Architectural Principles of the “Streamonas”
Data Stream Management System and Performance Evaluation based on the Linear Road Benchmark

Panayiotis Adamos Michael and D. Stott Parker, Jr.
Computer Science Dept. - Scientific Computing Lab.,
UCLA-Anderson School of Management
University of California Los Angeles
Los Angeles, California, USA
michaelp@cs.ucla.edu , stott@cs.ucla.edu

Abstract—Data Stream Management Systems (DSMSs) receive large overheads when queries directly access the serial non-indexed incoming stream. Our novel architecture, presented in this work, addresses this problem by indexing the incoming dataflow based on a specially designed data structure. The role of this data structure is as fundamental for our DSMS as the role of a Relation in a Relational DBMS. The architecture achieves reusability, query parallelism and O(1) constant time complexity access to streamed data. The system managed to run the maximum level of difficulty the Linear Road Benchmark has (10 expressways), demonstrating excellent performance results.

Keywords- streams; data stream management system; benchmark; query latency; index; spatio-temporal cuboid; overhead; historical reference; atomic streams; decomposition; spatio-temporal element; time hashing; space hashing; historical span; semantic space; streaming layer; querying layer

I. INTRODUCTION

In a streaming environment various Data Stream Management Systems (DSMSs) address the challenge of managing the large volume of data arriving at high dataflow rates to the system, by feeding the serial stream of incoming information into a graph of logical operators. More specifically, in Aurora [7] tuples flow through a loop-free, directed graph of processing operations. In Stream Processing Core [10] the construction of queries uses a dataflow graph consisting of processing elements that consume and produce streams of processing data through input and output ports respectively. The research work STREAM [12] feeds the flow of tuples through a query plan. As we analyze in section 3, the approach of directly feeding the flow of information through a query graph in a serial manner, creates a large overhead for the DSMS. In the same section we introduce the architectural principles of our novel architecture which decomposes, indexes and temporarily stores the incoming serial data stream, into a specially designed data structure. This data structure allows parallel queries to randomly access the streamed elements, thus avoiding direct access of the incoming serial stream by the queries.

The architecture demonstrates excellent performance results by running the Linear Road Benchmark [8, 9] at the maximum level of its difficulty (10 expressways), achieving an average query latency of 0.000026 seconds, 192,307 times faster than the 5 seconds hard real-time constraint the benchmark sets.

II. PREVIOUS WORK

Extensive bibliography exists on spatio-temporal databases as emphasized in [1]. A consolidation approach to temporal data models and calculus-based query languages is provided in [2]. The researchers in [3] introduce a stream processing paradigm of functional transformations (transducers) on streams. The authors in [4] describe the Tangram stream processor. The paradigm of transducers on streams is used throughout their system, providing a database flow computation capability. Aurora [7] has introduced an architecture based on a data-flow model and an algebra with a set of operators to express its stream processing requirements. Commercial database systems as analyzed in [8] are not suitable for streaming applications. A commercial system was able to run only 0.5 expressways (XWays) on the Linear Road Benchmark (LRB) in order to meet the maximum response times set by the benchmark. Aurora [7] was the first system to use the benchmark reaching a level of 2.5 XWays while meeting the response time requirements for Tolls. The project STREAM [12] has built a Data Stream Management System prototype which supports a large class of declarative continuous queries over continuous and traditionally stored data sets. In the terms of the STREAM project, the researchers have designed the CQL continuous query language. Borealis [11] has a distributed processing engine and was built upon the works of the projects Aurora [7] and Medusa [14]. TelegraphCQ [13] has an adaptive dataflow architecture for supporting a wide variety of intensive networked applications. The researchers in [15] have implemented a continuously adaptive, continuous query implementation (CACQ). In [5] the researchers studied the limitations of relational algebra and SQL in supporting sequence and stream queries. The Stream Mill project [6] has as goal to overcome the limitations of SQL in a streaming environment. SPC [10] has run the Linear Road
Benchmarks on a distributed architecture on a cluster of 85 nodes. SPC has achieved to satisfy the time constraints of the benchmark at a level of 2.5 XWays on a single node. In this work we analyze the architectural principles of the novel architecture of our Data Stream Management System “Streamonas”, which demonstrates excellent performance characteristics at a level of 10 XWays on a single PC.

III. ARCHITECTURAL PRINCIPLES OF “STREAMONAS”

In this section we present the architectural principles of our novel Data Stream Management System. The architecture of our system provides a general framework for many streaming environments. Throughout the paper (without limiting the system’s generality) we shall use the Linear Road Benchmark as an example application to clarify the various theoretical concepts.

A. Principle I: Constant time complexity O(1) access of temporal sequences.

Our system models the evolution of entities. For this reason a fundamental principle in its architecture is the modeling and constant time complexity O(1) access of temporal sequences which we name Atomic Streams (ASTs). For the LRB application, the evolving entities are cars moving on expressways (LRB simulates approximately 1.3 million cars moving on 10 expressways). The “Speed” attribute of a car entity (describing the actual speed of the car reported every 30 seconds by a sensor system), is modeled in our DSMS as an atomic stream. Figure 1 demonstrates a novel data structure used by the data stream engine of our system. We have named this data structure as Spatio-Temporal Cuboid Data Structure (ST-Cuboid). The following paragraphs describe the central role ST-Cuboid has for our system. One of the tasks of ST-Cuboid is to isolate and index temporal sequences of entities (atomic streams) arriving through the incoming serial stream. The incoming stream is decomposed into atomic streams based on a direct address table hashing method. The cardinality n of the universe of keys U of the streamed information is very small compared to the whole volume of temporal information (figure 1), making the specific indexing scheme very effective. Indicatively, for the Linear Road Benchmark application for a volume of 6.5 GBytes of information, a hash table with n=1,373,327 keys was sufficient and could fit in main memory. Once a tuple arrives in the system, is read and forwarded to the respective atomic stream based on the hash table. The process is very fast and has constant time complexity O(1). As we cannot store the whole spatio-temporal volume of historical information (in the general case), we should evaluate the number of the most recent values each AST should store. For the LRB the requirements of the application were satisfied with a historical reference of up to the 6 most recent values (including the latest one). In order to respect the semantics of an application, the time granularity of the timestamps of an AST as also the frequency of arrival of the tuples in the system, are parameters that must be taken into consideration by the application developer who is asked to define the historical span of the ST-Cuboid. For the LRB the “Speed” attribute of a car entity had time granularity in seconds, while for the ASTs of statistical metrics (such as the “Minute Average”) a minute time granularity was used. The frequency of arrival in the system was one tuple per car every 30 seconds (for the active cars on the expressways). Atomic streams are described by the general expression:

\[ \text{Space}[j].\text{Attribute}[t=\ast] \]  

where Space is the name of the universe of keys and j is a parameter taking values from this universe of keys. Figure 1 shows various instances of ASTs. We have used the symbol “Space” because we assume the universe of keys to be static i.e. it does not evolve over time. Extending our concept to cover domains of meaningful, static over time universes of keys we also refer to “Space” as “Semantic Space”. For the LRB application, Semantic Space consists of the keys of the cars. As an example, for the ST-Cuboid CAR the AST expression for the car having as key Car_Id with Car_Id="100" is \( \text{CAR} \{ \text{Car_Id} = 100 \}. \text{Attribute}[t=\ast] \). The expression “t=\ast” denotes reference to the whole temporal sequence of the attribute the AST models. The AST data type is used in the schema of an ST-Cuboid to model attributes with temporal behavior such as “Speed” and “Segment” (location). An ST-Cuboid schema may have attributes of data type Atomic Stream with the same or different historical spans. As an example the simplified schema for the CAR ST-Cuboid of the LRB application, is:

\[ \text{CAR} \{ \text{Car_Id} : \text{int}, \text{Speed}(t) : \text{AST(6)}, \text{Segment}(t) : \text{AST(6)} \} . \]

The parameter “t” of the attributes “Speed” and “Segment” denotes each attribute’s evolving nature over time. The AST data type, models the temporal sequences of “Speed” and “Segment” attributes with their historical span defined in parentheses, increased by one, in order to include the latest value (the value “6” for the LRB application denotes 5 historical values in addition to the latest one). It is very important to mention that for application development and data modeling, the ST-Cuboids have a role as fundamental for our DSMS as the role of a Relation in a Relational DBMS.

B. Principle II: Constant time complexity O(1) historical access of any tuple within the temporal scope of the ST-Cuboid.

The ability to access historical information with constant time complexity is also a fundamental principle of our system. In order to achieve this requirement we have an individual
index for each atomic stream. The index is a hash table for each one of the latest $\tau$ tuples stored in the system per AST.

The First-In-First-Out structure implementing the hash table as a cyclical list (i) enables the updating of the list in real-time (a cyclical list captures a predetermined fixed area of memory locations and does not require to update all elements of the AST upon a new append), (ii) stores only the latest $\tau$ tuples of a specific atomic stream, thus providing stable memory characteristics without overflows and (iii) enables the constant time complexity $O(1)$ access of any element (spatio-temporal element) in the AST. As an example in the LRB the expression referring to the latest speed element of a car entity having as key Car_Id with Car_Id = "100" is CAR[100].Attr[0].

The following general expression describes the operation of randomly accessing the spatio-temporal elements of the ST-Cuboid:

$$\text{Space}[j].\text{Attr}[0-k]$$

(Expr. 2)

Where $j$ is any valid key and $k$ is an index allowing the historical random access of the AST temporal sequence, having as maximum value the historical span $\tau$ of the atomic stream (Figure 1).

C. Principle III: Decoupling of Querying from Streaming.

Systems directly querying the serial incoming data stream do not have the capability to isolate the individual temporal sequences (atomic streams). Queries in these systems, access unnecessary out of the query scope elements, as the incoming stream is inherently serial and not indexed. The evaluation becomes more demanding when joins are involved, resulting to an increased risk for memory overflows. The architecture of our system by using the novel ST-Cuboid data structure described in the previous paragraphs, decouples in two parallel layers the querying process from the streaming process (Figure 2).

The architecture (i) enables the reusability of the streamed information stored in the ST-Cuboids, (ii) allows the real-time constant time complexity access of the stored information, (iii) provides an environment for parallel access of the ST-Cuboids by multiple queries and (iv) reduces the risk for memory overflows by minimizing the volume of accessed information.

The following equation describes the overhead of accessing unnecessary data elements when a query directly accesses the serial stream feed:

$$\text{Overhead} = \sum_{i=1}^{q} \sum_{j=1}^{S_i} \text{time}_i \times (\text{frequency}_j)$$

(Eq. 1)

Where $q$ is the number of queries running in the system, $S_i$ is the cardinality of the set of atomic streams per query $i$, unnecessarily accessed when they are not in the scope of the query, frequency$_j$ is the frequency of the tuples per atomic stream $j$ not in the scope of a specific query $i$ and time$_i$ is the whole duration the continuous query $i$ runs. Based on (Eq.1) the overhead even for a small number of queries can get very large as it accumulates over time the whole spatio-temporal volume of elements from bursty sources.

IV. EXPERIMENTAL RESULTS

In this section we provide the performance results of the Streamonas DSMS when it runs the Linear Road Benchmark [8, 9] on a single PC. Our system achieves to reach the maximum level of 10 expressways the Linear Road Benchmark supports, with an average query latency of 0.000026 seconds, 192,307 times faster than the 5 seconds hard real-time constraint the LRB sets. The worst case query latency of the system at the same level (10XWays) is 0.139580 seconds.

During the benchmarking, Streamonas achieved an average throughput of 66,226 tuples/second compared to 486 tuples/sec of Aurora as published in [8] and 100 tuples/sec of the commercial system also published in [8]. Figure 3 provides the mapping of the query latency during the 3 hour simulation of the LRB on our DSMS at the level of 10 XWays.
V. CONCLUSION

In this work we have introduced a novel DSMS architecture based on a specially designed data structure which (i) enables the reusability of the streamed information, (ii) allows the real-time constant time complexity access of the stored information, (iii) reduces the risk of memory overflows and (iv) provides an environment for parallel access of the streamed information by multiple queries. Our theoretical analysis is verified by excellent experimental performance results, as our DSMS managed to reach the maximum level of difficulty of the Linear Road Benchmark (10 XWays) with an average query latency of 0.000026 seconds, 192,307 times faster than the 5 seconds hard real-time constraint the benchmark sets.

ACKNOWLEDGMENT

Panayiotis Michael would like to thank his parents, Adamos Michael and Andriani Zachariadou-Michael for their continuous support to his research work.

REFERENCES