Stream Traffic Data Archival, Querying, and Analysis with TransDec

Final Report
METRANS Project 10-13

January 2011

Prof. Cyrus Shahabi

Computer Science Department
Viterbi School of Engineering
University of Southern California
Los Angeles, CA 90089

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the U.S. Department of Transportation, University Transportation Centers Program, and California Department of Transportation in the interest of information exchange. The U.S. Government and California Department of Transportation assume no liability for the contents or use thereof. The contents do not necessarily reflect the official views or policies of the State of California or the Department of Transportation. This report does not constitute a standard, specification, or regulation.

ABSTRACT

The goal of research was to extend the traffic data analysis of the TransDec (short for Transportation Decision-Making) system, which was developed under METRANS 09-26 research grant. The TransDec system is a real-data driven system to support decision-making in transportation systems. With TransDec, so far we have addressed the challenges in visualization, querying and management of dynamic and large-scale spatiotemporal transportation data, in particular, traffic sensors data and moving assets data.† With this proposal, building on our experience in implementing TransDec, we extended our research and technology development efforts under three specific tasks. First, we developed new techniques to create a streaming data archival repository that supports continuous querying and analysis of the vast amount of California transit data from RIITS (Regional Integration of Intelligent Transportation Systems) generated in the form of data streams. Second, we extended the current data-tier of TransDec to a distributed design to enable more scalable and stable computing environment. Finally, to demonstrate the benefits of the archived traffic datasets, we presented a novel proof-of-concept application, namely time-dependent optimal sequenced route (TD-OSR) planner using congestion prediction. This application exploits a subset of the real-world RIITS datasets, and these days we evaluating the ways to make it available for public use.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

DISCLOSURE

This project was funded in part under this contract to California Department of Transportation.

ACKNOWLEDGEMENTS

We acknowledge the United States Department of Transportation, California Department of Transportation and METRANS for their interest and generous support to this research. We would also like to thank RIITS to provide us the traffic sensor data for this research.

1. Introduction

Real-time data-driven framework of the transportation systems (i.e., a system driven by real data collected from the field) enables development of *data-driven* Intelligent Transportation Systems (ITS) for realistic and effective decision-making, planning, and management of the transportation systems. A data-driven ITS system must be able to handle various data types such as automatic vehicle location data (AVL), traffic congestion information, traffic incidents reports, road construction notices, driver information, CCTV video streams and snapshots, etc. Considering the large size of the transportation data, variety of the data (different modalities and resolutions), and frequent changes of the data, the integration, visualization, querying and analysis of such data for large-scale real-time systems are intrinsically challenging *data management* tasks. Due to these challenges, most of the current ITS applications only support limited data monitoring and analysis capabilities. Moreover, transportation researchers and policy developers, who mostly focus on developing algorithms and policies to enhance the efficacy of the transportation systems, often are restricted to simulation test-beds driven by synthetic or simplified data, and/or expensive and time-consuming field surveys to analyze the transportation systems and evaluate the proposed solutions.

In this research, we present a data-driven framework, dubbed TransDec, which enables real-time visualization, monitoring, querying, and analysis of dynamic transportation systems. We build TransDec on top of a three-tier architecture (presentation tier, query-interface tier, and data tier) that allows users to create customized spatiotemporal queries through an interactive web-based map interface in support of decision-making. With this architecture, we particularly address the fundamental data management and visualization challenges in 1) effective handling of dynamic and large-scale transportation data, and 2) efficient processing of real-time and historical spatiotemporal queries on transportation networks. To evaluate TransDec, we utilize a rich set of real transportation data which we have obtain and archive from RIITS[1] (Regional Integration of Intelligent Transportation Systems). The RIITS dataset [1] is collected by various organizations based in Los Angeles County including Caltrans D7, MTA-Metro, LADOT, and CHP. This dataset includes both inventory and real-time data (with update rate as high as every 1 minute) for freeway and arterial congestion, bus location, events, and CCTV snapshots. Moreover, in order to support diverse ITS applications, the TransDec data tier contains the transportation network of the entire US, as well as a wide variety of points-of-interest data provided by Navteq [2].

In addition to offering realistic and immersive virtualization of the traffic information and moving assets (e.g., busses, trains) through a web-based application, TransDec allows decision makers to issue various real-time and historical spatiotemporal queries about a) traffic at specific segments or sensor stations (with any user-defined level of aggregation at any desired timeframe), and b) moving assets and their location-based information.

[1] http://www.riits.net/

[2] http://www.navteq.com/

The successful implementation of the TransDec infrastructure in the previous stages of the project has facilitated the infrastructure and knowledgebase for two fundamental research lines. The first aims at devising an algorithm for compact and efficient data representation. Compact suggests that the data stored requires as little storage space as possible. The compactness of the data becomes a critical issue as the amount of data stored increases. Efficient representation means that, query times of the data are minimal and allow to work with the system in an interactive fashion. Then, exploiting the results of these lines of research, a new paradigm is presented. In this new storage paradigm the single point of storage, thus the single database server is traded for a cloud computing. This has many advantages, both in terms of storage scalability and maintenance and in terms of the availability of the data to all users as soon as it stored. We expect this new paradigm to dominate the research in geo-spatiotemporal databases in the near future and believe that the seeds we present within this research will play a significant role in it.

The remainder of this report is organized as follows. The two following sections, Sections 2 and 3, review the related transportation monitoring and simulation systems and describe the system's architecture. This facilitates the needed terminology and concepts that Section 4, which elaborate on the novel data representation methods, consists of. In Section 5, we introduce geospatial queries on a cloud decentralized storage system. Finally, we will conclude the report with conclusion and future work.

2. Related Systems

The latest developments in wireless technologies as well as the widespread usage of sensors have led to the recent prevalence of transportation network monitoring and simulation systems. The main functionality of such systems, in general, is to collect and archive data from distributed sensors and analyze the archived data for transportation applications such as planning and mobility measurement. Among the numerous transportation network monitoring and simulation systems that have been developed in the last decade, we review two of the most relevant systems, namely PeMS [2] and ADMS [3].

The freeway Performance Evaluation Monitoring System (PeMS) developed by UC Berkeley collects and stores data from loop detectors operated by Caltrans. The main goal of PeMS is to convert this freeway sensor data into intuitive tables and graphs that show traffic patterns on highways. Moreover, PeMS is used to spot bottlenecks, measure the efficiency of highways, and estimate travel times for highway segments based on the historical travel time data.

Similarly, ADMS developed by Smart Travel Lab in University of Virginia is a system which utilizes archived ITS data to provide information services to measure the performance and operation of Virginia transportation systems. ADMS, in addition to monitoring highways, enables users to query the database for real-time (and historical) weather and incident information for specific routes and segments.

Our work is fundamentally different from the aforementioned systems in several ways. First, while the scope of these systems is limited to collection, archival and analysis of the sensor data, with TransDec we fuse the sensor data with various spatiotemporal data (e.g., transportation network data, moving object data, point of interest data) to be able to support end-to-end ITS applications more realistically. Second,

PeMS and ADMS, for analysis of huge datasets, use traditional Database Management System (DBMS) specific analytical query processing tools. DBMSs can only support a set of predefined analytical queries. However, with TransDec we utilize more sophisticated query processing techniques that allow for ad hoc and complex analytical queries as well. Finally, in addition to real-time traffic monitoring and analysis, TransDec enables various other real ITS applications such as real-time moving asset tracking, route planning and location-based querying.

3. TRANSDEC Architecture

TransDec adopts a three-tier architecture where presentation, query-interface, and data management tiers are logically separated. With the three-tier architecture of TransDec, the query is initialized at the presentation tier interactively and sent to the query-interface tier where each request is formulated as structured query language (SQL) before interacting with the data tier. Apart from the usual advantages of modular system with well defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently as requirements or technology change. Figure 1 shows the three-tier architecture of TransDec. Below we elaborate on each tier in detail and describe the interaction between them.

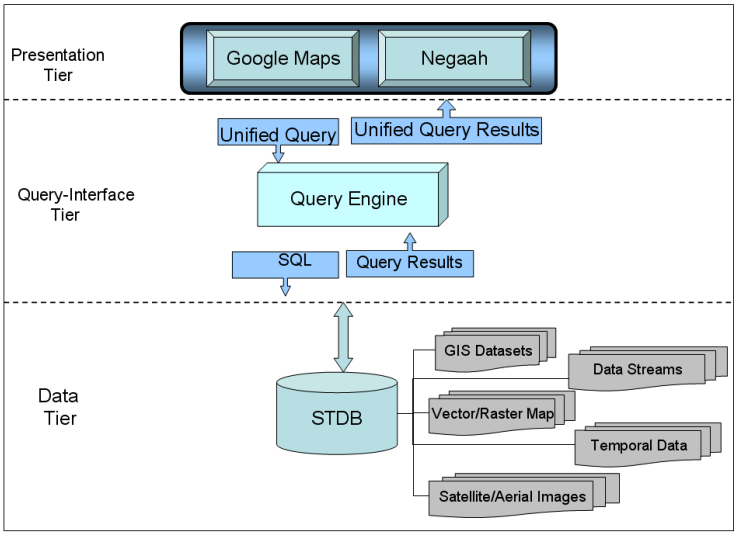


Figure 1. The TransDec architecture

3.1. Presentation Tier

One of the key distinguishing features of TransDec is the provision of an immersive environment that enables users to interact with the system and perform a wide range of ITS related spatiotemporal queries intuitively. The current graphical user interface (GUI) provides users with query flexibility by allowing them to query the spatiotemporal datasets based on a user-defined area and time interval. Specifically, users can selectively query and display different layers of information on desired regions, and move forward or backward in time for various query types. To implement the presentation tier, we have

integrated a new generation *web-based map application*, Google Map™, into TransDec as graphical user interface. As a second choice, we have also developed our proprietary interface (termed *Negaah*) that provides custom spatiotemporal queries not supported by typical web-based mapping applications.

The advantages of utilizing Google Map™ at the presentation tier are as follows: a) appealing and highly responsive visualization options, b) ability to zoom in and out to the desired scale and to pan to specific locations, c) flexible environment for as many layers of information as necessary to provide the level of detail required, and d) easy access through Internet via any browser (without any software or hardware installation). Additionally, given that Google Map™ is already ported to PDAs, porting TransDec to mobile computing devices is not a far-fetched future work. To support querying capability on mobile computing devices, we have a head start by studying multi-resolution vector data compression techniques [4] that effectively compresses the result of query windows, taking into account the client's display resolution.

3.2. Query-Interface Tier

With the query-interface tier, TransDec allows several independently developed GUIs to interact with our spatiotemporal data tier transparently. Our query-interface tier offers a universal standard for specifying the type of query (e.g., shortest path, range aggregate, etc.) and its parameters, as well as the returned results. Specifically, depending on the query type received from the GUIs, the query-interface tier constructs and sends the query to our data tier through low-level structured query language (SQL) commands for evaluation. In most cases, the query results are formulated into an industry standard XML[3] file before sending to presentation tier.

In addition, this tier includes a data fusion engine which continuously acquires sensor readings from RIITS web-services and stores them into our data repository. Data fusion engine utilizes WSDL[4] (provided by RIITS) in combination with SOAP[5] to retrieve the dynamic sensor data through a secure communication channel over the Internet.

3.3. Data Tier

The data repository of TransDec is a spatiotemporal database management system (STDBMS), namely Oracle 10g. STDBMS stores a variety of both dynamic (frequently changing) and static datasets such as highway sensor data, road network information (i.e., vector data), moving object trajectory data, point of interest data (e.g., hospitals, restaurants), terrain data, satellite and aerial imagery, and raster maps. Most of the data stored in the repository are labeled by both space and time to allow for a wide range of spatiotemporal (both time and space based) queries.

---

[3] http://www.w3.org/XML/

[4] http://www.w3.org/TR/wsdl

[5] http://www.w3.org/TR/soap/

In the data tier, our main focus is on developing index structures to answer spatiotemporal queries efficiently. Index structures are pre-computed data structures that are built off-line to speed up the evaluation of the queries issued on-line. Towards this end, in addition to incorporating numerous off-the-shelf indexing methods (e.g., R-Tree), we have introduced various versions of Voronoi Diagrams (VD) [5] as index structures for enabling efficient spatiotemporal querying on different types of data hosted in TransDec. The major data components of TransDec are as follows:

3.3.1. Sensor Data

TransDec, through RIITS, acquires traffic sensor data from approximately 6500 sensors (covering approximately 2000 miles) located on highways and arterial streets at the boundaries of Los Angeles County. The arrival rate of the data from each sensor is 1 reading/sensor/min. The storage space required for this streamed dataset is approximately 850 MB/day.

Each sensor is a separate data source that generates records with several attributes including sensor id, measure (value of reading), space (coordinates), and time (timestamp). The traffic measures captured by the sensors are speed, HOV speed, volume (number of vehicles) and occupancy (ratio of time vehicle is detected). Based on their spatial placement, we group the sensors as follows: a) individual sensors, b) spatially continuous sensors (segments), and c) spatially continuous segments (sections). Also, we consider the following time intervals to interact with the sensor data: a) a time point/range within a day, and b) day of week. These hierarchical categorizations with both space and time enables TransDec to process all roll-up and drill-down queries on traffic measures efficiently by navigating between various hierarchies of the time and space dimensions. For example, when computing the average speed at a network segment within five past Mondays from 08:00am to 08:30am, the weekend traffic data and unrelated segments can be discarded easily.

3.3.2. Transportation Network Data

TransDec contains a wide range of transportation network data (provided by Navteq ™) including highways, major and secondary roads, streets, railroads, bridges, etc., for the entire US. Each network segment is represented in the vector data format and described by more than 20 attributes such as direction, speed limit, zip code, paved, etc. We utilize transportation network dataset primarily when processing route planning and location-based queries.

3.3.3. Trajectory Data

Another spatiotemporal dataset of TransDec consists of the trajectory information collected from moving objects whose location in the space changes over time. The position of a moving object is sampled at discrete times, and a series of straight lines connecting successive positions represents the trajectory of the object. Currently, TransDec collects live location data from GPS equipped USC trams and student cell phones (GeoSim, 2008). All time variant vehicle and cell phone coordinates are stored and archived in the data tier.

3.3.4. Points-of-Interest (POI) Data

Point of interest data stored in TransDec (also provided by Navteq^{TM}) includes list of public locations, such as bus satiations, parking structures, hospitals, restaurants, etc. Each POI data record is represented as spatial point location in the database and includes various attributes such as phone number, street address, type, etc.

4. Spatiotemporal Summarization of Traffic Data Streams

Efficient summarization and accurate reconstruction of the historic traffic sensory data, allows for smaller and efficiently manageable transportation data storage systems. Existing data summarization (and archival) techniques are generic and are not designed to leverage the unique characteristics of the traffic data for effective data reduction. In this section a family of data summaries that take advantage of the high temporal and spatial redundancy/correlation among sensor readings is introduced.

In the past, different types of data reduction techniques have been widely used to reduce the size of the large sensor datasets. The prominent data reduction techniques are Wavelets [6], Single Value Decompositions (SVD) [7] and Principal Component Analysis (PCA) [8]. The main idea behind these techniques is to compactly store the main patterns in the data (i.e., sketches) in such a way that the dataset can be reconstructed back in its entirety from those patterns, with minimal loss of accuracy. Wavelets - a widely used technique in signal processing and image compression - compress large datasets by hierarchically decomposing the raw data and storing a small number of wavelet basis functions (i.e., wavelet coefficients) which best describe the data. Wavelets have been applied successfully in answering range-sum aggregate queries over data cubes [6], in selectivity estimation [9] and in approximate query processing [10,11,12]. Likewise, SVD and PCA represent a multivariate dataset using the smallest possible number of new variables (i.e., principal components) that are selected based on the statistical characteristics of the dataset. PCA reduce the size of the datasets by maintaining a sketch of archived historical data (i.e., small number of principal components and a transformed dataset). However, the main difference is that although these compression techniques enable approximate query processing on the set of sketches, most of them do not guarantee any error bounds on the query results. Specifically, depending on the spatial and temporal extent of the query, the variation between the actual result and the approximated result can be unacceptable. In contrast, our approach ensure both an error bound and probabilistic guarantee on the results to the spatiotemporal queries.

Another line of related work is data stream processing. In many streaming techniques, the structures similar with signatures (e.g., synopsis) are built online for real-time approximate query purposes, examples include equi-depth histograms and Haar wavelets [9,13], maintaining samples and simple statistics over sliding windows [14], data clustering and decision tree construction [15,16]. But most these research efforts focus on the application of online data monitoring rather than queries over historical dataset. Similarly, in some large streaming projects, queries over historical data streams do not receive much attention. In the area of sensor networks, such systems includes Aurora [17] which is designed for the purpose of managing data streams for monitoring applications

and Telegraph [18] from UC Berkeley which focuses on creating adaptive engine over querying streaming data from sensors. However, for other streaming projects, such as Coguar [19] from Cornell University which considers sensor network as a distributed database system, STREAM [20] which serves as a general-purpose data stream management system as well as Niagara [21] which is designed for internet XML query processing, do concern the historical queries, but the type of their queries do not include spatial and temporal filters as the spatiotemporal query defined in this paper.

For the data reduction in spatiotemporal domain, there have been several studies customized for specific types of spatiotemporal dataset. One of them is the work done by Cao et al. [22] on data reduction over trajectories of moving objects. They adopted line simplification approach from graphics field to reduce the storage size of trajectories. Unlike the traditional reduction technique, the line simplification based approach can guarantee a deterministic error bound, which is very similar with our error bounds structure. However, the approach of line-simplification aims at geographically simplify the presentation of individual moving objects trajectory, therefore heavily relies on the structure of trajectory data and not applicable on the traffic sensor data described in this paper.

4.1. Problem Definition

Our proposed approach builds on the observation that there is a strong correlation (both temporally and spatially) and redundancy present among the measurements of the single and multiple traffic sensor(s). For example, Figure 2(a) plots the average speed measurement from a single sensor located on I-10 East for two consecutive Mondays from 6 AM to 9 PM. As shown, both signals follow almost the same trend, and hence it is obvious that maintaining the two sets of measurements in their entirety is redundant. As another example, Figure (b) depicts a scatter plot of speed measurements (for Wednesday from 8 AM to 9 AM) from four different sensors, which are spatially close to each other on a segment of I-10 East. Similarly, there exists a strong correlation among the speed measurements of multiple sensors in spatial proximity. Given these observations, with our data summarization technique, we derive and maintain data signatures that represent typical patterns of the sensor readings that approximate the actual readings with bounded error. These data signatures, first enable us to store the streaming sensor data more efficiently by discarding the redundant sensor readings, and hence, provide cost effective data growth. Second, we can evaluate the spatiotemporal queries based on a small but informative summary of the sensor readings with sufficiently accurate results, rather than having to scan the entire datasets, which yields unacceptable response times. Specifically, with signature based approach, we only store the streaming data which falls outside of the signature (i.e., outlier) within a given error-bound; otherwise we discard the data since it is already represented by the signature. With our study, we use the large-scale traffic sensor dataset from the entire Los Angeles County highways for the past two years, collected by the TransDec system. Based on our experiments on this real dataset, we observe that our proposed approach can reduce the storage requirement up to 77% while maintaining high accuracy (with bounded-errors) on the query results.

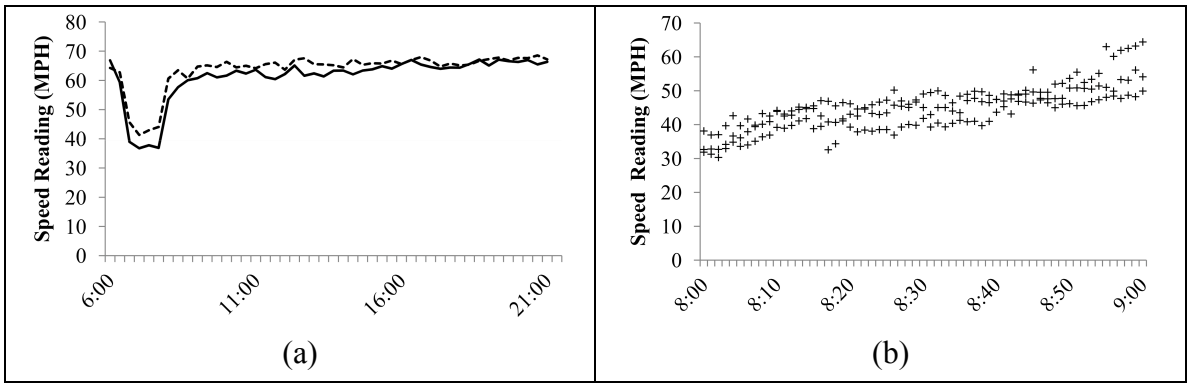


Figure 2 - Examples of sensor readings

In our study, we consider the readings collected from each sensor as a time-series with each reading observed by a sensor node at time *t*. Each sensor node is located on a road network segment. Each sensor reading contains multiple attributes (i.e., speed, volume and occupancy) describing the traffic behavior. In the rest of the discussion we will focus of the speed data. The speed value denotes the average speed during its sensor sampling time unit. The entire dataset, which contains all sensor speed readings collected in time interval $[T_s, T_e]$ is denoted by *D*. Our goal is to provide approximate results (with a bounded-error) to the queries that ask for the speed reading for a single sensor during time interval $[t_s, t_e]$\$, where $(t_s > T_s)$ and $(t_e < T_e)$. Note that other queries such as average query can be answered on top of this query defined here.

Since we are interested in answering the spatiotemporal queries within a specified error-bound, we define precision constraint which incorporates user specified precision parameters and enforces an approximate result to deviate from the exact result by (at most) ± error-bound ε with probability δ.

Definition: Precision Constraint

Let ε and δ denote relative error and probabilistic guarantee specified by users, respectively and let *A* and *Y* be the exact and approximate query result. The Precision Constraint states that the approximate result, *Y*, should hold a relative error of at most ε of the exact result *A* with probability of at least 1-δ:

$$P\left[\left|Y - A\right| \le \varepsilon A\right] \ge 1 - \delta \qquad \textbf{Equation 1}$$

4.2. Overview of Approach

One way of answering spatiotemporal queries approximately is to capture the underlying patterns from the sensor readings and use them instead of the exact readings. Towards this end, we create a concise but reasonably accurate pattern of a sensor or a group of sensors called *signature* by averaging the sensor readings. Hence, given a spatiotemporal historical query, we can use the signatures to represent the results rather than scanning

the entire historical data. However, when the signatures are not sufficient to represent the exact sensor readings within precision constraint, we store the sensor readings that violate the constraint as outlier. We consider both signatures and outliers as the data summaries to answer the spatiotemporal queries.

To further improve the storage efficiency, we explore temporal and spatial correlation in constructing data summaries based on the following observations. Like in most environmental monitoring sensor network deployments (e.g., pollution, temperature), the data generated by the traffic sensor nodes is highly auto correlated both in time and space. For example, the weekday readings from a specific traffic sensor usually follow a similar pattern. Similarly, the readings from multiple sensors located within a spatial proximity may also be strongly correlated, since the traffic flow hardly diverges or accumulates within a small spatial region, especially for the road segments with no exits/entries. These correlations can be captured accurately by constructing different types of data summaries from the historical data traces.

We now explain the construction and maintenance steps of the data summaries shown in Figure 3. Our approach involves three phases. At the data analysis and query-processing phase, we use the historical data to pre-compute the signatures and their corresponding outliers to support historical spatiotemporal queries submitted by users. At the data collection phase, we compare the incoming sensor readings to corresponding signatures to identify whether they violate the precision constraint. If precision constraint is violated, to avoid storing all such sensor readings, we conduct sampling among them with rate (1-δ) and only store the samples, otherwise, we discard the reading because we can use its signature value to represent it in the query processing.

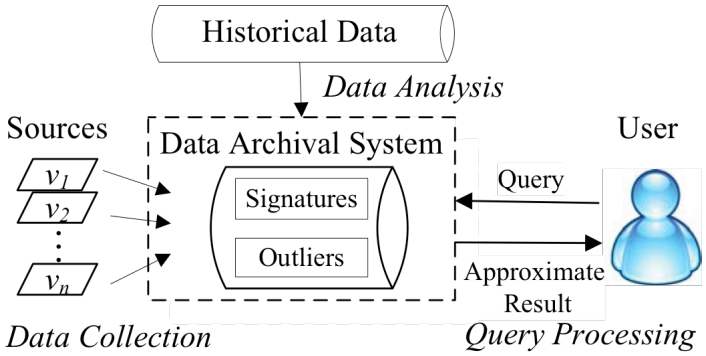


Figure 3 – Block Diagram f the System

The summarization process' result is presented in Figure 4, the solid line is the sensor's signature and the dash lines indicate the error bounds of a sample signature. The crosses represent outliers that are outside of the error-bounds. Note that readings identified as outliers, are stored with probability of (1-δ).

Accordingly, given a query, we not only utilize the signatures to provide approximate answers, but also incorporate the outliers when the signatures are not sufficient in satisfying the precision constraints. We argue that the combination of signatures and outliers can satisfy the precision constraints. The justification of our argument is as follows. For the query results (or part of it) from the outliers, they are 100 percent

accurate with no error, because we store the exact sensor reading as outlier. If the results are from signatures, based on the way we sample the outliers, there is (1-δ) probability that the exact result is within the error range ε. Hence, the combination of the results from signatures and outliers can guarantee the precision constraint.

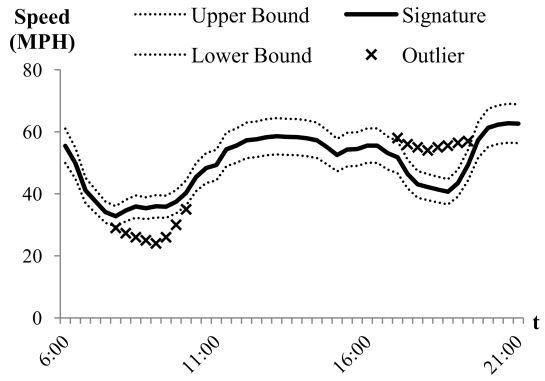


Figure 4 - Example of data summary

4.3. Summarization Methods

So far, we have formally defined the two components of data summaries: signatures and outliers. In this section, we explain our basic summarization technique. In this technique, we compute the daily signature of a sensor by averaging all historical sensor readings from that sensor. We repeat this process for all sensors. For each sensor, comparing the sensor reading to the corresponding value in its signature S identifies the outliers. Then, we examine $v_j(t)$ and S_j in the context of outlier definition to determine whether the reading is an outlier.

Sensors' signatures required storage is negligible with respect to the storage required for saving the entire data. However, if a signature is not representative enough (i.e., does not capture the typical patterns of its corresponding sensor), the storage needed to maintain the outliers can be high. We address this problem by maintaining several signatures for one sensor at different temporal and spatial scales. Meanwhile, we also aim to strike a compromise between the storage of signatures and outliers, and minimize the overall storage requirement of data summaries. Towards these ends, we propose two different data summarization techniques that exploit temporal and spatial correlations of the sensors.

4.3.1. Temporal Summary

In real-world road networks, the traffic patterns may show variations among different days within a week or even different seasons. We can accommodate for such diversities by using more than one signatures corresponding to different temporal scales. Trivially, increasing the number of signatures reduces the amount of storage needed by outliers.

We explain our temporal summary technique using the example in Figure 5 where we focus on three levels of temporal summaries. The leftmost level indicates the method using single signature for each sensor as discussed in the basic summarization. At the second level, we increase the granularity of temporal summaries by providing seven

signatures with each one representing a unique day in the week. Each signature at this level is computed by averaging all the sensor readings collected on the corresponding day. For example, Wednesday's signature of a sensor is the average of its sensor readings collected on Wednesdays in the historical dataset. At the third level, we increase the temporal granularity by introducing seasonal information. Based on each signature generated in the previous level, we derive three signatures with each one representing the sensor readings within a particular season. In this level, we create a total of 21 signatures for each sensor characterizing the sensor readings at different temporal scales.

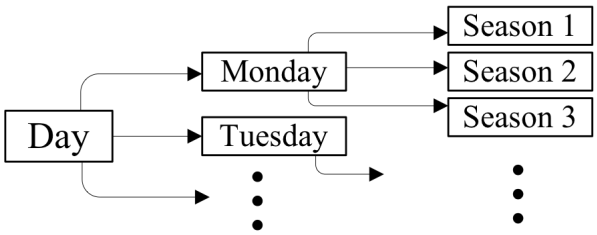


Figure 5 - Different Levels of Temporal Summaries

4.3.2. Spatial Summary

We exploit the fact that the traffic sensors co-located within a spatial proximity report similar readings and behaviors. Therefore, the sensor readings between two adjacent entries/exits, or between two adjacent intersections may show similar values. Therefore, instead of maintaining one signature per sensor, we can compute one common signature for a group of sensors. Subsequently, with spatial summarization, we aim at eliminating the redundant signatures.

We define two types of segments for spatial summaries that include groups of sensors, which have similar patterns: the segment between two adjacent exits/entries *E-Segment* and the segment between two adjacent intersections *I-Segment*. As illustrated in Figure 6 in a typical road network, each E-Segment includes a small number of sensors, and each I-Segment includes several E-Segments, corresponding to a larger number of sensors. To compute the signatures for each segment, we first identify the set of sensors located in that. Next, we calculate the average of the sensor readings from all sensors in this group and use it as the common signature for each individual sensor located in that group.

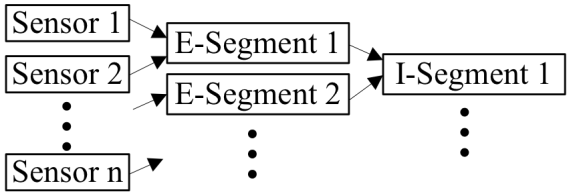


Figure 6 - Different Aggregation Level of Spatial Summaries

4.4. Queries

In this section, we introduce our proposed approach to answer spatiotemporal queries based on each type of data summaries. Given a query, asking for sensor readings during

a particular time interval $[t_s,t_e]$ for a particular sensor *i*, we perform the following four steps to generate the answer:

1. Partitioning the time interval $[t_s,t_e]$ into individual sensor sampling intervals (i.e., $[t_j,t_j+T]$), and initialize the empty result array to carry the result value for each sampling interval.
2. For each interval $[t_j,t_j+T]$, we extract date and time information, denoted as *d′* and *t′*. We use sensor_id *i*, *d′* and *t′* to search the database to check if any corresponding outlier stored. If we find such an outlier, we insert its value to the result array for interval $[t_j,t_j+T]$ and skip the third step, otherwise, we continue with the third step.
3. The sensor id, *i*, is employed to search for its corresponding signature *S*. Then, we utilize *t′* to find the corresponding position (*k*) of *S* representing the interval $[t_j,t_j+T]$ and insert S_k to the result.
4. Finally, once we have gone through all the individual sensor sampling intervals in $[t_s,t_e]$, we return the result array as the approximate answer to the query.

For the queries based on temporal or spatial summaries, we use the similar framework with a few changes in the third step. For temporal summaries, besides sensor id, *i*, we add date information *d′* from t_j to search for the corresponding signature *S*. Since we maintain several signatures instead of one per sensor, we need to identify the particular one for t_j by using *d′*. For spatial summaries, before searching for the signature, we identify the group ID of sensor *i*. Next, instead of sensor ID, we use group ID to find the group signature as the signature for sensor *i* in the third step.

4.5. Experiments

In our experiments, we use a large-scale and high resolution (both spatially and temporally) traffic sensor (i.e., loop detector) dataset collected from entire Los Angeles County highways. This dataset includes both inventory and real-time data for around 1800 traffic sensors covering approximately 3000 miles. The sampling rate of the streaming data is 1 reading/sensor/min. To evaluate the storage efficiency, we compare our summarization techniques with a baseline solution, which stores entire historical sensor readings. In the first two sets of experiments, we vary two precision constraint parameters: error range ε and probabilistic guarantee rate δ by comparing the storage requirement of the baseline approach, with our techniques using different summary strategies (i.e., temporal and spatial summaries). When varying one parameter, we set the value of the other one to 10%. In the third set of experiments, we fix the precision constraint (i.e., both ε and δ are set to 10 %), and compare the different combinations of our two summarization techniques in storage efficiency and signature size. The performance is measured as the storage requirement of each technique. For all the experiments, we use a PC running Windows with Intel 6420 Dual CPU 2.13G and 3.0 GB RAM.

4.5.1. Temporal Summaries

First, we compare the baseline approach (B), the basic summarization (BS) technique and two temporal summarization techniques based on two temporal scales: Weekday (W), Seasonally Weekday (SW). For SW, we define three seasons: spring, summer, fall with each one covering four months of the year, Jan-Apr, May-Aug, Sep-Dec, respectively.

Figure 7 shows that as ε and δ increase the overall storage requirement of our summarization techniques based on different temporal correlations decrease sharply as compared to the baseline approach. For the effect of ε and δ, we observe that as ε increase, we hold an increasing reduction rate on storage requirement. Specifically, when ε increases to 10, the storage requirement of our system is reduced by nearly 75% as compared to the baseline approach. As ε increase to 20, the reduction reaches 80% to 85% percent. This indicates that about 75% to 85% of the entire sensor readings are distributed within a small error range of the signatures. For δ, the storage size decreases linearly as δ increases. The reason is that we sample the outliers to store, so the storage size is proportional to the sampling rate. In general, the higher the temporal scales, the more signatures are stored, hence resulting in less number of outliers as shown in Figure Figure 7. From approach BS to W, the decrease in storage requirement is noticeable, but from W to SW, the two lines nearly overlap with each other, which indicates the amount of storage saving is negligible. One reason is that the traffic sensor readings hardly change across different seasons in Los Angeles. Figure 8 shows the size of signatures for SW is two times higher than that of W. In conclusion, to make a trade-off between the number of signatures and the storage efficiency, the temporal summary by weekday signatures is the proper temporal summarization approach to choose in Los Angeles.

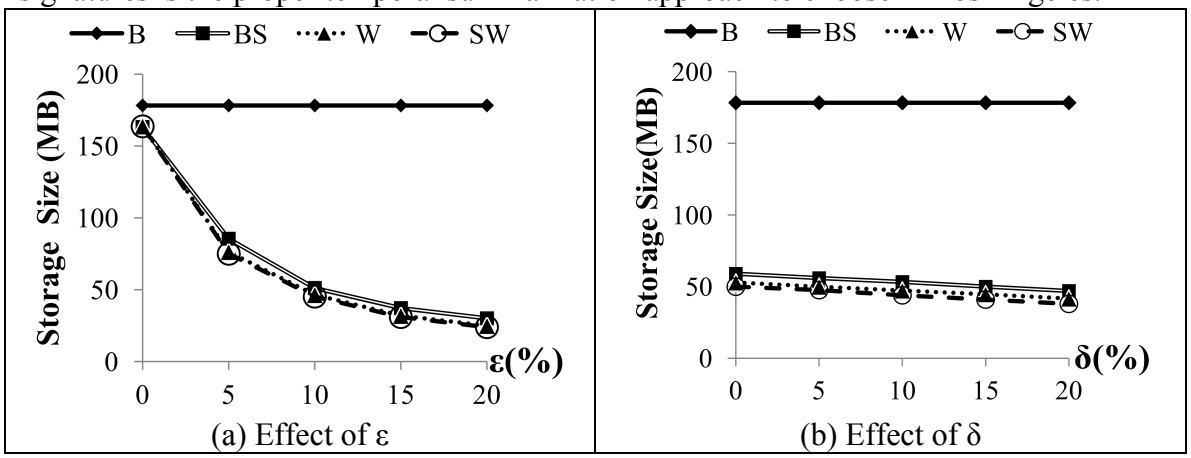


Figure 7 - The overall storage size for temporal summaries

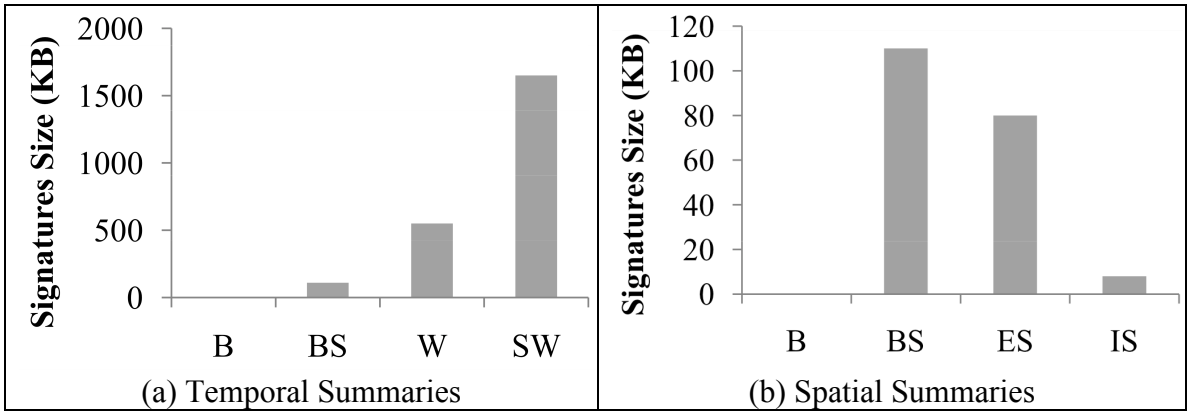


Figure 8 - The signatures size for different summaries

4.5.2. Performance of Spatial Summaries

We compare here the baseline approach (B) and basic summarization (BS) with two spatial summaries designed for sensors within an E-Segment (ES), and within an I-Segment (IS).

As shown in Figure Figure 9, the impacts of ε and δ are similar to those previous experiments. When we increase the spatial aggregation level by grouping sensors according to different size of segments (in our dataset, I-Segments are longer than E-Segments), the overall representation capability of each group signature for individual sensor is reduced; therefore the number of outliers increases. In particular, comparing IS with BS in both Figure 8 and Figure 7(b), although the signature size of IS is reduced significantly as compared that of BS, IS shows a sharp increase of outliers. One possible reason for that is the sensor readings generally fluctuate a lot within two adjacent intersections. We observe that both BS and ES require similar storage capacity in most cases, which means that sensors located between two exits mostly maintain similar speed readings hence the amount of outliers does not increase significantly. Moreover, the signature size of ES is smaller than that of BS. Hence, the ES based spatial summarization technique was the best one in this set of experiments.

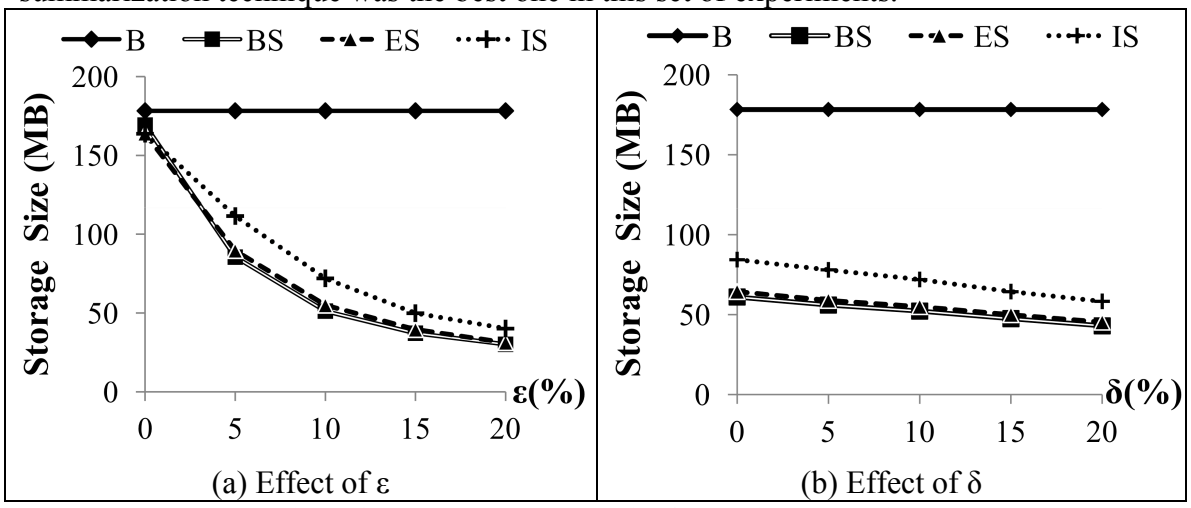


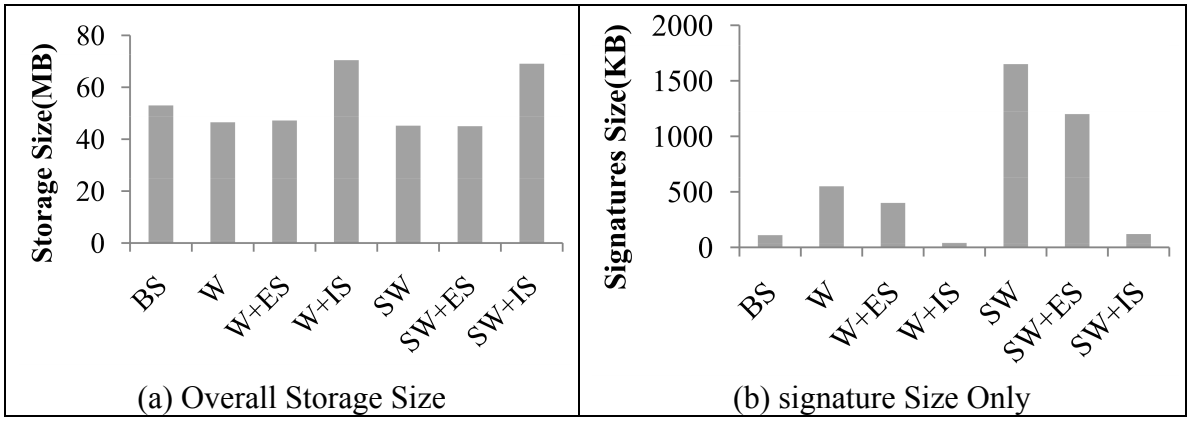
Figure 9 - The overall storage size for spatial summaries

4.5.3. Performance of Spatial & Temporal Summaries

With our previous two experiments, we fixed one type of summarization technique at one time to examine the effect of others. In our third set of experiments, we vary both the spatial and temporal summarization technique simultaneously to identify the optimal combination for our system. With this set of experiments, the same notation as in the last two experiments is used, (e.g., W+ES indicates the combination of week day summarization and E-Segment summarization). We fix both ε and δ to 10%.

Figure 10 shows the comparison of the seven combinations for overall storage size and signature size. As shown, although IS methods is useful in decreasing the amount of signatures significantly, it sacrifices the overall storage size because of the increasing number of outliers, so it cannot be considered as a part of the optimal choices. When comparing the performance of techniques including W and the ones including SW, the storage size does not change much, but the signature sizes of the ones with SW are much

larger than that of the ones with W. Therefore, we should also exclude SW technique from our optimal choices. Now, let us compare the remaining three choices: BS, W and W+ES. As shown, W and W+ES requires similar storage requirement that is much less than that of BS. Hence, we eliminate BS. But W+ES performs better by maintaining less signatures as compared to W. Hence, W+ES is the optimal solution for our system. By comparing the overall storage size of W+ES with that of the baseline approach shown in previous experiments, we observe that the storage requirement of our system as decreased by 77% percent of the baseline approach with ε=10% and δ=10%.



(a) Overall Storage Size (b) signature Size Only

Figure 10 - The performance of temporal & spatial summaries

5. Distributed data management algorithms

5.1. Introduction

With the recent advances in location-based services, the amount of geospatial data is rapidly growing. Geospatial queries (nearest neighbor queries and reverse nearest neighbor queries) are computationally complex problems which are time consuming to solve, especially with large datasets. On the other hand, we observe that a large variety of geospatial queries are intrinsically *parallelizable*.

Cloud computing implies a considerable reduction in operational expenses by providing *flexible* resources that can instantaneously scale up and down. The annual global market for cloud computing is estimated to surpass $100 billion in three years [23]. Thus, the big IT vendors including Google, Microsoft, IBM and Amazon are ramping up parallel programming infrastructures. Google's MapReduce programming model [24] provides a parallelization for processing large datasets and can do at a very large scale. For example, Google processes 20 petabytes of data per day with MapReduce [25]. Given recent availability of cloud services and intrinsic parallel nature of geospatial queries, a variety of geospatial queries can be efficiently modeled using MapReduce. Suppose we want to answer a reverse nearest neighbor query (RNN). Given a query point q and a set of data points P, RNN query retrieves all the data points that have q as their nearest neighbors [26]. Each point p_i finds its nearest neighbor p_k efficiently in parallel and emits $<p_k, p_i>$ in the map phase. Subsequently, all points having the same key, p_k will be grouped in the reduce phase by indetifying p_k's RNNs without any extra step.

Parallel spatial query processing has been studied in the contexts of parallel databases, cluster systems, and peer to peer systems as well as cloud platforms. We propose a MapReduce-based approach that both constructs a *flat* spatial index, Voronoi diagram (VD), and enables efficient processing of wide range of spatial queries including reverse nearest neighbor (RNN), maximum reverse nearest neighbor (MaxRNN) and k nearest neighbor (kNN) queries. These types of queries are widely used in decision support systems, profile-based marketing, bioinformatics and geographical information systems. For example, RNN query can help a franchise (e.g. Starbucks) decide where to put a new store in order to maximize the benefit to its customers. The experiments show that our approach scales well with the increasing number of nodes. For example, a VD can be generated by merging partial Voronoi Diagrams (PVD). Specifically, each mapper creates a PVD and a reducer combines all the PVDs to obtain a single VD. Clearly, dividing the most computationally complex piece of the algorithm, namely construction of PVDs, across multiple mappers improves the performance. To the best of our knowledge, our work is the first detailed attempt in processing geospatial queries with MapReduce in the cloud-computing context. This work was presented in the last IEEE conference on cloud-computing and was awarded with the best paper award [27]. Before we present our geospatial query processing approach, in this section we briefly introduce Voronoi Diagrams and MapReduce to prepare for the rest of the discussion.

5.1.1. Voronoi diagrams

A Voronoi diagram decomposes a space into disjoint polygons based on the set of generators (i.e., data points). Given a set of generators S in the Euclidean space, Voronoi diagram associates all locations in the plane to their closest generator. Each generator s has a Voronoi polygon consisting of all points closer to s than to any other site. Hence, the nearest neighbor of any query point inside a Voronoi polygon is the generator of that polygon. The set of Voronoi polygons associated with all the generators is called the Voronoi diagram (VD) with respect to the generators set. The polygons are mutually exclusive except for their boundaries.

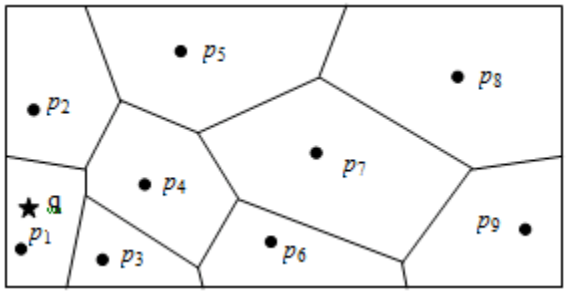


Figure 11 - A sample Voronoi diagram

Figure 11 illustrates an example of a Voronoi diagram and its polygons for nine generators. Various approaches have been proposed to generate *VD* in Euclidean space. We use *divide and conquer* approach to efficiently compute *VD* [28].

5.1.2. MapReduce

MapReduce is a popular programming model for parallel processing of large datasets. Programs written with this model are automatically parallelized and executed on thousands of commodity machines, collectively referred to as a cluster. Since it is a simple but yet a powerful model, it has been used in a wide variety of application domains such as machine learning, data mining and search-engines.

Hadoop is a popular open source implementation of the MapReduce, which runs on top of a distributed storage system called Hadoop File System (HDFS). In HDFS, a file is broken into multiple blocks, called *split*, which are then distributed among the machines in the cluster. All splits are of the same size with the exception of the last one, and the split size is configurable per file. MapReduce consists of two user-defined functions, namely a map function and a reduce function. Given a MapReduce task, with Hadoop first each mapper is assigned to one or more splits depending on the number of machines in the cluster. Second, each mapper reads inputs provided as a <key, value> pair at a time, applies the map function to it and generates intermediate <*key,value*> pairs as the output. Finally, reducers fetch the output of the mappers and merge all of the intermediate values that share the same intermediate key, process them together and generate the final output. The input of mappers and the output of reducers are stored on HDFS. Figure 12 shows how Hadoop processes the data of four split in parallel where there are three map machines and two reduce machines.

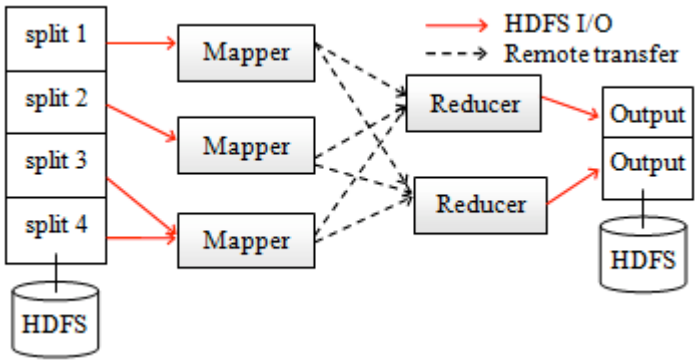


Figure 12 - The data flow in Hadoop Architecture

5.2. Constructing Voronoi Diagram with MapReduce

Voronoi Diagram construction is inherently suitable for MapReduce modeling because a *VD* can be obtained by merging multiple partial Voronoi diagrams (*PVD*). Specifically, each of *PVD*s can be created by the mappers in parallel and the reducer can combine them into a single *VD*. In this section, we show how a *VD* is built with MapReduce using *divide and conquer* method.

One of the techniques used to generate a Voronoi diagram is *divide and conquer* paradigm. The main idea behind the divide and conquer method is as follows. Given a set of data points *P* as an input in Euclidean space, first the points are sorted in increasing order by X coordinate. Next *P* is separated into several subsets of equal size. Subsequently, a *PVD* is generated for the points of each subset, and then all of the *PVD*s are merged to obtain the final *VD* for *P*.

MapReduce Solution: Given a set of data points sorted by X coordinate, each mapper reads an input split in the format of <data_point, dummy_value> (i.e., <key, value> pair). Note that the dummy_value does not have any purpose, i.e., it is used to follow the input format of MapReduce. Subsequently, each mapper generates a *PVD* for the data points in its split, marks the boundary polygons to be later used in the merge phase and emits the generated *PVD*s in the form of <constant_key, PVD_i> where *i* denotes the split number. The constant_key is common to all *PVD*s, so that all *PVD*s can be grouped together and merged in the subsequent reduce step.

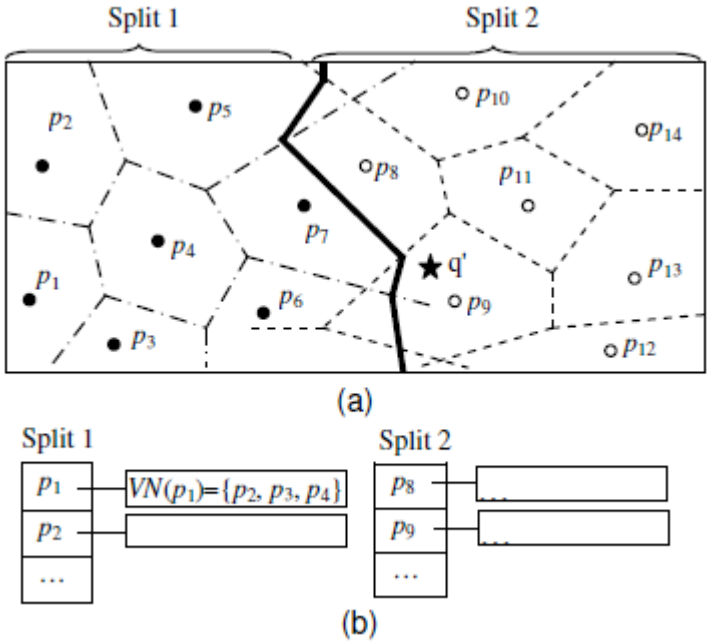


Figure 13- a) Merger of left and right PVDs b) A Voronoi diagram distributed on HDFS which will read by the mappers as input

Finally, a single reducer aggregates all *PVD*s in the same group and combines them into a single *VD*. In the merger phase, the boundary cells are detected first with a sequential scan, and then new Voronoi edges and vertices are generated by deleting superfluous boundary portions from *PVD*s. As the final output, the reducer emits each point and its Voronoi neighbors (*VN*).

Figure 13a illustrates parallel *VD* generation with two mappers and one reducer. Given a set of data points P= {$p_1, p_2, ..., p_{13}, p_{14}$} as input, first the points are sorted, then they are divided into two splits equal in size. The first mapper reads the data points in *Split 1*, generates a *PVD* and emits <1, PVD_1)>. The key value "*1*" is constant for all mappers. Likewise the second mapper emits <1, PVD_2>. Next the reducer aggregates these *PVD*s and merges them into a single *VD*. The final output of the reducer is in the following <key, value> format: <p_i, *VN*s(p_i)> where *VN*s(p_i) represents the set of Voronoi neighbors of p_i. For example, for point p_1 the output is < p_1, $VN(p_1)=\{p_2, p_3, p_4\}$> and for point p_{14} the output is < p_{14}, $VN(p_{14})=\{p_{10}, p_{11}, p_{13}\}$>. Analogously, a <key, value> pair is generated for each data point.

Once the reducer outputs the result, the output is also split into multiple blocks and scattered across the machines in the cluster to prepare for query processing. Suppose the output is divided into two equally sized splits. Consequently, the first split will contain the Voronoi polygons of $\{p_1,..., p_7\}$ and the second will include those of $\{p_8,\ldots, p_{14}\}$. Figure 13b shows how the generated *VD* is stored on HDFS.

5.3. Query processing

In this section, we discuss our proposed MapReduce-based approaches to answer a variety of spatial queries using Voronoi diagrams. For each query type, we first define the problem and then discuss our approach.

5.3.1. Reverse Nearest Neighbor Query (RNN)

Given a query point q and set of data points P, reverse nearest neighbor (RNN) query retrieves all data points $p \in P$ that have q as their nearest neighbors. There are two cases of RNN queries, namely, monochromatic RNN and bichromatic RNN. We focus on monochromatic RNN (MRNN) which has been extensively studied in spatial databases [29]. With MRNN, all objects are of the same type. Therefore, a point p is considered an RNN of q if there is no other point p′ where $d(p, p') \leq d(p, q)$. RNN query has been used in a wide range of applciations such as decision support systems, profile-based marketing, bioinformatics, and optimal location. For example, McDonald's has a marketing application in which the issue is to determine the business impact of restaurants to each other. A simple task would be to determine the customers who would be likely to use restaurants.

Figure 14 illustrates the computation of reverse nearest neighbor for a given set of data points in 2D space where each data point is considered a query point as well. The search expands from each point p_i simultaneously until p_k, the point closest to p_i, is found. Performing this task for each point p_i results in a circle centered at p_i with a radius $|p_i, p_k|$. Therefore, the nearest neighbor to each point lies on the perimeter of its corresponding circle. Given that p_k is p_i's nearest neighbor, p_k's reverse nearest neighbor includes p_i and possibly other points whose nearest neighbors are p_k as well. For example, p_5 is the nearest neighbors of p_2, p_3 and p_4 in Figure 14; therefore, p_5's RNNs are p_2, p_3 and p_4.

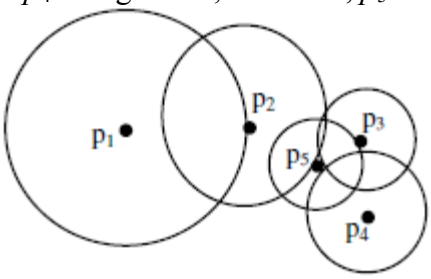


Figure 14 - The reverse nearest neighbor query

MapReduce Solution: The RNN problem defined above is intrinsically applicable to MapReduce. Given that each mapping operation is independent of the others and all maps can be performed in parallel, first each point p_i finds its nearest neighbor p_k efficiently and emits $<p_k, p_i>$ in the map phase. Next, all points having the same key, p_k will be

grouped in the reduce phase by indetifying p_k's RNNs without any extra step. The detailed map and reduce steps for processing an RNN query is as follows. Suppose the input to the map function is:

<p, Voronoi Neighbors (*VN*s)>, $\forall p \in P$

Map: Each point p_i finds p_k as its nearest neighbor. Recall that the nearest neighbor of a point lies within its *VN*s (see Property 4) and on average the number of *VN*s cannot exceed 6 (see Property 3). Therefore, in order to find the nearest neighbor we need to check only 6 points on average. Once p_k is found, the mapper emits the pair <p_k, p_i>, which states that p_k's reverse nearest neighbors include p_i.

Reduce: The reducer aggregates all pairs with the same key p_k (i.e., the result of each nearest neighbor search) and emits a set <p_{k1}, p_{k2},…,p_{km}>, which contains all points p_{ij} 0<j<m+1 whose nearest neighbor is p_k.

For example, p_2, p_3 and p_4 all have p_5 as their nearest neighbor in Figure 14. In the map phase, these points will locate p_5 as the nearest neighbor and emit the <key, value> pairs: <p_5, p_2>, <p_5, p_3>, <p_5, p_4>, respectively. Then in the reduce phase, every point with the same key, p_5 will be aggregated as <p_5, {p_2, p_3, p_4}>. This group forms the RNNs of p_5.

5.3.2. Maximizing Reverse Nearest Neighbor (MaxRNN)

A MaxRNN query [30] locates the optimal region *A* such that when a new point *p* is inserted in A, the number of RNNs for *p* is maximized. This query is known *optimal location problem* as well. The optimal region can be found using nearest neighbor circles.

DEFINITION 1 Nearest Neighbor Circle (NNC)
Given a point p and its nearest neighbor p', NNC of p is the circle centered at p' with radius |p, p'|.

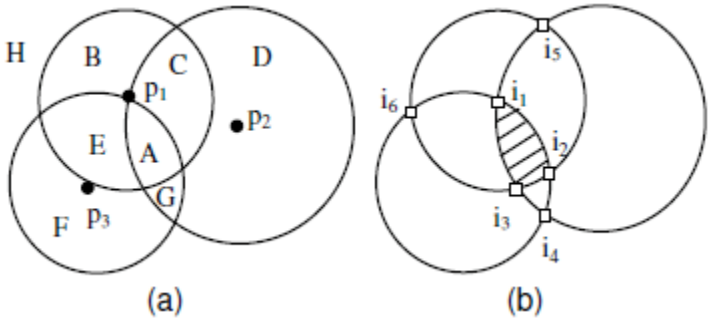


Figure 15 - Overlapping NNCs

Given a set of data points P, each point p_i finds its nearest neighbor p_k, and computes an NNC based on p_k. The region which is intersected by the highest number of NNCs is the answer to MaxRNN query. Figure 15a shows an example of MaxRNN with three points {p_1, p_2, p_3}. The nearest neighbors of p_1, p_2 and p_3 are p_3, p_1 and p_1, respectively. The boundaries of NNCs decompose the space into disjoint regions labeled as A, B, C, D, E, F, G and H. As shown in Figure 15b, region A, the overlapping region of all circles, is maximizing the size of RNN for the new point and is represented by a set of intersection points of NNCs which are covered by the maximum number of circles, i.e. {i_1, i_2, i_3}.

MapReduce Solution: The main motivation behind parallelizing a MaxRNN query with MapReduce is that MaxRNN queries need to process large datasets in its entirety which may result in an unreasonable response time. For example, in a recent study it has been showed that the computation of a MaxRNN query with 200,000 points takes several hours. Whereas, our MapReduce-based approach can answer the same query in a few minutes with 32 nodes and the performance can be improved as more nodes join the cluster.

With our approach, we compute a MaxRNN query in two MapReduce steps where the output of each step is given to the following step as input. First step finds the nearest neighbors of every point and computes the radiuses of the NNCs. The second step finds the intersection points that represent the optimal region. The detailed map and reduce steps are as follows. Suppose the input to the map function is in the following form:

<p, Voronoi Neighbors (*VN*s)>, $\forall_{p \in P}$.

Step 1: Each point p finds p' as its nearest neighbor among the *VN*s, and sets |p, p′| as the radius r of its NNC. Subsequently, for each point p_{vn} in the *VN*s of p, <p_{vn}, {p, r}> (i.e. <key, value>) is emitted as the intermediate output by the mappers. Subsequently, the reducers aggregate the intermediate output, such that for every p and its *VN*s a radius value is set. The final output of the reduce phase is as follows.

<(p, r), {Voronoi Neighbors with radius of their NNCs}>, $\forall_{p \in P}$

Step 2: First, each point p checks its *VN*s in order to find the NNCs with which it overlaps using the radius values computed in the first step. Two circles centered at p_1 and p_2 with radiuses r_1 and r_2 overlap if $|p_1, p_2| < r_1 + r_2$.

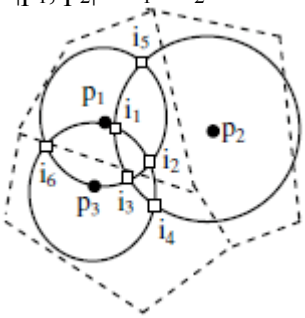


Figure 16 - MaxRNN computation

We expand our search to neighbors' neighbors in stages as follows. For each point p_{vn} in the *VN*s of p, the mappers emit <p_{vn}, {*VN*s - p_{vn}}> as output. Then the reducers receive the input from the mappers and emit it as is. This expansion is repeated until all overlapping NNCs are found for every point p. Once the expansion is terminated, the mappers compute the intersection points.

Subsequently, we count the number of NNCs covering each of the intersection points to identify the weights. An intersection point i can only be covered by the NNCs which overlap the NNC on which i exists. Since we have already found all NNCs at the end of the expansion phase, we find the weights of each i and emit <contant_key, (i, w(i))> pairs as output of the map phase where w(i) represents the weight of the intersection point i. At the concluding reduce step, all intersection points are grouped together since they

have the same constant_key, and then the highest weigthed intersection points are found and emitted as the final output.

Figure 16 shows NNCs and the underlying Voronoi polygons with three data points. At the first step, p_1 finds p_3 as the nearest neighbor among its *VN*s $\{p_2, p_3\}$ and sets $|p_1, p_3|$ as the radius of its NNC. The final output of the first MapReduce step for p_1 is $<(p_1, |p_1, p_3|), \{(p_2, r_2), (p_3, r_3)\}>$. In the map phase of the second step, p_1 checks its *VN*s $\{p_2, p_3\}$, finds that it overlaps $\{p_2, p_3\}$ and computes the intersection points $\{i_2, i_3, i_5, i_6\}$. The search for p_1 can expand to neighbors' neighbors, namely neighbors of $\{p_2, p_3\}$; however, for the sake of clarity we do not demonstrate the expansion phase. Each intersection point *i* then computes its weight by counting the NNCs covering *i.* For example, i_2 is covered by the NNCs of $\{p_1, p_2, p_2\}$, and $w(i_2)=3$. For each intersection point *i*, the second map phase emits $<1, (i, w(i))>$ pairs, i.e., $<1, (i_2, 3)>$, $<1, (i_3, 3)>$, $<1, (i_5, 2)>$, and $<1, (i_6, 2>$. The final reduce phase combines all pairs, finds the intersection points with the heighest weights who are emitted as the final output, i.e. $\{ i_1, i_2, i_3\}$.

5.3.3. k Nearest Neighbor Query (kNN)

Given a query point q and a set of data points P, k Nearest Neighbor (kNN) query finds the k closest data points $p_i \in P$ to q where $d(q, p_i) \leq d(q, p)$. Voronoi diagram has been showed to be an efficient method to solve kNN problem [31]. The *first* NN is found by locating q into corresponding Voronoi polygon (*VP*). The *second* NN to *any* location inside a Voronoi polygon *VP* (p_i) is among the *VN*s of p_i .

MapReduce Solution: We address kNN problem with one MapReduce step if all NNs of q are in the same split. Additional steps might be required for some query points which have NNs in more than one split. The map and reduce steps are discussed below. Suppose the input to the map function is as follows:

<p, Voronoi Neighbors (*VN*s)>, $\forall p \in P$

Map: Suppose we answer a 3NN query with two mappers and one reducer for query point q_1 in Figure 17. First, each mapper reads its split. All data points in the split are put in a hash map where each point is a key and *VN*s are values. Subsequently, the point p_1, the first NN, is found where q_1 is inside *VP* (p_1). Notice that only one split can have the *VP*(p_1) and only one mapper processing that split can locate *q*. The second mapper can simultaneously process the query point q_2 and other queries as well. Clearly, processing a batch of query points simultaneosuly improves the throughput considerably. However, for simplicity, we will move on the discussion assuming that there is only one query point. The second NN is among the *VN*s of p_1, $\{p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$. Suppose the second NN is p_2. The third NN is among the *VN*s of $\{p_1, p_2\}$. The *VN*s of p_1 are already known. The neighbors of p_2 are efficiently found by using the hash map. A distance from *q* to every point in the set of *VN*s is calculated and the point with the minimum distance is found as the third NN, i.e., $\{p_5\}$. Once all the neighbors are found, the mapper emits the query point q as key and the neighbor list as value, i.e., $<q_1, \{p_1, p_2, p_5\}>$. In some cases, a query point can be close to the boundary of a split (see q_3 in Figure 17) and some of its nearest neighbors can be in the neighboring split. In order to handle this case, when a point is found as a nearest neighbor, we check each of its *VN*s in the hash map if they are in the current split. If not, we flag the missing points and reprocess them in an additional MapReduce step as described below.

Reduce: Receives the query point as key and its neighbors as value, and emits as they are.

After the reduce step, the final output is checked if the query contains any flagged point. If some of the neighbors are in an another split, an aditional MapReduce step is required. In the second map phase of such a case, the query comes with its current kNNs and flagged points. The flagged points are located in the Voronoi diagram and the new candidate points are found. Next, the new set of kNNs is selected among the current kNNs and the candidate set. Finally, the mapper emits the new result.

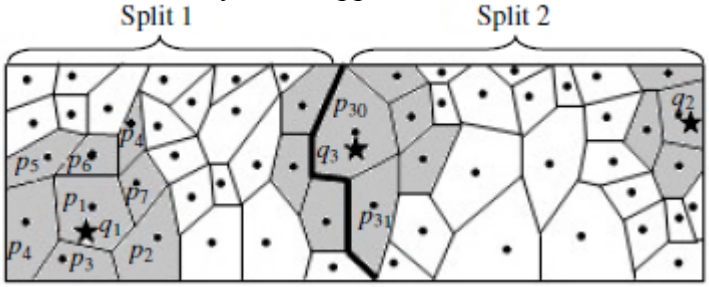


Figure 17 - visualization of processing multiple kNN queries on Hadoop

5.4. Performance Evaluation

5.4.1. Setup

With our experiments, we used two real Navteq datasets that consist of all financial institutions (FIN), auto services (AUT), restaurants (RES), other businesses (BSN) in the entire U.S., containing approximately 190000, 250000, 450000 and 1300000 data points, respectively. With these datasets, we evaluated the performance of our proposed techniques and correspondingly compared them with their closest related work while varying the number of employed nodes in the cluster.

We conducted our experiments on the Amazon EC2 cluster with extra large instances that run on 64 bit Fedora 8 Linux Operating System with 15 GB memory, 4 virtual cores, and 4 disks with 1,690 GB storage. The I/O performance of EC2 varies from 40 MB/s to 140 MB/s during a day depending on the hour [32]. In order to obtain consistent results, we conducted all experiments in off-peak hours. All experiments are implemented using Hadoop version 0.20.1. The configuration of Hadoop can considerably influence the performance of the query processing techniques. Accordingly, we performed all of our experiments with the same setup, where the replication factor of each file is set to 1 in order to avoid writing overhead and to enable compression of the data transferred between mappers and reducers. Moreover, each node runs two map instances and one reduce instance.

5.4.2. Results

In this experiment, we compared the performance of our proposed indexing approach (VD) with the MapReduce-based R-tree [33] in terms of index construction time and query response time. Figure 18a illustrates the effect of varying the number of nodes in

the cluster on the construction time of VD and R-tree, given the RES dataset. As shown, the construction times of both index structures decrease almost linearly as more parallelism is induced by the new nodes. Construction of the R-tree index takes less time than that of VD. This is because the mappers generating *PVD*s execute expensive computations and there is only one reducer merging the *PVD*s in the cluster. Figure 18b depicts a similar observation with the BSN dataset. Even though R-Tree is faster to build, our experiments show that VD outperforms R-tree in query response time. Figure 18c and d depict the time to compute the nearest neighbor (as an example query) for every point in the RES and BSN datasets, respectively. As shown, in both cases VD is at least five times faster than R-tree and R-tree does not scale well. This is because with VD, points can immediatelly find their nearest neighbors in their VNs in the map phase; however, R-tree does not guarantee that nearest neighbor of a point p is in the same split with p. Thus, with R-tree a reducer always verifies the answer.

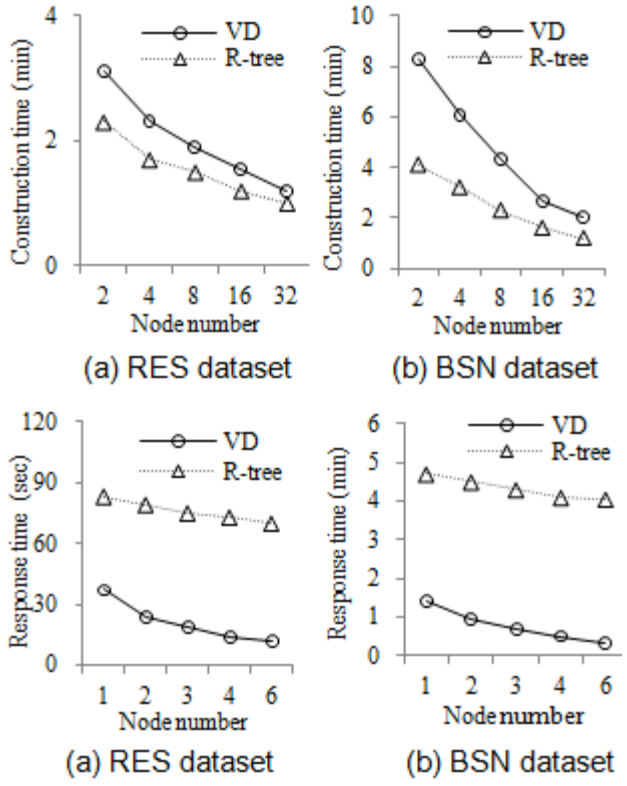


Figure 18 - ab) VD & R-tree construction cd) NN query on VD & R-tree

In this experiment, we study the impact of varying the number of nodes as well as size of the datasets on average response time of our proposed RNN query processing technique. Our approach processes RNN query for every data point simultaneously rather than a single data point. Figure 19a shows the average response time of processing a reverse nearest neighbor query for every data point on a single node without parallelism. As shown, the average response time slightly decreases with the increasing number of data points; hence, the overall throughput increases. With the state-of-the-art centeralized approach, FITCH [34], an RNN query for a single data point runs in a few seconds given a similar dataset in size with RES. Clearly, processing an RNN query for all the data

points in a dataset decreases the average response time by several orders of magnitude. Figure 19b illustrates the effect of node number on the average response time of processing an RNN query for BSN and RES datasets. As shown, the response time decreases linearly as more nodes join the cluster.

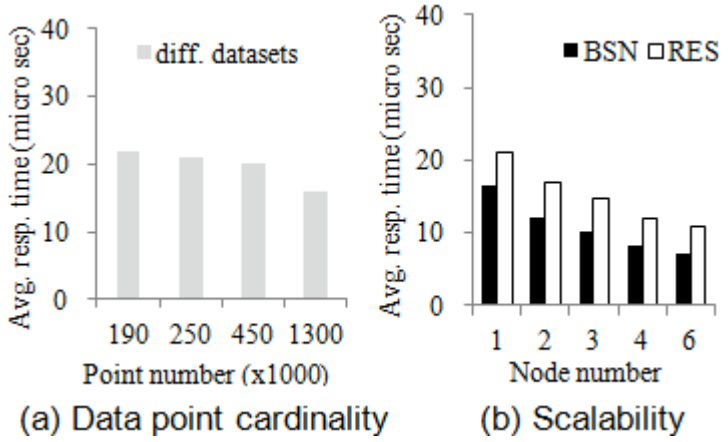


Figure 19 - Effect of node number on RNN

In this experiment,we study the performance of our proposed technique for MaxRNN query answering by varying node numbers with both BSN and RES datasets. We compare our Voronoi based approach with the best-known centeralized algorithm, MaxOverlap[30], which uses R-tree index structure. Both approaches implement the same algorithm with different data structures; therefore, the performance on a single machine without parallelism is similar for both. However, our approach is parallelizable and shows linear scalability in response time utilizing VD. Figure 20 illustrates the execution time with varying node numbers and datasets. As shown, parallelization reduces the response time *at linear rate* for both datasets. As the number of nodes doubles, we observe a performance increase from %30 up to %50.

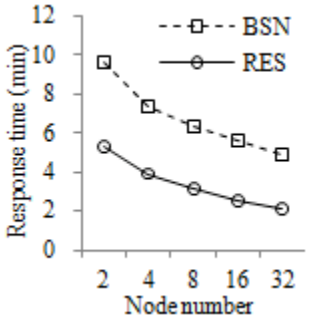


Figure 20 - Effect of node number on MaxRNN

In this experiment, we compare our approach (VD) with the existing MapReduce based kNN search (MRK) [35] in terms of response time. We study the impact of k and query cardinality on response time. Figure 21a presents the response of kNN queries (with BSN dataset) processed on a single node for varying k. As shown, VD outperforms MRK significantly. This is because, for a given number of data points *P* and given number of query points *Q*, MRK always outputs *P*×*Q* <key,value> pairs in the map phase independent of *k*. Due to the massive amount of data generated by the mappers, the

response time is high. The effect of k is not a substantial factor for VD as long as all nearest neighbors of a query point is in the same split. Otherwise, additional MapReduce steps might be required. Thus, for larger k values, the chance of having nearest neighbors in more than one split is also higher. Figure 21b shows the impact of query cardinality with uniform distribution on response time. MRK does not scale with the increasing number of queries due to the same reason discussed above. As more queries are processed simultaneously using VD, the throughput increases. This is because, before the queries are processed, the entire data is already fetched into the memory.

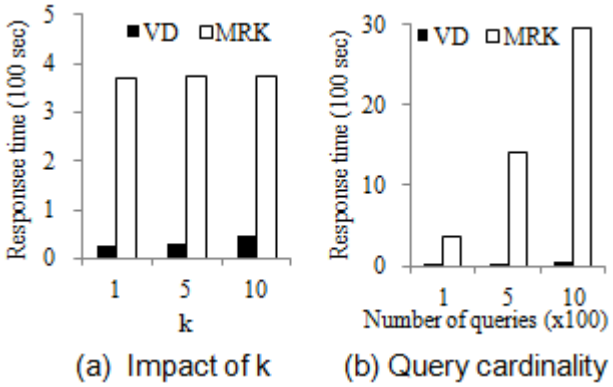


Figure 21 - kNN

6. Conclusion and Future Work

In this research, we developed a data-driven end-to-end decision making system, TransDec, that enables interactive and extensive querying of transportation related spatiotemporal datasets including traffic sensor data, trajectory data, transportation network data, and points-of-interest data. Particularly, we explained the design \of TransDec's architecture. We also introduced a set of advanced data representation and spatiotemporal queries supported by TransDec. We elaborated on the challenges with each line of research and presented our solution in each case.

References

[1] RIITS (Regional Integration of Intelligent Transportation Systems). http://www.riits.net/, Last visited December 25, 2008.

[2] Freeway Performance Measurement System. https://pems.eecs.berkeley.edu/, Last visited December 21, 2008.

[3] ADMS Smart Travel Lab. http://cts.virginia.edu/stl_adms.htm, Last visited December 21, 2008.

[4] Khoshgozaran, A. and et al. "A Multi-Resolution Compression Scheme for Window Queries Road Network Databases", SSTDM 2006.

[5] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In VLDB, 2004.

[6] Vitter, J.S. and Wang, M., "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets", Proc. of the 1999 ACM SIGMOD Conf., 1999, 193-204

[7] Wall, M. E.; Rechtsteiner, A. & Rocha, L. M., "Singular value decomposition and principal component analysis", 2003

[8] Jolliffe, I. T., "Principal Component Analysis", Springer-Verlag, 1986

[9] Matias, Y.; Vitter, J. S. & Wang, M., "Dynamic Maintenance of Wavelet-Based Histograms", Proc. of the 26rd Intl.Conf. on Very Large Data Bases (VLDB), 2000

[10] Chakrabarti, K.; Garofalakis, M.; Rastogi, R. & Shim, K., "Approximate Query Processing Using Wavelets",Proceedings of the 26th VLDB Conference, 2000

[11] Jahangiri, M.; Sacharidis, D. & Shahabi, C., "SHIFT-SPLIT: I/O Efficient Maintenance of Wavelet-Transformed Multidimensional Data",24th ACM SIGMOD International Conference on Management of Data, 2005

[12] Schmidt, R. R. & Shahabi, C., "ProPolyne: A Fast Wavelet-based Algorithm for Progressive Evaluation of Polynomial Range-Sum Queries(extended version)", VIII. Conference on Extending Database Technology, 2002

[13] Gibbons, P. B.; Matias, Y. & Poosala, V., "Fast Incremental Maintenance of Approximate Histograms",
Proc. of the 23rd Intl.Conf. on Very Large Data Bases (VLDB), 1997

[14] Datar, Y.; Gionis, A.; Indyk, P. & Motwani, R.,"Maintaining Stream Statistics over Sliding Windows",Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms, 2002

[15] Guha, S.; Meyerson, A.; Mishra, N.; Motwani, R. & OíCallaghan, L., "Clustering data streams",Proc. of the 2000 Annual Symp. on Foundations of Computer Science (FOCS), 2000

[16] Domingos, P. & Hulten, G., "Mining high-speed data streams", Proc. of the Sixth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 2000

[17] Abadi, D. J.; Carney, D.; Cetintemel, U.; Cherniack, M.; Convey, C.; Lee, S.; Stonebraker, M.; Tatbul, N. & Zdonik, S., "Aurora: a new model and architecture for data stream management",VLDB Journal, 2003

[18] Chandrasekaran, S.; Cooper, O.; Deshpande, A.; Franklin, M. J.; Hellerstein, J. M.; Hong, W.; Krishnamurthy, S.; Madden, S. R.; Reiss, F. & Shah, M. A., "TelegraphCQ: continuous dataflow processing",Proceedings of the 2003 ACM SIGMOD international conference on Management of data, 2003, 668-668

[19] Fung, W. F., "COUGAR: The Network is the Database",Proc. of SIGMOD Conference, ACM Press, 2002, 621-621

[20] Arasu, A.; Babcock, B.; Babu, S.; Datar, M.; Ito, K.; Nishizawa, I.; Rosenstein, J. & Widom, J., "STREAM: The Stanford Stream Data Manager", IEEE Data engeneering Bulletin, 2003, 26

[21] Naughton, J. , "The Niagara Internet Query System", Proceedings of the 26th VLDB Conference, 2000

[22] Cao, H.; Wolfson, O. & Trajcevski, G., "Spatio-temporal data reduction with deterministic error bounds", The VLDB Journal - The International Journal on Very Large Data Bases, Springer-Verlag New York, Inc, 2006, 23, 211-228

[23] K. Rangan.The Cloud Wars: $100+ billion at stake. Tech. Rep. Merrill Lynch, May 2008

[24] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI, 2004

[25] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. Commun. ACM, 51(1):107-113,2008

[26] F. Korn and S. Muthukrishnan. Influence Set Based on Reverse Nearest Neighbor Queries. In SIGMOD, 2000

[27] Akdogan A., Demiryurek U., Banaei-Kashani F. and Shahabi C., "Voronoi-based Geospatial Query Processing with MapReduce", IEEE Cloud-Com 2010. **Best Paper award**

[28] A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley, 2000

[29]Y. Tao, D. Papadias, and X. Lian. Reverse kNN Search in Arbitrary Dimensionality. In VLDB, 2004

[30] R.C. Wong, M.T. Ozsu, P.S. Yu, A.W. Fu, and L. Liu. Efficient Method for Maximizing Bichromatic Reverse Nearest Neighbor. In VLDB, 2009

[31] M.R. Kolahdouzan and C. Shahabi. Voronoi-Based k Nearest Neighbor Search for Spatial Network Databases. Proceedings of the 30th Very Large DataBase, pages 840-851, 2004

[32] D. Jiang, B. Chin, O. L. Shi, and S. Wu. The Performance of MapReduce: An In-depth Study. VLDB, 2010

[33] A. Cary, Z. Sun, V. Hristidis, and N. Rishe. Experiences on Processing Spatial Data with MapReduce. In SSDBM, 2009

[34] W. Wu, F. Yang, C.Y. Chan, and K.L. Tan. FITCH: Evaluating Reverse k-Nearest-Neighbor Queries on Location Data. VLDB, 2008

[35] S. Zhang, J. Han, Z. Liu, K. Wang, and S. Fend. Spatial Queries Evaluation with MapReduce. In International Conference on Grid and Cooperative Computing, 2009