THE DESIGN OF A BENCHMARK FOR GEO-STREAM MANAGEMENT SYSTEMS

Chao Shen

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2011

Shen, Chao. The design of a benchmark for geo-stream management systems. Master of Science (Computer Science), December 2011, 68 pp., 5 tables, 18 illustrations, references, 18 titles.

The recent growth in sensor technology allows easier information gathering in real-time as sensors have grown smaller, more accurate, and less expensive. The resulting data is often in a geo-stream format continuously changing input with a spatial extent. Researchers developing geo-streaming management systems (GSMS) require a benchmark system for evaluation, which is currently lacking. This thesis presents GSMark, a benchmark for evaluating GSMSs. GSMark provides a data generator that creates a combination of synthetic and real geo-streaming data, a workload simulator to present the data to the GSMS as a data stream, and a set of benchmark queries that evaluate typical GSMS functionality and query performance. In particular, GSMark generates both moving points and evolving spatial regions, two fundamental data types for a broad range of geo-stream applications, and the geo-streaming queries on this data.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Motivation

1.1.1. Sensor Technology

In recent decades, sensor technology has grown at an unbelievable pace. The sensor size is smaller, communication is faster, batteries can continue supplying power longer, networks are more stable, and overall price is lower. All these developments bring great benefits to people's daily lives. The following paragraphs will discuss the benefits in more details.

First of all, the miniature sensors' sizes are smaller than before. The coin-sized sensors are easier to be embedded in moving objects. In many countries of the world, cell phones are ubiquitous. Since the size of a typical cell phone is very small, people usually just leave it in the pocket or purse, and carry the cell phone every day for communication purposes. Now many cell phones are embedded with tiny-sized sensors and many applications are developed using data collected by these sensors. With the explosive growth of the smart phone market, fueled by faster data network infrastructure and the increasing need for mobile computation, today's mobile phone has become not just a communication device but also incorporated entertainment, office productivity, and navigation functionalities. Today, many cell phones have built-in GPSs (global position system) and various sensors and run a complete operating system, providing a standardized interface and platform for application developers. The fast development of smart phones enables a new and scalable way to collect location data through smart phone applications. There are many location based services (LBS) that are developed based on the cell phone's build-in sensors. Basically the LBSs are the services that use the objects' location information for services. These services include traffic monitoring system and patients' location monitoring system. These services strongly depend on the sensor networks and database management systems. The sensors gather information about location and sometimes involve other interesting data attributes, such as temperature, humidity, and, the most commonly used attribute, temporal data. After the sensors collect all the necessary information, they send the data to a particular data management system. The data management system should be able to store the data, process the data, and query the data when users want to find the information they need.

Additionally, because of the fast communication capacity of mobile devices, real time communication becomes possible. Many applications take advantage of that, such as applications with "push" abilities. Many built-in sensors can gather different kinds of information such as 3D acceleration, altitude, course and speed, position, latitude, longitude, and so on. Many applications use the information and the mobile communication to serve users. For example, when a user searches nearest restaurants in a mobile phone location based service application, the mobile phone gets the user's location from the built-in sensor, sends the location into its server, queries the nearest restaurants in its database management system, and sends the query results back to the user's phone. When the user is walking or driving, the location of the user is changing. The new locations will be sent from the mobile phone to the application in real time. The application will query the nearest restaurants again based on the new locations. Based on the search results, the application will pull the earliest information out of the queue, and push the newest search results into the queue of the phone's physical memory. By this way, a user can be updated with relevant information in real time. To help real time communication, instead of using a traditional relational tuple based data management system, some applications use a stream data management system. In this chapter, sections 2 and 3 discuss stream data and its management systems in more detail.

Fortunately, now the sensors' battery can last much longer than before. For instance, the iPhone's build in Nike + iPod sensor's battery can keep working for more than a thousand active hours. The sensor has a sophisticated built-in power management system. The power of the battery will only be consumed when the sensor is active. This system allows the the sensor to have a longer life than the power's maximum supply ability. Moreover, the sensor network technology becomes more mature. For example, the researchers are more capable of building advanced sensor networks with back up sensors. When setting up a sensor network, researchers usually prepare several back up sensor nodes to avoid data losses in case of sensor node failure. Therefore, in a sensor network, even when a sensor is out of battery power, other sensor nodes will take the dead sensor's duty, and continue sending data to the data management system.

The cost of the sensor is also decreasing. And because of the sensor's lower price, many daily used devices have have sensors built in. For example, while many latest versions of smart phones, tablets, and cameras all contain sensors, their prices are still affordable. More and more consumers can enjoy sensor technologies at affordable prices. The sensor based applications make people's lives more convenient. Because more people use sensors daily, there are more and more geo-streaming data generated. The huge amount of geo-stream data requires more efficient database management systems.

Overall, sensors are much easier to use, more convenient to set up, and very reliable for scientific research.

1.1.2. Geo-Stream

As mentioned by the above section, the continuing development in sensor technology allows researchers to gather information easily in a real-time pace and is widely used in people's daily lives and research labs. The wide use of the sensors in many different research areas generates huge amount of data. Much of this real-time data is geo-referenced stream data—namely geo-streams. Spatial data often involves an object represented by a two- or three-dimensional location on a geographic coordinate system, such as the latitude-longitude geographic coordinate system, but some objects may require a spatial extent. Temporal data is often a timestamp associated with the spatial data. Together, this forms spatio-temporal data that can represent objects in relation to time, e.g. the object's location at a given time.

While it may be interesting to know the object's location at any moment, the real-time trajectory is often more interesting. The sensor network monitoring the object can produce a data stream of changing location points that researchers refer to as streaming data, or in this case, geo-streaming data since it involves spatio-temporal data.

Many applications produce geo-streaming data, such as health care, emergency management, object positioning and identification, earth science, and traffic monitoring. Location based service is an area that could be strongly related with geo-stream data.

Location based services can be generally divided into 4 categories [14]. They are services of emergency, services of information, services of tracking, and services of entertainment. One typical application of emergency service is the hospital's emergency response. Many hospitals embed sensors in small electronic devices that could be easily carried by their patients. The sensors can continuously report a patient's position as a geo-stream to the hospital. When the sensor senses an emergency situation, such as abnormal pulse, temperature that is below or above a certain range, and palpitation that is suddenly disappearing, or when the patient feels uncomfortable and triggers the sensor's emergency button by himself or other witnesses, the hospital can find the location of the patient much easier. It will save time for the hospital to arrange an ambulance and analyze the information gathered by the sensors to prepare for the emergency treatment. There are also many applications on mobile devices that obtain users' location geo-stream data from their mobile devices and serve the users with local news, weather, stock information. Tracking services can be used to monitor traffic flows in road networks. In many countries, the traffic-monitoring system is developed or under development. Traffic-monitoring is taking a very important role in city planning and developing. Traffic-monitoring systems often generate geo-streaming point data and deliver the traffic flows into a data management system. Therefore, we can query the data stream for the historical trajectory of a moving object and possibly to discover the habits of a certain kind of object through long-term monitoring of their trips. The data can be used to do traffic analysis, behavior analysis, and crime control. One interesting application of entertainment services is to locate a user's nearby friends. When a user is moving, the geo-stream data of the user's location comes into a data management system. And the system can query for the friends who are very close to the user's current location. Because geo-steam data is real time, usually is location related, sometimes contains geo-related extent information, and often comes with a time stamp, location based services become more and more popular.

Not only do location based services generate a huge amount of geo-stream data, many other applications also generate an abundance of geo-stream data. For example, many applications are developed for monitoring weather situations. Weather-monitoring systems often generate spatial data to represent the regional events such rainstorms, snow weathers, and blizzards. The geo-stream data of the weather monitoring provides useful information that could be used to avoid injury, death, and asset loss in serious weather situations.

The resulting large data volumes require efficient management systems that traditional database management systems (DBMS) lack. For example, a traditional DBMS does not offer efficient modeling of spatial data types. Newer spatio-temporal database management systems (STDBMS), such as SECONDO [10], do offer those capabilities but often lack streaming capabilities.

1.1.3. Geo-Stream DBMS

Researchers have developed several stream management systems (SMSs) to reconcile the need of managing streaming data.

One example is Stanford's STREAM project, a general-purpose SMS that uses a specially designed continuous query language (CQL) with sliding window operators for stream-to-stream and stream-to-relation queries [5]. The paper states that more and more applications need streaming data support. The data stream can continue flowing into a data management system. Therefore, there is a need for queries that could handle not only the static data sets but also the data stream. Traditional data management systems hardly process the continue queries on the data stream. Therefore, they developed new query system to allow users easily issue continuous queries on stream data. Their queries are based on three different

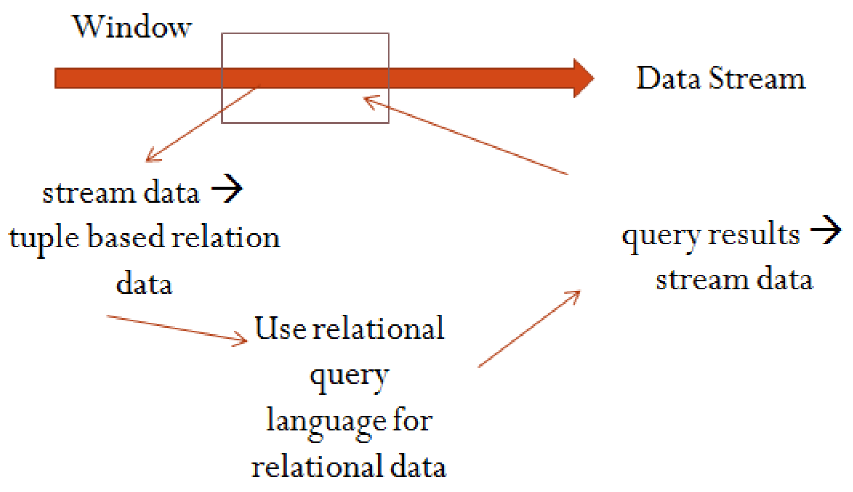

Window
Data Stream
stream data → tuple based relation data
query results → stream data
Use relational query language for relational data

Figure 1.1. STREAM's query system.

abstract parts: relational query language, window specification language, and relation-to-stream operator. The window specification language uses window approach to select limited data sets from the data stream, and transforms the data into relational tuples. It is a process that performs stream-to-relation operations. It first places a window in the data stream. When the data flows through the window, it is selected. The selected data can be taken out from the flow and processed into the form of relational tuples. In addition, it applies relational query language to query the target tuples. Most traditional query language, such as SQL, can perform a query for relational tuples. When the query returns results, it begins to run the relation-to-stream operations to output the results back to the data stream. The relation-to-stream operations can transform the relational tuples into streaming data. Therefore, the query did not break the data stream flow. The data comes into the query system as a stream and also comes out as a stream. Therefore, in general, every time a data stream brings the new data into a window, the window does three operations: stream-to-relation, relational query, and relation-to-stream. All the data goes into the window in the stream; the query results continue running out of the window and join into the stream. Fig. 1.1 shows the STREAM's query system.

Another example is Aurora, a joint project between Brandeis University, Brown University, and M.I.T. that also uses window operators and eventually led to the StreamSQL language [2]. It argues that the traditional DBMSs do not care about the historical data while giving most attention to current data. Furthermore, it states that traditional DBMSs always get exact query answers. Therefore, a traditional DBMS needs complete data to answer a query. Moreover, the traditional DBMSs lack the ability to handle the real time data. Streaming data is always incomplete and needs to be processed in real-time. For example, when a data management system accepts data stream, the input data continues flowing into the system. The data may not be complete when the system performs a query. As an instance, when we try to query the cars' trajectory in 2 hours from now, the data is not complete when we run the query. The query will finish after the 2 hours. And since the data stream continues flowing in the system, if we do not perform query in real-time, we may miss the right point of doing the query. As another example, if we perform a query of selecting a car's current speed, if a system can not perform the query in real-time and performs the query 2 minutes later, it will returns the cars's speed of 2 minutes later, but not the speed from the time we want to perform the query. Therefore, a system for streaming data has to perform queries from the incomplete data sets and do it in real-time. From the incomplete data sets, a query may not be able to return an exact answer, but it may return the approximate query answers. For example, for a monitoring system, it may need to perform approximate queries. The monitoring system's data is continuously generating all the time. The sensors for a traffic monitoring system can produce a huge amount of geo-streaming data every day. Usually, a monitoring system needs to pay a lot of attention to the historical data when it tries to find out abnormal objects in its data sets. When an abnormal event happened, it needs to trigger an alert to tell users to pay attention to the abnormal events. However, since the input data's amount is huge and continues to grow, the early streaming data may flow out of the management system already. Therefore, when we track the historical data, we may not be able to return the exact answer from the incomplete historical data sets which lead to approximate query answering. For the query processing steps, Aurora is similar to Stanford's STREAM project. It adopts the boxes and arrows paradigm. The data input stream comes into a box, queries are performed in the boxes, and the results go out from the box. The query results flow into applications as an output stream. Aurora designs connection points on the data stream path. The connection points are for applications, and they are where the applications connect with Aurora. When an application needs new queries, Aurora can add new boxes in the connection points on the path of data stream. When a data stream comes though the boxes, the system cut the unbound data stream to finite data sets. The bounders of the data sets are the bounders of the boxes. Then it performs queries in the boxes, and outputs the results back to the stream. In the data stream, the query results flow into the application that is connected with the connection nodes. When an application does not want a query anymore, Aurora can delete the boxes on the path. Since the applications often need to query the historical data, Aurora can usually store data for certain time period in the connection point. In a few cases, a connection node can have no upper-stream node and it may only have historical static data sets. But without the data that comes from upper-stream, the data in the connection node could not form a stream. If we only want to query for the historical data, Aurora can just push the data from the connection node going down as a down-stream even without upper-stream data. And down-stream data could pull the data into a box to execute queries. Fig. 1.2 shows the Aurora's query system.

The 2 examples above of stream data management systems (SMS) illustrates how a SMS is developed and what their basic features are. They are very famous SMSs, but they are not the only well-structured SMSs that have been developed. In recent decades, many researchers are focusing on stream data management area. They developed many SMSs and used them in a lot of different applications. Some labs are still developing their own SMSs for specific applications; some are modifying the existing data management system to make

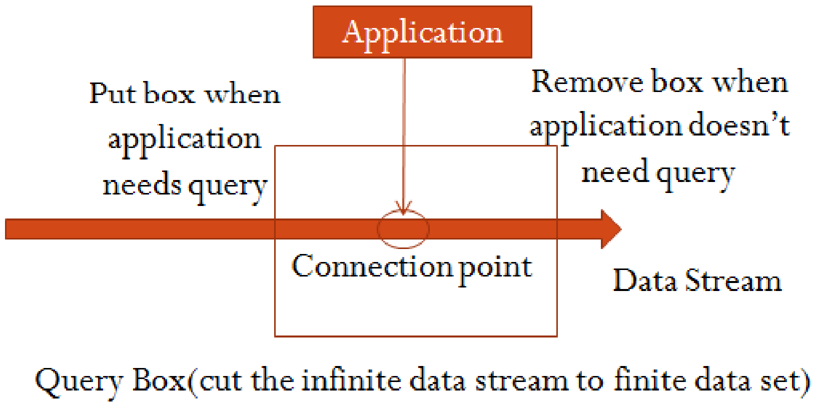

Application
Put box when application needs query
Remove box when application doesn't need query
Connection point
Data Stream
Query Box(cut the infinite data stream to finite data set)

Figure 1.2. Aurora's query system.

it more suitable for the stream data; some are optimizing a SMS for making it faster and more reliable.

For example, Borealis later superseded Aurora and offered additional features such as dynamic revising of query results and dynamic query modification [1]. Borealis is a distributed data management system for stream processing. The processing is distributed to different Borealis nodes in a network that has huge amount of query operations. Every node can access and perform query operations toward the data stream. Each node is able to do three different information revision operations toward the data stream. Information revision allows us to recover the data from problems or mistakes in the input. In the Aurora system, it can only insert data into a stream for the revision of information. In the Borealis system, it has the insert data ability just as Aurora; moreover, it can delete a data record from the stream, and replace a data record in the stream. This provides more revising control for Borealis. It also contains many attributes in the data stream, such as data importance and data arrival time, to optimize the query processing. With these attributes, it can use Vector of Metrics to weigh the data's impaction. In this way, we can get the most important or expected query results faster. Overall, Borealis is designed based on Aurora, but contains much more abilities than Aurora.

Another example is StreanInsight. Recently, Microsoft released the StreamInsight SMS, which extends the Microsoft SQL Server framework to offer stream management [4]. These SMSs are generally tuple-based that do not directly support geo-streaming objects.

Most of the example management systems open their source codes for the users. It's a great way to share knowledge. Users can download the source codes and see the entire structures of the systems from the codes. Moreover, most of the developers of stream data management systems also put their end-user applications on their website for free. The other developers can easily implement new applications based on the published source code and applications. And users can easily compare the traditional data management systems and geo-stream data management systems.

1.1.4. Benchmark for Geo-Stream DBMS

The extensive usage of the sensors generates a lot of geo-stream data. The enormous geo-stream data requires efficient stream data management systems for different applications. As said in previous section, there are many SMSs developed recently. The efforts to adapt or create management systems for geo-streaming applications require determining functionality and efficiency through performance evaluations. Benchmarks provide an evaluation platform by offering simplified experimental scenarios using scalable, well-structured, data sets and operations to test individual components or entire systems. This makes it simple to compare multiple systems under identical parameters while eliminating bias introduced when data or operations favor one system over another. However, SMS benchmarks are rare and there is no widely adopted benchmark for geo-streaming management systems in the literature. The close relationship between DBMSs, STDBMSs, SMSs, and GSMSs, does provide several relevant examples including SEQUOIA 2000 (spatial), DynaMark (spatial), Linear Road Benchmark (spatio-temporal streaming), BerlinMOD (spatio-temporal), COSTS (spatio-temporal), and 3-D Spatio-temporal Benchmark (spatio-temporal). The details of the above benchmark are discussed in the Chapter 2. Therefore, it is necessary to have a benchmark for those SMSs, especially for geo-stream data management systems. The thesis studies these benchmarks, and develops a brand new benchmark for geo-stream data management systems. This benchmark is named as GSMark.

1.2. Contribution

This thesis presents GSMark, a framework for evaluating two-dimensional GSMSs. Specifically, this thesis offers three contributions. First, this paper overviews the aforementioned benchmarks and provides a classification scheme to categorizes them as being spatial or spatio-temporal and either streaming or non-streaming. Second, it describes the three components of GSMark. The data generator (1) produces streaming point and polygon objects to cover various test scenarios. The set of queries (2) evaluate GSMS performance concerning spatio-temporal data, both static and streaming, and typical operations involving selection, join, aggregation, and continuous queries. The workload simulator (3) delivers geo-streaming data over a simulated unreliable network. Finally, this paper discusses the benchmark's data generator performance to demonstrate its potential in creating large data sets in reasonable time. GSMark v1.0 is available at http://powerranger.cse.unt.edu/GSMark.

1.3. Organize of the Thesis

The remainder of the thesis is organized as following. Chapter 2 discusses the related works. Several representative benchmarks are reviewed. Through comparing the benchmarks, the thesis identifies what need be developed for a geo-stream benchmark system. Chapter 3, 4 and 5 present the 3 major components of the benchmark. Chapter 3 introduces the data generator of the benchmark. It demonstrates how the data is simulated and the main principles of the traffic flows. Chapter 4 presents the details of the benchmark's work-load simulator. The work-load simulator responds to the data delivery and storage. Chapter 5 summarizes the query sets of the benchmark. It includes the most popular queries for the streaming data and most necessary operations for current geo-stream management systems. Chapter 6 presents several experimental results of the benchmark. Chapter 7 concludes the thesis, and suggests future works that could be done to improve the benchmark.

CHAPTER 2

RELATED WORK

2.1. Benchmark for Evaluating Data Management Systems

Even though there is not a widely adopted GSMS (geo-stream management system) benchmark, several benchmarks share similar characteristics. One popular benchmark is the TPC series that evaluates performance using business scenarios [17]. Adapting this series to STDBMSs (spatio-temporal data management system) is a challenge because of the spatio-temporal context. For example, a brokerage firm performing thousands of transactions per minute consisting of time, purchase/sell amount, and identification information, is significantly different from a weather storm that requires spatial extents and different processing techniques.

Attempts to adapt/create benchmarks for STDBMSs and SMSs (stream management system) have resulted in four general categories. Typically, a benchmark is either spatial or spatio-temporal and either streaming or non-streaming[1]. A spatial benchmark evaluates the system's functionality regarding only spatial aspects, such as the performance of joining polygons. Spatio-temporal benchmarks focus on both space and time; for example, it may evaluate performance when finding overlapping polygon regions during a time interval. A benchmark is streaming if it evaluates the system's streaming capabilities; otherwise, it is non-streaming. The following highlights various benchmarks in these categories, but note that the benchmark's goal determines the category. To clarify, let us consider a benchmark that generates spatio-temporal data. If the benchmark only performs spatial queries, it is a spatial benchmark, not spatio-temporal. If the data could represent streaming data, but the benchmark does not treat it as such, it is a non-streaming benchmark.

[1]There are benchmarks for temporal databases but they are outdated since temporal data is typically associated with spatio-temporal databases today.

2.2. Example Benchmarks

2.2.1. Linear Road Benchmark

The Linear Road Benchmark (LRB) is for stream management systems [6]. The authors identified that a SMS benchmark requires semantically valid data (i.e. not random) meaning that simulated[2] data must to be realistic. LRB provides this by simulating a variable tolling system that charges based on dynamic traffic congestion and accident proximity. The simulated data consists of vehicle location reports every thirty seconds and associated statistical information used for detecting and alerting vehicles of accidents and toll charge calculations. The system uses the MIT Traffic Simulator to generate moving vehicles on straight parallel roads with eight lane roads (four eastbound, four westbound) in a 100x100 square mile area. Accidents occur at randomly every 20 minutes, forcing other traffic to slow down according to a traffic spacing model.

The benchmark provides two query types: continuous and historical. The continuous queries involve variable toll calculations and accident notifications since these are dependent on current conditions. The non-continuous historical queries are account balance, daily expenditure, or travel time estimation queries that the vehicles issue at some fixed probability when producing a location report. The account balance query returns the current toll balance using data from the start of the benchmark execution until the request time. Daily expenditure queries returns the toll charged for a given day within the last ten weeks, excluding the current day. The travel time estimation query uses historical statistics to predict the travel time and toll charges based on the previous ten weeks of data.

While the benchmark does provide data and queries, it is limited in representing realistic situations. First, the road network is limited to parallel straight roads when real road networks have curves and intersections and it only considers traffic accidents as an impediment to traffic flow. More importantly, LRB reduces the vehicle's spatial location from an x-y coordinate to distance from the westernmost point of the expressway. This limits

[2]This paper uses the terms *synthetic* and *simulated* interchangeably.

spatial and temporal data exploration necessary for GSMSs. For example, a GSMS could represent dynamic global area events, such as a moving rainstorm or widespread fire, as polygons that affect different road segments at different times. Therefore, this is technically a spatio-temporal streaming benchmark; however, the data limits its spatio-temporal evaluation.

2.2.2. SEQUOIA 2000 Benchmark

Stonebraker et al. developed the SEQUOIA 2000 benchmark, a non-streaming spatial benchmark, for evaluating performance when involving earth science problems [16]. They identified that earth science applications typically require large databases of images, simulation output, complex data types, and sophisticated searches. The problems also tend to be on local, regional, national, and Earth scales and the benchmark reflects this by having three sub-benchmarks: regional for large regions, national for entire countries, and Earth for the entire globe. They further identify that the common data types for earth science problems are raster, point, polygon, and directed graph, and gather real data from NOAA (National Oceanic and Atmospheric Administration) Advanced Very High Resolution Radiometer sensors (raster data), the USGS (United States Geological Survey) Geographical Names Information System (point data), the USGS land-use/land-cover type dataset (polygon), and the USGS drainage network (directed graph).

The authors provide eleven queries to evaluate the capabilities. The first query evaluates creating and loading the database. Three queries cover the raster data type including listing time-sorted raster data, raster data aggregation using time and geographical region, and changing raster image spatial resolution. The next three queries involve finding a point given an identification value, finding polygons that intersect specified rectangular regions, and finding polygons with specific sizes within a given circular region. Queries 8, 9, and 10, are points, polygons, and raster image spatial joins and the final query is a recursion query using the drainage network directed graph.

The benchmark data is spatio-temporal data; however, it is a spatial benchmark because the temporal attribute is like any other non-spatial, non-temporal, attribute. For example, the joining of raster objects uses it as a filter and the retrieval of objects uses it as a sort key. This data could potentially be streaming data, but the benchmark treats it statically, thereby classifying it as a non-streaming spatial benchmark. The benchmark is also extensible to other problem types, as noted by the authors, but it is still limited to spatial evaluation.

2.2.3. 3-Dimensional Spatio-temporal Benchmark

The 3-Dimensional Spatio-temporal Benchmark by Dr. Werstein is for non-streaming STDBMSs [18]. He identified that most benchmarks at the time, such as SEQUOIA 2000, could not apply to spatio-temporal systems because they rely on spatial information while paying little attention to temporal attributes. The spatial information is also limited to two dimensions while many real-world examples use 3-D data. As a result, the data sets are often lacking and the queries limit the benchmark to spatial evaluation.

The benchmark fully evaluates the database's 2-D and 3-D spatio-temporal capabilities by extending the SEQUOIA 2000 benchmark data set, adding new functions to track historical characteristics, and providing queries that evaluate index creation, aggregation, object tracking, overlapping regions, region based data retrieval (including overlapping regions), point and polygon joining, and data updates/changes. This benchmark performs well at evaluating spatio-temporal databases; however, it is limited to non-steaming systems and does not provide any continuous queries that would be typical in a GSMS.

2.2.4. The DynaMark Benchmark

Myllymak et al. developed the DynaMark benchmark to evaluate middleware and database servers that support LBS, but it is applicable to any spatial database since the database benchmark component can stand-alone [15]. The data simulates the movement of mobile users represented by an ID, three-dimensional location, and timestamp, who update their location with an average periodicity. They also provide spatial queries categorized as proximity, K-NN, and sorted-distance. The proximity query finds all the objects within a 2-D or

3-D range. The K-NN queries represent users searching for other users or stationary objects and the sorted-distance query lists the objects based on distances relative to a reference point.

The data could be used as spatio-temporal streaming data; however, the benchmark's focus is on the spatial index performance as the system inserts and updates user locations. Furthermore, the spatial continuous point queries may use a polygon to limit the query range but they do not evaluate the system's capabilities involving a full range of spatial objects such as a streaming polygon overlapping a line segment. A GSMS benchmark would need to expand this and change the focus from indexing evaluation to streaming and spatio-temporal aspects.

2.2.5. COST Benchmark

Like the DynaMark benchmark, the COST benchmark is for LBS systems but is applicable to any spatio-temporal database system [12]; however, this benchmark is only concerned with indexing performance. The benchmark measures the system performance using average I/O and CPU time per index operation when performing updates and queries. The data is simulated objects with periodically updated 2-D locations. The data position queries include finding trajectories through an approximation function that uses historical data and are of three types: timeslice, window, and moving window. The timeslice query returns objects that intersect with a given 2-D rectangle while the window query returns objects that intersect a rectangle during a given time interval. The moving window query returns objects that intersect a trapezoid resulting from connecting rectangles at two different times.

The benchmark's workload generator is important because it must enable index evaluation as opposed to query response evaluation. The generator uses several parameters (number of objects, positioning, velocity skew, update arrival patterns, position accuracy, and workload duration) with an extension to Šaltenis's generator to produce synthetic data. The queries include spatio-temporal aspects; however, since the focus is on indexing, the benchmark does not fully evaluate a STDBMS nor is it concerned with streaming data.

2.2.6. BerlinMOD

The BerlinMOD benchmark, like LRB, simulates spatio-temporal data of moving vehicles on a road network [9]. The Berlin road network serves as a base with vehicles following the principle that drivers spend the majority of travel time between home and work or within a surrounding neighborhood, in this case, a 3-km radius around the home. During the week, the person travels between work and home (*labour trip*) following a typical temporal pattern. At home, they have four-hours of spare time in which they may make an additional trip. On the weekend, the person has two five-hour blocks for additional trips. A trip has a starting location, destination, follows the shortest path, and attempts travel at the speed limit. Acceleration, deceleration, and stop events occur with given probability to represent impediments such as traffic lights and narrow curves.

The benchmark includes two query sets, range and nearest neighbor, which cover five different aspects: object identity, dimension, query interval, condition type, and aggregation. The dimensional and query interval aspects are the most interesting because they define the spatio-temporal aspect of the benchmark. The dimension can be standard, meaning no spatial or temporal aspects, or can have spatial, temporal, or spatio-temporal aspects with the query interval defining the query as point, range, or unbounded (continuous). The query set covers 26 out of the 96 possible combinations since not all combinations are meaningful for evaluation purposes.

Overall, the BerlinMOD is one of the most comprehensive benchmarks for spatio-temporal database systems. The data contains significant and meaningful spatial and temporal components and the queries evaluate the key facilities. However, it is not a streaming benchmark since it treats the data statically. Even the continuous queries use the data statically. This benchmark partly servers as a base for GSMark by providing the moving vehicle concept and similar queries, but GSMark expands to include streaming polygons and a streaming data delivery system.

Table 2.1. Summary of the benchmarks. An X indicates that the benchmark belongs to that category. SP = Spatial Only, ST = Spatio-temporal, S = Streaming.

Benchmark	**SP**	**ST**	**S**	**Data**	**Notes**
Linear Road		X	X	Simulated	Limited data set (straight, parallel roads only, reduced spatial location)
SEQUOIA 2000	X			Real	Spatio-temporal data, but temporal is not evaluated; geared towards earth science problem
3-D Spatio-temporal		X		N/A	Dataset not explicitly explained, but suspected to be similar to SEQOUIA 2000
DynaMark	X			Simulated	Spatio-temporal data but temporal is not evaluated; geared towards index evaluation
COSTS		X		Simulated	Focused on indexing only; not intended for streaming systems
BerlinMOD		X		Simulated/Real	Complete STDBMS benchmark

2.2.7. Benchmark Summary

Table 2.1 summarizes the benchmarks. The Linear Road Benchmark and BerlinMOD stand out as providing the best foundational concepts for a GSMS benchmark. GSMark borrows the vehicle-road network model but extends it to a larger framework and offers data and queries suitable for geo-streaming data. The following section discusses the principles considered for data generation using the vehicle-road network concept.

2.3. The Brinkhoff Data Generator

There are two general approaches to benchmark data. If the benchmark evaluates overall performance, then the data should cover the important system components. If the benchmark is for evaluation under specific conditions, then the data can be limited to those conditions. The previously mentioned benchmarks exemplify this: SEQUOIA 2000 uses data representing typical earth science problems while BerlinMOD uses data to cover all spatio-temporal features. For a GSMS benchmark, the data should be streaming spatio-temporal data that is semantically valid; e.g., the temporal component should have some meaning in the context of moving objects.

The data generator framework by Thomas Brinkhoff encapsulates nine principles of moving objects (Table 2.2) derived from identifying moving object properties exhibited on a restrictive network [7]. Brinkhoff identified two types of data: synthetic and real. Generally, the preference is for real data since it allows performance evaluation under realistic conditions; however, it is sometimes difficult to obtain real data, determine the necessary extent of the data, and how much data to use. In contrast, synthetic data is easy to generate and one can design it to evaluate algorithms and structures under any condition, but at the cost of losing realism. Therefore, one goal is to balance real and synthetic data by using real data but with synthetic data included as necessary. A good example is a road network with vehicles. The vehicles move at different speeds along connected segments and face impediments, such as traffic jams and adverse weather conditions, while attempting the fastest path. In addition, there are different road classes and time-scheduled traffic such as a ferry lines. Data of real road networks is readily available and simulated vehicles can follow basic principles. This example and principles serve as a foundation for the GSMark data generator.

Table 2.2. Brinkhoff's principles of moving objects.

#	**Principle**
1	Moving real-world objects very often follow a network.
2	Most moving objects use a fast path to their destination.
3	Networks consist of classified connections, which have impact on the speed of spatio-temporal objects.
4	The number of moving objects will influence the speed of the objects if a threshold is exceeded.
5	The path of moving objects may change if the speed on a connection is modified.
6	The number of moving objects is a time-dependent function.
7	The speed of moving objects is influenced by a spatio-temporal function, which is independent of the network and of the objects moving on the network.
8	Moving objects belong to a class. This class restricts the maximum speed of the object.
9	Time-scheduled traffic may influence the speed and the paths of moving objects.

CHAPTER 3

GEO-STREAMING DATA GENERATOR OF THE GSMARK BENCHMARK

The GSMark data generator encapsulates Brinkhoff's principles by creating synthetic data sets that incorporate real data. The resulting data consists of a real road network with simulated moving vehicles influenced by traffic jams and weather conditions. The principles are actually relaxed some to prevent creating repetitive evaluation data and to focus on evaluating the streaming capabilities; adding an additional class system of constraints to vehicles already under similar constraints does not significantly add to evaluating a GSMS (geo-stream management system). The data generator can also stand alone and includes a tiger/line shape file converter. The following describes the data generator in more details.

3.1. Real Road Network

As mentioned in Chapter 2, it's better to have both simulated data and real data. Real data can present more realistic condition, but is hard to obtain. Synthetic data is easy to generated, but lacks the realism. It is necessary to make a balance of synthetic data and real data. The GSMark data generator creates moving vehicles on a real-world road network coming from the U.S. Census Bureau tiger/line shape files [8]. The tiger/line shape files are an extensive source but the file format is difficult to interpret by inspection and often requires a conversion tool. There are several converters that are easy to use, have most necessary features, but are not free. Most free converters are not easy to use, and lack important features. Therefore, GSMark designs its own converter to convert shape files. For pragmatic purposes, the data generator will use information from ESRI's GIS (geographic information system) software white papers to convert and clean (remove duplicate data points and road segments) the shape files into a simple text format resembling Feifei Li's dataset of real road networks from Florida State University [13]. The data generator also converts the files to

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
 <Document>
   <name>denton's shp to kml</name>
   <description>transform shp to kml</description>
   <Style id="linefordenton">
    <LineStyle>
      <color>7f00ffff</color>
      <width>4</width>
    </LineStyle>
    <PolyStyle>
      <color>7f00ff00</color>
    </PolyStyle>
   </Style>
   <Placemark>
     <name>denton</name>
     <description>denton's all lines</description>
     <styleUrl>#linefordenton</styleUrl>
     <LineString>
      <extrude>4</extrude>
      <tessellate>2</tessellate>
      <altitudeMode>clampToSeaFloor</altitudeMode>
      <coordinates>
        -96.9895,32.9893,0
        -96.9902,32.9893,0
        .
        .
        .
        -97.0489,33.2669,0
        -97.0489,33.2669,0
       </coordinates>
      </LineString>
   </Placemark>
 </Document>
</kml>
```

Figure 3.1. An example KML file resulting from converting a tiger/lines shape file. The file is easy to read, understand, and usable by Google Maps without special requirements.

the Google KML format allowing non-experts to easily read and visualize them. Google KML files can be executed by Google Earth and Google map. Google Earth have more functionalities that can present different views for users. For example, Fig. 3.1 shows a KML file excerpt for Denton, TX, and Fig. 3.2 shows it overlaying the Denton's Map in Google Earth. Close inspection reveals some errors, such as creeks appearing as roads, but the errors propagate from the original shape file and are removable by hand.

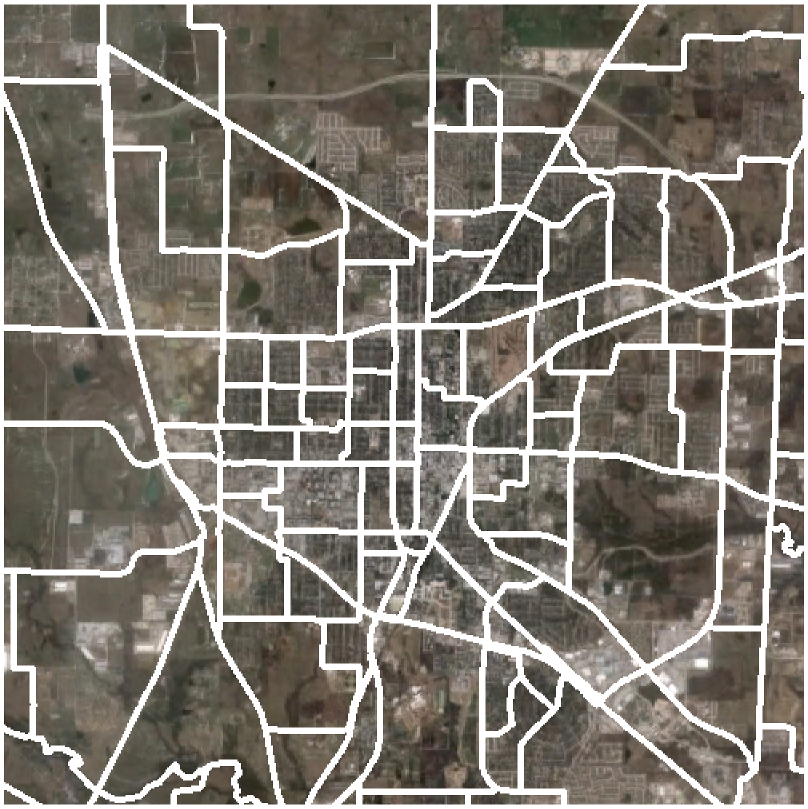


Figure 3.2. The data generator converted the tiger/line data file of Denton, TX, to the KML format. The road network is real enough to host simulated vehicles for evaluating a GSMS.

3.2. Routes and Routing

Each data point defining the road is a network node. The data generator will randomly select, using a uniform distribution, two nodes designated as the vehicle's start and destination nodes. The *A** algorithm then finds the shortest path (route) between these two nodes. The data generator ensures reasonable route distances with an adjustment to the *A** algorithm so that it adjusts the routes for every other vehicle to maintain a running average length (17 miles by default) as it generates vehicles. The *A** algorithm removes road segments from the end of the route if it too long or adds the shortest connecting segment to the end if it is too short. The removal of segments still guarantees the shortest route; however, adding segments do not guarantee the shortest route from the start node the new destination node, but it is still reasonable for evaluation purposes.

Algorithm 3.2.1: Generate Vehicles( )

```
$Let : v = list\ of\ vehicles$
      $end = user\ chosen\ end\ time$
      $nodelist = all\ road\ network\ nodes$
      $maxV = number\ of\ desired\ vehicles$
      $moving = flag\ indicating\ vehicle\ movement$

for $i \leftarrow 1$ to $maxV$
  do
    $v[i].startLocation \leftarrow x \in nodelist$
    $v[i].endLocation \leftarrow y \in nodelist | x \neq y$
    $v[i].route \leftarrow A^*(x, y)$
    $v[i].moving \leftarrow false$
    $temp \leftarrow randomInt(0, 9)$
    if $0 <= temp <= 2$
      then $v[i].time \leftarrow randomTime(0, 600)$
      else if $3 <= temp <= 6$
      then $v[i].time \leftarrow randomTime(600, end - 600)$
      else $v[i].time \leftarrow randomTime(end - 600, end)$

$Note : 600 = 10\ minutes$
```

The user specifies three parameters for the generator: start time, end time, and the total number of vehicles. Each vehicle is unique with only one route in the dataset and will have a random start time between the user specified times. Thirty percent of the vehicles start in the first 10 minutes, 30 percent in the last 10 minutes, and the remaining 40 percent in-between to represent two rush hour peaks. The variations in route and speed means that the number of moving vehicles may be different at any given moment. Algorithm 3.2.1 summarizes this process.

3.3. Assigning Speeds

Speed assignment concerns two items: road segments and vehicles. Real road network segments have a speed limit that vehicles generally adhere to or travel slower. For example, highways often have a 70 mi/h speed limit outside city limits but vehicles will travel below or at this speed to avoid speeding tickets and city streets may have 30 mi/h limits but vehicles often travel slower through intersections and turns. In addition, vehicles on a road segment typically travel the same speed to match the traffic flow. The data generator can use this as an assumption to simplify speed assignment. Each road segment can have a speed limit and any vehicles on the segment will travel at that speed unless impediments exist such as traffic jams or weather conditions. When impediments occur, simply lowering the speed limit forces the vehicles to travel slower. It is not necessary for the data generator to give every road segment a different speed limit to evaluate a GSMS because traffic impediments already factor vehicle speed changes into the dataset. The following two sections will discuss how the benchmark uses traffic jams and weather conditions to impede the traffic through adjusting the speed limit.

Algorithm 3.4.1: trafficJam( )

```
$Let : v = list\ of\ vehicles$
        $nodelist = all\ road\ network\ nodes$
        $maxV = number\ of\ desired\ vehicles$
        $moving = flag\ indicating\ vehicle\ movement$
        $movingV = listofmovingvehicles$
        $completedV = number\ of\ processed\ vehicles$
        $timestamp = current\ time\ being\ processed$
$sort(v)$
$timestamp \leftarrow 0; i \leftarrow 0; j \leftarrow 0; completedV \leftarrow 0$
while $completedV < maxV$
  do
    while $v[i].time = timestamp$
      do
        $movingV[j] \leftarrow v[i]$
        $movingV[j].moving \leftarrow true$
        $updateSegmentCounts()$
        $calcSegmentSpeeds()$
        $i \leftarrow i + 1$
    for $k \leftarrow 1$ to $movingV.size$
      do
        if $movingV[k].moving = true$
          then
            $move(movingV[k])$
            $updateSegmentCounts()$
            $calcSegmentSpeeds()$
            $checkDest(movingV[k])$
    $timestamp \leftarrow timestamp + 1$
    $completedV \leftarrow completedV + 1$
```

$$s_{new} = max(s_{min}, s_o - s_o * ((V_c - V_{max})/V_{max})) \tag{1}$$

The road segment capacity exerts influence on vehicle speeds; for example, a road segment with capacity of 10 but currently hosting 20 vehicles forces vehicles to decrease speed. In other words, this creates a traffic jam the data generator implements using Algorithm 3.4.1. Any vehicles arriving to an impacted segment must slow down and the addition of this vehicle further slows traffic. The proposed benchmark considers this by assigning a capacity and using a slowdown equation to decrease the segment's speed limit automatically when the current capacity exceeds the maximum capacity. The new speed s_n is the original speed s_o minus a proportion of the original speed based on the current capacity V_c and the maximum capacity V_{max}. As is, this equation can produce a speed less than or equal to zero, which would violate the traffic model since vehicles cannot have negative speeds and a speed of zero would permanently halt any vehicles on the segment. Research has shown that in heavily congested traffic, there is generally a minimum speed s_{min} that vehicles move [3]; therefore, the equation (Eq. 1) takes the maximum of either the minimum speed (by default 5 mi/h) or the calculated speed. The speed returns to the original value once the capacity equals or falls below the maximum.

There are two items to note. First, assigning different maximum capacities to the road segments is not necessary because the variation in current capacity is sufficient for evaluating a GSMS. This also coincides with the previous statement about directly assigning different speeds to road segments—it is not necessary since this impediment creates the speed variations. Second, the traffic jam is not a defined object in the data set. Instead, it is implicit in the distance between the vehicle coordinates.

3.5. External Events

External events, e.g. weather conditions, may affect vehicle speeds. The data generator creates moving polygons to represent such events based on user-defined area size, speed,

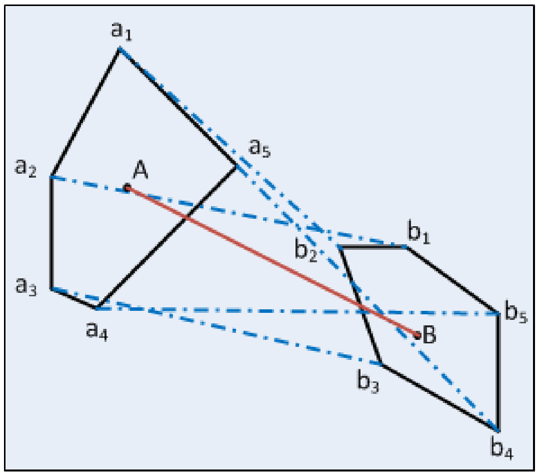


Figure 3.3. Polygon A moves from its current location to polygon B, changing shape as each point travels along the straight-line trajectory (blue dash lines) to a point in polygon B. Starting with a_1, the data generator mapped the points using the minimal distance and each point travels at a relative rate to the center point to ensure that all points arrive at their corresponding points at the same time.

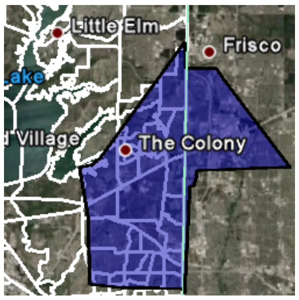
number of desired polygons, and number of polygon sides. It uses a simple method to generate the polygon. First, it selects a random starting location to represent the polygon's center and another for the ending location. It then creates a rectangle around the starting center point to serve as a polygon template. The dimensions of the rectangular are selected at random but limited to ensure the area matches the user's chosen size. Third, it conceptually divides the rectangle into four quarters and then randomly selects points of varying distances (maximum of about 17 miles) from the edge to create approximately one-fourth of the desired sides per quarter. For example, a request for a 200-face polygon causes the generator to select and connect 50 random points per rectangle quarter.

The polygon overlays the road network and moves at the user defined speed, changing shape, until it reaches the ending location. The benchmark simulates this by creating two polygons, one at the starting location and one at the ending location. The data generator maps the polygon points from the starting polygon to the ending polygon based on the minimum distance between the points of the two polygons. Alternatively, one can view this as polygon A morphing and moving until it reaches the shape and location of polygon B. Consider the two polygons in Fig. 3.3. There are five points in polygon B that point a_1 can map to from polygon A. In this example, the minimum distance from a_1 is to b_2 so the process maps those points and removes them from the remaining mapping process. Next, a_2 maps to b_1 followed by a_3 to b_3, etc. Each point of A moves along the straight line connecting it to its corresponding point of B at a relative speed such that all points meet their corresponding points at the same time.

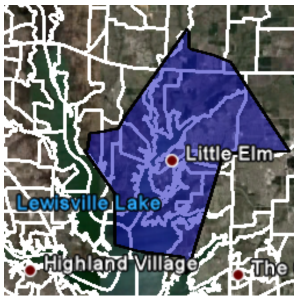
The moving polygon triggers a 70 percent decrease in the speed limits of road segments beneath it, thereby slowing the vehicles; once it passes, the speed returns to normal. The area affected is not the full polygon area however. The original rectangle serves as non-strict boundary for the affected area to reduce edge detection complexity as it moves. It is non-strict because the slowdown only affects full road segments. This simplification does not affect the benchmark's ability to evaluate a system because the data still exhibits the moving object properties and the polygon's primary purpose in evaluating spatio-temporal facilities is unaffected.

3.6. Output Data

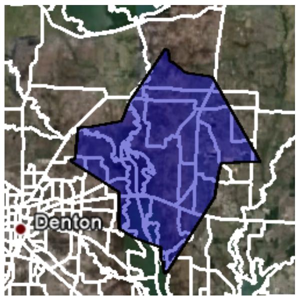
The data generator outputs the resulting data into both KML and TXT files. There are two groups of files: the road network dataset and the vehicle/polygon dataset. The road network dataset includes a KML file containing all nodes and edges and TXT files of network nodes (.node filename extension) and edges (.edge filename extension). The vehicle-polygon dataset has three files including a TXT file with all vehicle and polygon data and separate KML files for vehicles and polygons. Table 3.1 is a vehicle TXT file excerpt showing each vehicle with a unique identification, timestamp, and location. Fig. 3.5 shows the KML vehicle-polygon file containing 136 vehicles for 90 minutes on the California road network. The white lines trace the vehicle trips and the blue squares indicate their final destination.



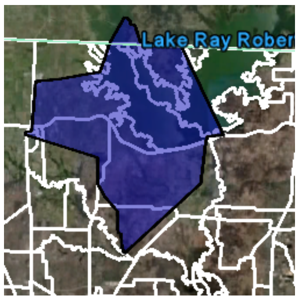
(a) Time 0



(b) Time 1



(c) Time 2



(d) Time 3

Figure 3.4. An example 200-face polygon moving northwest over the Denton, TX road network. At each time interval, the polygon has changed position and shape.

With only a few vehicles, the trips cover a significant portion of the road network and indicate that the data generator does create reasonable route distances.

Table 3.1. An example of the data generator's vehicle TXT output file.

Time Stamp	object ID	Object Position
2011-1-31 17:47:43	0	-117.491379 34.809978
2011-1-31 17:47:53	0	-117.465027 34.735233
2011-1-31 17:48:10	0	-117.421669 34.604073
2011-1-31 17:48:13	0	-117.41745 34.580677
2011-1-31 17:48:15	0	-117.400993 34.580963
2011-1-31 17:48:16	0	-117.400291 34.576096
2011-1-31 17:48:22	0	-117.400414 34.522202
2011-1-31 17:48:33	0	-117.401802 34.428623
2011-1-31 17:48:44	0	-117.492363 34.428772
2011-1-31 17:48:52	0	-117.563164 34.428162
2011-1-31 17:48:54	1	-120.824333 40.101688
2011-1-31 17:48:56	0	-117.597809 34.427448
2011-1-31 17:48:58	0	-117.612152 34.42421
2011-1-31 17:48:59	1	-120.808983 40.063011

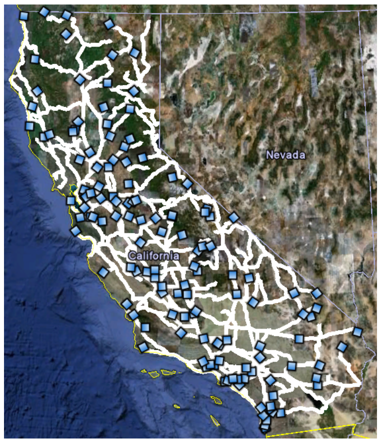


Figure 3.5. 136 simulated vehicles on the California road network for 90 minutes. Each blue square represents the vehicle destination and the while lines trace all the routes by the end of the dataset.

CHAPTER 4

WORKLOAD SIMULATOR OF THE GSMARK BENCHMARK

4.1. The Purpose of the Workload Simulator

First, the purpose of the workload simulator is to deliver a data stream to a GSMS (geo-stream management system). To evaluate the performance of a GSMS, we have to have a data stream that can be delivered to a GSMS. By using the data generator, we can have the data set ready for use, but the data is not delivered to a GSMS as a data stream. Therefore, we need the workload simulator to deliver the data records as a data stream. Different GSMSs may accept a data stream in different ways. To make the benchmark work for different GSMSs, we do not restrict the data delivery to the format of a single GSMS. Instead, we create a buffer in memory and deliver the data stream into that buffer. Any application that is able to access the buffer can share the data stream. Fig. 4.1 shows how the data stream flows into a GSMS.

In addition, the workload simulator should be able to not only simply deliver the data stream, but also simulate a real sensor network's complicated behaviors. This can make the simulated data stream more realistic. Therefore, the GSMark's data generator and workload simulator work together to simulate a sensor network that has remote sensors on each moving object. The simulated sensor network works as below: 1) each sensor can get the longitude, latitude and timestamp of a moving object when a record is generated, 2) the sensor network sends data each time unit (by default we use 1 second as a time unit), 3) several sensors (the exact number of sensors is defined by the users) are used to detect the edges of the moving regional events (such as a fire and a storm) and send the data records in each time unit. The time unit, number of moving objects, and number of sensors for moving regional events, strongly impact the performance of the benchmark. For example, the shorter the time unit

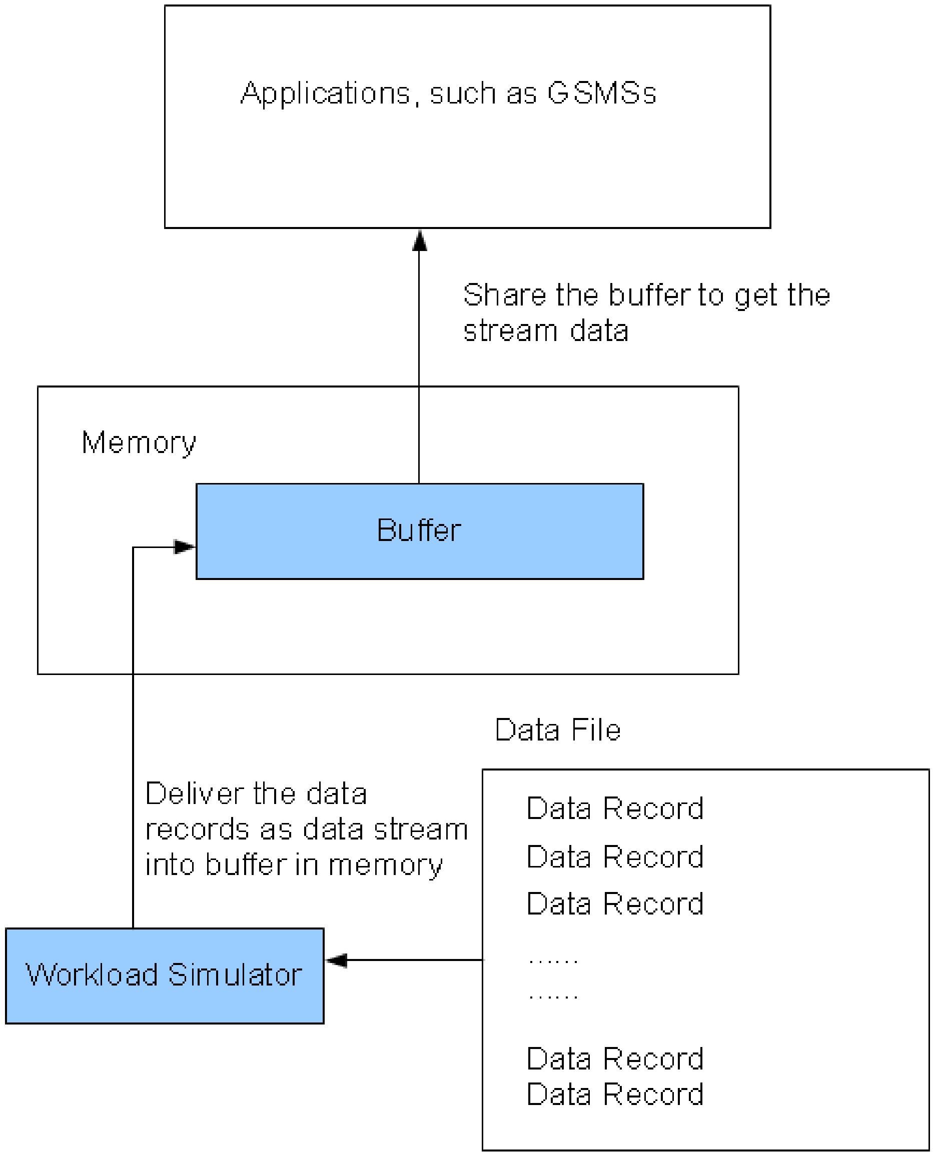

Applications, such as GSMSs
Share the buffer to get the stream data
Memory
Buffer
Data File
Deliver the data records as data stream into buffer in memory
Data Record
Data Record
Data Record
......
......
Data Record
Data Record
Workload Simulator

Figure 4.1. Data flow of workload simulator.

is, the slower the benchmark performs. The benchmark will also be slowed down when it has more moving objects since it needs to find out each object's position during every time unit. For a moving regional event, it is represented by a polygon. We use the nodes of the polygon to represent the sensors that report the geo-referenced information of the event. For example, a special event that is monitored by 200 sensors is simulated by a polygon composed by 200 nodes. Therefore, if we use more sensors to report the geo-referenced information, we need to check more sensors in each time unit. In other words, it means we need to check more nodes' positions each time unit which slows the benchmark dramatically. More details of the benchmark's performance's issue are discussed in chapter 6. Fig. 4.2 shows the simulated sensor network.

Furthermore, in some cases, the sensor network may behave abnormal. For example, the network may not be able to always deliver the data records at the same speed. If a moving object is in a bad signal area, the network's speed may be much slower than during normal conditions. Therefore, the workload simulator should be able to simulate the abnormal situation of a sensor network. This is further explained in the next section.

In conclusion, the workload simulator's purpose is to deliver the data records into the GSMS as a data stream and work with data generator to simulate a sensor network's realistic behaviors.

4.2. Components of The Workload Simulator

To simulate the data delivery steps, we need to have two components. One component to send data from the data sets that are generated from data generator; another component to receive the data that is sent from the first part and make the data stream accessible for the GSMSs. Therefore, the workload simulator has a *sender* and the *receiver*. The *sender* executes on a local machine but simulates a sensor by reading data object records from the data file and sending them to the *receiver* along with signal messages indicating the beginning and the end of the data stream. The *sender* uses the time stamps to simulate time. At the start of a *sender* cycle, the *sender* reads all the data for timestamp T_1 and sends it over a TCP/IP connection to the *receiver*. Then it sends all the data for timestamp T_2, and so on until the end of the dataset.

The *sender* also simulates sensor network delays. When it reads an object's record with timestamp T_i, there is a probability (50% by default) that the record will delay by a random amount of seconds d (0-5 seconds by default). If the delay is greater than one second, then

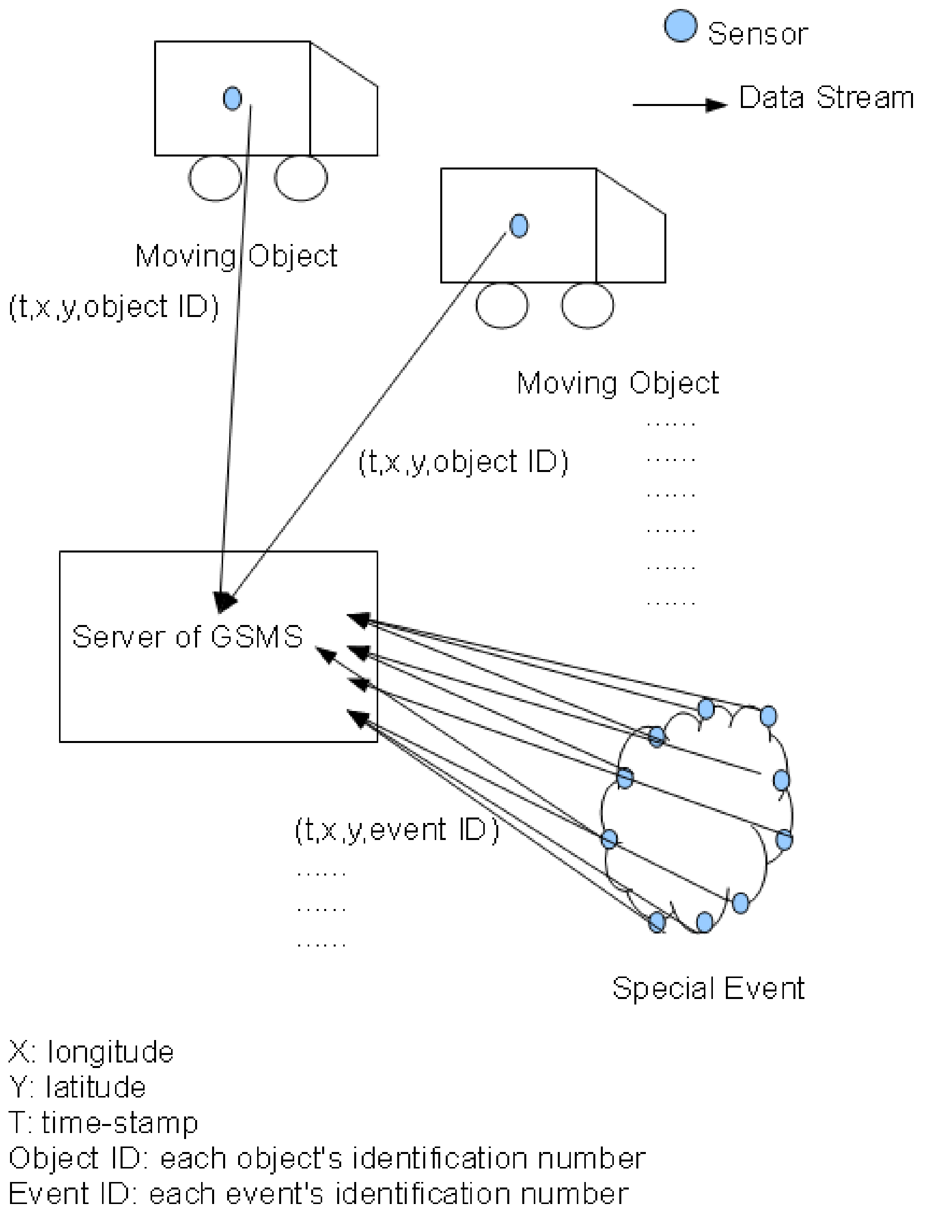


Figure 4.2. Simulated sensor network.

the record transmits at time T_i+d, creating an out-of-order sequence. In addition, the record also has a probability (50% by default) of delay by a random number of records. This creates an unknown time delay and, if large enough, the record will end up at the end of the dataset. The *sender* can delay the record by both time and record count; for example, it may delay by 5 seconds and then an additional 11 records. Once it is in its new sequence location, the sender may delay it again when it reads the record for a second time. In extreme cases, a record could propagate to the end of the dataset.

The *receiver* executes in an infinite loop with a *listener* waiting for the signal from the *sender* to receive data. Once signaled, the *listener* performs a reading operation that feeds the incoming data into a 101-element circular FIFO memory buffer until it reads the *sender*'s end of data signal. After reading this signal, the listener begins waiting for the next incoming data signal. The GSMS can directly use the resulting data in memory if shared, or the data can reside in a file or database.

In the following sections, the details of the *sender* and *receiver* are discussed.

4.2.1. Sender

The main purpose of the *sender* is to simulate the sensor network's data stream delivery. Therefore, we study the sensor network's realistic process of data stream delivery. The process has several important features that may impact the performance of a GSMS.

First, the most basic feature of the sensor's data stream delivery process is that the sensors send a regular data stream to a GSMS server. We assume a situation in which we have sensors on different moving objects and the sensors send the objects' geo-referenced information to a GSMS server every second. In this situation, the sensors will generate a data stream that continues feeding each moving object's geo-referenced information into a GSMS every second. We also assume the geo-referenced information is a object's longitude and latitude, and we do not consider altitude since we assume that each moving object moves on the ground.

Second, the delivery of the data records in the sensor network may be delayed. In a perfect network situation, all the data should arrive at the server on time. However, the network may be influenced by many factors, such as weather, obstructions between sensors and servers, and other signals interferences. Therefore, when a sensor sends a data record to a server, the delivery progress may be delayed since the network may be impacted by many factors.

Third, the data records may not be delivered in the order of their time stamps. In the entire sensor network, different sensors may be influenced by different factors making them perform the delivery progress differently; for example, one sensor's location is in a bad signal area. The data delivery speed of this sensor is slower than other sensors that are in good signal areas. This causes the data of this sensor to always arrive at the server later than the data of other sensors. If the delay is long enough (longer than 1 time unit), it may cause the disorder of the data records. To continue this example, let us assume that a sensor S1 is in a bad network situation. S1 sends a data record that has 10:14:25 AM as its timestamp. The other sensors send data records at the same time. If other sensors's delivery can be processed in real time, when other sensors' data records reach the server, it is 10:14:25 AM. However, the S1's data record is still in the delivery process since its network is slower. Let us assume that S1 has a 2 seconds delay. Its 10:14:25 AM data record reaches the server at 10:14:27 AM. Therefore, S1's 10:14:25 AM data record reaches the sever after the other sensors's data records that have timestamps of 10:14:26 AM. In the data stream, it will cause a disorder of data records.

Overall, we need to have the work-load simulator's *sender* to simulates the normal data delivery process, the delay situation, and the disorder situation.

4.2.1.1. *Normal data delivery process.* To simulate the regular data stream delivery, we first select the data records that have the same timestamps. Therefore, we need to search from the beginning of the data set that is generated by the data generator. If a data record's timestamp is as same as the first data record's timestamp, we select it. We finished selecting until we find a data record that has the timestamp later than the timestamp which we are selecting.

For example, we have a data set that has the first data record of the timestamp 11:23:34 AM. We select the first data record, and check the data record that is next to the first data record. If the second record's timestamp is as same as the first record, we select the second one. Let us assume the timestamps from the first data record to 13th data record are same. The timestamps of the 14th data record is 11:23:35 AM. Therefore, we need to keep selecting the data records from the first one to the 13th data record. We finish selecting right before the 14th data record.

Then we add messages at the beginning and the end of the selected group of data records. The message before the beginning of the data records should be the "Hello" message. It will tell the *listener* of the *receiver* that the *sender* is ready to send the data records. We will discuss the *receiver* in the next section. The message after the end of the selected data records should be the "Bye" message. The "Bye" message tells the *listener* of the *receiver* that the *sender* finished sending the selected data records.

After we selected the data records and added the two messages, we send the set of the selected data records to the receiver through a Java socket. The Java socket is a built-in class that is easy to use for connecting two machines through a TCP/IP connection. We assume the *sender* is running on one machine, and the *receiver* is running on another machine connected to by a Java socket. The data records are sent from the sender machine to the receiver machine through the connection. After the sender finishes sending the set of the selected data records, it waits until the next second to select the data records that have the next time stamps. Fig. 4.3 shows the progress of how the *sender* works in this step.

When we simulate the regular data delivery, we have to be aware of the data records that have previous timestamps. When we use work-load simulator to simulate a realistic sensor network, the delay and disorder situations may be generated. Therefore, when we select the data records of a timestamp, there may be some data records that have previous timestamps right after them or in-between. For those data records, when we simulate regular data delivery process, we include them in the selected data set and continue selecting next data records until we find a record that has a timestamp that is later than the timestamp we are selecting. In this way, we can deliver the data records which have the previous

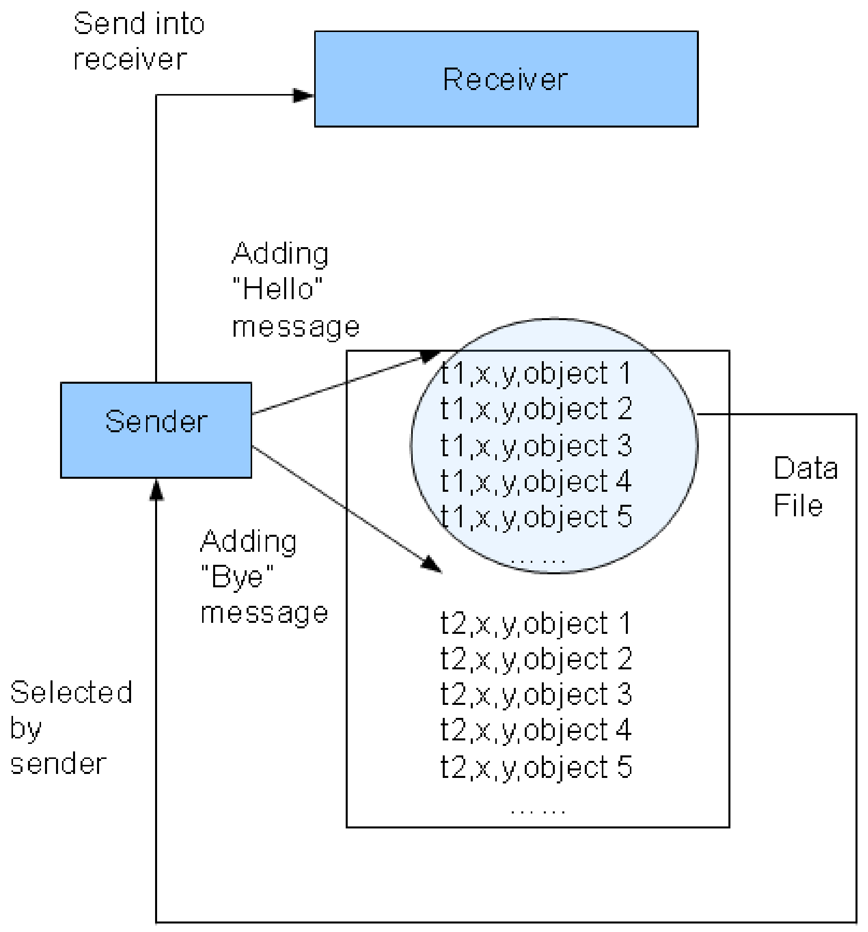


Figure 4.3. Sender's working process of sending regular stream.

timestamps to the server after the data records which have later timestamps.Therefore, the disorder situation is simulated.

For example, suppose we have a data set that has the first data record with timestamp 8:13:41 AM, the second data record with timestamp 8:13:40 AM, the third to the 13th data records with timestamps 8:13:41 AM and the 14th data record with timestamp 8:13:42 AM. The second data record is generated earlier than the first data record since it has a earlier timestamp. It is placed after the first data record because the *sender* simulates the disorder situation. When we are doing a regular data delivery, after we selected the first data record, we check the second one. Since the second data record has the earlier timestamp, we select the second one, and check the third one. Because, from the third data record to the 13th data record, the timestamps are as same as the first one, we select the data records until the 14th data record that has a later timestamp. After we place “Hello” and “Bye” messages before the beginning and after the end of the selected data set, we deliver the data set to the server. Therefore, the second record with previous timestamp will reach the server after the first record. The disorder situation is simulated completely.

The regular data delivery is the most important and basic feature of the work-load simulator. To make the data stream more realistic, the workload simulator also has to simulate the delay and disorder situations.

4.2.1.2. *Simulate delay.* The simulations of delay and disorder are related. If, because of a delay, a data record is delivered to the server after any other record that has a later timestamp, a disorder will occur. Usually, since we set default time unit as 1 second, if a delay lasts more than 1 second, it may cause a disorder. Through the simulation of delay and disorder, when we look the data records in a data stream at time T1, we may find not only the data records of timestamp T1 but also the data records of the previous timestamps. Although the delay and disorder are related, since we use different parameters and equations to compute the delay and disorder, we discuss them separately.

Before we introduce how the *sender* simulates delay, we need to define several parameters. 1) Delay possibility: we use this parameter to determine the possibility of delay. User can set this number as any number between 0 to 1. When we simulate a delay, the *sender* will use the uniform distribution to randomly generate a number between 0 and 1. If the generated number is less than the delay possibility, it will trigger the delay simulation. Since the *sender* uses uniform distribution to generate random numbers, every number has the same possibility to be generated. For example, if we set the delay possibility as 0.4, it has 40 percent possibility to be selected. 2)Max delay: the max duration of delay. 3) Delay duration: the duration of a simulated delay. The *sender* will randomly generate a number from 0 to 1. Then it will use this number to multiply the max delay. The result will be the simulated delay duration. Equation 2 shows the formula of computing the delay duration. 4) Deliver time: the time that a data record should be delivered to the server. When a data record is delayed, after we compute the delay duration, we use its timestamp to plus the delay duration to get the deliver time. The delayed data record is delivered to server at the deliver time. Equation 3 shows how to compute the deliver time.

$$DelayDuration = MaxDelay * random[0, 1] \tag{2}$$

$$DeliverTime = DelayDuration + timestamp \tag{3}$$

For example, a delay is simulated in the following way. We randomly generate a number in the range of 0 to 1. If we set the delay possibility as 0.5, and the random number is in the range of 0 to 0.5, a delay will happen. For the duration of a delay, we also randomly generate a number from 0 to 1. If the delay happens, we use the max delay to multiply the random number to get the delay duration. If the random number is 0.3, and the max delay is 20 seconds, the delay duration is 20*0.3=6 seconds. After we have the delay duration, we use the data record's timestamp to plus the delay duration to get a deliver time. If the data record's timestamp is 10:23:31 AM, the deliver time will be 10:23:31+6 seconds=10:23:37 AM. Therefore, the data record will be delivered at 10:23:37 AM.

4.2.1.3. *Simulate disorder.* For the disorder simulation, we use the similar strategy as the delay simulation. There are several parameters we need to introduce. 1) Disorder possibility: we use this parameter to determine the possibility of disorder. The *sender* can randomly generate a number. If the random number is less than the disorder possibility, the *sender* will simulate a disorder situation. 2) Max disorder range: the max range of a disorder. 3) Disorder range: the range of a disorder. To get this number, we use the max disorder range to multiply a random number. Equation 4 shows how to compute the disorder range. 4) Disorder index: the index where the data record will move to. We can get it by using the data record's index plus the disorder range. Equation 5 shows how to compute the disorder index.

$$DisorderRange = MaxDisorderRange * random[0,1] \tag{4}$$

$$DisorderIndex = DisorderRange + PreviousIndex \tag{5}$$

For example, the *sender* simulates a disorder by using following steps. First, we set up a number for disorder possibility. Then we random generate a number from 0 to 1. If the random number is less than the disorder possibility, a disorder will happen. For the range of disorder, we need to set up a parameter of the max disorder range. Then we generate a random number from 0 to 1. By using the random number to multiply the max disorder range, we can get the disorder range. Based the disorder range, we can compute the disorder index. If the data record is the 15th record in a data set, and its disorder range is 5, the disorder index will be 15+5=20. After we got the disorder index, the *sender* will delete the data record at the original place and insert it to the new index. Therefore, the *sender* will delete the 15th data record, the insert the contents of the 15th data record before the 20th data record. After all these steps, a disorder situation is simulated.

Overall, the simulations of disorder and delay make the behavior of a sensor network more realistic.

4.2.1.4. *Sender parameters's setting.* Table 4.1 using several examples to summarize how we define the parameters of the *sender*, and how they work.

4.2.2. Receiver

The *sender* performs the delivery of the data records while simulating the delay and disorder situations. After the data is sent from the *sender*, it is delivered to the *receiver*.

The *listener* part in the *receiver* always listens to the *sender*. After the *sender* sends a data stream, the *listener* will tell the *receiver* to receive the data stream and forward the

Table 4.1. An example set of the sender's parameters' setting.

parameter:
1. delay possibility: e.g. DelayPossibility=0.5 We generate a number from[0,1]. Assume we get 0.23. $0 < 0.23 < 0.5$ Then a delay happens.

2. max delay:
e.g. MaxDelay=20;
We generate a number a from (0,1]. Assume we get 0.3.
DelayDuration=a*maxDelay=0.3*20=6 seconds

3. disorder possibility: e.g. DisorderPossibility=0.5 We generate a number from[0,1]. Assume we get 0.31. $0 < 0.31 < 0.5$ Then a disorder happens.
4. disorder range: e.g. MaxDisorderRange=5 We generate a number from (0,1]. Assume we get 0.4. DisorderRange=5*0.4=2 Then we change the record's index and put in to the index+2.

data stream into a memory buffer. Fig. 4.4 shows the data flow from a *sender* to a memory buffer.

The "Hello" message and "Bye" messages in a data stream are working as the signals that trigger the *listener*'s actions. When a *listener* receives the "Hello" message, it will know that the *sender* is sending a data stream to the *receiver*. Then the *listener* will forward the data records of the stream to the main part of the *receiver*. When the *listener* receives the

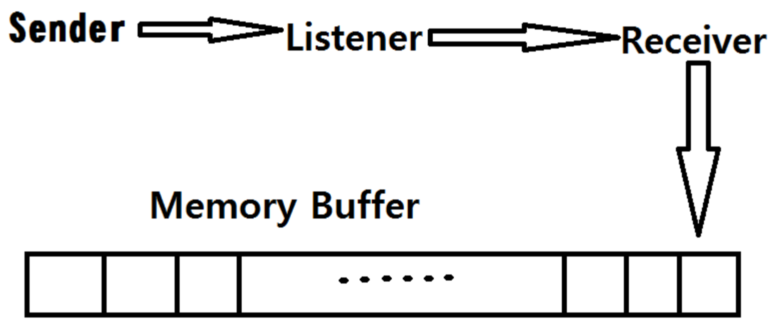


Figure 4.4. Workload simulator's data flow.

"Bye" message, it will know that the *sender* finishes sending a data stream. The *listener* will begin to wait for the next "Hello" message.

After the *listener* forwards the data records to the *receiver*, the *receiver* will create an array in memory. The array size is fixed. By default, we set the size of the array as 101. The *receiver* delivers the data records into the array one by one. Once the array is full, the *receiver* will rewrite the array from very beginning. For example, for the array size of 101, the *receiver* first writes the data records to the array from index 0 to 100. After it fills the array element of index 100 with a data record, it will write into the element 0 again.

The memory buffer stores the stream data temporarily. If any GSMS want to input the data stream, it can share the same memory location of the buffer.

Therefore, by using the *sender* and the *receiver*, the work-load simulator can successfully simulate a realistic sensor network.

After we have the data records that are generated by the data generator and delivered as a stream by the work-load simulator, the benchmark needs to use the query set to evaluate different GSMSs. The query set will be discussed in the next chapter.

CHAPTER 5

QUERY SENTENCES OF THE GSMARK BENCHMARK

5.1. The Goals of Queries

The goal of the queries is to use the data and queries that are common to the most GSMSs (geo-stream management system) to evaluate the GSMSs' performances. In particular, the benchmark data should consist of three attributes: time, object identification, and position, with queries focusing on those attributes and typical operations such as selects and joins. The following are queries designed for the proposed benchmark that cover selects, joins, aggregations, and historical trajectories, and represent realistic situations that a GSMS may encounter.

5.2. Schema Used for Queries

Each query is in both English and a geo-stream SQL with data types developed by Zhang et al. [11]. Table 5.1 gives the schema used for these queries.

Table 5.1. Schema for the benchmark queries. A data type prefixed with *s* indicates a streaming version of the data type.

vehicles(ID: int, location: spoint)
nodes(ID: int, location: point)
rain(ID: int, extent: sregion)
floods(ID: int, extent: sregion)
forestFires(ID: int, extent: sregion)
counties(name: sting, extent: region)

5.3. Measuring for Query Performance

Measuring the query performance may require more than a single execution and should incorporate multiple parameter combinations due to peculiarities that may result from randomization, small data sets, or peculiarities with the data. For example, Query 10 is more interesting if it returns data instead of an empty set resulting from 5% being too low. As suggested by BerlinMOD, 100 parameter combinations are often adequate and the number of execution should vary with the query. If the query is simple, a single execution may not accurately represent the result; if the query is complex, only a few executions may be necessary.

5.4. Query Sentences

Query 1: Find a vehicle's position given a specific vehicle identification number (VID).

```
SELECT a.location[now]
FROM vehicles a
WHERE a.ID=VID
```

This relatively simple, but common geo-streaming query, evaluates the system's ability to access streaming point data for a specific object and return its location. The *now* qualifier refers to the current system time, thus the query produces a single static result using the time at which the user issued the query.

Query 2: Find a vehicle's position given a specific vehicle identification number (VID) and time t.

```
SELECT a.location[t]
FROM vehicles a
WHERE a.ID=VID
```

Similar to Query 1, this also finds the vehicle location from streaming data but at a specific time, requiring the system to search historical data of the data stream.

Query 3: Find all vehicles that have passed a specific road network node in past 2 hours.

```
SELECT at(a.location[past 2 hours], b.location)
FROM vehicles a, nodes b
```

This continuous query evaluates the system using a join between a data stream and static data. The vehicles locations are streaming points while the road network nodes are static points. The *past 2 hours* qualifier creates a two-hour window from which the system returns vehicle data, including spatial location.

Query 4: Find all vehicles that rain is currently influencing.

```
SELECT a.ID
FROM vehicles a, rain c
WHERE inside(a.location[now], c.extent[now])
```

This query evaluates joining point and polygon data objects involving streams—both vehicles and rainstorms are streaming data. The system must join the two streams and calculate the intersection to find those vehicles that lay within the streaming rainstorm region.

Query 5: Find rainstorms that currently overlap with a flooded region.

```
SELECT c.ID, d.ID
FROM rains c, floods d
WHERE intersect(c.extent[now], d.extent[now])
```

This query evaluates a system's ability to join streaming data by searching for overlapping moving polygons. The query appears simple, but the mechanics require several calculations and operations that a typical mature GSMS would perform and it covers both spatial and temporal aspects.

Query 6: Find how many vehicles are currently close (within 5 miles) to a node.

```
SELECT COUNT(*)
```

```
FROM vehicles a, nodes b
WHERE b.ID=NodeID
      AND distance(b.location, a.location[now])<5
```

This aggregation query requires the GSMS to count of all the moving vehicles that are within in 5 miles to a road network node. The query requires joining stream and static data, but more importantly, it uses a compound predicate that requires a distance calculation and filter before the count aggregation.

Query 7: Find how many forest fires happen in a given county during the past 2 years.

```
SELECT COUNT(*)
FROM forestFires e, counties f
WHERE intersect(e.extent[past 2 years], f.extent)
      AND f.name=county
```

Similar to the previous query, this query is also a count aggregation query using stream and static data; however, this one is a continuous query using a window on the streaming data. In addition, the intersection operation requires a more extensive calculation than the previous query because of the spatial extents.

Query 8: Find a car's trajectory for the past 30 minutes.

```
SELECT trajectory(a.location[past 30 minutes])
FROM vehicles a
WHERE a.ID=VID
```

This is a continuous aggregation query that returns information from a single stream using a window and trajectory function. This is another common query for a GSMS and is particularly useful for location-based services.

Query 9: Find counties affected by a forest fire (FID) during the past 2 hours.

```
SELECT f.name
FROM forestFires e, counties f
WHERE duration(at(intersect(e.extent[past 2 hours],
      f.extent), TRUE)) <> 0 AND e.name = FID
```

Similar to previous queries, this continuous query will join streaming and static data, but this query finds the historical trajectory of a moving polygon in terms of the static regions it overlaps with as it moves. The predicate is also complex, covering several streaming operations and requiring comparisons.

Query 10: Find a forest fire that currently influences more that 5% of a county.

```
SELECT e.ID
FROM forestFires e, counties f
WHERE area(intersection(e.extent[now], f.extent))
      > 5\% * area(f.extent)
```

This query evaluates the system involving the intersection of streaming and filters static polygons by calculated results within the predicate. Specifically, it requires finding the intersecting area of moving regions with static regions, computing the overlap coverage, and selection using a comparison operator.

Query 11: Find whether a road network node is in a county or not.

```
SELECT b.ID
FROM nodes b, counties f
WHERE  inside(b.location, f.extent)
```

This is a query that handles the join of 2 static tables. A traditional STDBMSs can do join of a regional data and a point data. For a GSMS, it should have the same functionality to proceed the operations of static data.

Query 12: Find objects that are in a county(CountyName) at a certain time(t).

```
SELECT a.ID
FROM vehicles a, counties f
WHERE  inside(a.location[t], f.extent)
    AND f.name=CountyName
```

This is a query that needs to do the join of the stream data and the static data. The stream data records are the moving points. The static data is the polygon data. The query first selects the the county that is named CountyName. Then it needs to find the spatial information of the county. Finding extension information of a static region is a common functionality for GSMSs and STDBMSs. Moreover, since the cars' locations are coming into the system as a data stream, it tests the system for searching the moving points' spatial information when a particular time is given. And it has to do a join for the point location and the polygon's location to find out whether a car is in the county at time t or not.

Query 13: Count how many vehicles that have passed the node b in the past 2 hours.

```
SELECT COUNT *
FROM vehicles a, nodes b
WHERE at(a.location[past 2 hours], b.location)
```

This query is similar to query 3, but this query tests a GSMS's ability to do the counting operation. Almost all the traditional DBMSs have the functionality for aggregations, such as sum, count, average and so on. Count is one of the most frequently used aggregation. The query requires a GSMS to search for the locations of a moving point in a given time interval. It is one of the most important features of a GSMS. A moving object's trajectory in a certain time interval is usually very interested in a geo-stream. The query also requires the join for the static points and the moving points.

Query 14: Find the distance between 2 cars(CID1, CID2) at a given time t.

```
SELECT distance(a.location[t], b.location[t])
```

```
FROM vehicles a, vehicles b
WHERE a.ID=CID1 AND b.ID=CID2
```

This query performs a join of the vehicles' data. First, it needs to find one car that has the ID of CID1 and another car with the ID of CID2. For the 2 cars, the query requires a GSMS to get their spatial information based on their temporal information. A GSMS should return the 2 cars' locations of the time t. After that, the query requires a GSMS to compute the distance between the 2 locations. It tests a GSMS's ability of dealing with the spatial information of 2 point data in a stream.

Query 15: Find all the cars that have been closed (within 5 miles) to each other in past 2 hours.

```
SELECT a.ID, b.ID
FROM vehicles a, vehicles b
WHERE distance(a.location[past 2 hours],
    b.location[past 2 hours])<5
```

Similar as query 14, this query performs a join of the vehicles' data. First, it needs to find every car's location and compare the location with all the other cars' locations. The query needs a GSMS to continue doing the first step for all the data records that have previous 2 hours timestamps. If there is a timestamp that the distance of 2 cars is less than 5 miles, then we select the 2 cars as a pair.

Query 16: Find which car has run the longest distance in past 2 hours.

```
SELECT a.ID
FROM vehicles a
WHERE distance(a.location[now],
    a.location[now-2 hours])=
    MAX(distance(a.location[now],
    a.location[now-2 hours]))
```

This query finds a car's location now and 2 hours ago. Then it compares the locations to get the distance. The query requires a GSMS to find the distance for each car and compare the distances to find the max one. The car that has the max distance is selected. This query is similar to the query 15, but they focus on different aspects. Query 15 compares 2 moving points' positions in a stream while this query compares each moving point's position to itself of a previous timestamp.

Query 17: Find whether there were 2 cars met each other in past 2 hours.

```
SELECT a.ID, b.ID
FROM vehicles a, vehicles b
WHERE at(a.location[past 2 hours],
     b.location[past 2 hours])
```

Similar as query 15, but this query does not try to get the distance of 2 locations. This query compares whether 2 locations are same or not. For each timestamp, it needs to compare a car's location with all the other cars' locations. If the locations are same, we select the 2 cars as a pair. The query needs to continue doing this step for all the cars' every timestamp in past 2 hours. If 2 cars have been the same place in the past 2 hours, the query lists the IDs of the 2 cars. Query 15,16 and 17 give user several options to test a GSMS's ability of comparing the moving points' positions in a past time interval of a data stream. A GSMS has to be able to query the historical data records as well as handle the real-time stream's current data records.

Query 18: If the 2 cars (CID1, CID2) were in a same county at a timestamp (t), list the county name.

```
SELECT f.name
FROM vehicles a, vehicles b, counties f
WHERE a.ID=CID1 AND b.ID=CID2 AND
```

```
    inside(a.location[t], f.extent)
    AND insdie(b.location[t],f.extent)
```

Similar as query 12, This query tests a GSMS's ability to handle join of moving points and a static region. First it needs a GSMS to select the car CID1 and the car CID2 in a stream. Then the GSMS needs to select their locations of the timestamp t. After the GSMS selects the 2 cars' locations, it needs to join the points and the region's extent information to compare whether a moving point was in the region at time t or not. If the 2 cars were both in the region (county f) at time t, the county's name will be selected.

Query 19: If a car has been stayed in a same county for the past 2 hours, choose this car.

```
SELECT a.ID
FROM vehicles a, counties f
WHERE f.name=CountyName AND
    duration(inside(a.location[past 2 hours],
     f.extent))=2 hours
```

First, the query chooses a county based on its name. In addition, it needs to select a car's locations in last 2 hours. Then the GSMS needs to compare the car's locations and the county's extent region. If there is any car has been stayed in the county for 2 hours, we choose the car's id.

Query 20: From now on, if a car stays in a same county for 2 hours, select the car's id.

```
SELECT a.ID
FROM vehicles a, counties f
WHERE f.name=CountyName AND
    duration(inside(a.location[2 hours from now],
     f.extent))=2 hours
```

The query is similar to query 19, but it requires a GSMS which has the ability to do the query for future data. It first chooses a county based on its name. Then it needs to check every car's trajectory from now to 2 hours later. Some GSMSs (such as Auraro) will place a query node in the data stream. When the stream flows through the node, it will select all the cars' locations and store in the node. Then the GSMS needs to compare the cars' locations and the county's extent region. If there is any car has been stayed in the county for 2 hours, we choose the car's id. A GSMS should be able to query a moving object's future trajectory as well as historical trajectory.

CHAPTER 6

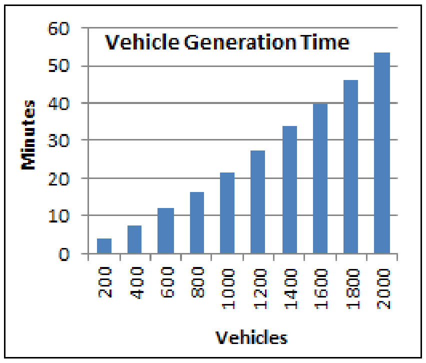
BENCHMARK PERFORMANCE ISSUES

6.1. Performance Experiments

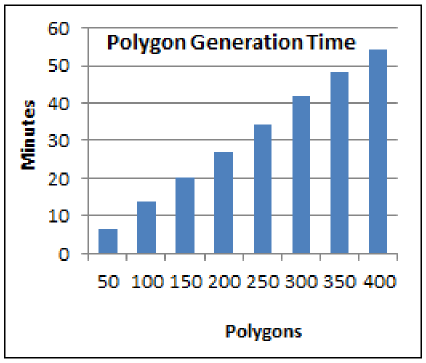
Evaluating a GSMS (geo-stream management system) requires large data sets, which implies that a benchmark data generator should produce data sets in a reasonable amount of time. We evaluated the GSMark data generator using a Dell^{TM} PowerEdge T110 server with a quad core Intel® Xeon® X3430 (8M Cache, 2.40 GHz) processor and 8GB of ram running CentOS 5.5. The test runs included examining the vehicle and polygon generations separately and then together to determine which component contributes most to generation time.

Fig. 6.1 shows typical results of a test run. For 100 minutes of data on the Denton road network, vehicle generation time (Fig. 1(a)) is exponential with 200 vehicles typically requiring only a few minutes but 2000 vehicles requiring over 50 minutes. There are three reasons for this. First, the A* algorithm is logarithmic; however, the road network density resulting from the tiger/line shape files is relatively high and the adjustment made to ensure reasonable route lengths causes an increase. Second, the generator must sort the vehicle list by start time before the vehicles begin moving since the traffic jam requires the current capacity, a value only known if it processes the vehicles in order. The sort function uses quick sort, but the random start times from a uniform distribution often produces the worst case $O(N^2)$. Third, the traffic jam function must scan all road segments with vehicles to check for over capacity and adjust the speed of vehicles if necessary.

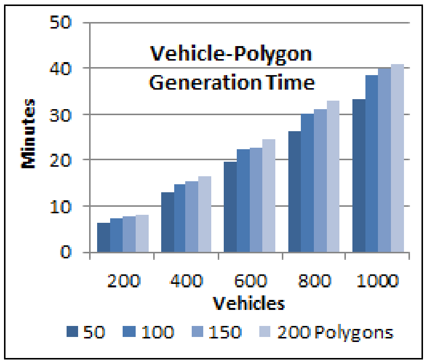
The polygon-only generation in Fig. 1(b) has similar generation times, but the relationship is linear. In this example, the polygons have 200 faces, travel at 10 mi/h, and are approximately 30 mi^2. The generation time is high mainly because, like the traffic jam function, it must check the segments it overlays and reduce the speed accordingly. Combining this with vehicle generation produces and overall exponential generation time (Fig. 1(c)), primarily because the vehicle generation is exponential.



(a) Vehicle Generation



(b) Polygon Generation



(c) Vehicle-Polygon Generation

Figure 6.1. Data generation performance. Vehicle generation is exponential while polygon generation is linear.

6.2. Parameters's Impaction of Performance

In the previous section, it shows that how the benchmark performs when we generate different numbers of vehicles, polygons, or both. All the parameters are setting as default. It's obviously, if we use better computer, the performance of the benchmark will improve. But not only the computer hardware impacts the performance of the benchmark, a user's setting of the parameters will also influence the benchmark's performance. In this section, we show the users how the parameters impact the performance and give some suggestions of setting up the parameters.

When users set up a parameter, users have to be careful about the number they want to use. For example, to have more realistic polygons, user may want to have more nodes to represent a polygon. When a polygon has more faces, its shape becomes more smooth. But when a user assigns more nodes for a polygon, the data generator has to generate, monitor and report more nodes each time unit. Workload simulator also has to send and receive more nodes each time unit. This will slow down the benchmark dramatically. To continue the example, a rain may have more than thousands faces. However, if we use so many nodes to represent a rain, the benchmark will perform really slow. And when a user wants to generate more cars in each time unit, it also costs more time to do so. Moreover, when we generate more cars in a time unit, we have to check the positions of the cars in each following time unit. We also have to analysis and the influence of each car for the traffic jam. Therefore, to set the parameters, a user should take a balance of both the realism and performance.

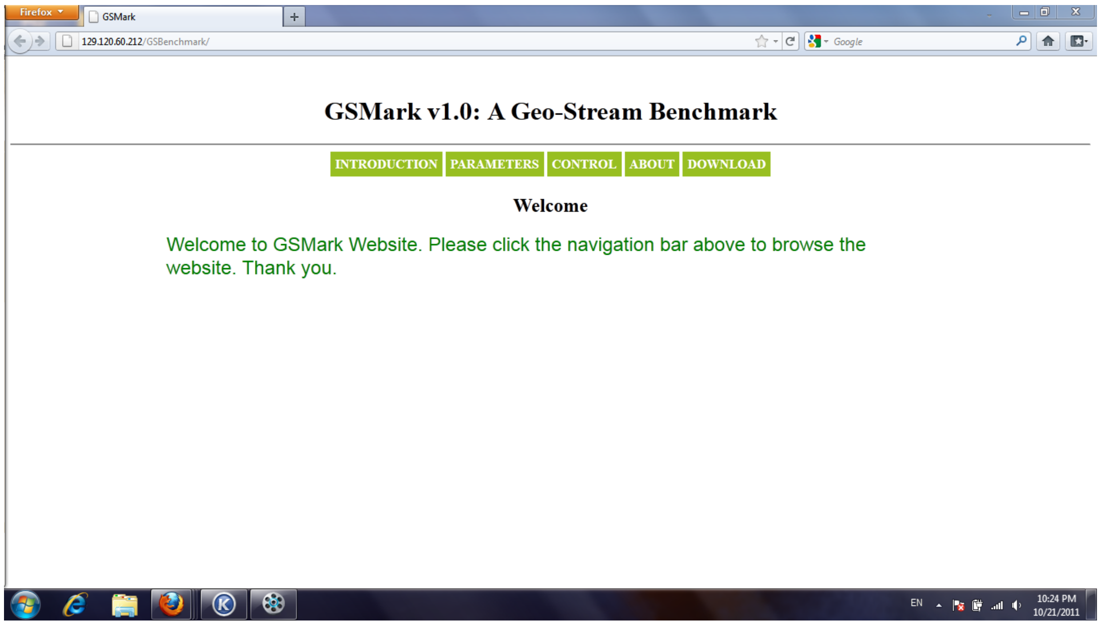
CHAPTER 7

INTERFACE DESIGN

To let users easily use the demo of the the benchmark, we designed a simple interface. The interface is made particular for the data generator which takes the most weight in the benchmark. For the security purpose and network issues, at this point, we do not allow a user to upload his own road network files. We have the Denton road network files in the server. And the road network is sufficient for the demo purpose.

7.1. Welcome Page

When a user browses the interface, the default page is the welcome page. The welcome page shows the welcome information to the users and notices the users to use the navigation bar to find the information they need. Fig 7.1 shows the welcome page.

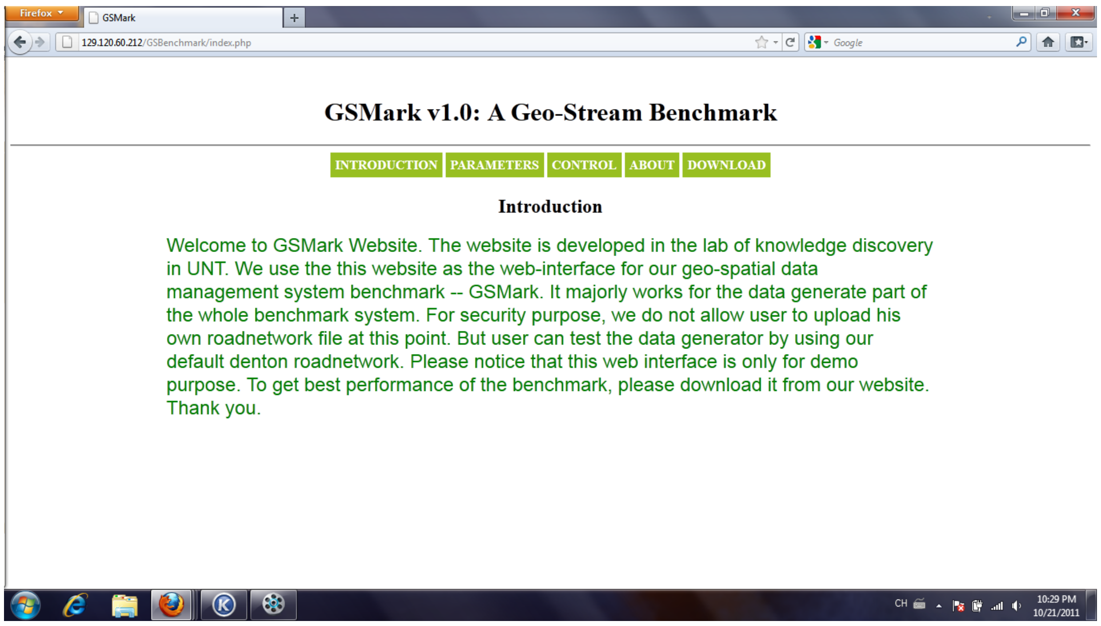


GSMark v1.0: A Geo-Stream Benchmark

Welcome

Welcome to GSMark Website. Please click the navigation bar above to browse the website. Thank you.

Figure 7.1. Welcome page.



GSMark v1.0: A Geo-Stream Benchmark

Introduction

Welcome to GSMark Website. The website is developed in the lab of knowledge discovery in UNT. We use the this website as the web-interface for our geo-spatial data management system benchmark -- GSMark. It majorly works for the data generate part of the whole benchmark system. For security purpose, we do not allow user to upload his own roadnetwork file at this point. But user can test the data generator by using our default denton roadnetwork. Please notice that this web interface is only for demo purpose. To get best performance of the benchmark, please download it from our website. Thank you.

Figure 7.2. Introduction page.

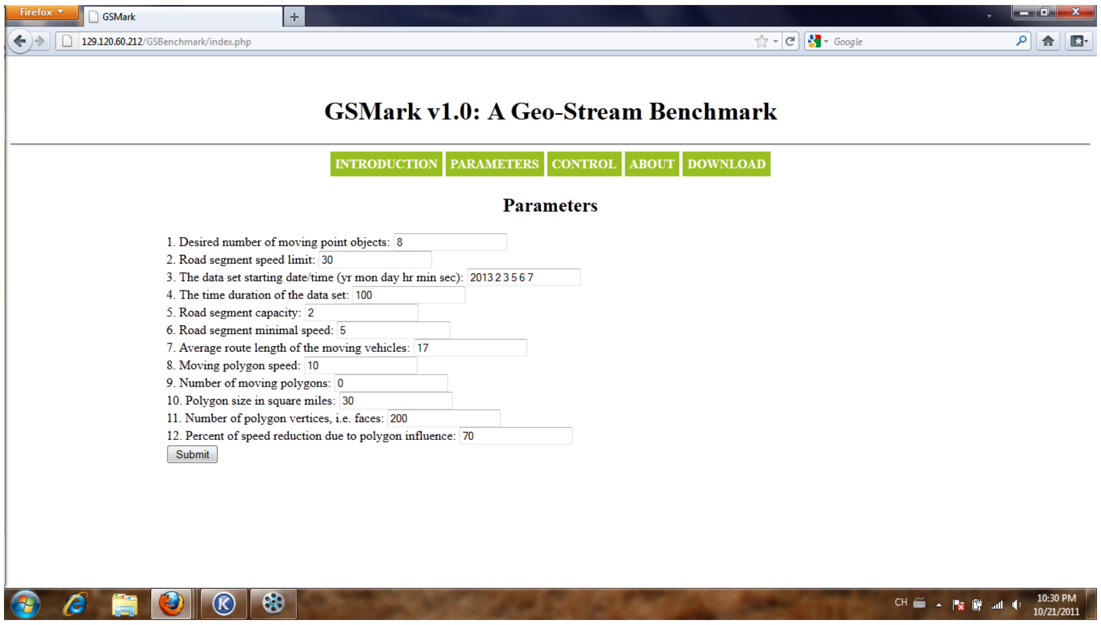
The green bar that is above the welcome message is the navigation bar. Users can click one of the link to go the interested page.

7.2. Introduction Page

When a user clicks the Introduction link in the navigation bar, he will reach the introduction page. The introduction page presents the purpose the benchmark and the interface and gives the basic background information for the benchmark. Fig 7.2 shows the introduction page.

7.3. Parameter Page

Users can click the Parameters link in the navigation bar to go to the parameter page. This page provides a user with 12 parameters. There is a TXT file of the parameters in the server. When we run the data generator, it will read the parameters from the TXT file. Through the parameter page, users have the access to the file's parameters that are related to the data generator. Fig 7.3 shows the parameter page.



GSMark v1.0: A Geo-Stream Benchmark

Parameters

1. Desired number of moving point objects: 8
2. Road segment speed limit: 30
3. The data set starting date/time (yr mon day hr min sec): 2013 2 3 5 6 7
4. The time duration of the data set: 100
5. Road segment capacity: 2
6. Road segment minimal speed: 5
7. Average route length of the moving vehicles: 17
8. Moving polygon speed: 10
9. Number of moving polygons: 0
10. Polygon size in square miles: 30
11. Number of polygon vertices, i.e. faces: 200
12. Percent of speed reduction due to polygon influence: 70

Figure 7.3. Parameter page.

The first parameter here is the desired number of moving point objects. A user can define how many moving points that he wants to generated. If we assume the moving points are cars, and a user set this parameter as 30, the data generator will generate 30 cars.

The second parameter here is the road segment speed limit. User can set a speed limit to each road segment from this parameter. All the moving points' speeds can not exceed the speed limit.

The third parameter here is the data set starting date and time. This parameter defines the beginning time of the artificial timestamps. The simulated moving vehicles may begin to move at this time, but not necessary. When set this parameter, a user should type year, month, date, hour, minute and second. Each number should be separated by a space.

The fourth parameter is the time duration of the data set. For example, if a user wants to simulate the traffic of 2 hours, he should set this parameter as 7200 seconds.

The fifth parameter is the road segment capacity. It is the capacity of a road segment. If the number of the moving vehicles on a road segment exceed this value, it will trigger a slow down function to calculate the vehicles' speeds. A traffic jam will be simulated.

The sixth parameter is the road segment minimal speed. When a traffic jam is simulated, the vehicles' speeds on the road segment can not be further slowed down than this value. This is the minimal speed for the vehicles that are in a traffic jam.

The seventh parameter is the average route length of the moving vehicles. When we do routing, we do not want to have a route that is too long or too short. We make the lengths of the routes follow a uniform distribution of the average route length.

The eighth parameter is the moving polygon speed. All the simulated special events' speeds follow a uniform distribution of this parameter. It's the average speed of all the simulated polygons.

The ninth parameter is the number of moving polygons. It is the number of the special events that a user wants to simulated. For example, if a user wants to simulate 2 special events, he can set this parameter as 2.

The tenth parameter is the polygon size in square miles. All the simulated polygons' sizes follow a uniform distribution of this parameter. Therefore, basically, this parameter is the average size of all the polygons.

The eleventh parameter is the number of polygon vertices. It means how many nodes a user wants to use to represent a polygon. More nodes are used, more smooth a polygon's shape is, but more slower the benchmark performances.

The twelfth parameter is the percent of speed reduction due to polygon influence. This is the influence of a polygon. When a polygon overlaps a road segment, the road segment's speed is decreased. For example, if a user sets this parameter as 70, when a polygon overlaps a road segment, all the vehicles on that road segment will be slowed 70 percent.

7.4. Control Page

After we set up the parameters, a user can click the Control link in the navigation bar to run or stop the benchmark. In this control page, a user can run the data generator, stop the data generator, and visualize the generated data set. Fig 7.4 shows the control page. We have a google earth plug in on this page for visualization purpose. If a user wants to

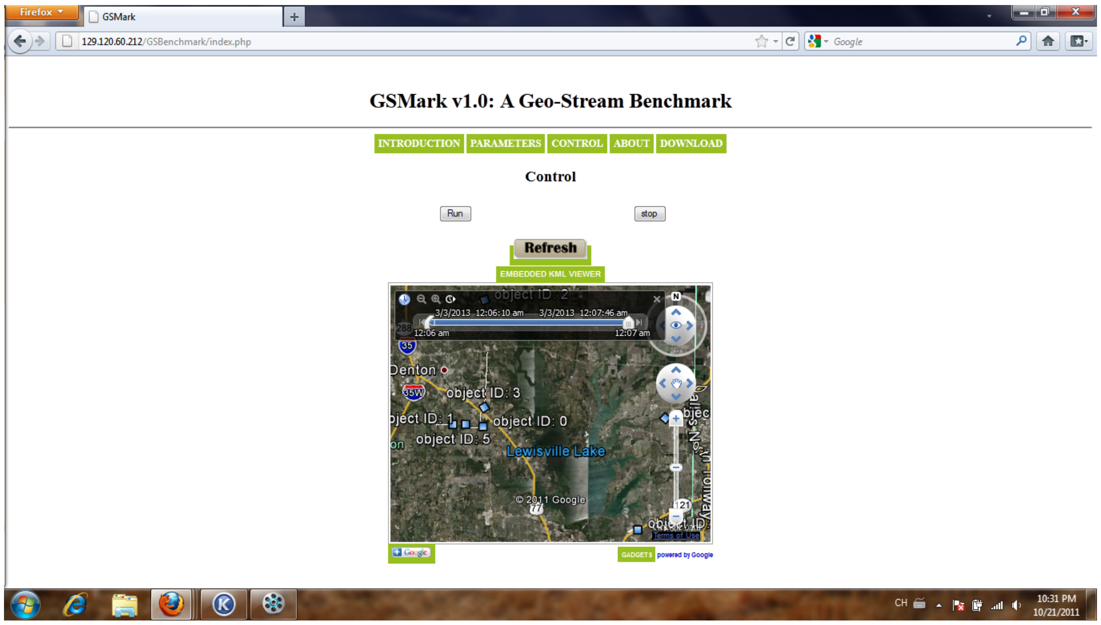


Figure 7.4. Control page.

visualize the data set, he may need to down load the google earth plug in to do so. However, if a user does not want to download it, he still can run the data generator and stop it at any time. To view the data set, he can go to the download page to download the data set, and use a TXT editor to view it.

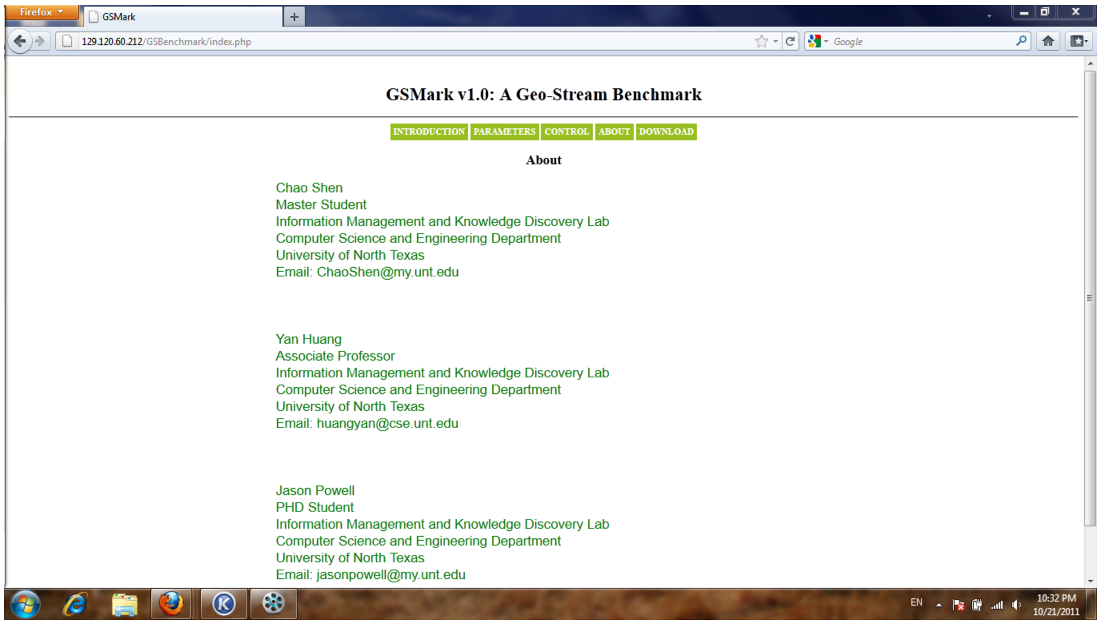
A user can click the run button to run the data generator and the stop button to stop the data generator. After running the data generator, a user needs to click the refresh button to refresh the google earth's visualization results of the new generated data set.

7.5. About Page

The about page shows all the authors that are related to this benchmark. This page presents their name, affiliation, and contact information. If a user has any question, he can contact them for the answers. Fig 7.5 show the about page.

7.6. Download Page

If a user clicks the Download link in the navigation bar, he can achieve the download page. This page provides users with the downloads of 3 different files, such as the point kml



GSMark v1.0: A Geo-Stream Benchmark

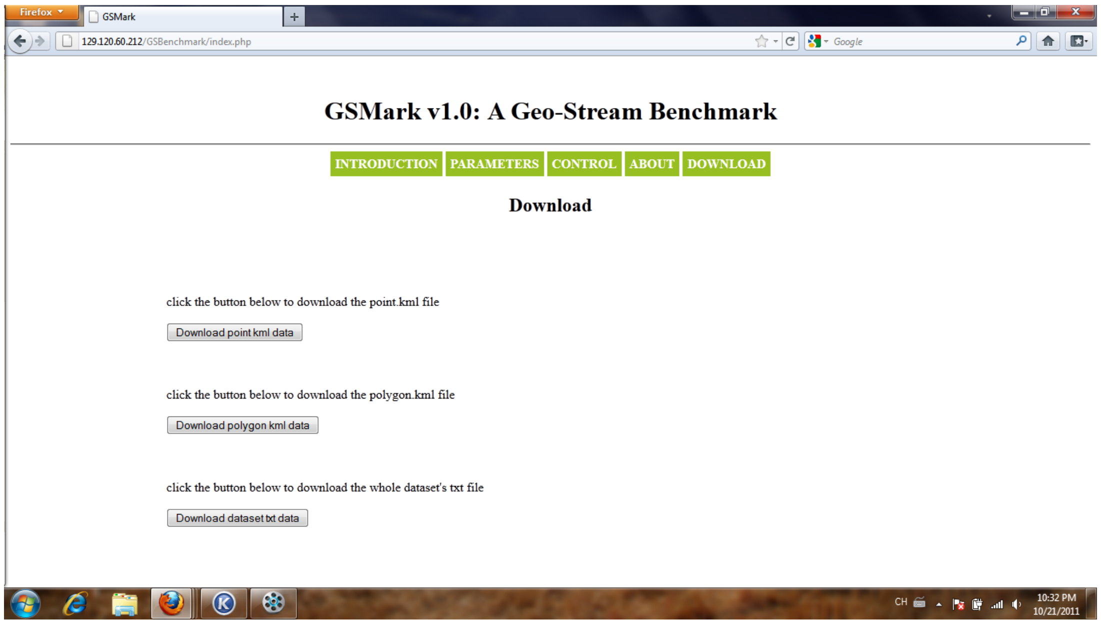
About

Chao Shen
Master Student
Information Management and Knowledge Discovery Lab
Computer Science and Engineering Department
University of North Texas
Email: ChaoShen@my.unt.edu

Yan Huang
Associate Professor
Information Management and Knowledge Discovery Lab
Computer Science and Engineering Department
University of North Texas
Email: huangyan@cse.unt.edu

Jason Powell
PHD Student
Information Management and Knowledge Discovery Lab
Computer Science and Engineering Department
University of North Texas
Email: jasonpowell@my.unt.edu

Figure 7.5. About page.



GSMark v1.0: A Geo-Stream Benchmark

Download

click the button below to download the point.kml file

click the button below to download the polygon.kml file

click the button below to download the whole dataset's txt file

Figure 7.6. Download page.

data file, the polygon kml data file, and the whole generated data set TXT file. Fig 7.6 shows the download page.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1. Conclusion

The benchmark studies recent research papers, and did a survey for streaming, spatio and spatio-temporal database mangement system benchmarks. After we did the classification for the research papers, it shows clearly what have been done recently toward streaming, spatio and spatio-temporal DBMSs (database management system), what the current systems lack, and what we can do to improve the current systems.

Through the survey, we have several observations. The recent growth in sensor technology allows easier real-time gathering of geo-streaming data that requires efficient geo-streaming management systems. Benchmarks for determining functionality and efficiency of GSMS (geo-stream management system) are lacking, therefore, this paper presented GSMark, a benchmark specifically for GSMSs. GSMark builds on spatial, spatio-temporal, and stream management system benchmarks to provide a data generator, query set, and workload simulator using streaming point and polygon objects moving on real road networks. The 3 parts of the benchmark can either work together for the whole benchmark system or work individually for particular research purposes. The data generator performance demonstrates acceptable generation times for reasonable data set sizes that can sufficiently evaluate a GSMS.

The data generator considers most principles that are generally used in streaming, spatio, and spatio-temporal benchmarks. For example, most current benchmarks consider traffic jam as an important feature for data simulation. Therefore, in the data generator of GSMark, it generates traffic jam when the number of car exceed the road capacity. A few benchmarks consider outside special events are another important feature for data generator. GSMark also generators special events, and check how it impacts the road traffic. A few benchmarks have simulated the peak hours for daily traffic. GSMark also simulates the traffic peak hours.

It also has many features that are not covered by previous benchmarks. For example, it generates polygons as special events. Most benchmarks use rectangles to represent special events. Using moving polygons to represent special events can test the geo-stream data management system's ability of dealing with polygons.

The workload simulator simulates delay and disorder situation of data delivery. Most benchmarks do not consider the delay and disorder situation. And by separating data generate and delivery processes, we give user more room to control each step. For example, users have more control over delay possibility, max delay, disorder possibility, and max disorder range. Based on the user's setting of those parameters, the GSMark could simulate more realistic traffic situations. And it also has more varieties. User can create different patterns for data delivery by setting parameter differently.

Query set contains most popular queries for current benchmark systems. For example, it has the join for points data, regional data, and point and region data. It covers the most important testings for the geo-spatial streaming data management systems. Several queries made specially for geo-spatial streaming data.

The whole benchmark system contains most popular features of current benchmark systems, and creates special features particularly for geo-spatial streaming data management systems. It conducts a reasonable testing method for GSMSs, and has potential to get expansion in the future.

8.2. Future Work

There are several future projects for this benchmark. First, even though it contains typical queries that cover typical operations, the query set is not comprehensive enough to evaluate a mature GSMS fully at this time. Second, a web-based interface could allow easier parameter customization and provide mechanisms to customize the road network, polygons, or vehicle distributions. In addition, the interface could also provide visuals about the resulting data set, such as a heat map with road segments colored by capacity or current speeds. Finally, it is possible to improve the data generation algorithms and the data generator could further incorporate traffic and driver patterns to increase realism; likewise, polygon movements could also follow patterns. Improvements like these could shift the benchmark into a series similar to the TPC series but with a geo-streaming context.

BIBLIOGRAPHY

[1] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik, *The Design of the Borealis Stream Processing Engine*, Second Biennial Conference on Innovative Data Systems Research (CIDR 2005) (Asilomar, CA), January 2005.

[2] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik, *Aurora: a new model and architecture for data stream management*, The VLDB Journal 12 (2003), 120–139.

[3] M. Van Aerde, B. Hellinga, M. Baker, H. Rakha, M. Van Aerde, B. Hellinga, M. Baker, and H. Rakha, *Integration: Overview of simulation features*, 1996.

[4] Mohamed Ali, *An introduction to microsoft sql server streaminsight*, Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research &#38; Application (New York, NY, USA), COM.Geo '10, ACM, 2010, pp. 66:1–66:1.

[5] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom, *Stream: The stanford data stream management system*, Technical Report 2004-20, Stanford InfoLab, 2004.

[6] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts, *Linear road: a stream data management benchmark*, Proceedings of the Thirtieth international conference on Very large data bases - Volume 30, VLDB '04, VLDB Endowment, 2004, pp. 480–491.

[7] Thomas Brinkhoff, *A framework for generating network-based moving objects*, Geoinformatica 6 (2002), 2002.

[8] U.S. Census Bureau Geography Division, *U.s. census bureau - tiger/line*, [Online; accessed June 11, 2011], http://www.census.gov/geo/www/tiger/.

[9] Christian Düntgen, Thomas Behr, and Ralf Hartmut Güting, *Berlinmod: a benchmark for moving object databases*, The VLDB Journal 18 (2009), 1335–1368.

[10] Ralf Hartmut Gting, Thomas Behr, Victor Almeida, Zhiming Ding, Frank Hoffmann, and Markus Spiekermann, *Secondo: An extensible dbms architecture and prototype*, Tech. report, 2004.

[11] Yan Huang and Chengyang Zhang, *New data types and operations to support geo-streams*, Proceedings of the 5th international conference on Geographic Information Science (Berlin, Heidelberg), GIScience '08, Springer-Verlag, 2008, pp. 106–118.

[12] Christian S. Jensen, Dalia Tiesyte, and Nerius Tradisauskas, *The cost benchmark-comparison and evaluation of spatio-temporal indexes*, DASFAA, 2006, pp. 125–140.

[13] Feifei Li, *Real datasets for spatial databases: Road networks and points of interest*, [Online; accessed June 11, 2011], http://www.cs.fsu.edu/˜lifeifei/SpatialDataset.htm.

[14] D. Mohapatra and S.B. Suma, *Survey of location based wireless services*, Personal Wireless Communications, 2005. ICPWC 2005. 2005 IEEE International Conference on, jan. 2005, pp. 358 – 362.

[15] Jussi Myllymaki and James Kaufman, *Dynamark: A benchmark for dynamic spatial indexing*, Proceedings of the 4th International Conference on Mobile Data Management (London, UK, UK), MDM '03, Springer-Verlag, 2003, pp. 92–105.

[16] Michael Stonebraker, Jim Frew, Kenn Gardels, and Jeff Meredith, *The sequoia 2000 storage benchmark*, Proceedings of the 1993 ACM SIGMOD international conference on Management of data (New York, NY, USA), SIGMOD '93, ACM, 1993, pp. 2–11.

[17] TPC, *Tpc - about the tpc*, [Online; accessed June 11, 2011], http://www.tpc.org/information/about/abouttpc.asp.

[18] Paul Werstein, *A performance benchmark for spatiotemporal databases*, In: Proc. of the 10th Annual Colloquium of the Spatial Information Research Centre, 1998, pp. 365–373.