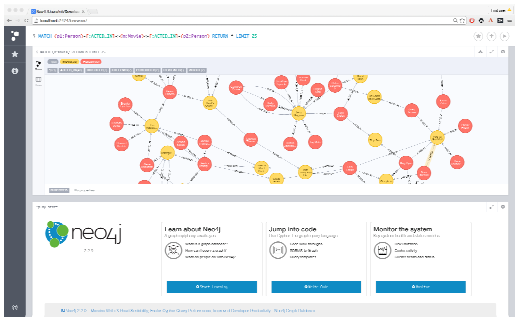
و پس از اجرای این پرس‌جو داده‌ها آماده پردازش هستند.

Neo4j -۲-۶

یکی دیگر از ابزارهای قابل استفاده برای تحلیل و اجرای پرس‌وجو بر روی داده‌های مکانی کتابخانه مبتی بر جاوا Neo4j است. این کتابخانه به مدل‌سازی مسائل و نگاشت آن‌ها به گراف کمک می‌کند.

از طرف دیگر وجود ابزار Neo4jSpatial با پیاده‌سازی توابع مکانی و روش اندیس‌گذاری RTree برای داده‌های مکانی باعث شده تا بتوانیم از این کتابخانه نیز برای پرس‌وجوهای مکانی استفاده نماییم.

Neo4j با فراهم آوردن محیط گرافیکی مناسب و توضیحات و مستندات مناسب به نظر محیطی کارا برای دانشمندان علوم کامپیوتر و داده است اما از نظر یک متخصص پایگاه‌داده انعطاف لازم را ممکن است نداشته باشد.



شکل ۶ نمونه ای از محیط Neo4j

۳- مدل ها و ایندکس ها

با توجه به گسترش داده ها و منابع اطلاعاتی، دیتاست های مورد استفاده نیز ممکن است بسیار بزرگ باشند و گاها انجام پرس و جو ها و بررسی مستقیم تمام رکورد ها نیز زمان بر باشند. به همین دلیل در پایگاه داده ها، ایندکس های مختلفی پیاده سازی شده است که یکی از این ایندکس ها، R-Tree می‌باشد. برای پیاده سازی این نوع ایندکس در بعضی پایگاه داده ها مثل PostGIS نیاز است که ابتدا ایندکسی به نام GiST ساخته شود و سپس ایندکسR-Tree پیاده شود. در ادامه ساختار این نوع دو ایندکس مورد بررسی قرار گرفته می‌شود.

R-Tree Index -۳-۱

این نوع ایندکس برای نخستین بار توسط گوتمن[۱۰] و برکلی[۱۱] برای تمام داده های مکانی و چند بعدی در سال ۱۹۸۴ ارائه شد. ساختار سلسله مراتبی و متعادل این نوع ایندکس شبیه ساختار ایندکس B-Tree می‌باشد اما برای ذخیره داده، روشی متفاوت را پیش می‌گیرد. ایده‌ی اصلی این ساختار، دسته بندی اشیا و قرار دادن اشیای نزدیک بهم در یک ناحیه مستطیلی می‌باشد. برای ساخت این درخت ابتدا اشیایی که فاصله شان از یکدیگر از یک آستانه مشخص کمتر باشد، در کوچکترین مستطیل ممکن قرار می‌گیرند و تمام اشیای موجود در آن مستطیل، در سطوح بعدی درخت با همان ناحیه در نظر گرفته شده، شناخته می‌شوند سپس مرز های ساخته شده نزدیک به هم در یک مستطیل بزرگتر قرار داده می‌شوند و به همین ترتیب این کار ادامه داده می‌شود تا یک مرز کلی ایجاد شود. در شکل ۷ روند ساخت این درخت قابل مشاهده است.

این ساختار ایندکس در هر نوع فضای مکانی قابل پیاده سازی می‌باشد. این بدین معنی است که اگر فضای مکانی دو بعدی باشد و داده ها را بتوان در دایره هایی با شعاع های مختلف در نظر گرفت، ساخت مستطیل های مرزی همچنان ممکن است و کوچکترین مستطیل های مرزی را می‌توان به اندازه قطر دایره داده ها، در نظر گرفت. نکته لازم به ذکر این است که ابعاد مرز-

[۱۰] Antonin Guttman

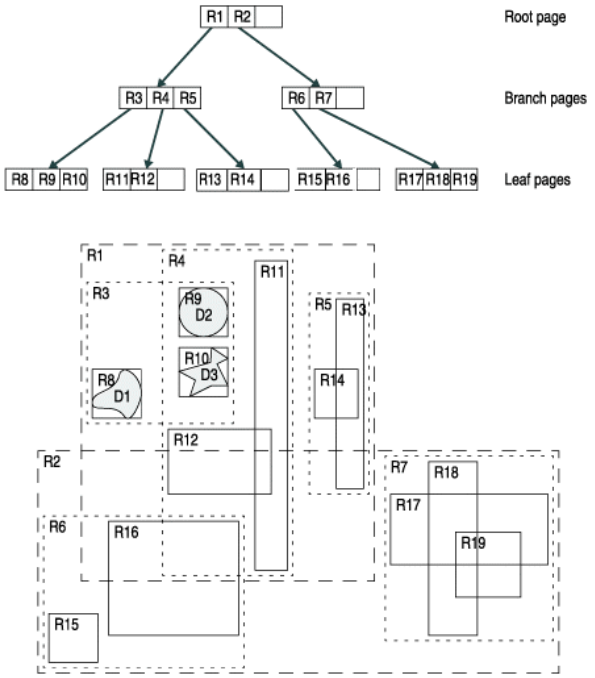
[۱۱] UC Berkley

های تشکیل داده شده، می‌تواند متغیری به جز مکان، مانند زمـان نیز باشد.

هرچند در ساختار تئوری R-Tree Index، در گره های برگ، داده ها باید ذخیره شوند اما در نسـخه جدیـد پایگـاه داده هـا بـه صورت پیش فرض اینگونه نیست و کپی از داده هـا در بـرگ نگـه داری نمی‌شود بلکه محدوده آن در برگ ذخیـره مـی‌شـود کـه در این صورت به ایـن نـوع ایـنـدکس، bounding-box-only R-tree گفته می‌شود کـه مزیـت هـای ایـن نـوع ایـنـدکس عبارتسـت از: استفاده بهینه از حافظه و زمان، کاهش فضای مورد نیاز بـرای بـه روز رسانی درخت و افزایش کارایی.

همانطور که ذکر شد، ساختار این نوع ایندکس، یـک درخـت متعادل است که این بدین معنی است که از ریشـه تـا بـرگ، هـر مسیری طی شود، از تعداد گره های یکسانی عبور می‌شود. یـا بـه عبارتی دیگر، تمام برگ ها در یک سطح قرار دارند.

ساختار R-Tree index، کـه در ایـن گـزارش مـورد اسـتفاده قرار گرفته است در گره های برگ علاوه بر کپی از کلیـد یـا داده، اشاره گری به گره های قبلی نیـز ذخیـره مـی‌شـود. همچنـین در گره های میانی و ریشه، مرز شامل مرز های کوچکتر و اشـاره گـر به گره های سـطوح پـایین تـر ذخیـره مـی‌شـوند. بـدین صـورت پیمایش بالا به پایین درخت و پیمـایش پـایین بـه بـالای درخـت فراهم می‌شود. در این نوع ایندکس، با توجه به ضـریب انشـعاب[۱۲] حداکثر و حـداقل تعـداد گـره هـای ذخیـره شـده در هـر سـطح مشخص می‌شود . [1] [2]



شکل ۷ ساختار R-Tree برای ایندکس مکانی

GiST Index -۳-۲

برای پیـاده سـازی R-Tree در postgis لازم اسـت کـه ابتـدا GiST بر روی پایگاه داده ساخته شود و سـپس روی آن، R-Tree را می‌توان در نظر گرفت. GiST، اشیا را براسـاس اینکـه موقعیـت مکانیشان نسبت به ناحیه مورد نظر، چگونـه اسـت تقسـیم بنـدی می‌کند و بدین ترتیب اشیایی که هیچ تداخلی با ناحیه ندارنـد در یک دسته، اشیایی که با ناحیه هم پوشانی دارند در دسته دیگـر و اشیایی که درون ناحیه هستند نیز در دسته سوم قرار می‌گیرند.

نکته مثبت پیاده سـازی R-Tree بـر روی GiST را مـی‌تـوان در دو مورد زیر خلاصه نمود :

۱- Null Safe: ایندکس بر روی داده هایی که سـتون مـورد نظرشان Null است، نیز ساخته می‌شود.

۲- قابلیت Lossines : این قابلیت زمانی اهمیت پیدا می‌کنـد که داده های GIS بزرگتر از 8K باشند. بدین ترتیـب در این مواقع، در ساخت ایندکس، قسمت هـای مهـم داده در ایندکس ذخیره می‌شود.

[۱۲] Branching factor

۴- پرس و جوی KNN

۴-۱ GeoSpark

پس از انجام مراحل نصب و راه اندازی GeoSpark با استفاده از زبان scala کوئری KNN برای دیتاست مربوطه را در مرحله ی اول بدون استفاده از ایندکس (شکل ۸) و در مرحله ی بعد با استفاده از ساختار ایندکس R-Tree (شکل ۹) اجرا می کنیم.

```
import org.datasyslab.geospark.spatialOperator.KNNQuery;
import org.datasyslab.geospark.spatialRDD.PointRDD;
import com.vividsolutions.jts.geom.GeometryFactory;
import com.vividsolutions.jts.geom.Point;
import com.vividsolutions.jts.geom.Coordinate;
import org.datasyslab.geospark.enums.FileDataSplitter;

val fact=new GeometryFactory();
val queryPoint=fact.createPoint(new Coordinate(-109.73, 35.08));
/* Range query window format: X Y */
val objectRDD = new PointRDD(sc, "/home/hduser/Desktop/all.csv", 2, FileDataSplitter.CSV, false);
/*
 * 2 is the starting column of spatial data in the input file.
 * FileDataSplitter.CSV means the data format is CSV. We CSV, TSV, WKT, GeoJSON and self-defined format mapper.
 * false means each spatial object doesn't need to carry the original input tuple with it.
 */
val resultSize = KNNQuery.SpatialKnnQuery(objectRDD, queryPoint, 5,false).size();
/* The number 5 means 5 nearest neighbors
 * The false means don't use spatial index.
 */
```

شکل ۸ پرس و جوی KNN بر روی بستر GeoSpark با استفاده از زبان Scala بدون استفاده از ساختار ایندکس

```
import org.datasyslab.geospark.spatialOperator.KNNQuery;
import org.datasyslab.geospark.spatialRDD.PointRDD;
import com.vividsolutions.jts.geom.GeometryFactory;
import com.vividsolutions.jts.geom.Point;
import com.vividsolutions.jts.geom.Coordinate;
import org.datasyslab.geospark.enums.FileDataSplitter;
import org.datasyslab.geospark.enums.IndexType;

val fact=new GeometryFactory();
val queryPoint=fact.createPoint(new Coordinate(-109.73, 35.08));
/* Range query window format: X Y */
val objectRDD = new PointRDD(sc, "/home/hduser/Desktop/all.csv", 2, FileDataSplitter.CSV, false);
/*
 * 2 is the starting column of spatial data in the input file.
 * FileDataSplitter.CSV means the data format is CSV. We CSV, TSV, WKT, GeoJSON and self-defined format mapper.
 * false means each spatial object doesn't need to carry the original input tuple with it.
 */
objectRDD.buildIndex(IndexType.RTREE,false);
/*
 * false means just build index on original spatial RDD instead of spatial partitioned RDD.
 */
val resultSize = KNNQuery.SpatialKnnQuery(objectRDD, queryPoint, 5,true).size();
/* The number 5 means 5 nearest neighbors
 * The true means use spatial index.
```

شکل ۹ پرس و جوی KNN بر روی بستر GeoSpark با استفاده از زبان Scala با استفاده از ساختار ایندکس

قبل از اجرای پرس و جوها باید فایل دیتاست را در محیط هادوپ اضافه کنیم که چون محیط هادوپ به گونه ای تنظیم شده که فایل ها را به صورت MB۱۲۸ ذخیره می کند پس در این صورت فایل دیتاست (MB۷۵۲) به صورت ۶ فایل MB ۱۲۸ تبدیل خواهد شد.

ابتدا در حالت بدون استفاده از ساختار ایندکس کتابخانه های مورد نظر را به برنامه اضافه کرده سپس نقطه ای که میخواهیم حول آن کوئری KNN را اجرا کنیم با استفاده از تابع createpoint تعریف می کنیم. در مرحله ی بعد RDD را با استفاده از فایل CSV حاوی نقاط ایجاد می کنیم. عدد ۲ در ورودی این تابع نشان دهنده ی ستون شروع شونده ی داده های مکانی می باشد. در نهایت با استفاده از تابع SpatialKNNQuery از داده های موجود کوئری می گیریم. عدد ۵ در ورودی این تابع نشان دهنده ی تعداد K در این پرس و جو می باشد. همچنین پارامتر ورودی false در این تابع نشان دهنده ی اجرای پرس و جو بدون استفاده از ساختار ایندکس می باشد.

در حالت بعدی همان روند با استفاده از ایندکس بر روی داده ها انجام می شود. همانطور که پیش از این نیز بیان شد در GeoSpark ایندکس ها بر روی هر RDD به صورت مستقل تعریف می شود. برای ساخت ایندکس بر روی داده ها از تابع buildIndex استفاده شده است و نوع ایندکس استفاده شده R-Tree می باشد.

نتایج حاصل از این پرس و جوها در قسمت ارزیابی و نتایج نشان داده شده است.

۴-۲ MemSQL

برای یافتن نزدیک ترین همسایگان به یک نقطه در MemSQL باید در وهله اول می‌توان از یکی از ابتدایی ترین روش‌ها یعنی مرتب‌سازی بر اساس فاصله تمامی نقاط دیتاست و یافتن K کمترین آن‌ها ها این مهم را تحقق بخشید. پرس‌وجوی لازم برای این کار با چند خط قابل به پیاده‌سازی است.

```
SELECT DISTINCT tid
FROM
  ADB
ORDER BY GEOGRAPHY_DISTANCE(loc, GEOGRAPHY_POINT(-72.1235, 42.3521)) ASC
LIMIT 10;
```

شکل ۱۰ پرس و جوی KNN بر روی MemSQL

نمونه خروجی این پرس‌وجو در شکل زیر قابل مشاهده است:

```
+------+
| tid  |
+------+
| 6708 |
|  173 |
| 3694 |
| 3646 |
| 3893 |
|  316 |
| 4405 |
| 4054 |
| 2628 |
| 4258 |
+------+
10 rows in set (0.62 sec)
```

شکل ۱۱ خروجی پرس و جوی KNN بر روی MemSQL

۳-۴- PostGIS

اگر پیدا کردن نزدیکترین خودروهای موجود به جایگاه خاصی مورد نظر باشد، کافی است پرس و جویی مبتنی بر الگوریتم KNN را روی دیتابیس مورد نظر اعمال کرد که بدین منظور راهکار های مختلفی وجود دارد، که در ادامه به بررسی آن ها در PostGIS پرداخته می‌شود.

ساده ترین راهکار این است که جدول یا جداول ترکیبی[۱۳] بر اساس فاصله تعریف شده در پرس و جو مرتب شود و K تا اولین رکورد ها را به عنوان نتیجه پرس و جو برگردانده شود. این الگوریتم در شکل ۱۲ قابل مشاهده است. اما باید توجه کرد در این روش، اگر تعداد رکورد های جدول یا جدول های مورد نظر، زیاد باشند یا اینکه K را تعداد بالایی در نظر گرفت، همین پرس و جوی ساده ممکن است مدت زمان زیادی طول بکشد.

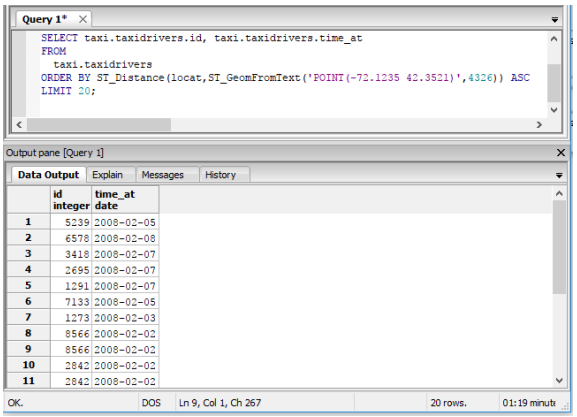
راهکار بعدی که برای پیاده سازی KNN می‌توان در نظر گرفت روش محدودیت ایندکس[۱۴] می‌باشد که در آن یک عدد شاخص یا به اصطلاح عدد جادویی[۱۵] نیاز است که با استفاده از آن، جعبه ای در محدوده ای که در پرس و جو بیان شده است، ساخته می‌شود و خودروهای قرار گرفته در آن جعبه به عنوان نتایج پرس و جو نشان داده می شود. این روش نیز در شکل ۱۳ آورده شده است. اما این روش نیز بدون نقص نیست و اگر رکوردی وجود نداشته باشد که در محدوده جعبه ساخته شده با عدد جادویی مورد نظر قرار گرفته باشد، هیچ نتیجه ای نشان داده نمی‌شود در حالیکه ما قصد داریم که نزدیکترین خودروها را پیدا کنیم هرچند نزدیکترین خودرو در فاصله بسیار زیادی از آن باشد.

اما روش اصلی که برای پیاده سازی الگوریتم KNN در postgis استفاده می‌شود، ارزیابی فاصله در محدوده مرز های ساخته شده در ایندکس R-Tree در postgis می‌باشد. طبق توضیح ساختار R-Tree، در این نوع ایندکس از مرز ها برای تعیین محدوده استفاده می‌کند که به همین دلیل نزدیکترین خودروها بدون تعیین هیچ فاصله دقیقی، به دست می‌آید. برای پیاده سازی KNN با ایندکس R-Tree ابتدا لازم است با دو عملگر فاصله ای که در مرتب سازی استفاده می‌شوند آشنا شد.

<->: عملگری است که فاصله دو بعدی دو geometry را بر-می‌گرداند. این عملگر همراه با Order By می‌آید و کمترین/بیشترین فواصل را بر می‌گرداند. در واقع این عملگر از محاسبه تک تک نقاط موجود با نقطه داده شده و سپس مرتب کردن فواصل و برگرداندن بیشترین/کمترین فواصل جلوگیری می‌نماید. این فاصله همان فاصله مراکز دو داده geometry می-باشد.

<#> : فاصله بین لبه های جعبه های در نظر گرفته شده در ساختار ایندکس را بر می‌گرداند.

با استفاده از این دو عملگر می‌توان ایندکس مکانی مناسب را پیاده کرد و پرس و جوی مورد نظر را بر اساس آن پیاده سازی کرد. بدین ترتیب، زمان اجرای پرس و جو از مرتبه O(N) به مرتبه $O(\log^{n}{}_{b})$ کاهش پیدا می‌کند. (b تعداد مرز های در نظر گرفته شده می‌باشد.)



شکل ۱۲ Simple KNN

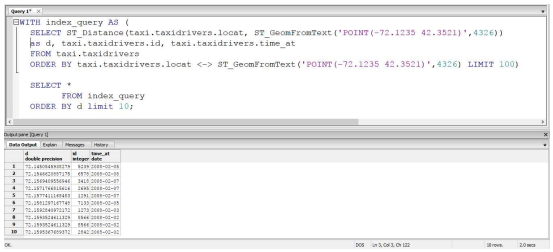
برای پیاده سازی KNN با استفاده از ایندکس در postgis، ایندکس هایی بجز R-Tree مانند B-Tree، GiST، BRIN و ... وجود دارد. که R-Tree و GiST در بخش های قبلی مورد بررسی قرار گرفته شد.

[۱۳] Joined Tables

[۱۴] Index constraint

[۱۵] Magic number

شکل ۱۳ Index Constraint KNN

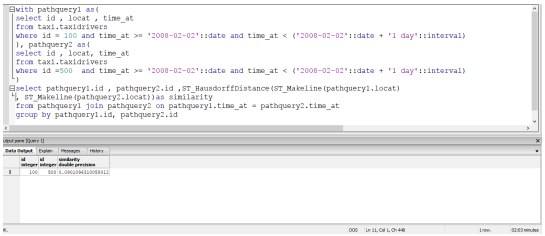


شکل ۱۴ Index Based KNN

۵- پرس و جوی شباهت سنجی مسیر ها

در پایگاه داده های مکانی، یکی از مواردی که می‌تواند مورد پرس و جو قرار گیرد، مسیر های مختلف و میزان شباهت مسیر ها می‌باشد. راهکاری که در پایگاه داده های مختلف از جمله PostGIS به کار برده می‌شود، این است که ابتدا مسیر های مورد نظر با استفاده از داده های geometry چون خط، چند ضلعی و ... ساخته شود و سپس به نحوی فاصله نقطه به نقطه این دو مسیر محاسبه شود و در نهایت فاصله کلی این دو مسیر و در واقع میزان اختلاف دو مسیر برگردانده شود.

برای این منظور در PostGIS از تابع ST_HausdorffDistance که در بخش قبل، توضیح داده شد استفاده می‌شود. برای مثال در یک پایگاه داده حاوی اطلاعات مکانی ماشین های مختلف در زمان های مختلف، ابتدا مسیر های دو ماشین را در یک روز به دست آورده و سپس با استفاده از تابع گفته شده، اختلاف دو مسیر بر حسب عدد اعشاری بر می‌گردانند. نمونه این پرس و جو و نتایج حاصل از آن را می‌توان در شکل ۱۵ مشاهده نمود.



شکل ۱۵ بررسی شباهت مسیر دو ماشین با شناسه ۱۰۰ و ۵۰۰

۶- ارزیابی و نتایج

۶-۱- دیتاست

یکی از مهم‌ترین و تاثیرگذار ترین پارامترهای مقایسه عملکرد پایگاه‌داده‌ها برای پرس‌وجوهای مختلف انتخاب پایگاه‌داده‌های مناسب است.

پس از بررسی دیتاست‌های مختلف در نهایت یک دیتاست با حجم حدودا ۸۰۰ مگابایت از مسیرهایی عبور ۱۰۳۵۷ تاکسی در بازه ۲ الی ۸ فوریه سال ۲۰۰۸ میلادی که توسط شرکت مایکروسافت جمع‌آوری و منتشر شده است، به عنوان دیتاست تست انتخاب شد.

در این دیتاست حدودا ۱۵ میلیون رکورد، در مجموع ۹ میلیون کیلومتر مسیر پیموده شده، متوسط زمان نمونه‌برداری ۱۷۷ ثانیه و متوسط فاصله بین هر دو نمونه‌برداری ۶۲۳ متر بوده است.

این دیتاست در هر رکورد شامل شناسه تاکسی، زمان نمونه‌برداری و محل نمونه‌برداری بوده است.

با خصوصیات ذکرشده برای این دیتاست پرس‌وجوهایی نظیر KNN و شباهت‌سنجی مسیرها برای این دیتاست پرس‌وجوهای معقولی به نظر می‌آید.

۶-۲- مشخصات کلاستر

مشخصات کامپیوتر های استفاده شده در این آزمایش به شرح زیر می باشد :

- CPU : تک هسته ای GH۲,۴
- RAM : ۲ گیگابایت
- HDD : SSD GB۲۰

برای اجرای پرس و جو های PostGIS و MemSQL از یک کامپیوتر با مشخصات فوق استفاده شده است و برای اجرای پرس و جو ها بر روی بستر GeoSpark از یک کلاستر با دو عدد WorkerNode با مشخصات فوق استفاده شده است.

۶-۳- نمودار ها و مقایسه ها

پس از اجرای پرس و جو ها بر روی بستر GeoSpark و محاسبه ی زمان اجرای آنها نتایج زیر در به دست آمد. در این جدول زمان اجرای هر RDD، حجم آن و محل اجرای آن

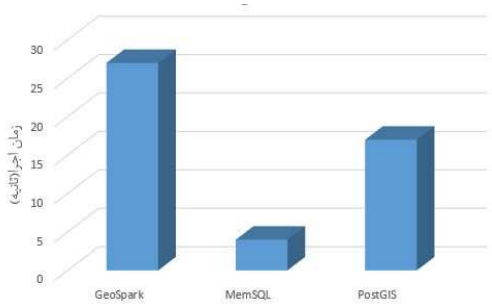
مشخص شده است. می توان مشاهده کرد که بیشترین زمان اجرا برای یک RDD در حدود ۱۲ ثانیه بوده است در نتیجه با افزایش حجم داده ها و اضافه شدن Worker node ها به راحتی می توان scalable بودن این روش را مشاهده کرد.

شماره	محل اجرا	زمان اجرا(S)	حجم(MB)
۱	Worker 2	12	128.1
۲	Worker 1	۱۲	128.1
۳	Worker 2	۷	128.1
۴	Worker 1	۷	128.1
۵	Worker 2	۶	128.1
۶	Worker 1	۶	112.2

در مجموع با استفاده از جدول زیر می توان نتیجه گرفت که هر کدام از worker node ها زمانی برابر با ۲۷ ثانیه کار کرده و حجم داده ی پردازش شده توسط هر کدام در این جدول نمایش داده شده است.

شماره	Node	زمان اجرا(S)	حجم(MB)
۱	Worker 1	27	384.2
۲	Worker 2	27	368.3

زمان اجرای پرس و جو ها به صورت کلی در نمودار شکل ۱۶ قابل مشاهده است. این نمودار زمان اجرای پرس و جوی KNN بدون استفاده از ایندکس را بر روی بستر های مختلف با یکدیگر مقایسه می کند. همانطور که مشاهده می شود برای دیتاست استفاده شده بهترین نتیجه با استفاده از پایگاه داده ی MemSQL به دست آمده است که زمانی برابر با ۴ ثانیه داشته است.



شکل ۱۶ مقایسه ی زمان اجرای پرس وجوی KNN بدون استفاده از ایندکس

۷- نتیجه‌گیری

در این گزارش ابتدا به معرفی پایگاه داده ها و مدل های ایندکس مورد استفاده پرداخته شد. سپس الگوریتم KNN با مثالی از پایگاه داده تاکسی ها نشان داده شد و نکاتی در حین پیاده سازی به تفکیک پایگاه داه های مختلف بیان شد. پس از آن الگوریتم های شباهت سنجی مسیر ها و نکات پیاده سازی آن در پایگاه داده های مختلف، بیان شد و در آخر هر کدام از الگوریتم های KNN و شباهت سنجی مسیرها با استفاده از ایندکس و بدون ایندکس در سه نوع پایگاه داده رابطه ای (PostgreSQL)، رابطه ای و درون حافظه ای (MemSQL) و غیررابطه ای و درون حافظه ای (GeoSpark) مورد ارزیابی و مقایسه قرار گرفت که MemSQL به عنوان بهترین پایگاه داده در دیتاست مورد استفاده، شناخته شد. البته ذکر این نکته حائز اهمیت است که در صورت نیاز به فضایی برای پردازش داده های بزرگ که بر روی یک کامپیوتر قابل ذخیره سازی نیست بهترین روش استفاده از الگوریتم های مقیاس پذیر[۱۶] است که بر روی بستر spark پیاده سازی شده اند.

مراجع

[1] I. Team, "IBM Informix R-Tree Index User's Guide," IBM, 2008.

[2] Antonin Guttman, Michael Stonebraker, "R-TREES: A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING," College of Engineering University of California, Berkeley, 1983.

[۱۶] scalable

P. team, "PostGIS 2.1.10dev Manual," Postgis, 2015. [3]

"MemSQL," [Online]. Available: https://docs.memsql.com. [4]

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma and M. Justin, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012. [5]

N. Roussopoulos, S. Kelley and F. Vincent, "Nearest neighbor queries," in *ACM sigmod record*, 1995. [6]

"GeoSpark," [Online]. Available: http://geospark.datasyslab.org/. [7]