Linear Road: A Stream Data Management Benchmark

Arvind Arasu*
Stanford University
arvinda@cs.stanford.edu

Mitch Cherniack†‡
Brandeis University
mfc@cs.brandeis.edu

Eduardo Galvez†
Brandeis University
eddie@cs.brandeis.edu

David Maier§
OHSU/OGI
maier@cse.ogi.edu

Anurag S. Maskey†
Brandeis University
anurag@cs.brandeis.edu

Esther Ryvkina‡
Brandeis University
essie@cs.brandeis.edu

Michael Stonebraker
MIT
stonebraker@csail.mit.edu

Richard Tibbetts
MIT
tibbetts@mit.edu

Abstract

This paper specifies the **Linear Road Benchmark** *for Stream Data Management Systems (SDMS). Stream Data Management Systems process streaming data by executing continuous and historical queries while producing query results in real-time. This benchmark makes it possible to compare the performance characteristics of SDMS' relative to each other and to alternative (e.g., Relational Database) systems. Linear Road has been endorsed as an SDMS benchmark by the developers of both the Aurora [1] (out of Brandeis University, Brown University and MIT) and STREAM [8] (out of Stanford University) stream systems.*

Linear Road simulates a toll system for the motor vehicle expressways of a large metropolitan area. The tolling system uses "variable tolling" [6, 11, 9]: an increasingly prevalent tolling technique that uses such dynamic factors as traffic congestion and accident proximity to calculate toll charges. Linear Road specifies a variable tolling system for a fictional urban area including such features as accident detection and alerts, traffic congestion measurements, toll calculations and historical queries. After specifying the benchmark, we describe experimental results involving two implementations: one using a commercially available Relational Database and the other using Aurora. Our results show that a dedicated Stream Data Management System can outperform a Relational Database by at least a factor of 5 on streaming data applications.

* This material is based on work supported by the National Science Foundation under Grant Nos. IIS-0118173 and IIS-9817799 (*), IIS-0086057 (†), IIS-0325525 (‡) and IIS-0086002 (§).

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

**Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004**

1 Introduction

In this paper we introduce the **Linear Road Benchmark** for Stream Data Management Systems (SDMS).

Stream data management has become a highly active research area and has inspired the development of several prototype systems including Aurora [1], STREAM [8], TelegraphCQ [4] and Niagara [5]. However, up until now there has been no way to compare the performance characteristics of these systems either to each other or to traditional data management systems configured to process streaming data (e.g., a Relational DBMS configured with triggers). Linear Road is designed to measure how well a system can meet real-time query response requirements in processing high-volume streaming and historical data. It has been endorsed by the developers of Aurora (out of Brandeis University, Brown University and MIT) and STREAM (out of Stanford University) as a basis for performance comparisons of stream processing approaches.

In this paper, we use Linear Road to compare the performance of an SDMS (Aurora) to a Relational Database configured to process stream data inputs.[1] Of course, our implementation of Linear Road over a Relational Database may not be optimal, and thus we invite others to implement Linear Road and report their numbers. Nonetheless, we believe that the results reported here show that a dedicated SDMS is far-better suited than a Relational Database for supporting streaming data applications.

Streaming data poses unique challenges to the design of a benchmark. For queries over this data to be meaningful, the input data must have *semantic validity* and not just be random. Because most stream queries are continuous, performance metrics should be based on *response time* rather than completion time. The benchmark must be *verifiable* even though results returned may vary depending on when they are generated. And the absence of a *query lan-*

[1] We did not get performance numbers for STREAM in time to include them in this paper. However, these numbers and a description of the STREAM implementation of Linear Road will be available on the STREAM Linear Road web page [10].

guage standard for stream queries means that the benchmark queries must be specified in a more general, though unambiguous way. Linear Road has been designed to meet each of these challenges.

Linear Road simulates an urban expressway system where tolls are determined according to such dynamic factors as congestion and accident proximity. Linear Road's traffic-based orientation is inspired by the increasing prevalence of *variable tolling* (also known as *congestion pricing*) [6, 11, 9] in urban traffic systems. Traffic congestion in major metropolitan areas is an increasing problem as expressways cannot be built fast enough to keep traffic flowing freely at peak periods. The idea behind *variable tolling* is to issue tolls that vary according to time-dependent factors such as congestion levels and accident proximity, with the motivation of charging higher tolls during peak traffic periods to discourage vehicles from using the roads and exacerbating the congestion. Variable tolling is becoming an increasingly popular option for urban planners due to its effectiveness in reducing traffic congestion and to recent advances in microsensor technology that make it feasible. Illinois, California, and Finland have pilot programs utilizing this concept. Moreover, both London and Singapore charge tolls at peak periods to let vehicles enter the downtown area using similar reasoning.

We begin in Section 2 by presenting the unique challenges that stream data introduces in designing a benchmark and describing the ways that Linear Road addresses those challenges. In Section 3, we specify the benchmark requirements. In Section 4, we describe experiments involving two implementations of the benchmark: one using a commercially available Relational Database (which we call "*System X*"), and one using Aurora. As we will show, a dedicated SDMS can outperform a Relational Database in supporting stream data applications by at least a factor of 5.

2 Challenges

Streaming data poses the following unique challenges to the design of a benchmark:

Semantically Valid Input: Input data to a stream benchmark should not be purely random but should have some semantic validity. A typical stream presents discrete measurements of a continuous activity (e.g., the movements of soldiers). The content of a stream should be consistent with this activity. For example, if the positions of a soldier are reported every 15 minutes, the positions of two consecutive reports should not differ by more than how far a soldier can travel in that time. To ensure semantic validity, input data to a stream benchmark should be produced using *simulation*.

Continuous Query (CQ) Performance Metrics: Stream queries are predominantly continuous, and therefore the typical database benchmark metric of "completion time" is inappropriate given that such queries never complete. Instead, more appropriate metrics for streams are:

- *Response Time*: What is the average or maximum difference between the time that an input arrives to an SDMS and the time when an SDMS outputs a computed response?
- *Supported Query Load*: How much input can a stream system process while still meeting specified response times and correctness constraints?

Many Correct Results: Any benchmark implementation should be validated to ensure that it produces results consistent with the benchmark specification. However, continuous queries results may depend upon evolving historical state or the arrival order tuples on a stream, and therefore several different results for the same query may be "correct". Validation should account for queries that have multiple correct answers.

No Query Language: There exists no standard query language for streaming systems, and therefore the query requirements for a stream benchmark should be language-agnostic, yet have a clear semantics.

Linear Road has been designed to meet each of the challenges listed above. The benchmark simulates an urban expressway system where toll charges are determined dynamically. Input data consists of a stream of *position reports* and *historical query requests*. Position reports specify the location of a vehicle on an expressway and are emitted by each vehicle every 30 seconds. A historical query request is issued by a vehicle with some fixed probability every time it emits a position report.

The benchmark requires processing a set of continuous and historical queries over this input stream. In processing position reports, a system must:

- maintain statistics about the number of vehicles and average speed on each segment of each expressway on a per minute basis,
- detect accidents and alert drivers of the accidents, and
- dynamically calculate toll charges based on segment statistics and proximate accidents, and notify and assess vehicles of these charges.

In processing a historical query request, a system will report an account balance, a total of all assessed tolls on a given expressway on a given day, or an estimated travel time and cost for a journey on an expressway. Each query answer must satisfy the response time and correctness requirements specified in this document, and the throughput that a system can sustain in meeting these requirements (as measured in the number of expressways, L of input it processes) constitutes the benchmark score (its **L-Rating**).

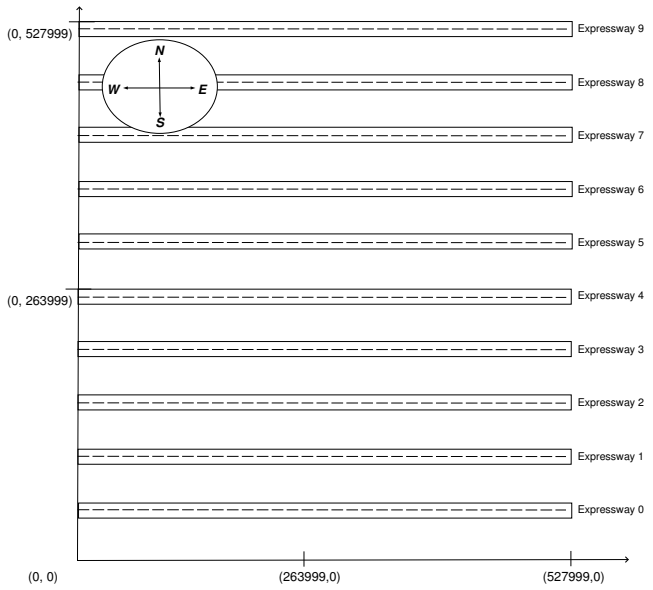

Figure 1. The Geometry of Linear City

Linear Road meets the above challenges of an SDMS benchmark:

- *Semantically Valid Input:* The input data to Linear Road is generated by the publicly available traffic simulator, MITSIM [12]. We describe the details of the simulated data in Section 3.1.1.
- *CQ Performance Metrics:* The *L-Rating* associated with Linear Road is a measure of *supported query load* in that it is a measure of the amount of input that an SDMS can process (as measured in number of expressways) while still meeting response time and correctness constraints (as specified in Section 3).
- *Many Correct Results:* For all queries that depend on some evolving state (e.g., account balance queries that depend on a table that is updated with every toll assessed), variation in response times can mean that multiple answers could be returned that are "correct". Linear Road includes two such queries, and validation for each of those considers all possible valid answers.
- *No Query Language:* All Linear Road queries are specified formally in the predicate calculus rather than a specific stream query language.

3 The Linear Road Benchmark

Linear City is a fictional metropolitan area that is the urban setting for the Linear Road benchmark. The city encompasses an area that is 100 miles wide and 100 miles long, and is divided into a grid such that the origin is the southwestern most point in the city, and coordinate (x, y)

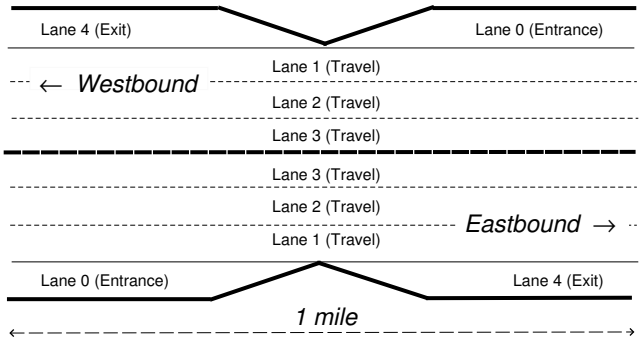

Figure 2. An Example Expressway Segment

is x feet east and y feet north of the origin. Linear City contains 10 parallel expressways numbered from 0-9 and running horizontally 10 miles apart, as illustrated in Figure 1. (For simplicity, there are no expressways that run vertically.) Each expressway has four lanes in each (east and west) direction: 3 travel lanes (lanes #1-3) and one lane devoted to entrance (lane #0) and exit (lane #4) ramps. Each expressway has 100 entrance ramps and 100 exit ramps in each direction, dividing it into 100 mile-long *segments*. Figure 2 shows an example segment.

Every vehicle in Linear City is equipped with a sensor that emits a *position report* that identifies the vehicle's exact coordinates every 30 seconds. (We assume that position reports specify coordinates with 100% accuracy.) Position reports are processed to generate statistics about traffic conditions on every segment of every expressway for every minute, including average vehicle speed, number of vehicles and existence of accidents. These statistics are used to determine toll charges for variable tolling. In addition, vehicles can issue queries to find out their current account balance with the expressway system, total tolls assessed on a given day and expressway, and travel time estimates.

For simplicity, we make the following assumptions about position reports:

1. *No Clock Skew:* A global clock is assumed as the basis for position report timestamps of all vehicles.
2. *No Position Interpolation:* The position of any vehicle at any time t is assumed to be exactly the position reported by that vehicle between times $(t - 30 \text{ sec}, t]$, or *unknown* if no position report was emitted within that range.
3. *Instantaneous Delivery:* A position report with timestamp t is made available to a stream processing system exactly t seconds after the start of the simulation. This is guaranteed by the data driver.

While simplistic, we justify these assumptions by pointing out that the purpose of the benchmark suite is to serve as a stress-test of systems performing stream processing and not to accurately model traffic patterns.

3.1 Linear Road Input

Input data for the Linear Road benchmark is generated by the MIT Traffic Simulator (MITSIM) [12] and stored in flat files. A separate *data driver* is responsible for reading these files and delivering this data in a manner simulating its arrival in real-time.

3.1.1 Simulation

Position reports are generated according to the following traffic model followed by the traffic simulator. The simulator generates a set of vehicles, each of which completes at least one *vehicle trip*: a journey that begins at an entry ramp on some segment and finishes at an exit ramp on some segment on the same expressway.[2] In making a vehicle trip, a vehicle is placed on the entrance ramp and accelerates at a rate allowed by the other traffic. It then merges onto the expressway and moves towards its destination at a rate determined by the degree of traffic congestion. When the vehicle reaches its destination, it moves to the exit ramp and decelerates. For each trip, the selected source location of a vehicle is uniformly distributed over all of the possible entrance ramps on the chosen expressway. The exit ramp is normally distributed with a mean segment location in the middle of the expressway (i.e., segment #50) and with a standard deviation of 20 miles. Hence, vehicles have an affinity for exiting in the downtown area. Once on the expressway, each vehicle proceeds according to a standard traffic spacing model built into the traffic simulator.

The simulator ensures that every vehicle emits a position report every 30 seconds, staggering them so that at every second, roughly $\frac{1}{30}$ of the reports for vehicles currently on the expressway are emitted. Every position report has a timestamp, which is an integer count of seconds since the start of the simulation. A vehicle never travels faster than 100 MPH, and therefore it will emit at least one position report from every segment it travels in. Further, every vehicle is guaranteed to average 40 MPH or less when entering and exiting an expressway and therefore it will emit at least one position report from an entrance ramp and one position report from an exit ramp for every vehicle trip.

The simulator generates one *accident* in a random location on each expressway for every 20 minutes of position reports. An accident occurs when two vehicles are "stopped" at the same position at the same time. A vehicle is stopped when it reports the same position in 4 consecutive position reports. Once an accident occurs in a given segment, traffic proceeds in that segment at a reduced speed determined by the traffic spacing model. The accident takes anywhere from 10-20 minutes to be cleared once it is detected. Until the accident clears, the vehicles involved in the accident continue to emit position reports. After either of the vehicles emits a position report that reveals that it has moved from the site of the accident, the accident is assumed to be cleared.

With 1% probability, every emitted position report is accompanied by a historical query request from the same vehicle. Of historical query requests, 50% are requests for account balances, 10% are requests for total daily tolls on a given day and expressway, and 40% are requests for travel time predictions. We specify how these historical queries should be processed in Section 3.2.3.

3.1.2 Stream Data

The stream data generated by the simulator consists of four types of tuples: *Position Reports* and historical query requests for *Account Balances*, *Daily Expenditures* and *Travel Time Estimation*.

Position Reports

A position report is a tuple of the form,

$$(\texttt{Type} = 0, \texttt{Time}, \texttt{VID}, \texttt{Spd}, \texttt{XWay}, \texttt{Lane}, \texttt{Dir}, \texttt{Seg}, \texttt{Pos})$$

such that $\texttt{Type} = 0$ identifies this tuple as a position report, $\texttt{Time}$ $(0 \ldots 10799)$[3] is a timestamp identifying the time at which the position report was emitted, $\texttt{VID}$ $(0 \ldots \texttt{MAXINT})$ is an integer *vehicle identifier* identifying the vehicle that emitted the position report, $\texttt{Spd}$ $(0 \ldots 100)$ is an integer reflecting the *speed* of the vehicle (in MPH) at the time the position report is emitted, and $\texttt{XWay}$, $\texttt{Lane}$, $\texttt{Dir}$, $\texttt{Seg}$ and $\texttt{Pos}$ are the following functions over the vehicle's (x, y) coordinates:

- $\texttt{XWay}$ $(0 \ldots L-1)$ identifies the *expressway* from which the position report is emitted
- $\texttt{Lane}$ $(0 \ldots 4)$ identifies the lane of the expressway from which the position report is emitted (0 if it is an entrance ramp ($\texttt{ENTRY}$), $1-3$ if it is a travel lane ($\texttt{TRAVEL}$) and 4 if it is an exit ramp ($\texttt{EXIT}$)).
- $\texttt{Dir}$ $(0 \ldots 1)$ indicates the direction (0 for Eastbound and 1 for Westbound) in which the vehicle is traveling when it emits its position report,
- $\texttt{Seg}$ $(0 \ldots 99)$ identifies the mile-long *segment* from which the position report is emitted, and
- $\texttt{Pos}$ $(0 \ldots 527999)$ identifies the horizontal *position* of the vehicle as a measure of the number of feet from the westernmost point on the expressway (i.e., $\texttt{Pos} = x$).[4]

[2] Note that some vehicles may not complete their vehicle trips by the end of the simulation period.

[3] There are 10800 seconds in a 3 hour simulation period.

[4] Strictly speaking $\texttt{Seg}$ is redundant given that position reports include $\texttt{Pos}$. However, several benchmark computations depend upon a vehicle's segment number, and therefore for convenience we include it in input position reports.

Historical Query Requests

A historical query request is either:

- *Account Balance*: a request for the vehicle's current account balance,
- *Daily Expenditure*: a request for the vehicle's total tolls on a specified expressway, on a specified day in the previous 10 weeks,
- *Travel Time*: a request for an estimated toll and travel time for a journey on a given expressway on a given day of the week, at a given time.

Account balance requests are tuples of the form,

$$(\texttt{Type} = 2,\ \texttt{Time},\ \texttt{VID},\ \texttt{QID})$$

such that `Type` identifies this tuple as an account balance request, `Time` is the time of the request, `VID` is the vehicle making the request, and `QID` is an integer *query identifier*. Daily expenditure requests are tuples of the form,

$$(\texttt{Type} = 3,\ \texttt{Time},\ \texttt{VID},\ \texttt{XWay},\ \texttt{QID},\ \texttt{Day})$$

such that `Type` identifies this tuple as an daily expenditure request, `Time` is the time of the request, `VID` is the vehicle making the request, `QID` is the query identifier, and `XWay` and `Day` ($1 \ldots 69$) identify the expressway and the day (1 is yesterday, 69 is 10 weeks ago) for which an expenditure total is desired. Travel time requests are tuples of the form,

$$(\texttt{Type} = 4,\ \texttt{Time},\ \texttt{VID},\ \texttt{XWay},\ \texttt{QID},\ \texttt{S}_{init},\ \texttt{S}_{end},\ \texttt{DOW},\ \texttt{TOD})$$

such that `Time` is the time of the request, `VID` is the vehicle making the request, `QID` is a query identifier, `XWay` is the expressway upon which the journey occurs (from segment $\texttt{S}_{init}$ to segment $\texttt{S}_{end}$), and `DOW` ($1 \ldots 7$) and `TOD` ($1 \ldots 1440$) specify the day of the week and minute number in the day when the journey would take place.

To avoid the complication of unpredictable event delivery order, the four types of input tuples are multiplexed together into a single stream of tuples consisting of the union of all fields. In order, these are: `Type`, `Time`, `VID`, `Spd`, `XWay`, `Lane`, `Dir`, `Seg`, `Pos`, `QID`, $\texttt{S}_{init}$, $\texttt{S}_{end}$, `DOW`, `TOD` and `Day`. Linear Road implementations can use the `Type` field to determine which fields are relevant for a given tuple.

3.1.3 Historical Data

Historical data summarizing 10 weeks worth of tolling history must be maintained by the system to answer historical query requests that refer to data dating prior to the start of the simulation. This data includes account data for all vehicles as well as toll charges and average speeds for every segment of every expressway for every minute over the previous 10 weeks. The historical data generator constructs two flat text files of comma separated values:

- File *TollHistory* consists of tuples of the form,

  $$(\texttt{VID},\ \texttt{Day},\ \texttt{XWay},\ \texttt{Tolls})$$

  such that there is one entry for every vehicle that uses an expressway during the 3 hour simulation (`VID`) for every day in the previous 10 weeks (`Day`) and every expressway (`XWay`). For every $(\texttt{VID},\ \texttt{Day},\ \texttt{XWay})$ combination, `Tolls` is the total amount in tolls spent on the expressway on day `Day` by vehicle `VID`.
- File *SegmentHistory* consists of tuples of the form,

  $$(\texttt{Day},\ \texttt{Min},\ \texttt{XWay},\ \texttt{Dir},\ \texttt{Seg},\ \texttt{Lav},\ \texttt{Cnt},\ \texttt{Toll})$$

  such that there is one entry for every day, `Day`, minute `Min`, expressway, `XWay`, direction, `Dir` and segment, `Seg`. The values of `Lav`, `Cnt` and `Toll` for each such entry reflect the average speed, number of vehicles and toll charge for the given segment on the given expressway at the given time.

Implementations of Linear Road can bulk load this data into any storage system and can do so offline so that the time for bulk loading is not included in the time to run the benchmark.

3.2 Linear Road Requirements

The Linear Road benchmark requires processing a fixed set of continuous and historical queries. These queries and their response time and accuracy requirements are discussed in detail below. Queries are described informally and specified formally in the predicate calculus.

Response time checks require that every output tuple, p, include two timestamps: one that identifies the time that p was emitted ($p.\texttt{Emit}$) and one that is the timestamp of the input tuple that triggered p to be generated ($p.\texttt{Time}$). `Emit` requires every system implementing Linear Road to invoke a system call to get the current time immediately prior to emitting p as output. `Time` is the timestamp of the input resulting in p's generation. For example, for any toll notification, p, $p.\texttt{Time}$ is the timestamp of the first position report from the same vehicle reporting its position in the segment for which the toll applies. This timestamp is generated by the simulator, and the data driver ensures that this timestamp is the time the tuple is made available to the stream processing system.

3.2.1 Toll Processing

Systems implementing Linear Road must calculate a toll every time a vehicle reports a position in a new segment, and notify the driver of this toll. Toll calculations are determined on the basis of the current congestion on the segment (as measured in terms of the number of vehicles and the average speed in the segment) as well as the proximity of accidents. We make a distinction between *toll notifications*

and *toll assessments*, which happen at different times. Every time a vehicle issues its first position report from a segment, a toll for that segment is calculated and the vehicle is notified of that toll. Every time a position report identifies a vehicle as *crossing* from one segment into another, the toll reported for the segment being exited is assessed to the vehicle's account. Thus, a toll calculation for one segment often is concurrent with an account being debited for the previous segment. If the vehicle exits at the exit ramp of a segment, the toll for that segment is not charged.

Toll Notifications

Table 1 expresses the conditions, output, recipients and response time requirements for toll notifications. The formalization is in terms of the set, P, of all position reports, and uses the following shorthand notation:

- For any position report, $p \in P$, $\overleftarrow{p}$ identifies the position report that was emitted by the same vehicle during the same vehicle trip immediately prior to p.[5] Because every vehicle emits a position report every 30 seconds during a vehicle trip, this can be defined formally as:

$$\overleftarrow{p} \;=\; q \in P \textit{ s.t.} \\ (q\,.\,\texttt{VID} = p\,.\,\texttt{VID} \;\wedge\; p\,.\,\texttt{Time} - q\,.\,\texttt{Time} = 30).$$

- Similarly for any position report $p \in P$, $\overrightarrow{p}$ identifies the position report during the same vehicle trip emitted immediately following p:

$$\overrightarrow{p} \;=\; q \in P \textit{ s.t.} \\ (q\,.\,\texttt{VID} = p\,.\,\texttt{VID} \;\wedge\; q\,.\,\texttt{Time} - p\,.\,\texttt{Time} = 30).$$

- For any vehicle identifier v and time t, $\textit{Last}_i\ (v,t)$ denotes the i^{th} position report emitted by v prior to t:[6]

$$\textit{Last}_i\ (v,t) \;=\; p \in P \textit{ s.t.} \\ (p\,.\,\texttt{VID} = v \;\wedge\; (30(i-1) \;\leq\; t - p\,.\,\texttt{Time} \;<\; 30i).$$

  For example, by the "*No Position Interpolation*" assumption, the current position of v at time t is always $\textit{Last}_1\ (v,t)$.

- For any timestamp, t (defined as an integer number of seconds since the start of the simulation), the "minute number of t" ($M(t)$) is the minute number in which t falls. That is,

$$M(t) = \lfloor \frac{t}{60} \rfloor + 1.$$

  Note that the first minute of the simulation is minute number 1.

[5] Obviously, this is undefined for the first position report of every vehicle trip.

[6] This function is defined for t and v provided that at time t, v was in the midst of a vehicle trip that began at least i position reports ago.

Trigger	Position report, q
Preconditions	$q\,.\,\texttt{Seg} \neq \overleftarrow{q}\,.\,\texttt{Seg},\ \ l \neq \texttt{EXIT}$
Output	(Type: 0, VID: v, Time: t, Emit: t' Spd: *Lav* $(M(t),x,s,d)$, Toll: *Toll* $(M(t),x,s,d)$)
Recipient	v
Response	$t' - t \leq 5$ Sec

Table 1. Toll Notification Requirements

The trigger for a toll notification to vehicle v of a charge for traveling in segment s is a position report, $q =$

$$(\texttt{Type}\text{: } 0,\ \texttt{Time}\text{: } t,\ \texttt{VID}\text{: } v,\ \texttt{Spd}\text{: } spd, \\ \texttt{XWay}\text{: } x,\ \texttt{Seg}\text{: } s,\ \texttt{Pos}\text{: } p,\ \texttt{Lane}\text{: } l,\ \texttt{Dir}\text{: } d).$$

As stated in Table 1, q triggers a toll notification if it reports that v is in a new segment since the last position report, but not in an exit lane. The tuple *output* consists of fields $\texttt{Type} = 0$ (identifying this tuple as a toll notification), VID (identifying the vehicle being notified of the toll), Time (specifying the time that q was emitted), Emit (specifying the time the toll notification is emitted), Speed (specifying the 5-minute average speed in the segment) and Toll (specifying the calculated toll). The *recipient* of the notification is v, and the *response* time requirement is 5 seconds between the time the position report was emitted (t) and the time the toll notification is sent (t').

The values calculated for fields Spd and Toll are expressed in terms of the functions, *Lav* and *Toll* defined in Table 2. *Lav* (short for "Latest Average Velocity") computes the average speed on some expressway x, segment s and direction d by averaging vehicle speeds over the 5 minutes that precede minute $m = M(t)$. Minute averages are expressed with the function *Avgs* (m,x,s,d) that specifies the average speed of all vehicles that emitted a position report from segment s of expressway x in direction d during minute m. Note that some vehicles might emit two position reports during this minute. This is accounted for in *Avgsv* (v,m,x,s,d) which calculates the average speed of vehicle v according to all of the position reports it emits during minute m. Finally, *cars* (m,x,s,d) returns the set of all vehicles that emit position reports from segment s on expressway x while traveling in direction d during minute m. Note that we use the notation, $p\,.\,(\texttt{XWay};\texttt{Seg};\texttt{Dir}) = (x;s;d)$ as shorthand for

$$p\,.\,\texttt{XWay} = x \;\wedge\; p\,.\,\texttt{Seg} = s \;\wedge\; p\,.\,\texttt{Dir} = d,$$

and use $\{\!|\ldots|\!\}$ to denote the contents of a bag.

By default, the value of Toll at time t for a segment is based on the average speed and number of vehicles reporting from the segment during minute $M(t)-1$. Specifically, if the LAV for the time interval from minute $M(t)-5$ to $M(t)-1$ is greater than or equal to 40 MPH, or if the number of vehicles on the segment ($numvehicles$) was 50 or less during minute $M(t)-1$, no toll is assessed. Otherwise,

the default toll is determined by the formula,

$$2 \times (numvehicles - 50)^2.$$

The basic intuition is to raise tolls when congestion is high so as to discourage drivers from contributing to worse congestion.

The toll calculation described above is issued for segment s unless an *accident* was detected 0-4 segments downstream of s as of minute $M(t)$. In this case, no toll is charged. Accident detection is discussed in Section 3.2.2.

Toll Assessments

Every time a position report identifies a vehicle as *crossing* from one segment into the next, the toll charge quoted to the vehicle when it first entered the segment that it is now leaving is assessed to the vehicle's account. Systems implementing Linear Road must keep track of all tolls assessed so that it can answer *Account Balance* queries that report the current balance of a vehicle, *Daily Expenditure* queries that report the total tolls assessed on a given expressway on a given day for a given vehicle, and *Travel Time Estimation* queries that use previous toll charges to estimate tolls for given segments on future days and times.

3.2.2 Accident Processing

Systems implementing Linear Road must detect accidents on the expressways as they occur (*detection*), and subsequently alert all vehicles in the vicinity (*notification*). As was discussed in Section 3.2.1, accident detection should also result in a reduction in tolls that are assessed within 5 segments upstream.

Accident Detection

A stream processing system should detect an *accident* on a given segment whenever two or more vehicles are *stopped* in that segment at the same lane and position. A vehicle is considered *stopped* if four consecutive position reports from this vehicle come from the same position (i.e., the same expressway, lane, position and direction). This is expressed formally in Table 3 with the predicates *Stop* and *Acc* . Predicate *Stop* (v, t, x, l, p, d) holds if the four most recent positions reports from v as of time t are from the same location. Predicate *Acc* (t, x, p, d) holds if there were two vehicles stopped as of time t at the same position p of expressway x in direction d.

Accident Notification

Once an accident is detected, every vehicle that enters into a segment in the vicinity of the accident must be notified so that these vehicles have the opportunity to exit the expressway and avoid the resulting congestion. The exact requirements for accident notification are summarized in Table 4.

Trigger	Position report, q
Precondition	$\exists_{s',0\leq i\leq 4}\ (s' = \mathit{Dn}\ (q\,.\,\texttt{Seg}, d, i) \wedge$ *Acc_in_Seg* $(M(t)-1, x, s', d))$, $q\,.\,\texttt{Seg} \neq \overleftarrow{q}\,.\,\texttt{Seg},\ l \neq \texttt{EXIT}$
Output	(Type: 1, Time: t, Emit: t', Seg: s')
Recipients	v
Response	$t' - t \leq 5$ Sec

Table 4. Accident Alert Requirements

The trigger for an accident notification is a position report $q =$

$$(\texttt{Type}: 0,\ \texttt{Time}: t,\ \texttt{VID}: v,\ \texttt{Spd}: spd,\\ \texttt{XWay}: x,\ \texttt{Seg}: s,\ \texttt{Pos}: p,\ \texttt{Lane}: l,\ \texttt{Dir}: d),$$

that identifies a vehicle entering a segment 0 to 4 segments upstream of some accident location, but only if q was emitted no earlier than the minute following the minute when the accident occurred, and no later than the minute the accident is cleared. This is expressed using the predicate

$$\mathit{Acc\_in\_Seg}\ (m, x, \mathit{Dn}\ (s, d, i))$$

that holds if there was an accident in the segment that is exactly i segments downstream of s, in expressway x and in the travel lanes for direction d during minute m.[7] The tuple *output* consists of the fields, $\texttt{Type} = 1$ (identifying this tuple as an accident alert), Time (specifying the time that q was emitted), Emit (specifying the time the notification is emitted), and Seg (specifying the segment where the accident occurred). The *response* time requirement is 5 seconds between the time that q was emitted (t) and the time the accident notification is sent (t').

Note that for a given accident, multiple accident notifications may be sent to the **same vehicle** if that vehicle does not exit the expressway and instead enters segments bringing it closer to the site of the accident. Repeated notifications are intentional, as this allows for vehicles that enter one of these segments *after* the accident occurs to be notified of the accident. Also, once a vehicle stops receiving accident notifications, it can assume that it has either passed the accident location or that the accident has been cleared.

3.2.3 Historical Query Processing

Aside from the continuous queries involving toll and accident notifications, systems implementing Linear Road must also be able to respond to historical query requests issued by vehicles. There are three types of historical queries. These are described below.

Account Balance Queries

A customer traveling on some expressway can request his account balance at any time. At the start of the simulation,

[7] The segment that is i segments downstream of s *Dn(s,d,i)* is MIN $(s + i, 99)$ if the direction is eastbound ($d = 0$) and MAX $(s - i, 0)$ otherwise.

$$
\begin{array}{rcl}
\textit{cars}\ (m,x,s,d) & = & \{p\,.\,\texttt{VID} \mid p \in P,\ m\ =\ M(p\,.\,\texttt{Time}),\ p\,.\,(\texttt{XWay};\texttt{Seg};\texttt{Dir}) = (x;s;d)\} \\
\textit{Avgsv}\ (v,m,x,s,d) & = & \mathsf{AVG}\ (\{\!|p\,.\,\texttt{Spd} \mid p \in P,\ p\,.\,\texttt{VID} = v,\ m\ =\ M(p\,.\,\texttt{Time}),\ p\,.\,(\texttt{XWay};\texttt{Seg};\texttt{Dir}) = (x;s;d)|\!\}) \\
\textit{Avgs}\ (m,x,s,d) & = & \mathsf{AVG}\ (\{\!|\textit{Avgsv}\ (v,m,x,s,d) \mid v \in \textit{cars}\ (m,x,s,d)|\!\}) \\
\\
\textit{Lav}\ (m,x,s,d) & = & \lfloor \mathsf{AVG}\ (\{\!|\textit{Avgs}\ (m-1,x,s,d),\ldots,\textit{Avgs}\ (m-5,x,s,d)|\!\})\rfloor \\
\\
\textit{Toll}\ (m,x,s,d) & = & \begin{cases}
2\cdot(|\textit{cars}\ (m-1,t,x,s,d)|-50)^2 \\
\quad \text{if } \textit{Lav}\ (m,x,s,d) < 40 \text{ and} \\
\quad |\textit{cars}\ (m-1,x,s,d)| > 50 \text{ and} \\
\forall_{0\le i\le 4}\ (\neg(\textit{Acc\_in\_Seg}\ (m-1,x,\textit{Dn}\ (s,d,i)))) \\
0, \quad \text{otherwise}
\end{cases}
\end{array}
$$

Table 2. Notation Used to Define Tolls

$$
\begin{array}{rcl}
\textit{Stop}\ (v,t,x,l,p,d) & \Leftrightarrow & \forall_{1\le i\le 4}\ (\textit{Last}_i\ (v,t)\,.\,(\texttt{XWay};\texttt{Lane};\texttt{Pos};\texttt{Dir}) = (x;l;p;d)) \\
\textit{Acc}\ (t,x,p,d) & \Leftrightarrow & \exists_{v_1,v_2,l}\ (l\ =\texttt{TRAVEL}\ \wedge\ v_1 \neq v_2\ \wedge\ \textit{Stop}\ (v_1,t,x,l,p,d)\ \wedge\ \textit{Stop}\ (v_2,t,x,l,p,d)) \\
\textit{Acc\_in\_Seg}\ (m,x,s,d) & \Leftrightarrow & \exists_{p,t}\ (t \in m\ \wedge\ \textit{Acc}\ (t,x,p,d)\ \wedge\ \lfloor \frac{p}{5280} \rfloor = s)
\end{array}
$$

Table 3. Notation Used to Define Accidents

every vehicle's account balance is zero, and thereafter the account balance at time t is the sum of all tolls assessed as of t. The requirements for account balance historical queries are summarized in Table 5. A historical query to return an account balance for a given vehicle is triggered by a request tuple $a =$

$$(\texttt{Type}\text{: } 2,\ \texttt{Time}\text{: } t,\ \texttt{VID}\text{: } v,\ \texttt{QID}\text{: } q).$$

The tuple *output* consists of the fields, $\texttt{Type} = 2$ (identifying this tuple as an account balance), `Time` (specifying the time that a was emitted), `Emit` (specifying the time the query response is emitted), `QID` (identifying the query that issued the request), `Bal` (the account balance calculated), and `ResultTime` (the time at which `Bal` was last updated). The balance is the sum of all tolls that were charged to the vehicle's account. This is expressed in terms of *tollset*: the set of all position reports that resulted in a toll charge being assessed. A subset of the position reports that generated alerts, *tollset* (v) consists of those position reports issued from some segment (s) whose subsequent position reports indicated that the vehicle did not exit the expressway from segment s. More formally,

$$
\begin{array}{l}
\textit{tollset}\ (v)\ = \\
\{p \in P \mid p\,.\,\texttt{VID} = v,\ p\,.\,\texttt{Seg} \neq \overrightarrow{p}\,.\,\texttt{Seg}, \\
\ p\,.\,(\texttt{XWay};\texttt{Dir}) = (x;d)\}.
\end{array}
$$

Function *Toll* specifies the toll calculation as described in Table 2.

That Linear Road requires answering account balance queries means that the tolls charged to each vehicle must be maintained in a timely fashion. Thus, the most substantial overhead resulting from inclusion of this historical query comes not from the cost of answering it but from the cost of maintaining the data required to answer it. The *response* time requirement is 5 seconds from the time the historical query request is issued to the time a response is emitted. The *accuracy* requirement specifies that the returned balance must have been accurate at some time, τ, in the 60 seconds prior to the time when the account balance request is issued. (Given that tolls can be issued at most once per emitted position report, this means the query has up to 3 possible correct answers.) This interval gives the stream processing system some flexibility as to when to update the balance of a vehicle as a result of assessing a toll. If a query request at time t is concurrent with some toll charges that have yet to be assessed to a vehicle's account, the system might choose to process the historical query before updating the account balance (potentially producing a result that is accurate for some time $\tau < t$), or waiting until the tolls have been assessed.

Trigger	Account balance request, a
Condition	-
Output	(`Type`: 2, `Time`: t, `Emit`: t', `ResultTime`: τ, `QID`: q, `Bal`: $\sum_{\substack{p\ \in\ \textit{tollset}(v),\\ p.\texttt{Time} \le \tau,\\ p\,.\,\texttt{Seg} \neq \textit{Last}_1\ (v,t).Seg}} (f(p))$ *s.t.* $f(p) =$ $\textit{Toll}\ (M(p\,.\,\texttt{Time}), p\,.\,\texttt{XWay}, p\,.\,\texttt{Seg}, p\,.\,\texttt{Dir})$
Recipient	v
Response	$t' - t \le 5$ Sec
Accuracy	$\tau \ge t - 60$ Sec

Table 5. Account Balance Requirements

Daily Expenditure Queries

A second historical query that can be issued in Linear Road is one that requests the sum of tolls spent on some express-

Trigger	Daily Expenditure request, d
Condition	-
Output	(Type: 3, Time: t, Emit: t', QID: q, Bal: $\sum_{p \in \textit{tollset}(v), \textit{Day}(p.\texttt{Time}) = d, p.\texttt{XWay} = x} (f(p))$ *s.t.* $f(p) =$ *Toll* $(M(p.\texttt{Time}), p.\texttt{XWay}, p.\texttt{Seg}, p.\texttt{Dir}))$
Recipients	v
Response	$t' - t \leq 10$ Sec

Table 6. Daily Expenditure Requirements

way on some day in the last 10 weeks (not including the current day or any day which ended within 5 minutes of t). The requirements for daily expenditure historical queries are summarized in Table 6. A historical query to return an account balance for a given vehicle is triggered by a request tuple $d =$

$$(\texttt{Type}: 3,\ \texttt{Time}: t,\ \texttt{VID}: v,\ \texttt{QID}: q,\ \texttt{XWay}: x,\ \texttt{Day}: n).$$

The tuple *output* consists of the fields, $\texttt{Type} = 3$ (identifying this tuple as a daily expenditure report), Time (specifying the time that d was emitted), Emit (specifying the time the query response is emitted), QID (identifying the query that issued the request), and Bal which is the account balance calculated. The value of Bal is the sum of all tolls from expressway x on day n that were charged to the vehicle's account.

To be able to respond to daily expenditure queries, systems implementing Linear Road must maintain 10 weeks worth of toll data per vehicle and expressway. Given the approximately 150,000 vehicles generated in a 3 hour simulation, this amounts to $150,000 \cdot L \cdot 70$ (roughly between 10 million and 100 million) rows.

Travel Time Estimation Queries

A historical query to return a time travel estimate is triggered by a request tuple $z =$

$$(\texttt{Type}: 4,\ \texttt{Time}: t,\ \texttt{VID}: v,\ \texttt{QID}: q,\ \texttt{XWay}: x,\ S_{init}: i,\ S_{end}: e,\ \texttt{DOW}: d,\ \texttt{TOD}: y)$$

In response, the system responds with a tuple of the form,

$$(\texttt{Type} : 4,\ \texttt{QID} : \texttt{q},\ \texttt{TravelTime} : \texttt{r}_1,\ \texttt{Toll} : \texttt{r}_2)$$

such that $\texttt{r}_1$ and $\texttt{r}_2$ are respectively, the predicted travel time and toll charge for the vehicle journey calculated on the basis of statistics maintained over the previous 10 weeks in the manner described below.

Let z be a request for a travel time and toll charge estimate for a journey from segment i to segment e on expressway x starting on day d and time y. Now, let t_j be the *expected arrival time* at segment j ($i \leq j \leq e$). Then,

$$\begin{aligned} y_i &= y, \text{ and} \\ y_{j+1} &= y_j + tav(x, j, d, t_j)\ (i < j \leq e) \end{aligned}$$

where $tav(Xway, Seg, DOW, TOD)$ computes the expected travel time from average vehicle speed for 10 weeks of data, for a given expressway, segment, day and time. Then, the expected travel time, $r_1 = y_e$ and the expected toll is

$$r_2 = \sum_{j=1}^{e-1} cav(x, j, d, y_j)$$

where $cav(Xway, Seg, DOW, TOD)$ computes the expected toll from average vehicle speed and number of vehicles for a given expressway, segment, day and time, using Table 2.

Systems implementing Linear Road must maintain 10 weeks worth of statistical data for each segment on the Linear Road expressways. The data that must be maintained for each of the $L \times 200$ segments includes a count of the number of vehicles in the segment and the *Lav* . Note that 10 weeks of historical data at 1 minute granularity for every segment requires maintaining $200 \cdot L \cdot 10 \cdot 7 \cdot 24 \cdot 60$ (roughly between 20 million and 200 million) rows. The response time requirement for the travel time query is 30 seconds.

3.3 Running the Benchmark

Aside from this document, the Linear Road benchmark web site [7] makes available four tools to assist researchers in the implementation of Linear Road:

- A *historical data generator* that generates a set of flat files containing historical toll data summarizing tolling activity over the 10 weeks prior to the simulation run,
- A *traffic simulator* (based on MITSIMLab [12]) that generates a set of flat files containing streaming input data for the benchmark,
- A *data driver* that delivers the data generated by the traffic simulator to a system in real-time, and
- A *validator* that verifies the correctness of query results as well as ensuring that response-time requirements are met.

The purpose of the benchmark is to determine the L-rating of a stream processing system: the maximum *scale factor* at which the system can respond to the specified set of continuous and historical queries while meeting their response time and accuracy requirements. It is assumed that the benchmark will be run with increasingly larger scale factors until one is found for which the requirements cannot be met. Once the queries are formulated in a given system, the benchmark is executed according to the steps below:

1. The *historical data generator* is run to generate flat files consisting of 10 weeks worth of historical data. Offline, this data can be loaded into the system's storage facility of choice.

2. The *traffic simulator* is run to generate L flat files, each of which consists of 3 hours of traffic data and historical query requests from vehicles reporting from a single expressway during rush hour. The *data driver* is then invoked to deliver this data in a manner simulating its arrival in real-time.

3. The system running the benchmark is configured to generate a flat file containing all output tuples (with timestamps reflecting the times of their generation and the times of the input tuples that triggered their generation) in response to the queries defined in the benchmark.

4. The *validation tool* is used to check the response times and accuracy of generated output to see if they meet the requirements of the benchmark.

A system achieves an *L-rating* for the benchmark if it meets its response time and accuracy constraints while supporting L expressways worth of input.

Systems implementing Linear Road must direct their output into a single flat file. Validation involves comparing the system's output with that generated as a reference set by the validation tool for the given input. The validation tool will read output from the flat files generated by the stream system and check the results to see if they meet the response time and accuracy requirements described previously. It is expected that most systems will produce accurate output, but will for some scale factor, be unable to continue meeting the response time constraints. When reporting its *L-rating*, a system should also specify the hardware configuration over which it ran.

4 Implementations & Experiments

In this section, we describe two implementations of the Linear Road benchmark and compare their relative performance. The first implementation is over a pre-release commercialization of Aurora [1] and is described in Section 4.1. The second implementation is over a commercially available Relational Database (System X) and is described in Section 4.2. Both systems were run on the same 3 GHz Pentium box with 2 GB RAM and running Linux. We compare the performance of these two implementations in Section 4.3 and show that a dedicated stream processing engine can outperform a Relational Database for streaming data applications (as measured in their respective scale factors) by a factor of 5.[8]

[8]The Aurora system we use in the benchmark is still a pre-Beta version of the commercial product, and we anticipate that this improvement factor will increase as the product matures.

Of the queries included in the benchmark, the *Travel Time Estimation* query is by far the most complex and difficult to express. Neither of the benchmark implementations described below supports this query and requests in the input for this type of query are ignored.

4.1 Linear Road in Aurora

Aurora uses a workflow-like boxes-and-arrows model for constructing queries over stream data [2]. The Aurora implementation of Linear Road consists of a *query network* of roughly 60 boxes and the following shared tables:

- *Vehicle Information:* Including, for every vehicle, such things as its last known location (expressway # + position + direction) and its account balance,

- *Stopped Cars:* Including all locations where cars are stopped and the cars involved,

- *Accidents:* Including all segments in close proximity to an accident and the time of the accident,

- *Segment Statistics:* Including for every segment of every expressway, and for every minute in the last 5 minutes, and

- *Toll History:* For every vehicle, expressway and day over the previous 10 weeks, the total tolls spent on the expressway.

Historical query requests are each handled separately from position reports, and each require 1-2 boxes to read from the appropriate tables (*Vehicle Information* for the account balance query and *Toll History* for the daily expenditure query) and process the results.

Position reports are processed by three consecutive subnetworks of the query network:

1. *Subnetwork #1* is responsible for detecting and recording when cars are stopped, and if for the ones that are, if they are in an accident. This subnetwork reads and writes the *Stopped Cars*, *Accidents* and *Vehicle Information* tables.

2. *Subnetwork #2* is responsible for maintaining statistics for every segment of every expressway with 1 minute granularity. This subnetwork reads from the *Accidents* table and writes to the *Segment Statistics* table.

3. *Subnetwork #3* is responsible for calculating and emitting tolls for those position reports that show a vehicle that has crossed into a new segment, and for emitting accident alerts for those position reports that show that the vehicle has entered a segment within 5 segments upstream of a recent accident. This subnetwork reads from the *Segment Statistics* table and emits results (toll notifications and accident alerts) to an output stream.

The subnetworks listed above are connected in sequence. Synchronization primitives between them ensure that no position report with timestamp t is processed by Subnetwork #3 before all position reports with timestamps of t - 1 minute or less have been processed by Subnetworks #1 and #2. The box-at-a-time scheduler of Aurora [3] is constrained only by these primitives and the availability of inputs to boxes when deciding what boxes are eligible to be scheduled.

4.2 Linear Road in System X

We built two implementations of Linear Road over System X. The first is a *trigger-based* implementation that uses the built-in trigger facility of System X to process position reports and historical query requests as they arrive. The second is a *polling-based* implementation that uses a data driver written in Perl to preload a dedicated relation with a second's worth of position reports every second and subsequently invoke a System X stored procedure. For both implementations, recovery logging was turned off. Because the polling-based implementation allows for batch processing of position reports, sensitivity analysis showed that it performed much better than the trigger-based implementation and therefore, only the polling-based implementation is presented here.

The Linear Road implementation over System X has much the same structure as the Linear Road implementation over Aurora. A stored procedure of roughly 300 lines of queries and accompanying code, this implementation also uses tables to store *vehicle information*, *stopped cars*, *accidents*, *segment statistics* and *toll history*. As well, there is an additional table to hold all input tuples delivered by the driver in the last second, and an additional table to receive the output results.

As with Aurora, historical queries are handled separately (with simple SQL queries). Position reports follow the same sequence of processing as with Aurora: first detect accidents; then generate statistics and calculate and emit toll and accident alerts. As much as possible, tuples are processed in batch mode. For example, after the arrival of a minute's worth of position reports, a query is run over these position reports to determine all segment statistics for that minute.

4.3 Results

In this section, we present experimental results from running the Aurora and System X implementations of Linear Road with varying numbers of expressways.

4.3.1 Scale Factor

Table 7 shows the L-factors achieved by Aurora and System X running Linear Road. An expressway's worth of input data consists of roughly 12 million position reports, 60,000 account balance query requests and 12,000 daily expenditure requests delivered in 3 hours. The corresponding output consists of roughly 2 million toll alerts and 28,000 accident alerts (as well as one historical query output for each historical query request). Therefore, on average, the System X implementation processed roughly 560 input tuples per second (delivering an average throughput of 100 tuples per second) while meeting the response time requirements of Linear Road, while Aurora processed roughly 2800 input tuples per second (delivering an average throughput of 486 tuples per second) for a factor of 5 performance gain.[9]

System X	*Aurora*
0.5	**2.5**

Table 7. L-Ratings for Linear Road

Table 8 shows the maximum response times for toll notifications for every run of System X and Aurora. For any given run, these numbers show the highest response time for an output toll notification such that the response time for any output q is equal to $q.\texttt{Emit} - q.\texttt{Time}$. Note that because timestamps are in the granularity of a second, the reported response times may be off by up to a second. That is, a response time of k calculated in this way indicates that the actual response time is some time, t, such that $k - 1$ Sec $< t < k + 1$ Sec.

One can observe from this table that when either system first fails to meet the benchmark requirements for some number of expressways, it fails substantially. Aurora succeeds with 2.5 expressways but has a worst-case response time of roughly 3 minutes with 3 expressways. System X succeeds with 0.5 expressways but has a worst-case response time of roughly 33 minutes for 1.0 expressway. The degree to which a system fails depends on how early during the 3 hour run the system first starts to fall behind (i.e., the first input that fails to meet the response time requirements). When this occurs, it must be the case that input tuples are being backed up on the input queue and soon it becomes the case that response time requirements fail before processing even begins for these inputs. Aurora first fails with 3 expressways in processing an input position report with timestamp, 7931 (roughly 2.3 hours into the benchmark), and therefore tuples are only accumulating in the input queue for the last 40 min or so of the run. System X first fails with 1 expressway in processing an input position report with timestamp, 4761 (roughly 1.3 hours into the benchmark), and therefore tuples are accumulating in the input queue for the last hour and 40 minutes of the run. Because System X fails earlier in its run, its worst-case response time is much higher. Note that when run with 1.5 and 2 expressways, System X fails even sooner and with many more input tuples idling in the input queues, and reports worst-case response times of roughly 4.5 and 14.5 hours respectively.

[9] Because System X was unable to meet the benchmark requirements for 1 expressway, results were generated at the granularity of half of an expressway.

XWays	System X	Aurora
0.5	3	1
1.0	2031	1
1.5	16346	1
2.0	52443	2
2.5	-	2
3.0	-	196

Table 8. Max Response Times for Tolls (Sec)

4.3.2 Discussion

Our results suggest that a dedicated SDMS can outperform a Relational Database system in processing stream data by at least a factor of 5. If the response time requirements were made more strict (e.g., 3 seconds for toll alerts rather than 5 seconds), then Aurora still meets the requirements for 2.5 expressways, but System X may then fail to meet the response time requirements for 0.5 expressways. Unfortunately, the 1 second granularity of timestamps we used stopped us from confirming this result in time for the paper deadline, and thus only the factor of 5 improvement is known with certainty.

The purpose of this benchmark is to stimulate creative thought on how to meet the challenges of large scale streaming data applications. To this end, the goal of our initial experiments described above, was to see how a stream data management system that was architected for exactly these kinds of applications, would compare to a relational database system that was configured to process queries in response to pushed data. Our numbers suggest that a dedicated SDMS is far better suited for stream data applications than a relational database. We readily acknowledge that our implementation of Linear Road in a Relational Database may not be optimal, and so we invite others to implement this benchmark and report their numbers.

5 Conclusions

This paper presents *Linear Road*: a benchmark and accompanying toolkit for comparing the capabilities of systems that perform stream data management. Linear Road is inspired by the increasing prevalence of "variable tolling" in highways systems throughout the world. Based on a fictional urban area with a simple geometry, Linear Road simulates a traffic monitoring system that maintains current and historical statistics over each 1 mile segment of each expressway, detects and alerts drivers of accidents, calculates tolls based on segment statistics, accidents and frequency of use, and supports historical queries that report account balances, daily expenditures and predicted travel times and tolls.

After outlining the challenges in formulating a stream data benchmark in Section 2 and describing the benchmark itself in Section 3, we described two implementations of Linear Road: one using a commercially available Relational Database system ("System X"), and the other using a prerelease commercialization of Aurora. Our experimental results showed that Aurora has an L-factor of 2.5, whereas System X has an L-factor of 0.5, thus showing a factor of 5 performance gain resulting from using a dedicated Stream Data Management System to process stream data. In fact, the performance gain is likely higher than this, but time constraints before the paper deadline did not allow us to refine our time precision to establish this for certain.

Beyond serving as a basis for comparison, the purpose of this benchmark is to stimulate creative thought in the design of Stream Data Management Systems. We invite others to run the benchmark on their own systems and contribute to this discussion.

References

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 12(3), August 2003.

[2] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB '02)*, pages 215–226, Hong Kong, China, August 2002.

[3] D. Carney, U. Cetintemel, A. Rasin, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Operator Scheduling in a Data Stream Manager. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB '03)*, pages 838–849, Berlin, Germany, September 2003.

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In M. Stonebraker, J. Gray, and D. DeWitt, editors, *Proceedings of the 1st Biennial Conference on Innovative Database Research (CIDR)*, Asilomar, CA, January 2003.

[5] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In J. N. Weidong Chen and P. Bernstein, editors, *Proceedings of the Special Interest Group on Management of Data (SIGMOD)*, Dallas, Tx, June 2000.

[6] Conjestion Pricing: A Report From Intelligent Transportation Systems (ITS, May 2002. URL: www.path.berkeley.edu/ leap/TTM/Demand_Manage/pricing.html.

[7] The Linear Road Benchmark Website, 2004. URL: http://www.cs.brandeis.edu/linearroad/.

[8] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olsten, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In M. Stonebraker, J. Gray, and D. DeWitt, editors, *Proceedings of the 1st Biennial Conference on Innovative Database Research (CIDR)*, Asilomar, CA, January 2003.

[9] R. W. Poole. HOT Lanes Prompted by Federal Program, November 2002. URL: http://www.rppi.org/federalhotlanes.html.

[10] STREAM project page for Linear Road Benchmark, 2004. URL: http://www-db.stanford.edu/stream/cql-benchmark.html.

[11] A Guide for Hot Lane Development: A U.S. Department of Transportation Federal Highway Administration White Paper. , March 2003. URL: www.itsdocs.fhwa.dot.gov/JPODOCS/REPTS_TE/13668.html.

[12] Q. Yang and H. N. Koutsopoulos. A Microscopic Traffic Simulator For Evaluation of Dynamic Traffic Management Systems. *Transportation Research C*, 4(3):113–129, June 1996.