

CS848 Presentation Report

(Aurora: a new model and architecture
for data stream management)

Qian (Kevin) Chen

Student number: 20191995

E-mail: q3chen@cs.uwaterloo.ca

Feb 06, 2006

Department of Computer Science

University of Waterloo

Waterloo, Ontario, Canada

1 Introduction

Traditional DBMSs focus on business data processing and are designed to meet the business requirements. This paper introduces one new system, Aurora, which is used to manage data streams for monitoring applications. The question comes if there are any particular reasons why we need a new system instead of using the traditional DBMSs. This paper gives five reasons as following:

(1). The basic computation model is wrong. Traditional DBMSs are a *human-active, DBMS-passive (HADP)* model. People tend to think a DBMS as a passive repository storing a large amount of data elements. They try to pull out the data when they need them by querying on the DBMS. Aurora system, on the other hand, is a *DBMS-active, human-passive (DAHP)* model. The monitoring application gets the data from external data source (e.g. Sensors). The system notifies humans when abnormal activity is detected.

(2). It is a challenge for Traditional DBMSs to store time-series information. Traditional DBMSs only care about current value. Monitoring applications require not only current records but also some history of values reported in a data stream. One example is to track the location of items, such as the laptops attached with electronic property stickers. If we use Traditional DBMSs to store time-series information, we have two choices: (a) Encoding the time series as current data in normal tables. In order to get the history data, we need to require the data over tables. Therefore, it slows down the performance. (b) Encoding time series information in binary large objects to achieve physical locality. It is expensive to find the individual values with time-series information.

(3). "Most monitoring applications are trigger-oriented."⁽¹⁾ Traditional DBMSs treat triggers and alerters as second-class citizens. They cannot scale a large amount of triggers per table.

(4). No traditional DBMSs can answer approximate query. DBMSs assume that data are synchronized. Therefore, the querying results are exact. But in stream-oriented applications, data arrive asynchronously and are often lost, stale, or intentionally dropping for processing reasons (e.g. load shed).

(5). Monitoring applications have real-time requirements. DBMSs assume that the application has no real-time requirements.

For the above reasons, monitoring applications can not be implemented by using traditional DBMSs. The characteristics of monitoring applications are streams of information, triggers, imprecise data, and real-time requirements. There exists a large class of such applications. (e.g. monitoring applications for physical facilities and GPS) . Currently the technology can only used on costly items like automobiles due to the high cost of such monitoring applications.

2 Aurora system model

There are two kinds of data sources in Aurora system. One comes from computer programs that generate values at regular or irregular intervals. The other is from hardware sensors. Each data source has a unique source identifier. Aurora system gives a timestamp for incoming tuple. A data stream is made up with several such data sources. They look like as following:

StockID, Time, Price	ID, Time, Position
(MSF, 1:00, \$20)	(1, 2:00, 3)
(IBM, 1:00, \$16)	(2, 2:00, 5)

Aurora system is mainly used to process incoming data streams in the way defined by an application administrator. The fundamental elements in Aurora system are boxes and arrows that

are similar to those in process flow and work flow systems. Boxes represent processing operations. Data stream, in terms of tuples, flow through a loop-free, directed graph. Output streams are sent to applications, which must be implemented to process the asynchronous tuples in an output stream. Arrows represent a collection of streams with common schema. Fig.1 illustrated the high-level system model.

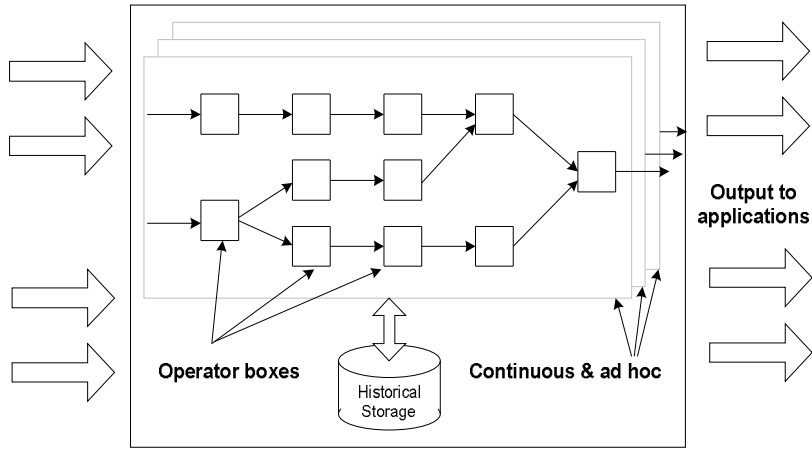


Fig.1. Aurora system model

Source: D.J. Abadi et al.: Aurora: a new model and architecture for data stream management

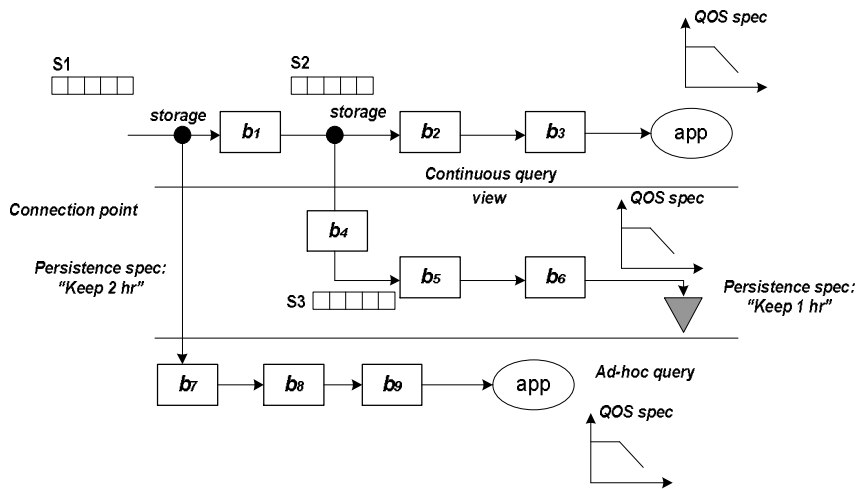


Fig.2. Aurora query model

Source: D.J. Abadi et al.: Aurora: a new model and architecture for data stream management

2.1 Query model

Each mode processes flows based on QoS (Quality-of-service) *specifications* – each output in Aurora system is related with two-dimensional QoS graphs that indicate the utility of the output in terms of several performance-related and quality-related attributes.

Aurora supports three query models that all use the same building blocks. As we can see on the Fig.2, the topmost path is a *continuous query*. As data stream flows on boxes, it has been processed and no data elements are stored.

The dark circles on the input arcs to boxes *b1* and *b2* represent *connection points*. A connection point support dynamic modification to the network. The new boxes can be added or deleted from the connection point. A connection point can also cache data streams passed on it as persistent storage for a specified period. In Fig 2, the period is 2 hours.

The middle path in Fig.2 represents a *view*. You can only see the path without the connected application. *QoS specifications* are also used to measure the importance of the view. An application can connect to the end of path at any time.

The bottom path represents an *ad hoc query*. An ad hoc query can initiate a search for a connection point and then attach itself to this connection point. An ad hoc query tries to find the earliest time T stored in the connection point and give the answers from T until the query is done.

3 Aurora optimization

In traditional DBMSs, the main objective of query optimization is to minimize the number of iterations. Aurora system has different strategies to optimize queries.

- (1). Computation requirements. Even if the amount of computation is very small for each operator to process those data stream, it is possible to have a large quantity of boxes.
- (2). Data rates may be high sometime.
- (3). Dynamic changes may occur over time. The query optimization has to be done without offline.

3.1 Dynamic continuous query optimization

At the beginning, the user constructs an unoptimized Aurora network. Aurora system gathers runtime statistics such as the average cost of box execution and box selectivity during execution. Instead of optimizing the whole network, it selects a portion of the network to optimize. Furthermore, it will find all connection points that surround the subnetwork to be optimized.

- (1). *Inserting projections* is done not by application administrator but the operators. The optimizer will use operator signatures that describe the attributes that are used and produced by the operators.
- (2). *Combining boxes* is to combine two boxes into a single box to reduce some cost.
- (3). *Reordering boxes* is to change the orders of the boxes.

3.2 Ad hoc query optimization

Aurora processes ad hoc queries in two steps by constructing two separate subnetworks. Each is attached to a connection point. The historical subnetwork runs first. The initial boxes in an ad hoc query pull information from the B-tree. Each node of the B-tree corresponds to the connection points. When the historical operation is done, Aurora changes mode to push-based mode to continue processing.

4 Run-time operation

“The basic purpose of an Aurora run-time network is to process data flows through a potentially large workflow diagram.”⁽¹⁾ Fig.3 illustrates the basic Aurora architecture. The storage manager maintains the box queues and managing the buffer. The scheduler picks a box for execution, finds what processing is required, and passes a pointer to the multithreaded box processor. The QoS monitor and the load shedder work together. The QoS monitor continually monitors system performance and triggers the load shedder when it detects an overload situation and poor system performance. The load shedder then sheds load until the performance of the system reaches an acceptable level.

4.1 QoS data structures

There exist the three criteria to decide the QoS in Aurora system. They are *Response*, *Tuple drops*, and *Values produced*.

4.2 Storage management

Aurora Storage Manager (ASM) is used to store all tuples needed by Aurora network. There are two kinds of storage management. They are *Queue management* and *Connection point*

management. Queue management manages storage for those tuples passed through an Aurora network. Connection point management arranges the storage for connection points.

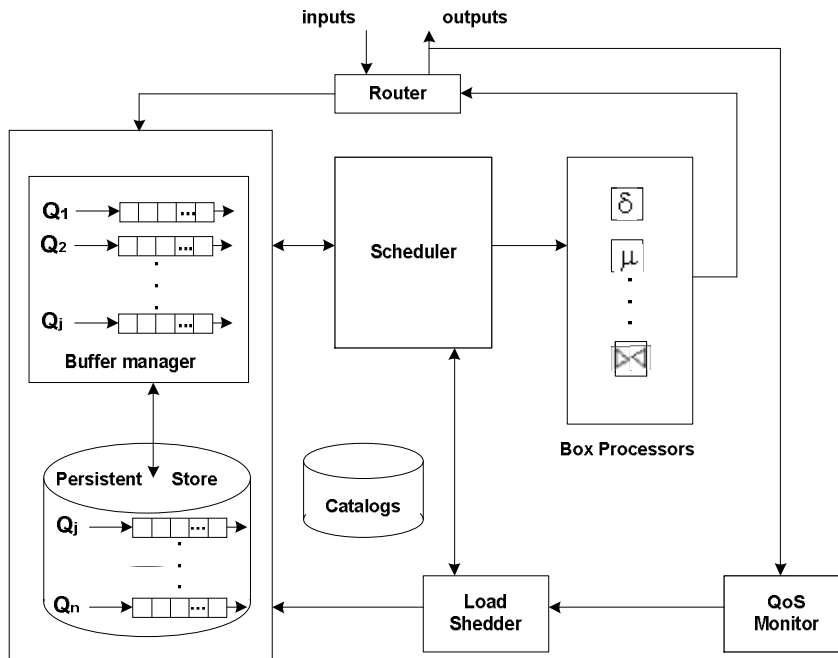


Fig.3 Aurora run-time architecture

Source: D.J. Abadi et al.: Aurora: a new model and architecture for data stream management

In terms of Queue management,

- (1). ASM can manage a large amount of queues of tuples with variable length by keeping track two points (head and tail).
- (2). ASM can scale arbitrarily and tuples can be kept in main memory, disk or both.
- (3). ASM always try to keep all the required blocks in main memory that corresponds to the top-priority queues.
- (4). ASM is aware of the size of the queue and its contiguity on disk.

One the other hand, connection point management will use B-tree to organized the historical tuples in most cases. If the amount of retained history is less than the maximum window size of the successor boxes, no extra storage is asked. Otherwise, ASM uses B-tree to arrange history tuples on storage key and creates B-tree index blocks. The scheduler will add some new boxes for an ad hoc query and give those boxes priorities. Correspondingly, ASM prefetches B-tree index blocks for worthy index structures.

4.3 Real-time scheduling

Scheduling in Aurora is a complex problem, which is related with several issues including large system scale, dynamic performance requirements, dependencies between box executions, and multiple accesses to secondary storage. Aurora system not only maximizes overall QoS but also tries to reduce overall tuple execution costs.

- (1). *Train scheduling* is used to describe a batch of mutiple tuples as input to a single box. Aurora system exploits two basic nonlinearities when processing tuples, and then reduce overall processing costs. (a). *Interbox nonlinearity*. The goal of the system is minimize the number of I/O operations processed per tuple. The system schedules the data stream to reduce tuple trashing when there is not enough buffer space. The system also schedules the tuples between two boxes without ASM intervening. (b). *Intrabox nonlinearity*. The goal is to minimize the number of box calls made per tuple by processing complete trains at once. It will reduce the overhead such as calls to box code and context switch.

(2). *Priority assignment.* The priority of an output is to meet the run-time requirement. Aurora currently considers two approaches for priority assignment. (a). A state-based approach assigns priorities to outputs based on their expected utility under the current system state and then picks for execution. The utility of an output is decided by evaluating the lost of QoS if the execution of the output is deferred. (b) A feedback-based approach. It continuously monitors the performance of the system and dynamically reassigns priorities to outputs.

(3). *Putting it all together. Superbox scheduling* is to describe scheduling actions that push a tuple train through multiple boxes. In this case, The system asks storage manager to put all boxes into memory.

(4). *Scheduler performance.* There are some measurements against Aurora prototype.(a)The time used in the scheduler can be reduced by a factor of 0.48 when we shift from a box-at-a-time scheduling discipline to using tuple trains. (b) Adding a simple version of superbox scheduling decreases the time spent in the scheduler by an additional factor of 0.43. (c) The overall execution costs are also reduced.

4.4 Introspection

Aurora uses static and dynamic introspection techniques to predict and detect overload situations.

(1). *Static analysis* is to determine if the system have enough computational power.

(2). *Dynamic analysis* is to determine the system performance if the system is under expected condition, unpredictable, long-duration spikes in input rates.

4.5 Load shedding

When Aurora system detect an overload, it will decrease the tuple volume by load shedding. This can be done by dropping tuples, which is similar to dropping overflow packets in packet-switching networks. Aurora has two methods to reduce the volume, at the same time, without potential problems.

(1). *Load shedding by dropping tuples.* For dynamic analysis, the effect of load shedding can use delay-based QoS. If the waiting time for output is beyond certain thresholds, then we continue the load-shedding process until the latency is acceptable. For static analysis, it can use drop-based QoS. We try to put the drop box as far upstream as possible until with hit the point where we find that the drop box will affects other output.

(2). *Semantic load shedding by filtering tuples.* It drops tuples in a more controlled way instead of randomly selected. Basically, it means that it drops less important tuples by using filters. The effect of load shedding can use value-based QoS. The system observes the value ranges to create a filter predicate.

5 SQuAl: the Aurora query algebra

The Aurora [S]tream [Q]uery [A]lgebra (SQuAl) supports seven operators that are used to construct Aurora networks (queries).Some operators are *order-agnostic* (*Filter, Map, Union*) others are *order-sensitive* (*BSort, Aggregate, Join, Resample*).

(1). Filter is like a case statement and can route input tuples to alternative streams.

$Filter(P1, \dots, Pm)(S)$ $P1, \dots, Pm$ are predicates over tuples on the input stream, S.

(2). Map is similar to relational projection but can apply any functions to tuples.

$Map(B1 = F1, \dots, Bm = Fm)(S)$ $B1, \dots, Bm$ are names of attributes and $F1, \dots, Fm$ are functions over tuples on the input stream, S

(3). Union can merge two or more streams into a single output stream.

$Union(S1, \dots, Sn)$ $S1, \dots, Sn$ are streams with common schema.

(4). *BSort* is an approximate sort operator that takes the form:
BSort (Assuming O) (S) *O = Order (On A, Slack n, GroupBy B1, . . . , Bm) A, B1, . . . , Bm*
A, B1, . . . , Bm are attributes and *n* is a nonnegative integer

Evaluating *BSort* with *Slack = 2* results in the behavior illustrated in the diagram in Fig. 4.

Time	1	2	3	4	5	6	7	...
Input Stream	1	3	1	2	4	4	8	...
Buffer	1 - -	1 3 -	1 3 1	3 1 2	3 2 4	3 4 4	4 4 8	...
Output Stream			1	1	2	3	4	...

Fig.4. An example application of *Bsort*

Source: D.J. Abadi et al.: Aurora: a new model and architecture for data stream management

(5). *Aggregate* applies “window functions” to sliding windows over its input stream. This operator has the form: *Aggregate (F, Assuming O, Size s, Advance i) (S)*
F is a “window function” *O = Order (On A, Slack n, GroupBy B1, . . . , Bm)* is an order specification over input stream *S*, *s* is the size of the window and *i* is an integer.

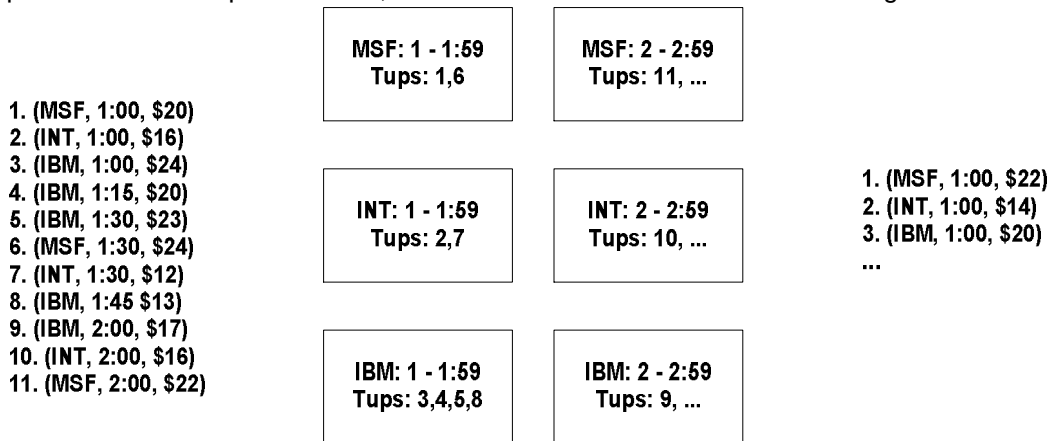


Fig.5. An example trace of *aggregate*

Source: D.J. Abadi et al.: Aurora: a new model and architecture for data stream management

(6). *Join* is a binary join operator that takes the form:
Join (P, Size s, Left Assuming O1, Right Assuming O2)(S1, S2)
P is a predicate over pairs of tuples from input streams *S1* and *S2*, *s* is an integer, and *O1* (on some numeric or time-based attribute of *S1*), and *O2* (on some numeric or time-based attribute of *S2*).

(7) *Resample* is an asymmetric, semijoin-like synchronization operator.
This operator takes the form: *Resample (F, Size s, Left Assuming O1, Right Assuming O2) (S1, S2)* *F* is a “window function” over *S1*. *s* is an integer, *A* is an attribute over *S1*, *O1* (on some numeric or time-based attribute of *S1*) and *O2* (on some numeric or time-based attribute of *S2*).
Suppose that *X* is a stream of position report from soldiers. *H* is a stream of ‘heartbeats’ that are emitted every 15 min. *t=10* min, We can see the following example.

Heartbeat (<i>H</i>)	Platoon X (<i>X</i>)	Resample (<i>F</i> , Size 10, On Time, Assuming <i>O</i>)
1. (2:00)	1. (1, 2:08, 3)	1. (2, 2:00, 2)
2. (2:15)	2. (2, 2:03, 2)	2. (1, 2:00, 3)
3. (2:30)	3. (2, 2:11, 1)	3. (1, 2:15, 2.33)
...	4. (1, 2:12, 1)	4. (3, 2:00, 1)
	5. (1, 2:21, 3)	5. (2, 2:15, 2.5)
	6. (2, 2:14, 4)	6. (3, 2:15, 2.33)
	7. (3, 2:10, 1)	7. (3, 2:30, 4)
	8. (1, 2:34, 4)	...
	9. (3, 2:19, 2)	
	10. (2, 2:28, 3)	
	11. (3, 2:25, 4)	
	12. (3, 2:41, 1)	
	...	

Fig.6. An example trace of resample

Source: D.J. Abadi et al.: Aurora: a new model and architecture for data stream management

5.3 Query examples

Query 1: Produce an output whenever *m* soldiers are across some border *k* at the same time (with border crossing detection determined by the predicate $Pos \geq k$).

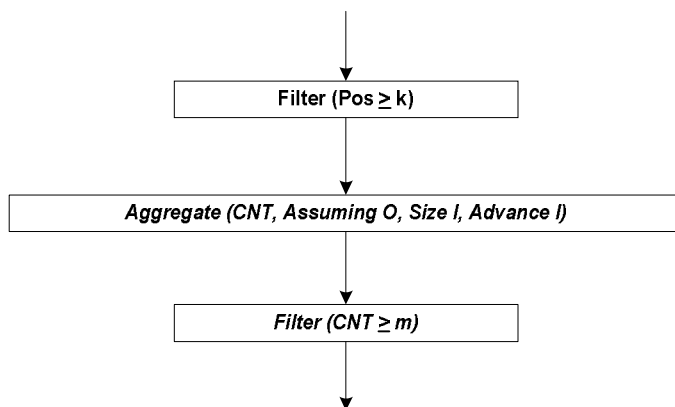


Fig.7. Query 1

Source: D.J. Abadi et al.: Aurora: a new model and architecture for data stream management

(Sid, Time, Posn)		(Sid, Time, Posn)		(Time, Cnt)		(Time, Cnt)
(1, 1, 34)	Filter	(1, 1, 34)	Aggregate	(1, 2)	Filter	(2, 5)
(1, 2, 38)		(1, 2, 38)		(2, 5)		(3, 3)
(2, 1, 24)		(3, 1, 35)		(3, 3)		(3, 3)
(3, 1, 35)		(3, 2, 38)		...		
(3, 2, 38)		(2, 2, 31)				
(3, 3, 18)		(4, 2, 36)				
(4, 1, 21)		(5, 2, 31)				
(5, 1, 20)		(4, 3, 30)				
(2, 2, 31)		(2, 3, 41)				
(4, 2, 36)		(5, 3, 31)				
(4, 3, 30)		...				
(5, 2, 31)						
(1, 3, 28)						
(2, 3, 41)						
(5, 3, 31)						
...						

Fig.8 An example trace of query 1

Source: D.J. Abadi et al.: Aurora: a new model and architecture for data stream management

An *example* trace of this query on a stream of soldier reports (assuming $k = 30$, $m = 3$, and $n(\text{Slack}) = 1$) is shown in Fig. 13.

6 Related Work and Conclusions

Aurora system may related with the following research fields: (1) query indexing (2) active databases (monitoring conditions) (3) adaptive query processing techniques (4) continuous query system (5) stream data query processing architectures (6) SEQ model (7) The Chronicle Data Model (8) materialized views.

Aurora system may benefit from and contribute to the following research fields: (1) temporal databases, main-memory databases and real-time databases (2) scheduling tasks in real-time and multimedia systems and databases (3) The congestion control problem in data networks and its load-shedding mechanism (4) approximate query answering (load shedding)

Aurora system can functionally correct, but there is no optimization and load shedding. The teams are working on competing scheduling algorithms and extending the functionality of ASM.

There are two further works for Aurora system: (1) will provide support for distribution. (2) will extend basic data and processing model to deal with missing and imprecise data values, which are common for the sensor-generated data streams

References

1. D.J. Abadi et al.(2003) Aurora: a new model and architecture for data stream management