Real Time application of Database Management System using monitoring of Input

Anita Shrotriya[1], Devendra Kumar Sharma[2]

[1]Asst. Prof., Dept. of Computer Science, Global Institute of Technology, Jaipur
[2]Asst. Prof., Dept. of Computer Science. Anand Institute of Technology, Jaipur

Abstract*—This paper introduces input monitoring applications, which is different from conventional business data processing. The fact that a system can process and react to inputs from many sources (e.g., sensors) rather than from human operators, a need for another fundamental architecture of a DBMS for this application area is required. Traditional DBMSs have been oriented toward business data processing, and consequently are designed to address the needs of the applications below:*

1. *They have assumed that the DBMS is a passive repository storing a large collection of data elements and that human initiate queries and transactions on this repository.*
2. *They have assumed that the current state of the data is the only thing that is important. Hence, current values of data elements are easy to obtain, while previous values can only be found by decoding the DBMS logs.*
3. *DBMSs assume that data elements are synchronized and that queries have exact answers. In many stream-oriented applications, data arrives asynchronously and answers must be computed with incomplete information.*
4. *DBMSs assume that applications require no real-time services.*

Keywords— Monitoring, input stream, query Model, logs, and transaction.

I. INTRODUCTION

Monitoring applications are applications that monitor continuous streams of data. This class of applications includes military applications that monitor readings from sensor by soldiers (e.g., blood pressure, heart rate, and position), financial analysis applications that monitor streams of stock data reported from various stock exchanges.

First, monitoring applications get their data from external sources (e.g., sensors) rather than from humans issuing transactions. The role of the DBMS

in this context is to alert humans when abnormal activity is detected.

Secondly, consider a monitoring application that tracks the location of items of interest, such as overhead transparency projectors and laptop computers, using electronic property stickers attached to the objects. Ceiling-mounted sensors inside a building and the GPS system in the open air generate large volumes of location data. If a reserved overhead projector is not in its proper location, then one might want to know the geographic position of the missing projector. In this case, the last value of the monitored object is required. However, an administrator might also want to know the duty cycle of the projector, thereby requiring access to the entire historical time series.

Third, most monitoring applications are trigger-oriented. If one is monitoring a chemical plant, then one wants to alert an operator if a sensor value gets too high or if another sensor value has recorded a value out of range more than twice in the last 24 hours. Every application could potentially monitor multiple streams of data, requesting alerts if complicated conditions are met.

Lastly, many monitoring applications have real-time requirements. Applications that monitor mobile sensors (e.g., military applications monitoring soldier locations) often have a low tolerance for stale data, making these applications effectively real time. The added stress on a DBMS that must serve real-time applications makes it imperative that the DBMS employ intelligent resource management (e.g., scheduling) and graceful degradation strategies (e.g., load shedding) during periods of high load.

II. PROBLEMS

Monitoring applications are very difficult to implement in traditional DBMSs. In addition, to store time-series information one has only two choices. First, he can encode the time as tuples in normal tables. In this case, assembling the historical time series is very expensive because the required data is spread over many tuples, thereby dramatically slowing performance.

Alternately, he can encode time series information in binary large objects to achieve physical locality, at the expense of making queries to individual values in the time series very difficult.

Moreover, if a monitoring application had a very large number of triggers or alerter, then current DBMSs would fail. The only alternative is to encode triggers in some middleware application. Using this implementation, the system cannot reason about the triggers (e.g., optimization), because they are outside the DBMS.

Lastly, no DBMS that we are aware of has built-in facilities for approximate query answering. The same comment applies to real-time capabilities. Again, the user must build custom code into his application.

In this paper, we try to implement a prototype system model, which is designed to better support monitoring applications.

The system illustrates design issues that would arise in any system of this kind. Monitoring applications are applications for which streams of information, triggers, imprecise data, and real-time requirements are prevalent. We expect that there will be a large class of such applications.

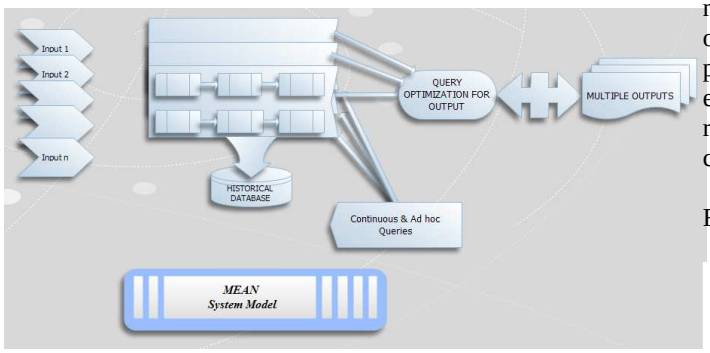
We named the model as **MEAN System Model**.

III. IMPLEMENTATION OF MEAN SYSTEM PROTOTYPE

Data is assumed to arrive from a variety of data sources such as computer programs that generate values at regular or irregular intervals or hardware sensors. We will use the term data source for either case. In addition, a data stream is the term we will use for the collection of data values that are presented by a data source. Each data source is assumed to have a unique source identifier and our system timestamps every incoming tuple to monitor the quality of service being provided.

The basic job of the *MEAN system* is to process incoming streams. It is fundamentally a data-flow system in which process and work flows are represented by arrows and boxes. Hence, tuples flow
through a loop-free, directed graph of processing operations (i.e., boxes). Ultimately, output streams are presented to applications, which must be programmed to deal with the asynchronous tuples in an output stream. It can also maintain historical storage (usually columnar databases), primarily in order to support ad-hoc queries.

Figure 1



MEAN System Model

1. OPERATIONS OF OPERATORS IN THE MEAN MODEL

There are number of operators that are suggested by us in support of this model implementation.

a. **Streamed Operator**: operators that operate on sets of consecutive tuples from a stream ("windows") at a time. Every streamed operator applies an input (user-defined) function to a window.
b. **Slide Operator**: Slide advances a window by "sliding" it downstream by some number of tuples. This tuple is used to continuing monitoring of tuples that are newly added and therefore used in real time queries. For eg. Fire a query that calculates the average value of stock in last three hours in a company.
c. **Disjoint Slide Operator**: resembles Slide except that consecutive windows have no tuples in common. Rather, It effectively partitions a stream into disjoint windows. This is useful, for example, when calculating daily stock indexes, where every stock quote is used in exactly one index calculation.
d. **Latch Operator**: resembles disjoint Slide but can maintain internal state between window calculations. This is useful for "infinite window" calculations, such as one that maintains the maximum or average value of every stock, maintained over its lifetime.

Streamed operations are operators that act on a single tuple at a time. The **Filter operator** screens tuples in a stream for those that satisfy some input predicate. A special case of Filter is **Drop**, which drops random tuples at some rate specified as an operator input. **Map** applies an input function to every tuple in a stream. **GroupBy** partitions tuples across multiple streams into new streams whose tuples contain the same values but grouped with a given condition. Finally, **Join** pairs tuples from input streams.

2. MEAN QUERY MODEL WITH SCHEDULING

MEAN Model is designed to handle real time queries, creation of views and Ad hoc queries. The performance of the model is compared with existing Aurora model on the account of three QoS parameters as: Response, Tuples drop and value produced. The major objective of a real time system is to execute the transaction before its deadline expires. This is the reason they are used for complex transactions with time constraints attached to them.

Figure 2

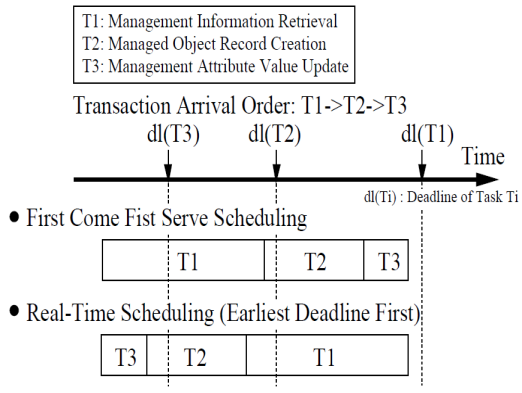
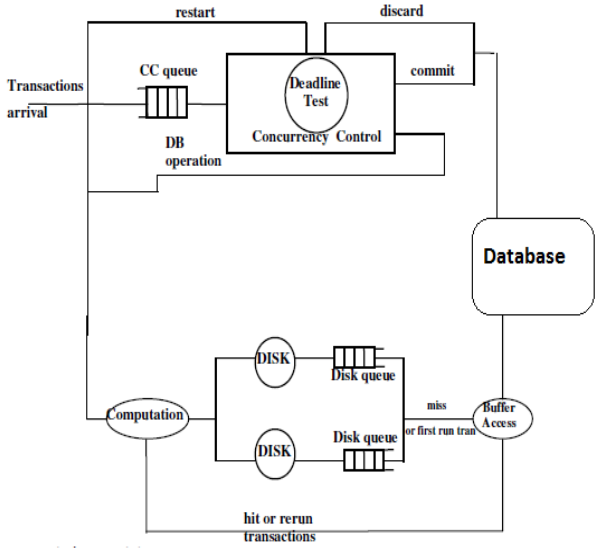


Figure 2 illustrates the difference between non real-time scheduling and real-time scheduling. A FCFS (First Come

First Serve) algorithm, a non real-time scheduler, assigns a higher priority to a transaction that arrived earlier. A Priority Scheduling (Earliest Deadline First) algorithm, a typical real-time scheduler, assigns a higher priority to a transaction that has an earlier deadline. In this example, three transactions arrive in the following order: T1 |T2 | T3, and the order of their deadlines is dl(T3) | dl(T2) | dl(T1).

Figure 3



3. OPTIMIZATION AND FUTURE WORK

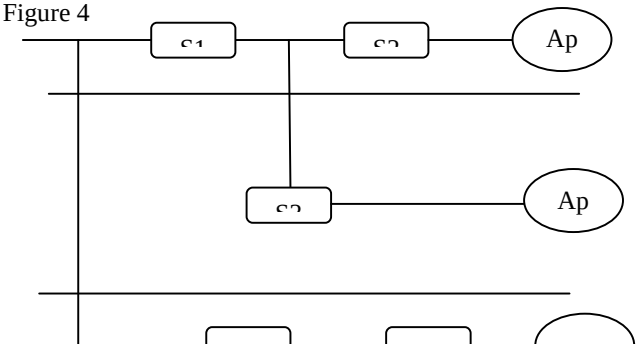
As illustrated above in the MEAN system model supports two parameters: Continuous and Ad-hoc queries. Beside these two, database views can also be used for querying purpose. Figure 4 illustrates how and at what layer these querying occurs.

The most important challenge in any real time database model is optimization and updating the data without going offline.

Fragmentation is a new proposed technique that can be used for updating data. The operator can fragment the data into different pieces which can be reassembled later as per the query requirements and the type of query fired (as shown in figure 4). The type of fragmentation can also be *Transparent* or *non-Transparent*, depending upon the type of application.

S1, S2…S5 are the data storage blocks which are utilized by different applications at different query levels. New storage blocks can be added or deleted from the model. These storage blocks can also act as buffer cache to store data streams passed on it as persistence storage for a specific period.

Figure 4



Continuous query

views

Ad hoc queries

IV. CONCLUSION

This MEAN System database model is factious and can be implemented in any field of real time database implementation. However, optimization and correction further in this model and its approach is likely to be possible.

. The basic drawback in this system is that it cannot interpret, or we can say that, cannot deal with missing or incomplete data values generated by real time censors. We are currently working to link this model with other existing models of this kind and finally compare the results in our upcoming papers.

REFERENCES

[1] Design and implementation of an emergency environmental response system to protect migrating salmon in the lower San Joaquin River, California Nigel W.T. Quinn a,*, Karl C. Jacobs ba Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA Department of Water Resources, Sacramento, CA 95814, USA Received 11 July 2005; received in revised form 30 October 2005; accepted 21 December 2005 Available online 17 April 2006

[2] On Optimistic Concurrency Control for Real-Time Database Systems Amer Abu Ali Faculty of Information Technology, Philadelphia University, Jordan.

[3] Real-time Database Experiences in Network Management Application Yoshiaki Kiriha NEC Corporation C&C Research Laboratories4-1-1 Miyazaki, Miyamae-ku, Kawasaki, Kanagawa 216, JAPAN Stanford University Computer Science Department Stanford, CA 94305 E-mail: kiriha@db.stanford.edu August 30, 1995

[4] Towards Process-Oriented Tool Support for Knowledge Discovery in Databases Rudiger Wirth1, Colin Shearer2, Udo Grimmer1, Thomas Reinartz1, J org Schl osser3, Christoph Breitner3, Robert Engels4, and Guido Lindner.

[5] 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.

[6] B. Kao and H. Garcia-Molina, "An Overview of Real-Time Database Systems," Proceedings of NATO Advanced Study Institute on Real-Time Computing, Springer-Verlag , 1993.10.
[7] Y. Kiriha et al., "An Automatic Generation of Management Information Base (MIB) for OSI based Network Management System," Proceedings of IEEE GLOBECOM 91, pp. 649- 53, 1991.12.