A Parallel Query Engine for Interactive Spatiotemporal Analysis

Mihir Sathe
Dept. of Computer Science
Univ. of Southern California
msathe@usc.edu

Craig A. Knoblock
Information Sciences Institute
Univ. of Southern California
knoblock@isi.edu

Yao-Yi Chiang
Spatial Sciences Institute
Univ. of Southern California
yaoyic@usc.edu

Aaron Harris
Dept. of Mech. Engineering
Univ. of Southern California
arharris@usc.edu

ABSTRACT

Given the increasing popularity and availability of location tracking devices, large quantities of spatiotemporal data are available from many different sources. Quick interactive analysis of such data is important in order to understand the data, identify patterns, and eventually make a marketable product. Since the data do not necessarily follow the relational model and may require flexible processing possibly using advanced machine learning techniques, spatial databases or similar query tools do not make the best means for such analysis. Moreover, the high complexity of geometric operations makes the quick interactive analysis very difficult. In this paper, we present a highly flexible functional query engine that 1) works with multiple schema types, 2) provides fast response times by spatiotemporal indexing and parallelization, 3) helps understand the data using visualizations and 4) is highly extensible to easily add complex functionality. To demonstrate its usefulness, we use our tool to solve a real world problem of crime pattern analysis in Los Angeles County and compare the process with other well known tools.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Design, Performance, Experimentation, Human Factors, Algorithms

Keywords

Spatiotemporal analysis, spatial join, parallelization, indexing, visualization

SIGSPATIAL '14, Nov 04-07 2014, Dallas/Fort Worth, TX, USA

http://dx.doi.org/10.1145/2666310.2666437.

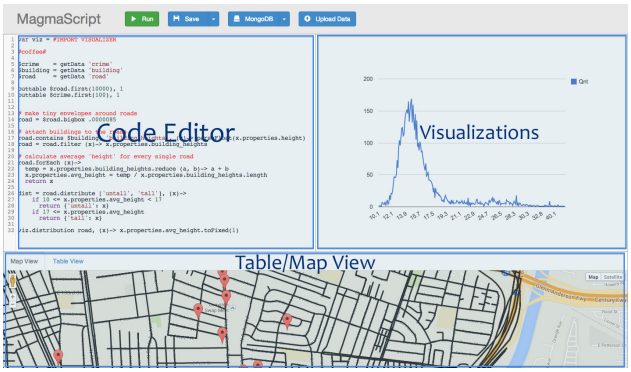

Figure 1: User interface for our tool

1. INTRODUCTION

Finding patterns in spatiotemporal data is much like criminal investigation. You collect, integrate, aggregate the data and make some queries to find some leads. You further drill down to make some educated guesses about possible patterns and then you query further to test your hypotheses. Fast response time queries are very important for such tasks. The high complexity of most geometric operations increases the response time of such queries. Another important requirement for such analysis is quick visualization which ranges from map-plotting to visualizing non-spatial quantities like aggregations and frequency distributions. Moreover, this analysis often needs advanced operations like anomaly detection and clustering which are not strictly spatial or temporal.

Through this paper, we present a unique system that provides efficient query processing through in-memory indexing and parallelization. It also provides data visualization and can be extended easily to add advanced functionality like machine learning. Our tool is deployed as a web application. Figure 1 shows the User Interface (UI) for our tool. The UI is divided into three different panels: the top left panel to write queries, the right panel for visualizations and bottom panel for tabular, and the map view of the data.

In the following section, we provide a motivating example of the analysis of statistical crime data in Los Angeles County. Section 3 contains details about the query model and capabilities of our system followed by a description of

the architecture in section 4. In section 5, we demonstrate the use of our tool to solve the problem described in section 2. We then evaluate our system in solving the given problem as compared to other well-known tools. Finally, we describe the related work and conclude with a discussion of our contribution and future scope of the project.

2. MOTIVATING EXAMPLE

Strategic crime analysis is the study of crime information integrated with sociodemographic and spatial factors to determine long term "patterns" of activity [2]. We obtained data about a total 148,638 crimes in Los Angeles (LA) County since October 1st 2013 from the LA County Sheriff's Department[1] and various city level police departments[2]. Most crimes include latitude and longitude, date and time of occurrence, date and time of report and category of the crime (provided by individual agencies). We also have data about the geographical borders of cities in the county. We will analyze the crime patterns in the cities of Compton and West Hollywood. Figure 2 shows the heat maps of the crimes in those cities. Red spots indicate the highest concentration of the crimes.

As a motivating example, we find the types of crimes that frequently occur spatiotemporally together. Since these crimes are more likely to be related to the same entity (e.g. person, gang, or riot), this analysis helps us profile the behavior of such an entity and hence can give better insight to law enforcement agencies. For example, if we find that drunk driving and assaults occur together frequently in a certain city, strict enforcement of driving laws near the places that serve alcohol will help reduce the number of assaults as well.

The first part of this problem involves identifying spatiotemporally co-occuring crimes followed by analysis of the association rules between these categories. We demonstrate the solution of this problem using our tool and present the results in Section 5. In Section 6, we compare the solution of this problem with our tool against solutions with well known tools.

3. QUERY MODEL

We make queries easy to read and write by using higher order functions. These functions take stateless 'example' functions as parameters and apply them on an entire dataset. This makes queries concise and parallelization-ready. We allow users to write queries in JavaScript, CoffeeScript or a combination of both.

3.1 Spatiotemporal Operations

Our tool supports all common spatiotemporal operations like joins and aggregations. More complex and domain-specific functionality can be added to this tool as an extension. All the operations take place over or between data collections. A collection here is an ordered list of objects where all objects represent similar entities such as a crime or a city. The objects inside the same collections can have different sets of properties or schemas.

All the join queries between two collections can be written in the following format:

<collection 1> . <operator> (<collection 2>, <label>, <selector>)

[1]http://shq.lasdnews.net/CrimeStats/LASDCrimeInfo.html
[2]http://www.crimemapping.com/

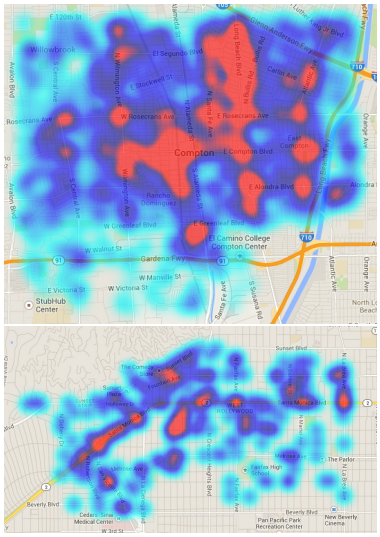

Figure 2: Crime heatmaps in the cities of Compton (above) and West Hollywood (below)

```
//Extension file HELLO.ext
(function() {
  return {
    sayHello: function() { return "Hello"; }
  }
})();
//Use in the script
var hello = #IMPORT HELLO
outtext ( hello.sayHello() ) // will print Hello
```

Figure 3: Procedure to create a simple extension

If an object from **Collection2** satisfies the criteria for join with an object in **Collection1**, part of the object of **Collection2** returned by the **selector** function is added to the given object of **Collection1** under the key specified by parameter **label**.

We support all common spatial **operators** like contains, covers, intersects and temporal operators like within, together and around. All operator inverses are supported as well. In figure 6, line 6 shows an example of join with the containedBy operator.

3.2 Extensions

Users can write their own extensions to add any desired functionality to the tool, allowing them to use this tool as an integrated environment for analysis. Moreover any existing algorithm implementation in JavaScript or CoffeeScript can be imported as an extension with little to no modification. Figure 3 shows the procedure to create and use a simple extension.

4. ARCHITECTURE

Users can import their data from relational databases, CSV files, GeoJSON files or MongoDB collections. All data is converted to the spatiotemporal GeoJSON format and stored in in-memory data store. Data can also be exported to the same sources.

Users write queries in a browser-based code editor. When a user runs a script, ther code is sent to a JavaScript sandbox on the server that contains an in-memory data store

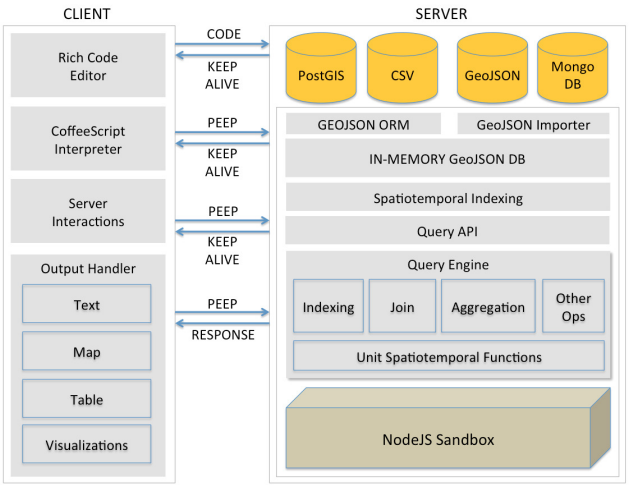

Figure 4: Architecture and request-response model

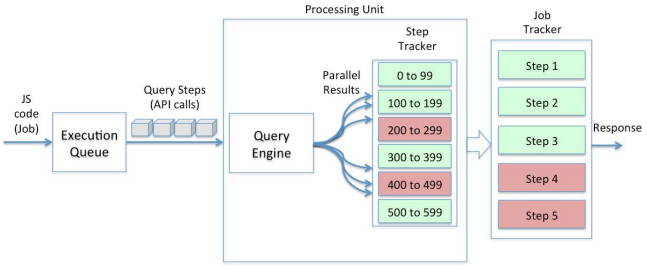

Figure 5: Workflow at the server side

and has full access to the query model described previously. Code from all imported extensions is also added to the sandbox. Asynchronous operations keep the clients waiting using KEEP ALIVE signal. In this case, a client keeps on sending PEEP signals until the response is received. Figure 4 shows the client-server interaction and overall architecture.

4.1 Parallel Processing Infrastructure

The main challenge for our parallelization is to put all the pieces of the output together correctly once all processes are done executing queries. The execution pipeline consists of a processing unit that fetches one instruction at a time from the execution queue, decides what collections to split and sends it to the processes to execute. Every query or job has a unique job identifier (ID) that helps identify the result. Once a process is done executing its part of the job, it passes on the results to the result unit which keeps track of the number of expected process outputs and number of outputs received. Once all the outputs are received, it updates the in-memory store with the new output (if required) and notifies the processing unit of the completion by giving it a callback. The processing unit will look for the next statement in the execution queue for the same job and proceed in the same way. If there are no more statements to be executed, it will end the processing and the server will send the response with all the results upon next request by the browser. Figure 5 shows the server side workflow including the parallel processing.

Association Rule	Support
Assault, Aggravated Assault, Vehicle/Boating Laws	23.86%
Theft/Larceny, Vehicle break-in/Theft, Assault	22.13%
Assault, Grand Theft Auto, Motor Vehicle Theft	20.61%

Association Rule	Support
Vehicle/Boating Laws, Narcotics, Non-aggravated Assault	29.91%
Drunk/Alcohol/Drugs, Theft/Larceny, Vehicle/Boating Laws	26.49%
Vehicle/Boating Laws, Aggravated Assault, Non-aggravated Assault	22.22%

Table 1: Association rules for spatiotemporally associated crimes from Compton (above) and West Hollywood (below) with their support

4.2 Indexing

Indexing is the heart of efficient spatiotemporal operations. We provide existing in-memory implementations of R-Tree[4] for the spatial indexing and Interval-Tree[3] for temporal indexing of the objects. Indices are created on all the spatial and temporal collections upon their creation unless otherwise specified.

5. PROBLEM SOLVING

As mentioned before, we have data about crimes and geographic boundaries of cities in the LA County. Given this dataset, we need to find the types of crimes that occur together. After placing crimes inside the given cities using a join, we find 7,289 cases in Compton and 2,839 cases in West Hollywood.

The next step is to find the co-occurring crimes. For this, we progressively search a specified number of meters around every crime to find crimes that occur within a specified number of hours after the previous crime. We then run the same function recursively on every crime selected by the above procedure until we no longer find any crimes nearby. We found 461 such series from Compton and 117 series from West Hollywood.

Once we find all such sets from our sample space, the next step is to find the types of crime that occur together most frequently in a set. For this, we use the apriori algorithm [5]. The basic idea of apriori is that any association can not be stronger than the strength of its weakest subset. We provide apriori algorithm as an extension to our tool. Table 1 shows the most commonly associated crime types in spatiotemporally nearby crimes. We have included the top 3 association rules with highest support for both cities. Support indicates the frequency with which these rules occur in the population.

Figure 6 shows code for solving this problem. Notice the very compact instructions for solving a complex problem.

6. EVALUATION

We do performance analysis of our tool against well known tools for solving the problem from Section 2 in terms of 1] the spatial join and 2] the recursive pattern discovery.

```
var apriori = #IMPORT APRIORI
#coffee#
searchNear = (point, meters, hours)->
  return [] if point.seen
  point.seen = 1
  crimes.containedBy(point.buffer(meters), 'sp', (x)->1)
  .filter((x)-> x.properties.sp)
  .within(point.temp_env(hours, 1), 'time', (x)->1)
  .filter((x)-> x.properties.time)

series = (point, meters, hours, _series)->
  nearby = searchNear point, meters, hours
  return _series if nearby.length == 0
  nearby.forEach (near)->
    series near, meters, hours, _series.push(near)

apriori.init (crimes.map (crime)-> series(crime, 100,
     48)), 15
outtext apriori.run()
```

Figure 6: Code for solving problem from section 2

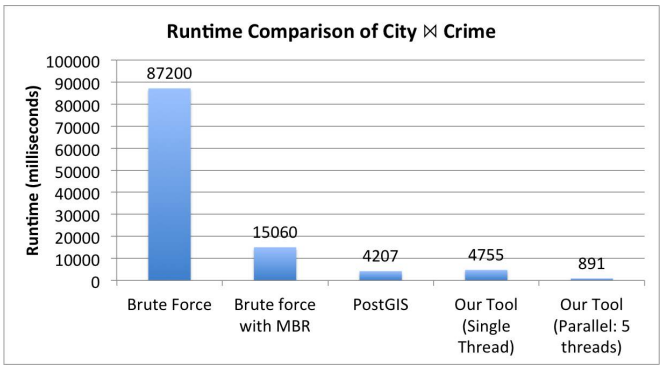

Figure 7: Performance analysis of various tools and methods of execution for spatial join between collection 'crime' and 'city'

For the purpose of evaluation, we take the spatial join between all cities and crimes. Collection 'city' has 250 polygons and collection 'crime' has 148,638 points. We compare the runtime for the join using a brute force approach (unindexed join), brute force but filtered by minimum bounding rectangle (MBR) approach, indexed join with PostGIS, single threaded indexed join with our tool, and finally parallel indexed join with our tool. Figure 7 shows the run times for the join. The parallel index join clearly outperforms everything else.

In this section, we compare the performance of our tool with the R language for running the algorithm for co-occurrence. Note that this is fairly complex since it potentially issues n^2 **contains** operations (**over** operations in R) during its runtime. Our tool outperforms R for both cities (Figure 6) mainly because of the indexing, which speeds up the large number of contains operations and brings down the runtime significantly.

7. RELATED WORK AND CONCLUSION

Similar tools include GIS systems, spatial databases and spatial extensions of some other analysis tools. Our tool is not designed to do everything that these tools can do but to outperform these tools in terms of ease, flexibility and efficiency in ad-hoc spatiotemporal analysis.

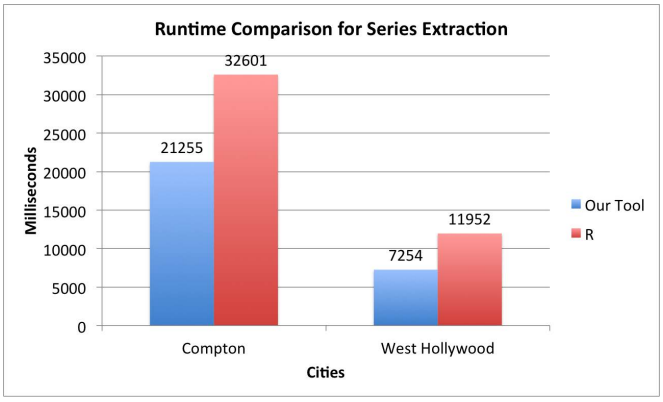

Figure 8: Performance analysis of our tool against R in running the algorithm to extract a series of spatiotemporally nearby crimes

We use the visual and interactive approach of GIS while giving users added flexibility through a programming model like spatial R[3] does. We also optimize our tool to perform efficient joins and aggregations like spatial databases. Unlike our tool, most spatial tools do not have an out of the box support for temporal data. There are other tools that attempt the map-reduce model for spatial operations such as Hadoop GIS[1]. However, these are designed for a much higher scale of data and are not particularly interactive.

We present a novel tool that helps make quick interactive spatiotemporal queries on the data in many different formats. It is a great tool to perform quick interactive analysis for pattern discovery. It provides a powerful operation set, wide range of visualizations, and fast processing with parallel execution. It outperforms other well known tools in solving fairly complex and ad-hoc problems with limited amount of data.

In the future, we would like to scale up to multiple machines. We would also like to explore in-memory caching systems and add support for raster data.

References

[1] Ablimit Aji et al. "Hadoop GIS: a high performance spatial data warehousing system over mapreduce". In: *Proceedings of the VLDB Endowment* 6.11 (2013), pp. 1009–1020.

[2] Rachel Boba. *Introductory Guide to Crime Analysis and Mapping*. United States: COPS, 2001.

[3] TH Cormen et al. *Introduction to Algorithms MIT Press*. 2003.

[4] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*. Vol. 14. 2. ACM, 1984.

[5] Yanbin Ye and Chia-Chu Chiang. "A Parallel Apriori Algorithm for Frequent Itemsets Mining". In: *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*. 2006, pp. 87–94. DOI: 10.1109/SERA.2006.6.

[3]http://cran.r-project.org/web/views/Spatial.html.