The Linear Road Benchmark

Richard Tibbetts tibbetts@mit.edu
Mike Stonebraker stonebraker@mit.edu

Abstract

Several senior DBMS researchers have questioned whether specialized stream-oriented DBMSs are necessary. In other words, maybe stream applications can be done readily with current DBMSs and middleware products. The purpose of this paper is to describe a benchmark based on a plausible scenario. Following this description, results are presented on runs of this benchmark for three different specialized storage systems as well as for a popular commercial DBMS.

It will be seen that the specialized systems outperform current DBMSs by an order of magnitude or more, and clearly demonstrate the payoff for this class of system software.

I Introduction

In this paper we introduce a benchmark based on a traffic scenario. Traffic congestion in major metropolitan areas is an increasing problem. Clearly, it is not possible to build expressways fast enough to keep traffic flowing freely at peak periods. There have been several attempts to reduce traffic by providing high occupancy vehicle (HOV) lanes, which have met with limited success. On the other hand, governments have begun to explore charging vehicles a toll, which is based on the amount of traffic using the road. The reasoning is that higher tools should be charged at peak periods to discourage vehicles from using the roads during that period. Illinois [XXX], California [YYY] and Finland [ZZZ] have pilot programs utilizing this concept. Moreover both London and Singapore charge tolls at peak periods to enter the downtown area, using similar reasoning.

Of course, to enforce a congestion-sensitive toll, it is necessary to monitor the traffic on the expressways in question. There are two architectures that could be used. First, sensors could be embedded in the road or on roadside facilities. Cars could carry "fast path" devices that would reflect the signal from the roadside systems, and the account of the individual vehicle could be decremented. Alternately, the vehicle could have an active device with a longer range that could broadcast its position to a data collection network that could be removed from the side of the road. Both the "smart road dumb car"" and "dumb road smart car" architectures are being followed in the current pilots.

Of course, any major metropolitan area has its own peculiarities. These include water obstacles in New York, San Francisco, and Boston, as well as hills in Los Angeles and Seattle. Modeling any given urban area is a challenge in its own right. To avoid this fidelity problem, we model a highly stylized urban area. Moreover, to avoid the necessity of modeling curves, lane drops, and other obstacles, we make the road network very linear. As a result, we call this benchmark The Linear Road Benchmark.

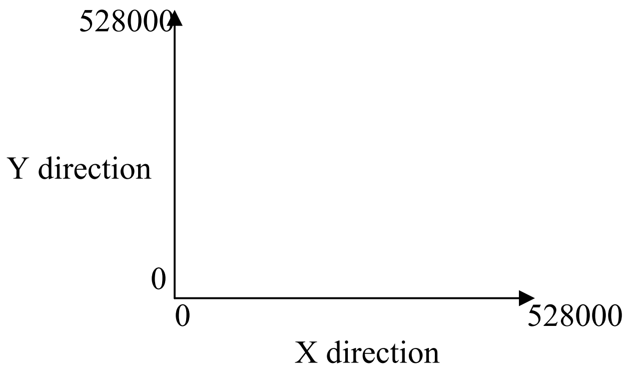
Lastly, many traffic simulations entail a significant control component. Often, the goal is to model the effect of HOV lanes, metering of cars at entrances to the expressway, etc. In order to avoid extending our simulation into this control realm, we purposely leave out of our benchmark any control mechanisms.

In Section 2, we begin with a description of our linear road scenario. Then, in Section 3 we discuss the specifics of the benchmark test harness. Section 4 describes our experiences with the benchmark in four different database systems, three streaming and one classic RDMS. Section 5 contains our results and section 6 our conclusions.

II The Linear Road Scenario

2.1 Expressway Geometry of Linear City

Consider a metropolitan area (which we call Linear City) that is 100 miles (528000 feet) wide and 100 miles (528000 feet) long. Hence, the area starts at (0, 0) and extends to (528000, 528000), as shown in Figure 1.



The Geometry of Linear City
Figure 1

There are 10 expressways in Linear City, each running horizontally. The first one runs from (X, Y) position (0, 52800) to (X, Y) position (528000, 52800), i.e. it is horizontal and 10 miles above the lower edge of Linear City. Successive expressways are spaced 10 miles apart and parallel to the first one. Hence, the 10th expressway runs from position (0,528000) to (528000, 528000) and is 100 miles above the lower edge of Linear City. For simplicity, there are no expressways that run vertically. The assumed center of Linear City is the vertical line from (264000, 0) to (264000, 528000).

Each expressway has 100 onramps and 100 offramps in each direction, dividing it into 200 segments (100 eastbound, 100 westbound). The ramps are approximately a third of a mile long, and allow cars to accelerate or decelerate. The onramp for a segment puts cars onto the segment just after the start of the segment. The offramp comes just before the end of the segment.

Each expressway contains three travel lanes in each direction, for a total of six lanes. According to the traffic simulator used in this benchmark, the capacity of each of the 200 segments is 150 cars. This allows for a reasonable separation between cars at 60 miles per hour. During rush hour, the roads may be over capacity, and there may be more than 30,000 cars on each expressway. The traffic simulator models the resulting slowdown when a segment is over capacity.

It will not always be possible for the tracking system to be described to keep up with the requirements of the benchmark. Hence, it will often be necessary to simulate L expressways, where $L < 10$. Expressways are simulated independently. A car may travel on more than one expressway during the course of the simulation, but it will of course travel on only one expressway at a time.

2.2 Vehicles

There are 1,000,000 vehicles registered in Linear City that can use the expressways. Each is equipped with a sensor that will report its position and speed every 30 seconds, but only if the vehicle is on an expressway, on an entrance ramp or on an exit ramp. Each car goes no more than 100 MPH; hence each car reports its position in each segment it travels in at least once.

Speed is an integer number of miles per hour. Position is a pair of integers (X-loc, Y-loc) on the grid defined by Figure 1. For simplicity, we assume that the sensor is accurate to within a few feet; hence the travel lane of a vehicle can be deduced from its reported position.

To simplify the implementation of the benchmark, the traffic network preprocesses the data coming from the cars. Hence, the actual message reported through the sensor network from each vehicle is:

(car_id, timestamp, speed, expressway#, travel_lane, direction, X-position)

The clock(s) that perform the timestamping are assumed accurate; hence there is no issue with clock skew.

2.3 The Simulation

Vehicle trips are generated according to two scenarios to be described. Each trip is from an entry ramp, EN, to an exit ramp, EX. The vehicle is placed on the entrance ramp at EN, begins reporting its position every 30 seconds, and accelerates at a rate allowed by the other traffic. It then merges onto the expressway, and the traffic simulator moves the vehicle toward EX at a rate allowed by the traffic congestion. When the vehicle reaches EX, it moves to the exit ramp and decelerates. When it leaves the exit ramp completely, it stops reporting its position.

The following two scenarios are envisioned:

Rush hour: the arrival rate is set so there are an average of 200 cars in each segment. Tolls are assumed high enough, so that 10% of the cars exit in each segment that charges a toll, rather than paying it.

Rush hour – accident: parameters are set so there are an average of 200 cars in each segment. An accident is generated at a random location in each direction on each expressway. At the moment the accident is cleared, another one is generated randomly to replace it. Traffic is assumed to proceed by the incident at a reduced speed in the remaining travel lanes. The traffic spacing model handles this calculation. Each car in the five upstream segments has a 10% probability of exiting the expressway to take advantage of the negative toll.

2.4 Tolls

The tracking system charges tolls for use of the expressway under certain conditions. The toll for each one-mile segment is zero if average velocity in the segment is at least 40 MPH. Otherwise the toll is based upon the number of cars in the segment.

This toll is reported to each vehicle when calculated along with the average velocity in the upcoming segment. The expressway authority has elected to reduce congestion by waiving the toll if the vehicle exits at the next interchange. In order to notify drivers in time for them to take advantage of this offer, the “toll” message must be delivered within 15 seconds of the toll calculation. Since all cars exit at the ends of the expressway, this toll waiver does not apply on the ending segments of the expressways.

If the tracking system fails to deliver toll messages within 15 seconds, it will be termed “over capacity”.

If the vehicle chooses not to exit, the toll is subtracted from the customer’s account. The expressway authority, in its infinite wisdom, gives a frequent traveler discount. Hence, subsequent tolls are divided in half, after a car has spent Y dollars in a given day on a given expressway.

A customer traveling on the expressway can ask for his balance at any time. This balance must be accurate through the last segment traveled, i.e. the one previous to the one he is currently in. He can also ask for his expenditure during any day of the last 10 weeks by expressway. Hence, the system must keep track of daily expenditure for L freeways for 70 days for 1M cars, a total of L * 70M observations.

2.6 Accidents

The tracking system also detects accidents, which are defined to be one or more cars that are stopped in a travel lane. More precisely, a car is stopped if four successive 30-second readings come from the same geographic location. At this moment an accident is registered at this position, and a lane of traffic is blocked. The traffic simulator then routes cars around the accident, until it is cleared.

Once an accident is registered, every vehicle in the 5 upstream segments is notified that there is a downstream accident, so that it can seek an alternate route if desired. A customer is given a "negative toll" of X if he exits at his current segment. Negative tolls are continued until the accident is cleared. Negative tolls are an incentive to keep accidents from causing too much havoc.

Lastly, once an accident is registered, tolls are not charged in any segment that is within 10 segments upstream of an incident. Tolls are not resumed until 20 minutes after the accident is cleared.

2.7 Ad Hoc Queries

Cars do not just passively drive down the highway, racking up tolls. Additional functionality is offered by the system, which cars can access by issuing ad-hoc queries. There are three different types of queries supported by the system:

Account Balance – Request for the current account balance. Includes the car id.

Daily Expenditures – A request for the day's expenditures for a given car. Includes the car id.

Travel Time Prediction – A request for the expected travel time between two segments on a specific expressway. Included with the request is the time of day and day of the week.

The system responds with the average travel time and average toll for the last 10 weeks from the initial segment to the final segment at the given time of day and day of week. It must keep this historical data on a 1-minute granularity. Hence, there are

(L expressways) *(200 segments) * (1440 minutes/day) * (70 days) ~ L * 20M observations.

This calculation must be accurate; i.e. the time of day must be advanced by the predicted time required to traverse each segment. Hence, the predicted travel time in the subsequent segment depends on the average speed in the current segment.

III The Scenarios

III Benchmark System Specifications

3.1 Traffic System Simulator

A road network simulator drives the linear road benchmark. At the core is a traffic simulator, MITSIMLab. This system generates cars and enters them onto the expressway. For each simulated expressway, the source location of a car is uniformly distributed over all of the possible entrance ramps. The exit ramp is normally distributed with a mean of 50 and a standard deviation of 20 miles. Hence cars have an affinity for the "downtown area". Cars choose eastbound or westbound ramps as appropriate. Once on the expressway, each car proceeds according to a standard traffic spacing model built into the traffic simulator.

The processed output of the simulator is written out to files. A 2-hour period is recorded for use in the benchmark. Before those two hours are recorded, the simulator is run for one hour to achieve a stable state of the road network.

In addition, there are ad-hoc queries. These queries are one of: account balance queries (50%), expenditure for a given day by freeway (10%) or travel time prediction queries (40%). Each time a car proceeds through a segment, it has a 1% chance of generating an ad-hoc query.

3.2 Input Specification

The system being benchmarked receives 4 input streams. The primary stream is events from the traffic monitoring system, which indicate the position and speed of cars. The other three event streams are queries from cars on the network, querying for account balance, expenditure for a given day, or travel time prediction. The events in each stream are tuples as follows:

Car location events	(0, timestamp, car_id, speed, expressway, travel_lane, direction, X-position)
Account balance query	(1, timestamp, query_id, car_id)
Today's expenditure query	(2, timestamp, query_id, car_id)
Travel time query	(3, timestamp, query_id, expressway, initial_segment, final_segment, time_of_day, day_of_week)

For simplicity, every element in every tuple is represent able with a 32-bit signed integer. Query_id is a unique identifier used to associate responses with their queries. The encoding for the non-obvious fields is as follows:

Timestamp	Seconds since start of simulation.

Speed	Miles per hour
Travel_lane	0=Ramp, 1=Left lane, 2=Middle, 3=Right
Direction	0=East, 1=West
Time_of_day	Seconds since 0:00
Day_of_week	0=Sunday, 1=Monday, … 6=Saturday.

To avoid the complication of unpredictable event delivery order, these 4 input streams are multiplexed together by the benchmark driver. The purpose of the number 0-3 at the start of each tuple is to allow the application to demultiplex the events. The driver is a small program that delivers events to the application being benchmarked at their appropriate time. Events are delivered in a binary format. For simplicity, every event is padded to 32 bytes (8 4-byte integers). The first two fields will be event type and timestamp on all events.

The driver reads its data out of a flat file. Applications that cannot be driven with the standard driver program may create their own equivalent driver program to read the same file format and issue events to their system. The file simply has tuples each on their own line, in the order in which they should be delivered. The fields are in ASCII text, separated by commas.

3.3 Historical Data

In order to be able to respond to queries about travel time, account balances and daily expenditures, there has to be a starting point for the simulation.

TODO: Explicit specification of all the necessary historical data.

3.4 Tracking System Specification

The vehicle tracking system should compute the following metrics:

a) Number of cars in each one mile segment, a total of L * 200 segments. It computes this metric once a minute as follows. It picks a specific timestamp, say on the minute boundary. Then, it identifies the most recent message from each car, and assigns it to the correct segment. Lastly, it counts the cars in each segment. The tracking system can use a different timestamp for each direction of each expressway, if it wishes to.

b) Latest Average Velocity (LAV) in each direction for each one-mile segment. This is computed as the average of the speeds of the cars in each segment.

c) Composite Average Velocity (CAV) in each direction in each one-mile segment. This is computed as the average velocity of the last four LAVs reported.

3.4 Output Specification

The system being benchmarked has out output 5 similar streams. The first stream is a notification of tolls given to each car every 30 seconds. The second is a notification of traffic accidents, given to all cars in the 5 upstream segments when an accident is detected. The other three streams are responses to the three different kinds of queries. The following table describes the format of the streams:

Toll notification	(0, car_id, toll, segment_speed)
Accident notification	(1, accident_position)
Account balance response	(2, query_id, balance)
Expenditure response	(3, query_id, expenditure)
Travel Time response	(4, query_id, travel_time, toll)

Balances and expenditures are measured in hundredths of a dollar. Travel time is measured in seconds.

3.4.1 Toll notification

After a car first reports its position in a new segment, the toll system has 15 seconds to respond with the toll for that segment and the current average speed in that segment. If the car exits at the offramp of this segment, no toll is charged. If the car continues on to the following segment, the toll is charged to the car's balance and a new toll is calculated.

The toll is calculated based on the number of cars in the segment, and the average velocity of the cars on the segment. Average velocity should be calculated from all reports on the segment over the last 5 minutes. The number of cars on the segment should be accurate to within 1 minute.

The toll will be 0 if the average velocity is greater than 40 MPH. If the velocity is less than 40MPH, the toll will be

$$basetoll \times (cars - 150)^2$$

However, if the total expenditures for an individual car exceed Y dollars in a given day, each additional toll is discounted by 50%. The discounted toll should be reported to the car in the toll notification, and charged to the account properly.

TODO: Consider if the toll notification should also include a new balance, if there was a toll on the previous segment.

3.4.2 Accident Notification

An accident is detected when a car is in the same location for 4 successive 30-second readings. Once an accident is registered, every vehicle in the 5 upstream segments is notified of the accident. Any car that exits at its current segment is given a negative toll of $X. Negative tolls persist in the 5 upstream segments until the accident is cleared.

Lastly, once an accident is registered, tolls are not charged in any segment that is within 10 segments upstream of an incident. Tolls are not resumed until 20 minutes after the accident is cleared.

3.4.3 Account Balance Response

The account balance response must take into account all historical tolls as well as all tolls while have been charged to the account during the simulation.

3.4.4 Expenditure Response

The expenditure response should total all tolls that have been charged today.

3.4.5 Travel Time Prediction

The travel time prediction must give an accurate prediction of the travel time and toll. Calculations for every segment cannot be made using the start time; they must take into account the time required to traverse previous segments.

3.5 Validation System

Along with the input files and the input processing system will be an output processing and validation system. This will be a set of flat files, as above, with a reader program. The flat files will be a reference set of output for the given input. The reader will read output from the benchmarked system and from the flat files, and check the benchmark output for correctness. Because of possible variation in some parts of the system, a certain margin of error in most calculations will be allowed. The validation system will look for large errors, and also for excessive lag. It is expected that most systems will produce accurate output, but will at some stage be unable to continue meeting the quality of service guidelines.

TODO: Specify the precise margin of error.

3.6 Required Quality of Service

The following table summarized the quality of service required of the benchmark system.

Response	Allowable lag
Toll notification	15 seconds from the time a car reports its position in a new segment.
Accident notification	15 seconds from the second report of a car in the same position.
Account balance	60 seconds from query.
Expenditure	90 seconds from query
Travel time	120 seconds from query

3.7 Running the Benchmark

For ultimate simplicity, this benchmark should be run on a single processor Linux box. The number of expressways, L, should be increased until the box cannot keep up with the input data stream. Hence, the output for any software system is the number of expressways that it can run for each scenario.

IV The Traffic Systems Utilized

The linear road benchmark was run on three prototype stream processing systems, Medusa from M.I.T., Aurora from Brown, and Streams from Stanford. Other stream projects were invited to participate but declined for one reason or another. The benchmark was also run at Brown on a popular commercial relational DBMS under the supervision of a person skilled in the use of the product.

In the remainder of this section we briefly describe the implementation of the benchmark on the four systems. Then, in Section 5, we present the results obtained.

4.1 Medusa Implementation

4.2 Aurora Implementation

4.3 Streams Implementation

4.4 RDBMS Implementation

We built three different implementations of the benchmark in the commercial system. We label these "obvious", "as little as possible" and "middle ground" respectively.

In the obvious implementation, we entered the incoming messages directly into the database as they appeared from an application program. Each one was an individual insert transaction. Each ad-hoc query was similarly entered from a second application program. Finally, a third application program constructed the required aggregates for each expressway once a minute and then computed the tolls for each car, all in SQL. The net result was updates to the account balance for each car. As a side effect of the update, the output message for formulated for each car.

Hence, the obvious implementation used SQL for as much of the application as possible. The downside of the obvious implementation is poor performance, since each input message turns into a transaction.

Therefore, we tried a second implementation "as little as possible". This implementation attempts to get the best performance possible by doing as little of the application in the DBMS as possible. Specifically, a copy of the account balance table is kept in the application as a main memory data structure. The entire toll calculation is performed in the application. Lastly, the account balance table is updated in the application and the actual DBMS table updated in bulk to obtain recoverability. The ad-hoc queries are handled one by one by an application as in the obvious implementation.

"As little as possible" uses the DBMS primarily as a persistent file system and the heavy lifting is all done in the application program. "As little as possible" was designed to get the ultimate performance from commercial software, though it is unlikely that a serious implementation would be done this way.

The third implementation was a middle ground. It performed the same algorithm as the obvious implementation but "batched" up input messages into groups of 500 and then bulk-inserted a batch of messages into the database. This cut down on the overhead of inserting the input messages into the database, and provided performance that was intermediate between the two implementations already described.

VI Results

Work in the application is
4.5