UNIVERSITY OF CALIFORNIA

Los Angeles

Architecture, Data Model and Real-Time Performance Evaluation of the Streamonas Data Stream Management System

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Computer Science

by

Panayiotis Adamos Michael

2015

ABSTRACT OF THE DISSERTATION

Architecture, Data Model and Real-Time Performance Evaluation of the Streamonas Data Stream Management System

By

Panayiotis Adamos Michael
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2015
Professor Vasilios I. Manousiouthakis, Co-Chair
Professor Douglass Stott Parker, Jr., Co-Chair

Within the challenging environment of data streams characterized by large volumes of data, having high data-flow rates, most Data Stream Management Systems (DSMSs) process information directly from the incoming serial data stream.

A radically different approach is followed by Streamonas DSMS presented in this Dissertation. The DSMS processes its information after the serial stream is restructured into its own, novel, object-oriented, parallel data model called the Spatio-Temporal Cuboid (ST-Cuboid).

By doing so, Streamonas has achieved both record-level performance and efficiency. As published, through extensive experiments over the Linear Road Benchmark (LRB), the single-cpu/single-core Streamonas is the first and still the only DSMS to have reached on a single cpu, the maximum level of difficulty of the benchmark, i.e. 10 XWays.

Distributed Streamonas, also presented in this Dissertation, has recently achieved on the LRB a level of 550 XWays which is also a record. The work has been submitted for publication.

While query latencies on the LRB reported by excellent DSMSs are on the order of a second, Streamonas demonstrated impressive query latencies on the order of a microsecond (single-cpu/single-core Streamonas) and millisecond (Distributed Streamonas).

Enforcing database consistency in real-time is so difficult that researchers often view the concept as a trade-off of accuracy for efficiency. Within this context, researchers have developed pioneering frameworks including semantic load shedding, data reduction and application-specific summary techniques.

Based on its framework named “Real-Time Consistency” the novel Streamonas architecture has been able to achieve 100% accurate results with excellent performance characteristics, without trade-offs, so that consistent database states are materialized within time intervals of less than 400 μsec. These results have been achieved for large distributed enterprise-like applications where the in-memory distributed database reaches 65 GBytes and throughput exceeds 730,000 tuples/sec.

Our Thesis is that the novel architecture of Streamonas, following three fundamental principles, supported by the novel, object-oriented, parallel ST-Cuboid data-structure and the rest of novel sub-systems and frameworks presented in this dissertation, has created a radical change to the perspective the research community views data-stream management, permitting the development of new kinds of improved Data Stream Management Systems in the years to come.

The dissertation of Panayiotis Adamos Michael is approved.

Todd D. Millstein

Carlo Zaniolo

Vasilios I. Manousiouthakis, Committee Co-Chair

Douglass Stott Parker, Jr., Committee Co-Chair

University of California, Los Angeles

2015

This work is dedicated to

my parents Adamos Michael and Andriani Zachariadou-Michael

TABLE OF CONTENTS

LIST OF FIGURES, LIST OF TABLES

Tables

ACKNOWLEDGMENTS

This work, a result of a multi-year effort, would not be feasible without the understanding, encouragement, guidance and education I received from my PhD advisor and co-author in the resulting “Streamonas DSMS” publications, Professor Douglass Stott Parker, Jr. -- I am grateful to him.

I need also to thank the co-Chair of my PhD Committee, Professor Vasilios I. Manousiouthakis for the extensive analyses we jointly performed on the applicability of my thesis in Real-Time Optimization, a research direction we shall definitely pursue in the near future.

I am also grateful for the knowledge I have received from my M.Sc. advisor and PhD Committee member, Professor Carlo Zaniolo. During my first years of academic life at UCLA, Professor Zaniolo’s strong and pioneering theory educated me, inspired me and allowed me to understand significant properties and challenges of the streaming environment.

I need also to thank Professor Todd D. Millstein for serving on my PhD Committee and for examining my research from different perspectives providing me invaluable feedback.

This work would not be feasible without the continuous financial support from my parents Adamos Michael and Andriani Zachariadou-Michael. I need also to thank my uncle Loizos Michael Prodromou, financial advisor in NY, who supported the last phase of my research.

I’m grateful also to the Fulbright Commission in Cyprus for the scholarship they awarded me after a competition, for my M.Sc. studies at UCLA. Finally, I need to thank the Chairs, Faculty, the Graduate Student Affairs Office and the Administration of the Computer Science Department who were always by my side and by the side of my PhD classmates all these years, helping us towards achieving our goal: To make an impact on the World.

VITA

January 8, 1969	Born, Famagusta, Cyprus
1993	B. S., Electrical Eng. / Computer Eng. National Technical University of Athens
1994-1995	Compulsory Military Service, Cyprus
1995-2000	Compunet Ltd. Start-up, Cyprus, Founder
2000-2002	M.S., Computer Science University of California Los Angeles Under a Fulbright Scholarship
2002	Chorafas Foundation, Award
2002	Approved Petition by the UCLA Graduate Office to follow two concurrent independent degrees an M.B.A. at UCLA-Anderson and a PhD in Computer Science at UCLA Henry Samueli School of Engineering
2005	Stream Engineering Ltd. High-Tech Start-Up, Cyprus, Founder. Funded by the Ministry of Commerce, Industry and Tourism of Cyprus
2006	M.B.A. UCLA-Anderson M.B.A. Thesis "Stream Engineering Ltd."
2010 – 2012	Cyprus Investment Promotion Agency *Governmental Organization under the Ministry of Commerce, Industry and Tourism of Cyprus* Senior Officer, Strategic Planning, Policy and Information Analysis Unit, responsible for the promotion of investments in the ICT sector.
2012 – current	Foreign Direct Investment Science and Technology Consulting Ltd., Cyprus. Founder, Associate Consultant to IBM on "Smart Cities"

Publications

“Streamonas” Data Stream Management System Publications

Submitted for Publication

[1] P. A. Michael, D. S. Parker. “The Multicore Distributed Streamonas Architecture – and its Performance Evaluation based on the Linear Road Benchmark”. Submitted for publication.

[2] P. A. Michael, D. S. Parker. “Embarrassingly Parallel Query Processing with the Real-Time Consistency Framework of the Distributed Streamonas DSMS”. Submitted for publication.

Already published works

[1] P. A. Michael, D. S. Parker. “Architectural Principles of the Streamonas Data Stream Management System and Performance Evaluation based on the Linear Road Benchmark”. In Proceedings of the 2008 International Conference on Computer Science and Software Engineering (2008 CSSE), IEEE, Wuhan, China, December 2008.

[2] P. A. Michael, D. S. Parker. “The Semantic Space-Time models of the Streamonas Data Stream Management System”. In Proceedings of the 2009 World Congress on Computer Science and Information Engineering (2009 CSIE), IEEE, Los Angeles / Anaheim, USA, March 2009.

[3] P. A. Michael, D. S. Parker. "Real-Time spatio-temporal data mining with the Streamonas Data Stream Management System". In Proceedings of the 10th International Conference on Data Mining, Protection, Detection and Security (Data Mining 2009), Wessex Institute of Technology, UK, Crete, Greece, May 2009.

Co-author in other publications (2002-2003)

[1] Alfonso F. Cardenas and Panayiotis A. Michael, "Image Stack Stream Model of Multimedia Data" Proceedings of the 2002 Conference on Distributed Multimedia Systems, San Francisco, CA., September 26-28, 2002. http://www.mmss.cs.ucla.edu/DMS2002.pdf

[2] Alfonso F. Cardenas, Panayiotis A. Michael and Bassam Islam, "Stack Database Model/View of Multimedia Data" Proceedings of the 2002 International Conference on Information and Knowledge Engineering, Las Vegas, Nevada, June 24-27, 2002. http://www.mmss.cs.ucla.edu/IKE2002.pdf

[3] Alfonso F. Cardenas, Raymond K. Pon, Panayiotis A. Michael, Jen-Ting T. Hsiao "Image Stack Viewing and Access" In Journal of Visual Language and Computing, Vol. 14, Academic Press Ltd., pp.421-441, 2003. http://www.mmss.cs.ucla.edu/CardenasJVLC2003.pdf

CHAPTER ONE

ARCHITECTURAL PRINCIPLES OF THE "STREAMONAS" DATA STREAM MANAGEMENT SYSTEM AND PERFORMANCE EVALUATION BASED ON THE LINEAR ROAD BENCHMARK

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.

1. 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.

2. 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 Benchmark 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.

3. 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.

3.1. 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 1-1 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 1-1), 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).

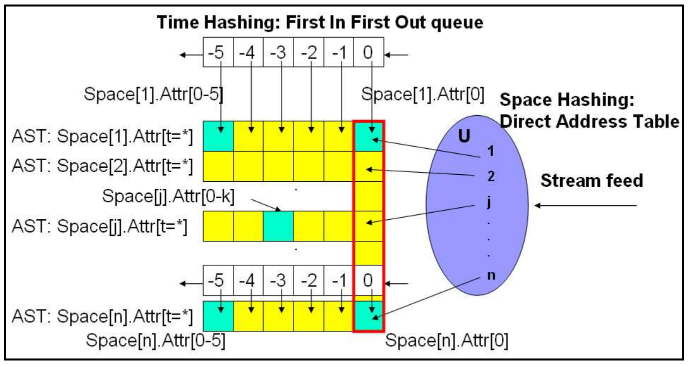


Figure 1-1. The Spatio-Temporal Cuboid Data Structure

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:

Space[j].Attribute[t=]* (Expr. 1)

where Space is the name of the universe of keys and j is a parameter taking values from this universe of keys. Figure 1-1 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 CAR[100].Attribute[t=*]. The expression “t=*” 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 :

CAR(Car_Id:int, Speed(t):AST(6), Segment(t):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.

3.2. 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 τ 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 τ 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:

Space[j].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 τ of the atomic stream (Figure 1-1).

3.3. 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 1-2).

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.

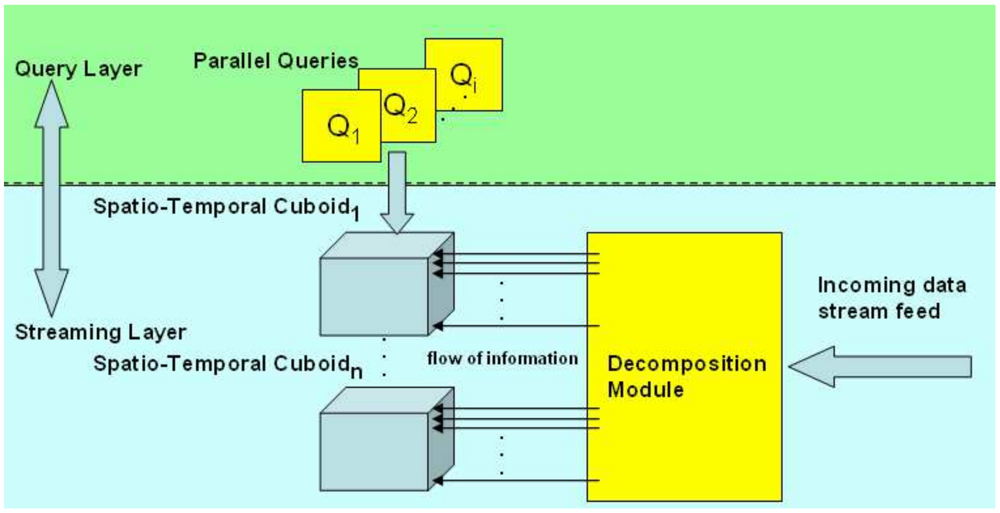


Figure 1-2. The Architecture of "Streamonas"

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

$$\textbf{Overhead} = \sum_{1}^{i=q} \text{time}_i \times \sum_{1}^{j=S'_i} (\text{frequency}_{ij}) \qquad \textbf{(Eq. 1)}$$

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_{ij} 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.

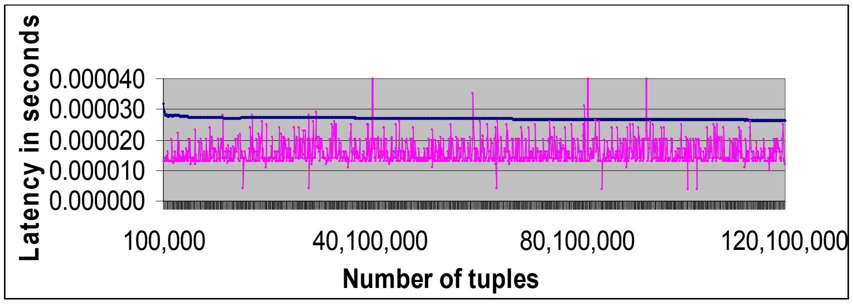


Figure 1-3. Query latency mapping of Streamonas during the 3 hour simulation with load from 10 XWays

4. 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 1-3 provides the mapping of the query latency during the 3 hour simulation of the LRB on our DSMS at the level of 10 XWays.

5. 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.

6. Contributions of the Dissertation

The dissertation makes the following significant contributions:

1. The design, implementation and performance evaluation, of a novel, parallel, object-oriented data model with name Spatio-Temporal Cuboid, which transforms the serial incoming stream, arriving to a Data Stream Management System. Data are temporarily stored in the First-In-First-Out Spatio-Temporal Cuboid data structure, and are made available for query processing. A Spatio-Temporal Cuboid as its name suggests, represents a multi-dimensional slice of data in a spatio-temporal space.

   Through object-oriented expressions, Spatio-Temporal cuboids, enable the constant time complexity O(1) access of temporal sequences in their native form (named as atomic streams), as also the constant time complexity O(1) access of any element streamed into the Data Stream Management System (named as spatio-temporal elements), providing extremely fast performance to the system. This approach is very novel in introducing a layer of abstraction above streams, and this is unique among Data Stream Management Systems.

2. Novel formal models defining the space-time semantics upon which the Streamonas architecture is based, as also the fundamental temporal semantics for application development on the prototype system. These models are:
   a. The Fact Generation Model which models the continuous fact (tuple) generation by a system of sensors, streamed to the Data Stream Management System.
   b. The Query Result Generation Model, describing continuous result generation by the Data Stream Management System from continuous queries.

c. The Semantic Space-Time model for Streaming, which decomposes streams based on a fundamental process for the system with the name Stream Semantic Decomposition.

d. The Semantic Space-Time model for Querying, allowing multiple queries to execute in parallel.

3. The Data Stream Management System architecture design for a single-cpu system with the name “Streamonas”, which decouples streaming from querying in two independent layers and uses Spatio-Temporal cuboid data structures following a database schema, to model the underlying database. The architecture (i) enables the reusability of the streamed information stored in the Spatio-Temporal 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.

4. A novel Distributed Data Stream Management System architecture for large-scale, enterprise level applications, designed and implemented on a multicore cluster environment, based on the principles of the single-cpu Streamonas. The Distributed Architecture has a Synchronous data-stream processing engine and enables both synchronous as also asynchronous query processing.

5. The enforcement of database consistency in real-time based on the Spatio-Temporal Cuboids data model providing 100% accuracy in data stream processing, along with in depth technical analysis.

We believe that the above contributions of the dissertation will influence the next generation of data stream management systems.

The Thesis is organized in five Chapters:

Chapter One presents the novel Spatio-Temporal Cuboid Data Structure and its indexing scheme. It also provides the architectural design and presents the principles of the novel "Streamonas" Data Stream Management System. The Chapter concludes by presenting simulation results, demonstrating the impressive performance of the system which is the first single-cpu system, to effectively run the Linear Road Benchmark at its highest level of difficulty, i.e. 10 expressways.

Chapter Two presents the novel formal models defining the space-time semantics upon which the Streamonas architecture is based, as also the fundamental semantics for application development on the system. The Chapter concludes with extensive presentations of results from simulations, presenting different aspects of the performance and stability characteristics of the system.

Chapter Three demonstrates the power and suitability of the system to perform data mining in real-time over spatio-temporal subsequences. The excellent performance characteristics of the system to maintain microsecond level latencies while evaluating complex dynamic clustering expressions are also presented.

Chapter Four presents the novel Distributed Streamonas Architecture. The distributed architecture addresses the challenges of the large-scale distributed environment by successfully implementing a fragmented database model of Spatio-Temporal Cuboids. The novel multicore architectures specially designed for the distributed system are also presented along with experimental results from the Linear Road Benchmark. In these experiments, Distributed Streamonas managed to create a new record on the Linear Road Benchmark by running a load of 550 Xways, reaching also a new record in efficiency by supporting 13.75 expressways/core.

Chapter Five addresses the problem of accuracy and semantic correctness in data stream management systems. Performance improvements made possible by the Spatio-Temporal Cuboid model permit improved consistency, and in fact our implementation has achieved 100% accuracy, creating a breakthrough in data stream processing. The Chapter concludes by presenting a novel asynchronous sub-system which re-uses the data stored in the Spatio-Temporal Cuboid database in an embarrassingly parallel querying manner, attaining extremely high levels of performance.

7. REFERENCES

[1] C. Zaniolo, S.Ceri, C.Faloutsos, R.T. Snodgrass, V. S. Subrahmanian and R. Zicari. *"Advanced Database Systems"*. Morgan Kaufmann, 1997.

[2] R. T. Snodgrass (ed.), I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Käfer, N. Kline, K. Kulkanri, C. Y. T. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. *"The TSQL2 Temporal Query Language"*. Kluwer Academic, Norwell, MA, 1995.

[3] D. Stott Parker. "Stream Data Analysis in Prolog". *In L. Sterling, ed., The Practice of Prolog, Cambridge, MA: MIT Press*, 1990 (abstract available at: http://www.cs.ucla.edu/~stott/sdb/abstracts.html).

[4] D. S. Parker, R. R. Muntz, H. L. Chau. "The Tangram stream query processing system". *In Proc. of the Fifth International Conference on Data Engineering*, 1989.

[5] Y. Law, H. Wang, C. Zaniolo. "Query Languages and Data Models for Database Sequences and Data Streams". *In Proc. of the 30th VLDB Conference 2004*.

[6] http://magna.cs.ucla.edu/stream-mill/

[7] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. "Aurora: a new model and architecture for data stream management". *The VLDB Journal*, 2003.

[8] A. Arasu, M.Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbets. "Linear Road: A Stream Data Management Benchmark". *In Proc. of the 30nd Intl. Conf. on Very Large Data Bases*, 2004.

[9] http://www.cs.brandeis.edu/~linearroad/

[10] N. Jain, L. Amini, H. Andrade, R. King, Y. park, P. Selo, C. Venkatramani. "Desing, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core". *In Proc. of the 2006 ACM SIGMOD Intl. Conference on Management of Data*, 2006.

[11] D J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. "The design of the Borealis stream processing engine". *In Proc. of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[12] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein and J. Widom. STREAM:"The Stanford Stream Data Manager". *In Proc.of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, 2003.

[13] S. Chandrasekaran, O. Cooper, A. Seshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurhty, S. Madden, V. Raman, F. Reiss and M. Shah. "TelegraphCQ: Continuous dataflow processing for an uncertain world". *In Proc. of the 2005 Conference on Innovative Data Systems Research (CIDR)* , 2003.

[14] M. Balazinska, H. Balakrishnan and M.Stonebraker. "Load Management and High Availability in the Medusa Distributed Stream Processing System". *In Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, 2004.

[15] S. R. Madden, M. A. Shah, J. M. Hellerstein and V. Raman. "Continuously adaptive continuous queries over streams". *In Proc. of the 2002 ACM SIGMOD Intl. Conference on Management of Data,* 2003.

[16] M. Svensson. *"Benchmarking the performance of a data stream management system"*. MSc thesis report, Uppsala University, November 2007.

CHAPTER TWO

THE SEMANTIC SPACE-TIME MODELS OF THE STREAMONAS DATA STREAM MANAGEMENT SYSTEM

ABSTRACT

Four novel models define the fundamental temporal semantics upon which the Streamonas DSMS has been architected, as also the fundamental temporal semantics for application development on the system. Extensive experimental results demonstrate the power of the theoretical models as also the stability and scalability of the system when this is tested with a load from 2, 4, 6, 8 and 10 expressways when running the Linear Road Benchmark. The extensive experimental results demonstrate the effectiveness of the theoretical Semantic-Space Time models as the system has reached the maximum level of difficulty of the benchmark (10 expressways) with an average query latency of 0.000026 seconds, 192,307 times faster than the 5 seconds hard real-time constraint the benchmark sets.

1. INTRODUCTION

Various Data Stream Management Systems (DSMSs) have been proposed in order to meet the challenging streaming environment. This environment is characterized by the large volume of data arriving at the DSMS from bursty data stream sources, as also by the requirement for high throughput and low query latency, in order to serve a large number of continuous queries. The architectural principles of the Streamonas Data Stream Management System have been presented in [1]. In the same work the excellent performance results received by Streamonas while running the Linear Road Benchmark (LRB) are also presented. Based on the results published in the research works [1, 10, 11, 12, 13, 14], Streamonas is the first Data Stream Management System which managed to run the Linear Road Benchmark at the maximum level of its difficulty (10 expressways) while meeting the benchmark's hard real-time constraints. As presented in [1], Streamonas has an average query latency of 0.000026 seconds, 192,307 times faster than the 5 seconds hard real-time constraint the benchmark sets. This research work presents the theoretical framework upon which, the Streamonas Data Stream Management System has been architected. Four novel theoretical models are presented. Section 3 presents the Fact Generation Model which models the continuous fact (tuple) generation by a system of sensors, streamed to the DSMS. In the same section we present the Query Result Generation Model. The model describes the continuous result generation by the DSMS from continuous queries. In section 4 we present the model of the streaming engine of Streamonas. The model, named Semantic Space-Time model for Streaming, decomposes streams based on a fundamental process for our system with the name Stream Semantic Decomposition. The process is serialized ensuring the internal to the system, transparent to the queries, preservation of the temporal semantics of the streamed data. In section 5 the Semantic Space-Time model for Querying is presented. The model is based on the well preserved semantics of the underlying Semantic Space-Time model for Streaming, while it is totally decoupled from it. This independence allows for multiple queries to run in parallel, each one having its own temporal semantics, at different granularities. The novel Semantic-Space Time grid defined in sections 4 and 5, provides the logical framework for application development on Streamonas. The theoretical framework presented in this work has been applied for the development of the queries for the benchmarking of Streamonas. In the experimental section of this paper, we present experimental results which demonstrate the stability and scalability of the system when it runs the LRB with a load from 2, 4, 6, 8 and 10 expressways.

2. Previous work

Many research works for modeling time and time granularities in Databases, Data Mining and Temporal Reasoning, are presented in [2]. In [3] it is emphasized that extensive bibliography exists on spatio-temporal databases. A consolidation approach to temporal data models and calculus-based query languages is provided in [4]. The researchers in [15] introduce a stream processing paradigm of functional transformations (transducers) on streams. The authors in [7] describe the Tangram stream processor. The paradigm of transducers on streams is used throughout their system, providing a database flow computation capability. Aurora [10] has introduced an architecture based on a data-flow model and an algebra with a set of operators to express its stream processing requirements. The project STREAM [18] 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 [17] has a distributed processing engine and was built upon the works of the projects Aurora [10] and Medusa [20]. TelegraphCQ [19] has an adaptive dataflow architecture for supporting a wide variety of intensive networked applications. The researchers in [22] have implemented a continuously adaptive, continuous query implementation (CACQ). In [16] the researchers studied the limitations of relational algebra and SQL in supporting sequence and stream queries. The Stream Mill project [8] has as goal to overcome the limitations of SQL in a streaming environment. The research work [9] begins the discussion of a SQL-based standard for streaming databases.

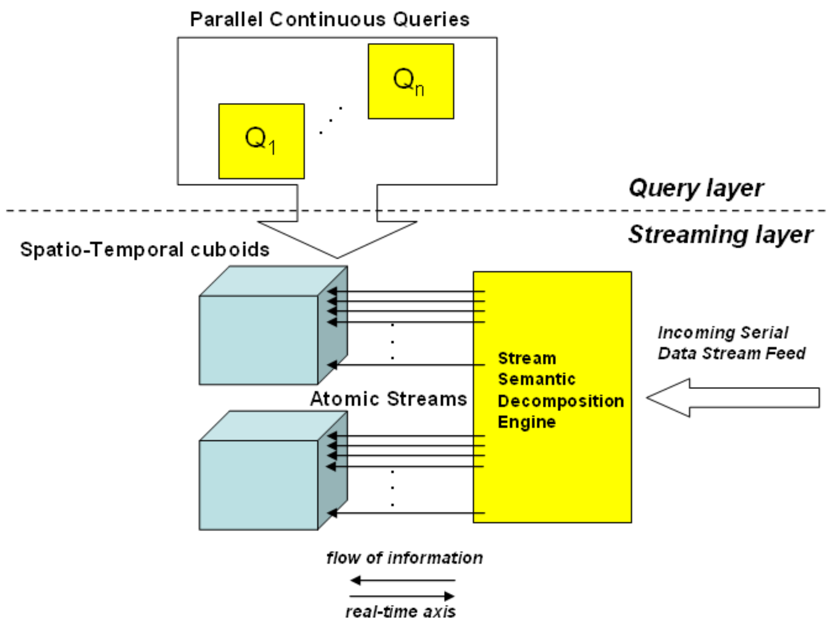


Figure 2-1. A diagram of the Architecture of "Streamonas" based on [1]

It discusses some deep model differences that exist between Oracle CQL and StreamBase StreamSQL. The same work introduces a new model to capture ordering and simultaneity through partial orders on batches of tuples. The researchers in [6] introduce a new architecture for stream systems, out-of-order processing (OOP), which avoids ordering constraints. The architecture has been applied on Gigascope [21] and NiagaraST [23]. The architecture frees stream systems from the burden of order maintenance by using explicit stream progress indicators, such as punctuation or heartbeats, to unblock and purge operators.

In [1] the architectural principles and the high performance characteristics of the Streamonas DSMS based on the Linear Road Benchmark are analyzed. As the theoretical framework presented in this work is linked to the architecture of Streamonas DSMS the following sub-section provides the outline of the architecture originally published in [1].

2.1. Previous Work - The Architecture of Streamonas

The architecture of Streamonas decouples streaming from querying in two independent layers, allowing parallelism at the query level, while keeping uninterrupted the flow of data at the streaming layer [1]. The system indexes the incoming serial data flow based on its specially designed data structure named Spatio-Temporal Cuboid. The Spatio-Temporal Cuboid data structure has a role as fundamental for the Streamonas DSMS as the role of a Relation in a Relational DBMS.

In this work (sections 3, 4 and 5), we present the four novel theoretical models which define the fundamental temporal semantics upon which the Streamonas DSMS has been architected, as also the fundamental temporal semantics for application development on the system.

In order to better present the architecture, we have created a diagram (Figure 2-1) based on [1]. As shown in figure 2-1, the incoming serial data stream feed is decomposed into individual temporal sequences which we name atomic streams. The process is named Stream Semantic Decomposition and it is fundamental for our DSMS. Both the Stream Semantic Decomposition Engine, as also the Spatio-Temporal Cuboids shown in figure 2-1, belong to the Streaming Layer of the architecture.

Queries in the Query Layer of the architecture, can access the Spatio-Temporal cuboids in parallel. The Spatio-Temporal Cuboid data structure is a set of atomic streams. In the general case, this set is static, defined a-priori.

Each atomic stream is uniquely identified by its key and accessed randomly by an expression of the form:

Space[j].Attr[t=0-k]

where j is any valid key and k is an index allowing the historical random access of the temporal sequence of the atomic stream. As an example in the LRB the expression referring to the latest segment (location) element of a car entity having as key Car_Id= "200", is : CAR[200].Segment[0].

Decoupling of querying from streaming is one of the three fundamental principles of the architecture of Streamonas. The other two principles are: (i) Constant time complexity O(1) access of temporal sequences and (ii) Constant time complexity O(1) historical access of any tuple within the temporal scope of the ST-Cuboid.

In-depth theoretical analysis of the architecture and its principles can be found in [1].

3. Fact Generation Model and Query Result Generation Model

The generation of facts in Streamonas follows an absolute time domain as in [2]. The DSMS materializes discrete time instants over a discrete time domain of the system. The base granularity of the system is the CPU clock cycle. The DSMS can support multiple time granularities [2] depending on the application requirements. Other higher level granularities such as milliseconds, seconds etc. can be formed by the CPU clock cycle granularity grouping periodically [2] into these higher level granularities.

In real-time systems we want the base granularity (CPU clock cycle) to be as fine as possible, or equivalently we would like the computational frequency to reach infinity. Current technology sets a greatest lower bound to the base granularity or equivalently a lowest upper bound to the computational frequency making the term “real-time” a theoretical limit which current processing systems cannot achieve (infinite processing frequency).

The generation of a fact by a sensor to be streamed into a DSMS can be modeled at a time instant t at the base granularity of the CPU clock cycle as:

Fact Generation Model:

$$(t, f_{id}(t)) \leftarrow f_{id}(t) \qquad \textbf{(1)}$$

with t being a time instant variable at the base granularity (CPU clock cycle) receiving continuously incremental values by following the “current” CPU clock cycle while this later discretizes real-time. Equivalently we can use the expression: “t follows the CPU clock cycle”, generating a real-time axis.

The unique function f_{id} is an abstract function which models the fact (tuple) generation semantics. This abstract function gave birth to the concept of Atomic Streams as this is described in [1] and for this reason it is named as Atomic Stream generating function (AST generating function). We name the fact $(t, f_{id}(t))$ as a Spatio-Temporal element.

As an example, a sensor monitoring temperature deployed on a geographic location with coordinates (x_1,y_1,z_1) can be modeled with the AST generating function $temp_{x1\ y1\ z1}(t)$ with t following the CPU clock cycle, generating a respective atomic stream.

When we model a sensor environment, the AST generating function f_{id} : (i) May be known. We can use as an example a single sensor sending a beacon-pulse every second. (ii) May not be known but it can be observed or data-mined by the generated facts as of the current CPU clock cycle. As an example, the speed of a car in the LRB is measured and transmitted to the DSMS. While the function describing the individual speed of a car is not known apriori, the facts generated by the abstract function f_{id} can be observed and models can be developed with data-mining methods in order to approximate it.

The AST generating function is a very powerful concept also used in the querying engine of the DSMS itself. Every result produced by a query logic follows the Fact Generation Model, generating an Atomic Stream with timestamps based on the system clock of the DSMS which is available to all applications.

Based on the above and expression (1) we thus receive:

Query Result Generation Model:

$$\textbf{(t, query\_logic}_{id}\textbf{ (t))} \leftarrow \textbf{query\_logic}_{id}\textbf{ (t)} \qquad \textbf{(2)}$$

Query_logic$_{id}$(t) has replaced the very general notion of f_{id} denoting the query processing logic applied on the system on a continuous manner thus implementing the concept of continuous queries on Streamonas. While as explained above, $f_{id}(t)$ is not necessarily known, query_logic$_{id}$(t) of the Query Result Generation model is known and well defined. The result of a query_logic$_{id}$(t) can be of any data type, either system defined or user defined, producing atomic streams of objects such as

query_logic$_{id}$: {(t_0, object$_0$), … , (t_q, object$_q$), …}.

4. Semantic Space–Time model for Streaming

The Semantic Space-Time model for Streaming (Streaming SST-model) is a theoretical model representing the semantics of the streaming process in the DSMS. During the process of its arrival to the system, each spatio-temporal element is timestamped with the CPU clock cycle of its arrival. By using the base CPU clock cycle for time-stamping, the theoretical model enforces that (i) the arrival process is serialized i.e. each STE is exclusively managed during a sequence of clock cycles (in the example of figure 2-2 we have assumed a sequence of 2 clock cycles) and (ii) each STE is uniquely timestamped by the first CPU clock cycle of this sequence.

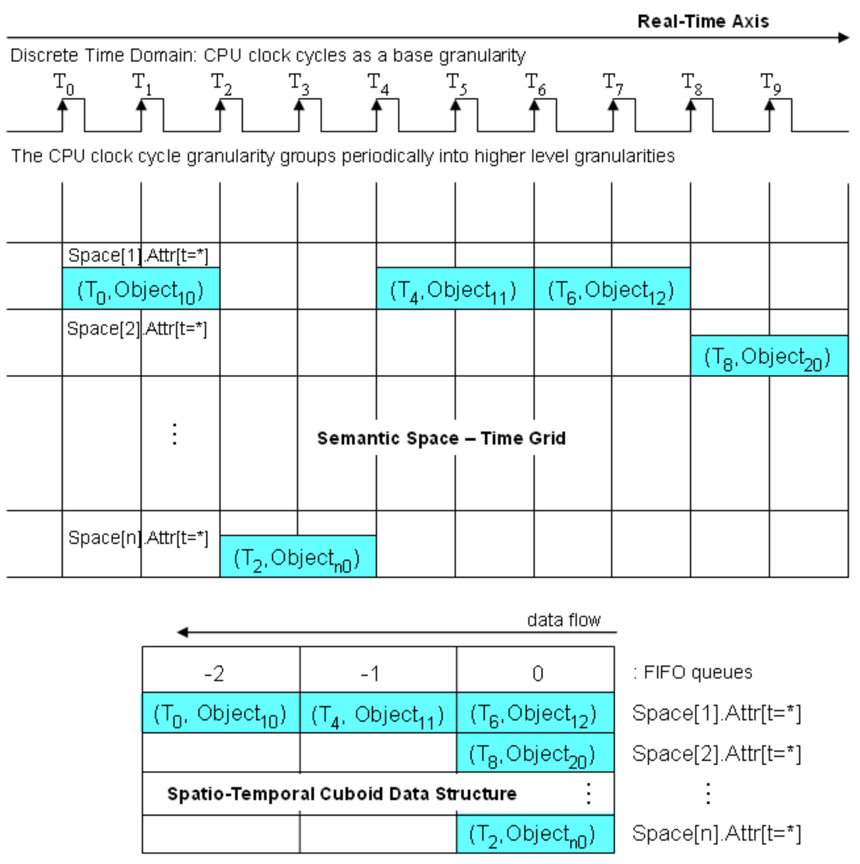


Figure 2-2. The Semantic Space – Time Model for Streaming

We want to stress that we refer to the theoretical model, in reality time-stamping is performed at a higher granularity than the CPU clock cycle based on the overhead required by the system per STE arrival and based on the clock precision of the system while always satisfying (i) and (ii). Figure 2-2 demonstrates an example of a Streaming SST-model. A grid is formed by the atomic streams spanning horizontally, crossed by the CPU clock cycles in the vertical direction (Semantic Space – Time grid). The theoretical model decomposes the individual atomic streams based on their AST unique identification and assigns the spatio-temporal elements (STEs) on their unique position on the grid.

The spatio-temporal grid position is uniquely identified by the semantic space identification (space-id) of the STE combined with the CPU clock cycle which marked the initiation of the processing of each STE (time-id), at the streaming layer of the DSMS in a serialized manner. In the example of figure 2-2, the AST with space-id= “Space[1].Attr[t=*]” is shown with the last 3 spatio-temporal elements arrived as :

{(T_0, $Object_{10}$), (T_4, $Object_{11}$), (T_6, $Object_{12}$)}. In the same example, the STE “$Object_{11}$” is uniquely identified and accessed by the combined space-id = “Space[1].Attr[t=*]” and time-id=T4, or equivalently, by the path expression Space[1].Attr[T4] (the indexes i, j used in the example for the enumeration of the objects as $Object_{ij}$, have only been used to help the explanation of the model and they do not consist part of its semantics).

In the example presented in figure 2-2 each atomic stream of the Spatio-Temporal Cuboid (ST-Cuboid) stores the last 3 spatio-temporal elements arrived in it, in the atomic stream’s FIFO queue. As seen in the example, the ST-Cuboid data structure, compresses the representation of the semantic space – time model for streaming.

The serialization process and the strict total order of the time-stamps, deterministically define the strict total ordering of the STEs in the FIFO queue in their respective atomic stream.

The serialization process, unique timestamping and the decoupling of streaming from querying approach of Streamonas preserves the semantics that can be derived from the incoming facts (STEs) which may be useful for different classes of applications.

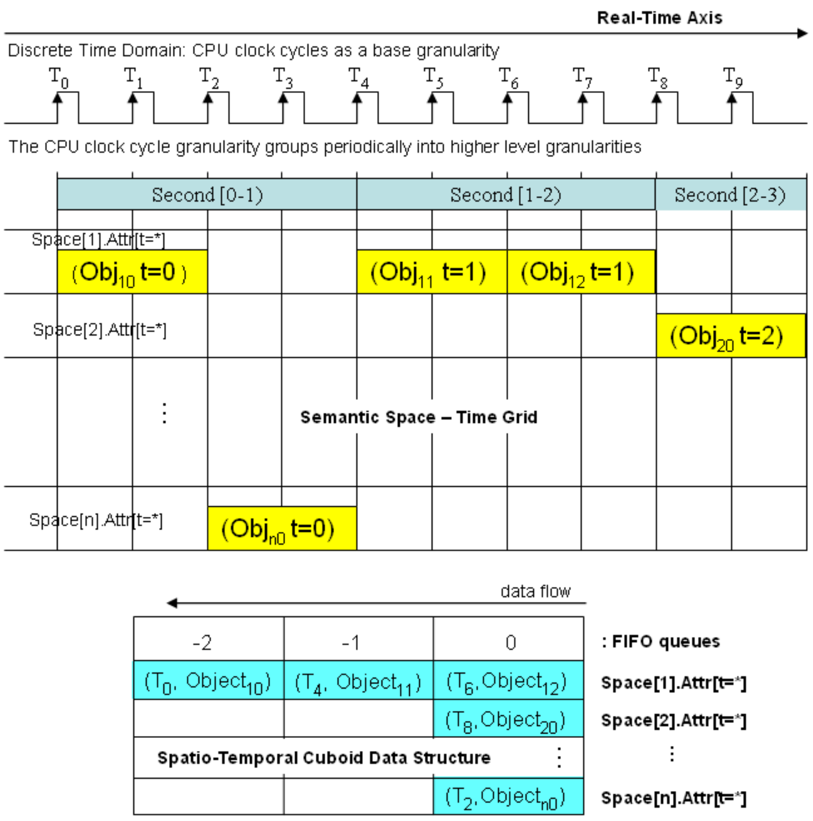


Figure 2-3. The Semantic Space – Time Model for Querying

5. Semantic Space – Time model for Querying

In the Streamonas architecture, querying is decoupled from streaming. As presented in figure 2-3, the CPU cycles and the associated T_i timestamps from streaming (Figure 2-2) are hidden, without explicitly affecting the application semantics. The Semantic Space – Time model for Querying (Querying SST-model) reveals the temporal semantics encapsulated within the objects of the spatio-temporal elements.

As seen in figure 2-3, the STE-objects have revealed their encapsulated timestamps with a time granularity in seconds. The encapsulated timestamps regenerate and semantically represent the strict total order of generation of the STE objects based on the fact generation model (e.g.

from a sensor subsystem). Figure 2-3 demonstrates how the temporal sequence of the timestamps in seconds, modeled by the Querying SST-model, is projected on a real-time axis modeled by the Streaming SST-model.

One important aspect of the decoupling of streaming from querying with the two independent SST-models, is that the application developer may focus on the semantics of the Querying SST-model without being affected by the streaming process. The serialization of the streaming process (described in the previous section) preserves the temporal semantics as they were generated (e.g. by a sensor system) to be used by the Querying SST-model. More specifically, the serialization process of the Streaming SST-model enforces (i) a strict total order of the STEs based on their time of arrival in the DSMS and (ii) a strict total order of the STEs in their respective atomic streams based on their arrival time in the DSMS. In the figure of the Querying SST-model (Figure 2-3), objects Obj_{11} and Obj_{12} have been generated by the sensors during the same second (t=1) while at the Streaming SST-model (Figure 2-2) the same objects during the serialization process are timestamped with T4 and T6 respectively.

The arrival order is indexed in the FIFO queue of each atomic stream. With the assumption that the order of the encapsulated timestamps is preserved upon arrival of the STEs at the DSMS (network protocols such as tcp/ip can ensure reliable and in order delivery of packets of information), the serialization process of the Streaming SST-model preserves the time semantics of the encapsulated information.

Temporal information at different granularities is incorporated within the STE object itself along with additional information. In the LRB application a car transmits its speed as also the time of measurement in seconds. An object tuple is then instantiated (e.g. (Obj12: (speed=60, t=1)) which is then transmitted to the DSMS. The DSMS receives the object through the serialization process at timestamp T6 instantiating the STE by encapsulating the received object as (T6, (Obj12: (speed=60, t=1))).

Different time granularities and different time models may be encapsulated within the objects. Thus from the temporal data encapsulated in the same objects, multiple projections of timestamps on the Real-Time axis may define multiple querying models at different granularities. In the example in figure 2-3, the time model uses a "second" time granularity modeling each second as a half-closed interval. The Semantic Space – Time model for Querying with its own Semantic Space – Time grid allows the modeling of such a time model which is independent from the streaming process time model (described by the Semantic Space-Time model for streaming). As mentioned earlier in this section, multiple time models may be incorporated within the same object depending on the application semantics. Each one of the multiple time models may be modeled by its respective Querying SST-model.

We realize from the above that from the same single Streaming SST-model, a number of independent Querying SST-models may be generated, enabling for query logic to be developed for different classes of applications and application semantics.

As a result the decoupled streaming/querying architecture with its associated Semantic Space – Time models allows the flexibility in query modeling while it is re-using the data stream objects stored in the Spatio-Temporal cuboid data structure in the streaming layer.

In their work the authors in [9] provide the example of the stream S2:

S2(value;time)=(10;1) (20;1) (30;3) (40;4)

mentioning that the two tuples (10;1) and (20;1) with time=1 seem to "evaporate" since they have the same timestamp and no further temporal distinction is possible between them. We shall use the same example in order to demonstrate how the architecture of Streamonas based on its SST-models does not allow “evaporation” of tuples.

During the serialized process of stream semantic decomposition and based on the SST streaming model, the DSMS uniquely time-stamps the tuples thus avoiding the “evaporation” of the two tuples, i.e. for the specific example the system would generate an atomic stream of the form:

AST_1: (T1; (Obj_{11}:(10;1))) (T2; (Obj_{12}:(20;1)))

(T3; (Obj_{13}:(30;3))) (T4; (Obj_{14}:(40;4)))

At the querying (application) layer the Querying SST-model hides the timestamps from streaming and reveals the temporal semantics encapsulated within the objects. For a filter of the form:

Obj_{ij} : (* , t=1)

the Querying SST-model receives from the atomic stream the STEs (Obj_{11}:(10;1)) and (Obj_{12}:(20;1)). The STEs are uniquely identified based on the timestamps T1 and T2 of the Streaming SST-model thus avoiding “evaporation”.

Each different application can manipulate the same atomic stream based on its individual semantics. As an example it can either choose to use any of the values of the STE-objects based on a criterion (e.g. the latest received value would be 20 in our example), or it may choose to apply a statistical function on the values of both objects (e.g. in case where the values of the objects are collected through a sampling process).

6. Experimental Results and Performance Evaluation

Aurora [10] was the first system to use the benchmark reaching a level of 2.5 XWays while meeting the response time requirements for Tolls. SPC [11] has run the Linear Road Benchmark 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. Commercial database systems as analyzed in [12] 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. The work in [14] reached a level of 1.5 expressways on the benchmark. The experimental results of Streamonas at the highest level of difficulty of the benchmark (10 XWays) has been published in [1]. In this section we provide the extensive experimental results for each level of difficulty of the benchmark demonstrating the power of the theoretical SST-models we present.

6.1. Query Latency

Table 2-1 provides the maximum and average query latency of the system when the difficulty level of the LRB scales up. The table provides the results when the system received load from 2, 4, 6, 8 and 10 expressways, thus reaching the maximum level of difficulty the benchmark supports (i.e. 10 XWays). At the level of 10 XWays the system demonstrates an excellent performance, being 192,307 times faster than the 5 seconds hard real-time constraint the benchmark sets. Both the maximum query latency and the average query latency were estimated based on all tuples streamed in the system during the 3 hr simulation at each level. The average query latency reported in Table 2-1 includes also the time for query-specific operations that bring the database into a consistent state on a per-tuple basis.

Table 2-1. Maximum and Average query latencies at various levels of difficulty of the benchmark

#XWays	Maximum Query Latency (seconds)	Average Query Latency (seconds)
10	0.139580	0.000026
8	0.182828	0.000025
6	0.192315	0.000024
4	0.082589	0.000018
2	0.023516	0.000016

If we had not included the time for real-time consistency operations, the query latency would have been lower. In-depth analysis of the concept (real-time consistency) will be provided in a future publication.

6.2. System Throughput

Table 2-2 provides the average throughput and average incoming flow rate for levels 2, 4, 6, 8 and 10 of the benchmark.

Table 2-2. Average Throughput and Average Incoming flow at various levels of difficulty of the benchmark

#XWays	Average Throughput (tuples/sec)	Average Incoming Flow Rate (tuples/sec)
10	66,226	20,368
8	67,081	22,546
6	66,285	23,681
4	66,829	30,611
2	67,125	34,406

6.3. Volume of data

Table 2-3 provides information regarding the volume of the streamed data in the DSMS within the 3 hour simulation, the size of the input file, the number of tuples the Data Generator tool of the Linear Road Benchmark has streamed in our DSMS, the simulation duration and the maximum number of cars each one of the XWay levels includes. The Data Generator tool applied delays between bursts of data, adjusting for the different volumes of data streamed in the DSMS at each level, during the 3 hour simulation.

Figure 2-4 demonstrates a graph of the average query latencies at various levels of difficulty of the benchmark. The averages were estimated based on the incremental counting and the cumulative sum of the query latencies, converging to the values presented in Table 2-1. As referred in section 6.1, query latencies include also the time for query-specific real-time consistency operations, which are applied upon arrival of each tuple. Figure 2-5 provides a detailed graph of the query latency for 2, 4, 6, 8 and 10 XWay levels during the 3 hour simulation for each level. The query latency at each level was sampled every 100,000 tuples. Both figure 2-4 and figure 2-5 demonstrate the excellent stable characteristics of the system.

Table 2-3. Volume of Data: Size of the input file, number of tuples, simulation time and maximum number of cars at various levels of difficulty

#XWays	**Size of input file (MB)**	**# tuples**	**Simulation duration (seconds)**	**Maximum # of cars**
10	6,536	120,332,680	10792	1,373,327
8	5,183	96,135,927	10784	1,098,476
6	3,866	72,100,236	10786	824,421
4	2,553	47,977,090	10785	549,481
2	1,241	24,085,768	10790	244,708

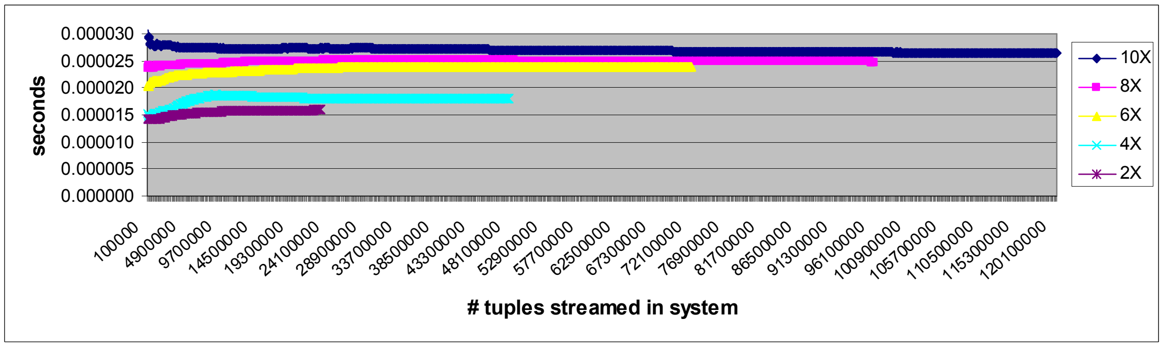


Figure 2-4. Average query latencies of the system at various levels of difficulty of the benchmark. Averages were estimated incrementally during a 3 hour simulation for each level

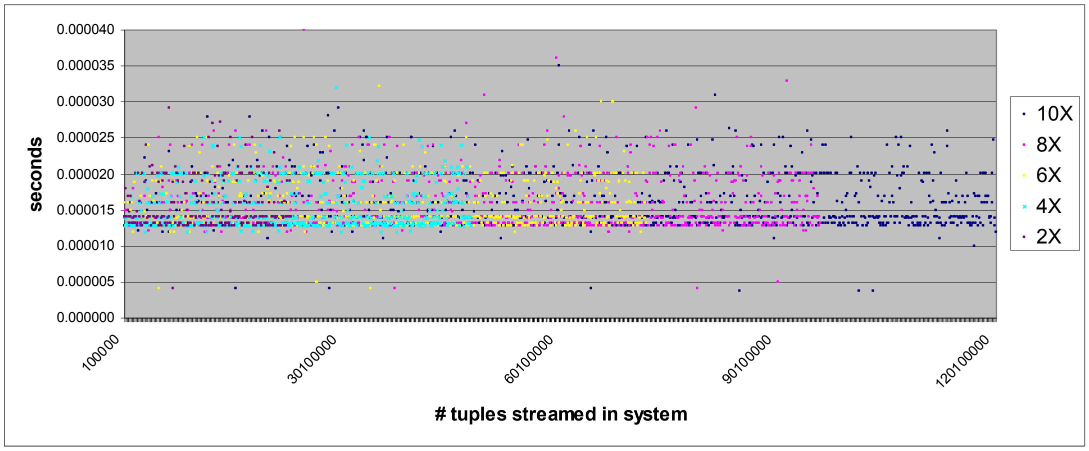


Figure 2-5. Query latencies of the system at various levels of difficulty of the benchmark. The query latency at each level was sampled every 100,000 tuples during a 3 hour simulation

7. Conclusion

As analyzed in [1], the Streamonas DSMS managed to run the maximum level of difficulty the Linear Road Benchmark supports (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. The excellent experimental results have been achieved based on a well defined theoretical framework which is presented in this work. The models presented describe the fundamental temporal semantics upon which Streamonas has been architected, as also the fundamental temporal semantics for application development on the system.

Four novel theoretical models have been presented. (i) The Fact Generation Model models the continuous fact (tuple) generation by a system of sensors, to be streamed in the DSMS. (ii) The Query Result Generation Model models data stream flows from continuous query results. (iii) The Semantic Space-Time model for Streaming with the Stream Semantic Decomposition applies a serialization ensuring the preservation of the temporal semantics of the incoming atomic streams and (iv) the Semantic Space-Time model for Querying which is extensively used for application development on Streamonas.

The integrated architecture based on the Streaming and Querying SST-models results in the following very important and powerful concept:

The independent semantics per query application over the same atomic streams are applied in a parallel manner based on multiple Querying SST-model(s), decoupled by the streaming semantics which are internally and transparently preserved by the system based on the serialization process of the Streaming SST-model.

8. References

[1] P. A. Michael, D. S. Parker. "Architectural Principles of the Streamonas Data Stream Management System and Performance Evaluation based on the Linear Road Benchmark". *In Proc. of the 2008 IEEE International Conference on Computer Science and Software Engineering (2008 CSSE)*, Wuhan, China, December 2008.

[2] C. Bettini, S. Jajodia, S. X. Wang. *"Time Granularities in Databases, Data Mining, and Temporal Reasoning"*. Springer 2000.

[3] C. Zaniolo, S.Ceri, C.Faloutsos, R.T. Snodgrass, V. S. Subrahmanian and R. Zicari. *"Advanced Database Systems"*. Morgan Kaufmann, 1997.

[4] R. T. Snodgrass (ed.), I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Käfer, N. Kline, K. Kulkanri, C. Y. T. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. *"The TSQL2 Temporal Query Language"*. Kluwer Academic, Norwell, MA, 1995.

[5] H. Moon, C. Curino, A. Deutsch, C.-Y. Hou,
C. Zaniolo. "Managing and Querying Transaction-time Databases under Schema Evolution". *In Proc. of the 34th VLDB Conference, Auckland, New Zealand*, 2008.

[6] J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos,
T. Johnson, and D. Maier. "Out-of-Order Processing: A New Architecture for High-Performance Stream Systems". *In Proc. of the 34th VLDB Conference, Auckland, New Zealand*, 2008.

[7] D. S. Parker, R. R. Muntz, H. L. Chau. "The Tangram stream query processing system". *In Proc. of the Fifth International Conference on Data Engineering*, 1989.

[8] H. Thakkar, B. Mozafari, and C. Zaniolo. "Designing an Inductive Data Stream Management System: the Stream Mill Experience". *In Proc. of the Second International Workshop on Scalable Stream Processing Systems, Nantes, France,* 2008.

[9] S. Zdonik, N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, M. Cherniack, U. Cetintemel, and R. Tibbetts. "Towards a Streaming SQL Standard". *In Proc. of the 34th VLDB Conference, Auckland, New Zealand*, 2008.

[10] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. "Aurora: a new model and architecture for data stream management". *The VLDB Journal*, 2003.

[11] N. Jain, L. Amini, H. Andrade, R. King, Y. park, P. Selo, C. Venkatramani. "Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core". *In Proc. of the 2006 ACM SIGMOD Intl. Conference on Management of Data*, 2006.

[12] A. Arasu, M.Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbets. "Linear Road: A Stream Data Management Benchmark". *In Proc. of the 30th Intl. Conference on Very Large Data Bases*, 2004.

[13] http://www.cs.brandeis.edu/~linearroad/

[14] M. Svensson. *"Benchmarking the performance of a data stream management system"*. MSc thesis report, Uppsala University, November 2007.

[15] D. Stott Parker. "Stream Data Analysis in Prolog". *In L. Sterling, ed., The Practice of Prolog, Cambridge, MA: MIT Press*, 1990 (abstract available at: http://www.cs.ucla.edu/~stott/sdb/abstracts.html).

[16] Y. Law, H. Wang, C. Zaniolo. "Query Languages and Data Models for Database Sequences and Data Streams". *In Proc. of the 30th VLDB Conference 2004*.

[17] D J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. "The design of the Borealis stream processing engine". *In Proc. of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[18] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein and J. Widom. STREAM:"The Stanford Stream Data Manager". *In Proc.of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, 2003.

[19] S. Chandrasekaran, O. Cooper, A. Seshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurhty, S. Madden, V. Raman, F. Reiss and M. Shah. "TelegraphCQ: Continuous dataflow processing for an uncertain world". *In Proc. of the 2005 Conference on Innovative Data Systems Research (CIDR)*, 2003.

[20] M. Balazinska, H. Balakrishnan and M.Stonebraker. "Load Management and High Availability in the Medusa Distributed Stream Processing System". *In Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, 2004.

[21] C. Cranor, T. Johnson, O. Spataschek, V. Shkapenyuk. "Gigascope: A Stream Database for Network Applications". *In Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, pages 647-651, June 2003.

[22] S. R. Madden, M. A. Shah, J. M. Hellerstein and V. Raman. "Continuously adaptive continuous queries over streams". *In Proc. of the 2002 ACM SIGMOD Intl. Conference on Management of Data*, 2003.

[23] Tucker, P., et al. "Exploiting Punctuation Semantics in Continuous Data Streams". *IEEE Trans. on Knowledge and Data Engineering, 15(3)*, May 2003.

CHAPTER THREE

REAL-TIME SPATIO-TEMPORAL DATA MINING WITH THE "STREAMONAS" DATA STREAM MANAGEMENT SYSTEM

ABSTRACT

Data Stream Management Systems (DSMSs) have not yet reached a mature enough stage to effectively run data mining algorithms, as they still face challenges within the streaming environment. Streamonas DSMS, as presented in a recent publication, is the first DSMS to reach the maximum level of difficulty supported by the Linear Road Benchmark which is 10 Expressways. The powerful engine of Streamonas can manage an input stream of 20,368 tuples/second with an average query latency of 0.000026 seconds, 192,307 times faster when compared to the 5 seconds maximum query latency the benchmark allows. The on-line data mining over streams presented in this work, is the first effort to apply spatio-temporal data mining algorithms on the Streamonas DSMS system. Dynamic Clustering of spatio-temporal subsequences in real-time has been performed successfully, within the large space, high bandwidth, heavy load Linear Road Benchmark streaming platform. Dynamic Clustering queries have been expressed in a novel SQL-like language, which we name Streamonas-SQL.

1. INTRODUCTION

This work uses as a platform the streaming environment of the Linear Road Benchmark (LRB) [1, 2, 12-15] at the maximum level of its difficulty, i.e. 10 XWays. The high-performance results of the Streamonas Data Stream Management System (DSMS) on the LRB have been published in [1, 2]. The engine of Streamonas while tested at the maximum level of difficulty the Linear Road Benchmark supports (10 XWays) managed an input stream of 20,368 tuples/second with an average query latency of 0.000026 seconds, 192,307 times faster when compared to the 5 seconds maximum query latency the benchmark allows. The results of this work, presented in the following sections, demonstrate that Streamonas can effectively perform large space, high bandwidth, heavy load dynamic clustering of spatio-temporal subsequences in real-time within the Linear Road Benchmark streaming platform, at an average query latency of 0.000027 seconds. The dynamic clustering querying is expressed in a novel SQL-like language with the name Streamonas-SQL.

2. Previous Work

Representation of the structure of an event sequence with Non-deterministic Finite Automata (NFA) has been used by [9, 11]. NFA are also used by the SASE event language [3, 4] in order to read query-specific event sequences efficiently from continuously arriving events. Other event systems are based on fixed data structures such as trees [8] , finite automata [9] and Petri nets [10].

The researchers in [3] achieve excellent performance results based on their own event generator platform. In this work we present our first efforts to apply on-line spatio-temporal data mining over streams on the Streamonas DSMS platform within the streaming environment of the Linear Road Benchmark along with performance evaluation. For pattern matching we have used as similarity metric the correlation coefficient between a given pattern and the incoming streaming information. While the correlation coefficient is a simple widely used methodology for off-line similarity measurement, the work [6] referred by [5], analyze that for Complex Event Processing and pattern matching in sequences of rows,

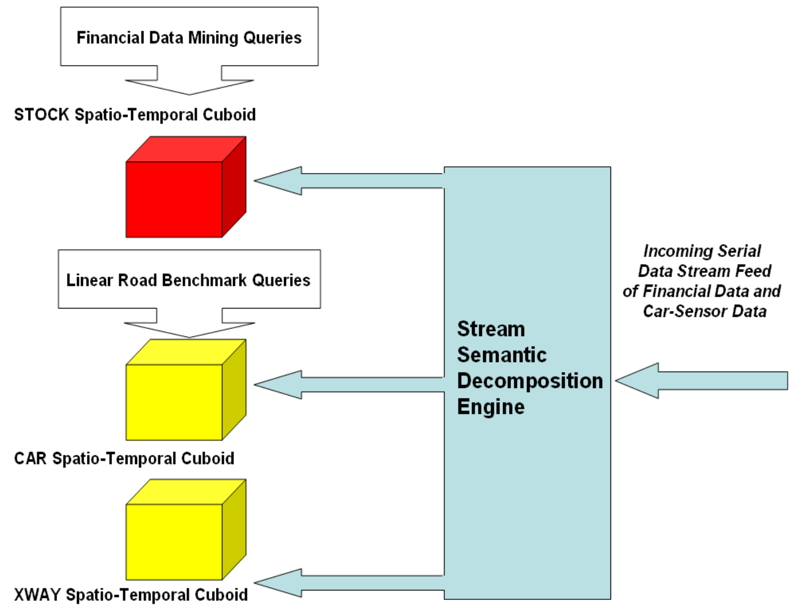

Financial Data Mining Queries
STOCK Spatio-Temporal Cuboid
Linear Road Benchmark Queries
CAR Spatio-Temporal Cuboid
XWAY Spatio-Temporal Cuboid
Stream Semantic Decomposition Engine
Incoming Serial Data Stream Feed of Financial Data and Car-Sensor Data

Figure 3-1. Architecture and Database Design of an Integrated Streamonas platform for simultaneous Data Mining queries, with different semantics, on Financial and Linear Road Benchmark streams.

the correlation aggregate is illegal when applied on groups of rows, as the two groups, depending on their respective filtering predicates, may have different number of rows. In [19] it is emphasized that extensive bibliography exists on spatio-temporal databases. The researchers in [24] introduce a stream processing paradigm of functional transformations (transducers) on streams. Aurora [23] has introduced an architecture based on a data-flow model and an algebra with a set of operators to express its stream processing requirements. The project STREAM [25] has built a Data Stream Management System prototype which supports a large class of declarative continuous queries over continuous and traditionally stored data sets.

3. Spatio-temporal data-mining on Streamonas

3.1. Architectural Outline

As extensively analyzed in [1, 2] the Streamonas architecture follows a two layer architecture where streaming is decoupled from querying (Figure 3-1). The incoming serial data stream is decomposed into specially designed data-structures, which store uniquely identified temporal sequences with the name Atomic Streams (ASTs). Special data structures called Spatio-Temporal Cuboids (ST-Cuboids) isolate and index the atomic streams. The incoming serial stream consists of data from the stock market, as also from car sensors. The Stream Semantic Decomposition Engine, decomposes the data into semantically meaningful temporal sequences (Atomic Streams), which populate the respective spatio-temporal cuboids.

3.2. Database Design

ST-Cuboids have a role as fundamental for our DSMS as the role of a Relation in a Relational DBMS. Figure 3-1, presents the spatio-temporal cuboids of the database supporting simultaneously the Linear Road Benchmark queries as also the Financial Data Mining queries. We would like to emphasize the database centric nature of Streamonas which allows the integration of multiple streaming sources with different semantics within the same database model. This database centric logic, allows the reusability of the streaming information, by allowing queries of different applications to run in parallel and access the underlying database.

In an equivalent manner as in the Relational Databases theory, the ST-Cuboid Database Schema is the collection of schemas for the ST-Cuboids in the database. In order to support the data mining application within the streaming environment of the Linear Road Benchmark we define the following ST-Cuboid Database Schema:

STOCK (Stock_Id:int, Price(t):AST(10))
CAR (Car_Id:int, Speed(t):AST(6), Segment(t):AST(6))
XWAY (XWay:int, Segment:int, Direction:int, Lane:int,
AVGS:AST(6), LAV:AST(6))

The ST-Cuboid *STOCK* stores the financial atomic streams while the ST-Cuboids CAR and XWAY store atomic streams generated from the Linear Road Benchmark simulating generation of data from car sensors.

The ST-Cuboid schemas include attributes of type Atomic Stream as also their historical span in parentheses (e.g. historical span is 10 for the atomic stream STOCK.Price(t), while it is 6 for the atomic stream XWAY.AVGS(t)).

The attribute Stock_Id describes the unique identification of each stock. The attribute STOCK.Price(t) of type atomic stream, models the evolving price of each stock. The attribute CAR.Car_Id models the static over time unique identification of each car, while CAR. Speed(t) of type atomic stream, models the evolving speed of each car. CAR. Segment(t) models the evolving segment (location) of each car on an XWay. In an equivalent manner the attributes of the ST-Cuboid XWAY, XWay, Segment, Direction and Lane are defined. The attributes AVGS and LAV of the ST-Cuboid XWAY model statistical information based on the specifications of the LRB.

3.3. Real-time clustering of spatio-temporal subsequences based on the correlation similarity metric

As also emphasized in [17], central to all goals of cluster analysis is the notion of the degree of similarity. For real-time cluster analysis of spatio-temporal patterns on the Streamonas Data Stream Management System, we have used correlation as a similarity metric [17]. More specifically we have used the correlation coefficient computational version analyzed in [18] (Eq. (1)) :

$$r = \frac{\Sigma xy - \frac{\Sigma x \, \Sigma y}{n}}{\sqrt{(\Sigma x^2) - \frac{(\Sigma x)^2}{n}} \sqrt{(\Sigma y^2) - \frac{(\Sigma y)^2}{n}}} \qquad \text{Eq. (1)}$$

3.4. Simplicity of Querying with Streamonas-SQL

Querying on Streamonas is expressed with an SQL-like language with the name Streamonas-SQL. Figure 3-2 shows the SQL statement expressing the query used during our experiments. While we shall extensively analyze Streamonas-SQL in a future publication, in this work we provide a general description of the Streamonas-SQL statement presented in Figure 3-2, emphasizing its simplicity. Streamonas-SQL manages tuples of atomic streams, rather than tuples of non-temporal information. The SELECT clause of the statement, returns the atomic streams that satisfy the predicate over time, the FROM clause of the statement refers to the ST-Cuboid STOCK and the predicate Correlation_function (Price, Pattern1) > 0.95, receives as arguments two atomic streams (the stock price and the pattern) and returns their correlation based on equation Eq (1). The query is being re-evaluated upon each arrival of a new tuple. Efficiency is achieved by evaluating only the delta results of the query within the scope of the new arrival. It is important to mention that Streamonas-SQL differs from other research works such as [3, 4, 6] as it does not include the pattern definition within the Query Expression. The object-oriented design of Streamonas allows the pattern to be stored as a data object in the database (as an atomic stream) and encapsulates the processing logic of each atomic stream in a respective function which we name Atomic Temporal Function (ATF). Correlation_function (Price, Pattern) is an example of an ATF. The query in figure 3-2 dynamically evaluates the members of a cluster of spatio-temporal subsequences for any stock in the database, where the correlation coefficient of the subsequence with the pattern is larger than 0.95. As shown in the experimental results, the population of the cluster with new members is performed with an average query latency of 0.000016 seconds.

4. EXPERIMENTAL RESULTS

4.1. PREPARATION OF THE SERIAL DATA STREAM AND TESTBED

We have used stock market data from [16]. Real data from seven indexes (NASDAQ 100, SP 500, Dow Jones, Diamonds, QQQQ and Spyder) have been multiplexed together and stored in a single file of 122.7MB. This single file has been used as an initial stress-test of the system at high-bandwidth. The tuples were also enumerated for reference purposes during the experiment. A number of 6,385,295 tuples were included in the file. A second file was also created by multiplexing the stock data with the car sensor data of the Linear Road Benchmark at a level of 10XWays. This second file has a size of 6.3GB consisting of 126,717,975 tuples. The Linear Road Benchmark environment was used in order to test the effectiveness of Streamonas under heavy load. A single PC was used having as CPU a Pentium4 processor running at 3GHz with 4GB RAM and a 100GB hard disk. During the second experiment we have used the same data driver tool of the LRB as in [1] and [2].

4.2. High Bandwidth Dynamic Clustering of spatio-temporal subsequences in real-time with standalone performance evaluation on Streamonas

Our first experiment measured the performance of the Streamonas DSMS when applying dynamic clustering on a number of 7 stocks at high bandwidth. A "W" shaped pattern (Figure 3-2) was chosen as the center of the cluster. The cluster was being populated in real-time with a number of spatio-temporal patterns from any stock that would satisfy the predicate. Two members of the cluster and their respective correlations are presented in Figure 3-2. By increasing the threshold to r > 0.99, two only spatio-temporal patterns satisfied the predicate. The bandwidth of the experiment was 63,461 tuples/second while the average query latency (of the sample points) was 0.000016 seconds (Figure 3-3a). The first experiment was performed by reading data directly from disk and includes the overhead from the stream semantic decomposition. A second experiment was performed by buffering decomposed stock data in memory and then streaming it into the system. The clustering was performed at an average bandwidth of 225,371 tuples/second with an average query latency (of the sampled points) 0.000004 seconds (measurements do not include the Stream Semantic Decomposition overhead) (Figure 3-3b).

4.3. Large Space - High Bandwidth – Heavy Load Dynamic Clustering of spatio-temporal subsequences in real-time within the Linear Road Benchmark streaming platform

The experiments in section 4.2 were performed over a relatively small number of stocks (7 stocks). We wanted to stress test the Streamonas system under the heavy load of the Linear Road Benchmark as described in [1, 2]. We have applied the same dynamic clustering described in 4.2 on the financial information multiplexed with the car-sensor data, stored in a 6.3GB file as described in section 4.1.

4.3.1. Dynamic Clustering under the Heavy Load of the LRB

A first experiment measured the performance of the system within the Linear Road Benchmark environment at its maximum level of difficulty (10XWays), loaded also with the dynamic clustering queries for the financial data. The average, maximum and minimum query latencies for this experiment (based on all tuples streamed into the system) are: 0.000026 seconds, 0.109826 seconds and 0.00003 seconds respectively.

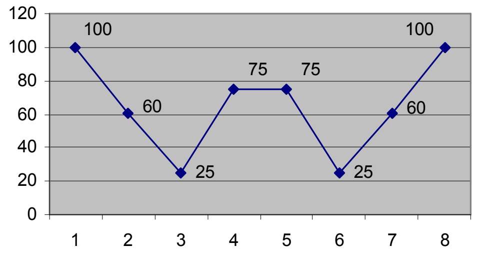
4.3.2. Large Space – High Bandwidth – Heavy Load Dynamic Clustering

We wanted to stress test Streamonas in a scenario where patterns are searched within a larger space than the semantic space [2] defined by the 7 stocks. For this reason we performed a second experiment where we applied dynamic clustering of spatio-temporal subsequences over the large semantic space of the speeds of the 1,373,327 cars of the LRB (10 XWays), simultaneously with the dynamic clustering of the 7 stocks (historical span was 6 for all ASTs). In an example application we wanted to identify patterns of driving behavior (e.g. any car accelerating or decelerating based on the predefined pattern). The average, maximum and minimum query latencies for this experiment are: 0.000027 seconds, 0.188051 seconds and 0.00003 seconds respectively, evaluated based on each one of the tuples streamed into the system. The bandwidth mapping of the system during the 3hr simulation is presented in Figure 3-4.

Streamonas-SQL Query:

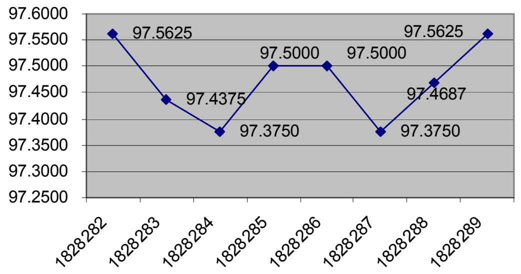
```
SELECT *
FROM Stock
WHERE
Correlation_function
( Price, Pattern1 ) > 0.95
```

Cluster Center: Pattern 1



Dynamic Cluster for r>0.95
Average Bandwidth of streaming data: 63,461 tuples/sec
Average Query Latency 0.000016 seconds

Stock:4 Correlation Coef.: 0.991831 Tick#: 1828289



Stock:7 Correlation Coef.: 0.954671 Tick#: 1157213

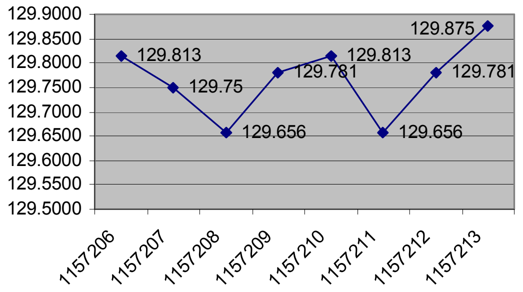


Figure 3-2. A Streamonas-SQL Query developing dynamic clusters of spatio-temporal subsequences in real-time.

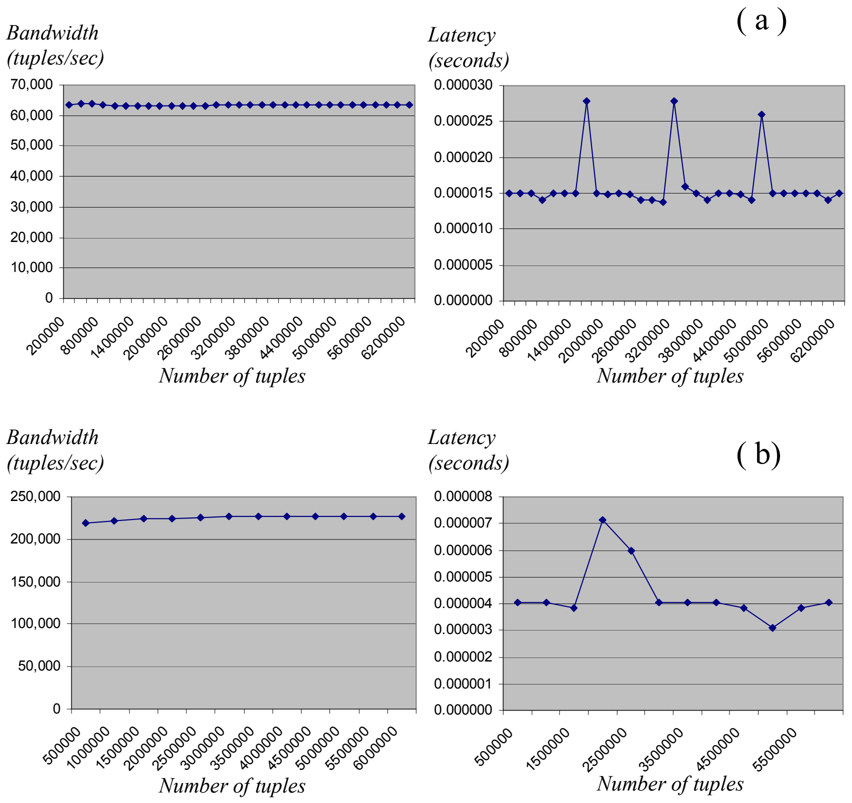


Figure 3-3. Average Bandwidth and Average Latency during Dynamic Clustering of Spatio-Temporal subsequences on Streamonas.

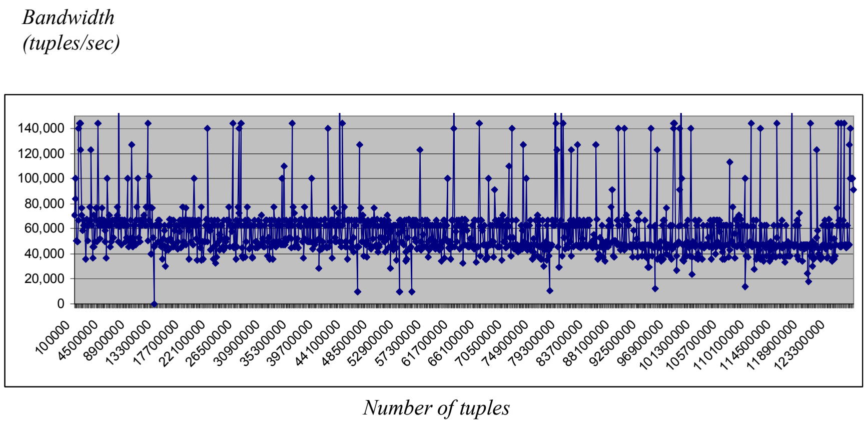


Figure 3-4. Bandwidth mapping during Dynamic Clustering of Spatio-Temporal subsequences on Streamonas for financial data and car-sensor data within the environment of the Linear Road Benchmark.

5. Conclusions

We have performed Large Space (more than 1.3 million evolving entities), High Bandwidth (63,461 tuples/sec on average), Heavy Load (6.3GB of data) Dynamic Clustering of spatio-temporal subsequences in real-time on the Streamonas Data Stream Management System. The experiments were performed within the environment of the Linear Road Benchmark at its maximum level of difficulty (10 XWays). In all our experiments, Streamonas performed excellently with an average query latency of 0.000027 seconds. Dynamic clustering was expressed in the novel Streamonas-SQL language.

6. References

[1] P. A. Michael, D. S. Parker. "Architectural Principles of the Streamonas Data Stream Management System and Performance Evaluation based on the Linear Road Benchmark". *In Proc. of the 2008 International Conference on Computer Science and Software Engineering (2008 CSSE)*, IEEE, Wuhan, China, December 2008.

[2] P. A. Michael, D. S. Parker. "The Semantic Space-time models of the Streamonas Data Stream Management System". *In Proc. of the 2009 World Congress on Computer Science and Information Engineering (2009 CSIE)*, IEEE, Los Angeles / Anaheim, USA, March 2009.

[3] E. Wu, Y. Diao, S. Rizvi. "High-Performance Complex Event Processing over Streams". *In Proc. of the 2006 ACM SIGMOD Intl. Conf. on Management of Data,* Chicago, Illinois, USA, 2006.

[4] D. Gyllstrom, Y. Diao, E. Wu, P. Stahlberg, H. Chae, G. Anderson. "SASE: Complex Event Processing over Streams", *CIDR*, 2007.

[5] N. Jain, J. Gehrke, J. Widom, H. Balakrishnan, U. Cetintemel, M. Cherniack, R. Tibbetts, S. Zdonik. "Towards a Streaming SQL Standard". *In Proc. of the 34th VLDB Conference*, Auckland, New Zealand, 2008.

[6] Anonymous: Pattern Matching in Sequences of Rows, SQL standard proposal, http://asktom.oracle.com/tkyte/row-pattern-recogniton-11-public.pdf, March, 2007.

[7] S. Reza, C. Zaniolo, A. Zarkesh, J. Adibi: Expressing and optimizing sequence queries in database systems. ACM *Transactions Database Systems. 29 (2): 282-318*, 2004.

[8] S. Chakravarthy, V. Krishnaprasad, E. Anwar, S. Kim. "Composite events for active databases: Semantics, contexts and detection". *In VLDB*, 1994.

[9] N.H. Gehani, H.V. Jagadish, O. Shmueli. "Composite event specification in active databases: Model and implementation". *In VLDB*, 1992.

[10] S. Gatziu and K.R. Dittrich. Events in an active object-oriented database system. *In Proc. of the 1st International Conference on Rules in Database Systems*, 1993.

[11] Y. Diao, M. Altinel, H. Zhang, M.J. Franklin, P.M. Fisher. "Path sharing and predicate evaluation for high-performance XML filtering". *TODS, 28(4), 467-516*, December 2003.

[12] A. Arasu, M.Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbets. "Linear Road: A Stream Data Management Benchmark". *In Proc. of the 30th Intl. Conference on Very Large Data Bases*, 2004.

[13] Linear Road Benchmark, http://www.cs.brandeis.edu/~linearroad/

[14] N. Jain, L. Amini, H. Andrade, R. King, Y. park, P. Selo, C. Venkatramani. "Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core". *In Proc. of the 2006 ACM SIGMOD Intl. Conference on Management of Data*, 2006.

[15] M. Svensson. "Benchmarking the performance of a data stream management system". *MSc thesis report*, Uppsala University, November 2007.

[16] Price-Data, http://www.price-data.com/

[17] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* Springer, pp. 453-457, 2001.

[18] S. Bernstein, R. Bernstein. *Elements of Statistics II.* McGraw Hill, pp. 344-349, 1999.

[19] C. Zaniolo, S.Ceri, C.Faloutsos, R.T. Snodgrass, V. S. Subrahmanian and R. Zicari. *"Advanced Database Systems"*. Morgan Kaufmann, pp. 100-124, 1997.

[20] J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos,
T. Johnson, and D. Maier. "Out-of-Order Processing: A New Architecture for High-Performance Stream Systems". *In Proc. of the 34th VLDB Conference*, Auckland, New Zealand, 2008.

[21] D. S. Parker, R. R. Muntz, H. L. Chau. "The Tangram stream query processing system". *In Proc. of the Fifth International Conference on Data Engineering*, 1989.

[22] H. Thakkar, B. Mozafari, and C. Zaniolo. "Designing an Inductive Data Stream Management System: the Stream Mill Experience". *In Proc. of the Second International Workshop on Scalable Stream Processing Systems*, Nantes, France, 2008.

[23] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. "Aurora: a new model and architecture for data stream management". *The VLDB Journal*, 2003.

[24] D. Stott Parker. "Stream Data Analysis in Prolog". In L. Sterling, ed.,
The Practice of Prolog, Cambridge, MA: MIT Press, 1990 (abstract available at: http://www.cs.ucla.edu/~stott/sdb/abstracts.html).

[25] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein and J. Widom. STREAM:"The Stanford Stream Data Manager". *In Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, 2003.

CHAPTER FOUR

THE MULTICORE DISTRIBUTED STREAMONAS ARCHITECTURE – AND ITS PERFORMANCE EVALUATION BASED ON THE LINEAR ROAD BENCHMARK

ABSTRACT

Scaling up Data Stream Management Systems (DSMS) from a shared memory model to a message-passing model introduces a number of challenges, with the interconnection network being a serious bottleneck for overall system performance. The challenges are still greater when query latency needs to be maintained at very low levels in order for the system to have real-time characteristics. Another factor to be considered is that efficiency is of increasing importance globally, both for environmental and economic sustainability, attracting strong interest from Governments and Regulatory organizations.

This work presents the Distributed Streamonas DSMS. It rests on a distributed, object-oriented, parallel data model of spatio-temporal information – ST-Cuboids – that can provide both high performance and efficiency in Stream Management. The result is an innovative, highly parallel and scalable DSMS with extremely low query latency, on the order of a millisecond. In this paper we describe its data model and implementation of its multicore architecture, and explain how it reached a performance of 550 Xways on the Linear Road Benchmark as well as an average efficiency of about 14 Xways per core, which to our knowledge are both records.

The system demonstrated robustness in supporting large-scale enterprise level applications, by managing a data volume above 350 GBytes in less than 3 hours and an average backbone network data rate of almost 600 Mbits/sec, monitoring approximately 75 million evolving entities (cars) with an average query latency below 400 μsec. The above mentioned excellent

characteristics make the distributed system ideal for large-scale, real-time applications within the framework of "smart city" infrastructures.

1. Introduction

For almost a decade, the Linear Road Benchmark has been established as a common benchmarking platform on which Data Stream Management Systems (DSMSs) can demonstrate novel characteristics and permit comparisons in performance.

The pioneering Aurora DSMS [1] both introduced the benchmark and proved that relational DBMSs were not suited for data streaming applications. Since then a number of DSMSs have been designed with their researchers reporting results on the benchmark. The scale used by the benchmark is based on the number of expressways supported by a DSMS in an Intelligent Transportation System scenario, while meeting real-time performance constraints. With a single-cpu/single-core system, Aurora reported 2.5 Xways on a scale having a maximum of 10 Xways.

This benchmark subsequently inspired hundreds of efforts to achieve greater performance with a single-cpu/single-core system. The challenge of reaching 10 Xways has been a motivation for many novel approaches, including the Streamonas system described in this paper. Ultimately Streamonas [23, 24, 25] succeeded in achieving 10 Xways.

Recently interest has shifted from single-cpu systems to cluster systems and cloud computing. The first cluster DSMS to report results on the benchmark was the SPC system [16]. Subsequently a performance breakthrough was achieved by the SCSQ system, with an impressive 512 Xways.

By analyzing the benchmarking results we have found that cluster DSMSs have not yet achieved high levels of efficiency – to our knowledge the maximum reported efficiency is 1.33 Xways/core. Efficiency is an issue of increasing importance globally, with strong interest from Multinational Corporations, Governments and Regulatory Organizations [15].

In this work we introduce Distributed Streamonas. As a result of its novel architecture, it has been able to reach a level of 550 Xways. At the same time the system achieves high efficiency, supporting 13.75 Xways per core, a tenfold improvement on results reported by cluster DSMSs.

Migrating from a single-cpu/single-core system to a distributed architecture entails specific challenges and requires a specialized distributed design, as we present in section 4.1. In section 4.2 we present a novel *Multicore Fragmentation Interface for the Streaming Layer,* and in section 4.4 the *Hybrid Message Passing/Shared-Memory Interface between Streaming and Querying Layers* of Distributed Streamonas.

The Hybrid Message Passing/Shared-Memory Interface enables parallel multicore query processing. Two novel node architectures are provided in Distributed Streamonas: the *1-1 Fragment-to-Core Architecture* (section 4.6.1), appropriate for real-time applications with heavy computational requirements, and the *M-N Fragment-to-Core Architecture* (section 4.6.2) maximizing efficiency for more generic applications.

In section 5 we present an outline of the novel characteristics of the Streamonas Distributed Architecture, highlighting both its synchronous and asynchronous subsystems. Within the same section we describe the suitability of the distributed system for the framework of "smart city" infrastructures.

In order to better illustrate the novel characteristics of the distributed architecture, results from our experiments while running the Linear Road Benchmark application with a load of 550 Xways are used throughout the paper. Experimental results are presented in section 6.

2. Related Work

A large number of recently published works on Data Stream Management Systems have expanded the already extensive bibliography on spatio-temporal databases. Much of this work emphasizes data flow processing. For example, Aurora [1] tuples flow through a loop-free, directed graph of processing operations. Medusa [6] is a distributed version of Aurora which is intended to be used by multiple enterprises that operate in different administrate domains. Borealis [2] is a distributed system that inherits core stream processing functionality from Aurora and distribution functionality from Medusa. Earlier architectures also adopted data-flow methods. For example, the paradigm of Transducers was used throughout the Tangram [27] stream processor, which provided a database flow computation capability.

TelegraphCQ [9] is a recent adaptive data-flow architecture, supporting a wide variety of intensive networked applications. CACQ [22] implements continuously adaptive, continuous queries. The Stream Processing Core [16], also a data-flow system, enables the execution of multiple stream processing applications simultaneously on a large cluster of machines. Each application is expressed in terms of a data-flow graph consisting of processing elements (PEs) that consume and produce streams of data. The work was the first to analyze the implementation of the Linear Road Benchmark [3, 20] on a cluster (85 nodes/170 cores).

The STREAM project [4] DSMS prototype supports a large class of declarative continuous queries over continuous and traditionally stored data sets. Continuous Query Language (CQL) [5] reuses the formal foundations and large body of implementation techniques for relation-to-relation languages such as relational algebra and SQL by adding a small set of stream-to-relation and relation-to-stream operators. The Stream Mill system [29], through its Expressive Stream Language (ESL), defines user-defined aggregates (UDAs) on logical and physical windows and optimizes their computation. ESL treats data streams as unbounded ordered sequences of tuples. GSQL in Gigascope [10] is a pure stream query language with SQL-like syntax. All inputs in GSQL are streams, and the output is a data stream. Out-of-order processing [21] in NiagaraST and Gigascope provides an architecture that avoids ordering constraints by using explicit stream progress indicators.

In more recent work, cluster-based Data Stream Management System architectures [11, 17, 34] apply Map-Reduce [12] and Map-Reduce-Merge [32] programming models in data stream processing. Map-Reduce [12] is a programming model that enables highly-scalable (many terabytes of data on thousands of machines) and parallel data processing on large clusters of commodity machines. Map-Reduce specifies a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. While Map-Reduce is applied on large datasets in hundreds of special-purpose applications, it does not address the challenges of a streaming environment, especially real-time query latency.

The work Map-Reduce-Merge [32] adds a Merge phase to Map-Reduce that can efficiently merge data already partitioned and sorted by map and reduce modules. The work demonstrates that Map-Reduce-Merge can express relational algebra operators especially supporting joins of heterogeneous datasets. This can be a significant extension of the Map-Reduce programming model, as joins are fundamental for processing data across application, company, or even industry boundaries. Joining databases can permit data miners to extract more comprehensive rules than they could individually.

System design has profound effects on DSMS performance; this is clear in published results on the LRB (Linear Road Benchmark). Streamonas [23, 24, 25] was the first system to run the maximum level of the LRB (i.e. 10 Xways) on a single cpu having a query latency of microseconds, for example. This level of performance stems both from the data model and parallel architecture.

The work SCSQ-plr [33] subsequently exploited application-specific stream splitting (splitstream) functions where the user specifies non-procedural stream partitioning and replication. Performance on a cluster of 6 nodes (48 cores) reached a breakthrough-level of 64 Xways on the Linear Road Benchmark. The work [34] then followed a split, map, merge paradigm (splitstream, mapstream, mergestream) in which an incoming stream is split into parallel sub-streams, with expensive query operators applied on these sub-streams in parallel. A tuple can be sent to one specific DSMS node (partitioning) or to many DSMS nodes (replication). The first level of the parasplit architecture performs a random routing through the window router while at the second level an application-specific stream splitting functions distribute tuples to a number of continuous query operators. Each continuous query operator merges inputs and performs query logic. Answers from queries are then merged. By using a cluster of 70 compute nodes and 560 cores, this design achieved an order of magnitude higher stream processing rate over prior published results with an impressive L-rating of 512.

Other adaptations of Map-Reduce have been proposed recently. CaaaS [11] allows a long-standing SQL query instance to run cycle by cycle, each cycle processes a chunk of data from an infinite data stream. The project scales-out analytics computation by implementing Map-Reduce during each cycle. The system succeeded in running 1 Xway of the LRB on a machine with 2 CPUs. The Project DEDUCE [17] combines the benefits of Map-Reduce and stream processing by providing periodic analysis of large amounts of stored data with Map-Reduce to generate a model that is then used to process a stream of incident updates with a stream processing system. The project did not report any LRB results.

By contrast with this work, Distributed Streamonas presented here does not have a data-flow architecture or use Map-Reduce. For query processing it uses a new object-oriented spatio-temporal data model that is implemented with stream processing. As we explain next, it is this architecture that allows it to achieve 550 Xways and improved efficiency.

3. Outline of the Single-CPU/Single-Core Streamonas Architecture

In this section we describe the single-cpu/single-core Streamonas architecture published earlier in [23, 24, 25]. This gives the general outline for the Distributed Streamonas Architecture presented in the next section.

The Streamonas Architecture is based on three principles, which are served both by its architecture and by its unique parallel, object-oriented ST-Cuboids data model.[1] These principles are to support: (i) the decoupling of streaming from querying, (ii) the random access of any temporal sequence stored in the system and (iii) the random access of any data element in the ST-Cuboid database. Thus the Streamonas architecture follows a fundamentally different philosophy from other DSMSs, and particularly from data-flow-centric architectures. Because so many state-of-the-art DSMSs have a data-flow centric architecture, throughout this paper we will contrast them with Streamonas in order to better characterize its unique features and novelty.

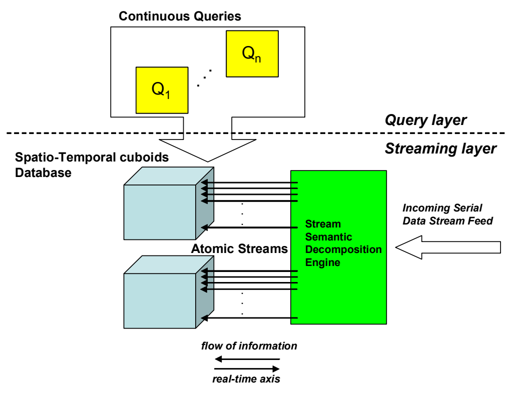


Figure 4-1. Single-cpu/Single-core Streamonas Architecture

[1] All the concepts of spatio-temporal cuboids used also throughout this paper, are defined along with further examples and diagrams in [22, 23], and can be accessed through the IEEE Xplore digital library at http://ieeexplore.ieee.org/Xplore/home.jsp or through the project's website at http://www.Streamonas.org

Decoupling of Streaming from Querying

As shown in figure 4-1, the Streamonas architecture uses two layers: the Streaming Layer and the Query Layer. By structuring the system in this way, we facilitate the uninterrupted flow of the data streams within the system. At the same time processed information is <u>not</u> query specific, allowing the system to provide data independence.

By contrast, data-flow centric systems apply query logic in-series with the streaming process, as tuples flow within a graph of query operators. Decoupling streaming from querying removes constraints on the flow of the data by the complexity of the query logic, which can cause bottlenecks and overflows when applied in-series.

ST-Cuboid data model

Streamonas does not use a serial data model as it can avoid accessing the incoming serial stream directly. Instead Streamonas uses its own, parallel, object-oriented ST-Cuboid data model after transforming the serial incoming stream. An ST-Cuboid abstracts the data arriving from a serial stream into parallel temporal sequences associated with evolving entities. These object-oriented, temporal sequences are named Atomic Streams.

As its name suggests, an ST-Cuboid represents a multi-dimensional slice of data in a spatio-temporal space. Slices are increasingly common substream-like abstractions in data analysis systems. In a nutshell, Streamonas allows flexible construction and query of slice objects, and takes maximal advantage of stream processing in processing queries.

The combined object-oriented data abstraction and change of data representation is a key difference of Streamonas from other DSMS. Adopting a traditional stream data model with minimal abstraction can gain performance advantages in some situations. However, it also loses data independence and limits query optimization. Similarly, requiring serial data access and data-flow query processing of the serial stream directly with a graph of query operators or an SQL expression imposes implementation constraints. Adopting an object-oriented spatio-temporal data model between the query and stream layers permits new kinds of performance improvement.

Some DSMS implementors may see this interposition of data abstraction as “unfair” for performance comparisons, since it changes the nature of the stream computation. It is clear though (as we have used a testbed of standard hardware and software), that similar approaches hold potential for future DSMS designs. This change is also clearly in keeping with DBMS traditions of data independence and query optimization; the history of DBMS is one of data abstraction.

In any event, the change to ST-Cuboids is a natural one, and Streamonas-SQL uses them instead of serial streams:

```
SELECT *
FROM ST-Cuboid
WHERE
{predicates}                                (Expr. 1)
```

Because of the principles mentioned above, the ST-Cuboid abstraction allows (i) random access of any temporal sequence in the database (ii) random access of any data element in the database – in each temporal sequence – providing significant real-time characteristics to the system. A basic reason that random access of any data element is not easy in data-flow architectures is that, as data is forwarded within a graph of query operators it is not easy to maintain a unique identifier for each data element that is constant over time.

The fundamental principles of Streamonas are novel for data-stream systems and require an innovative, technical design, illustrated in Figure 4-2. ST-Cuboids can model entities as evolving by storing their respective temporal sequences. An entity's attribute, instead of having a flat value of a standard type, has a temporal sequence of type atomic stream.

The elements within each temporal sequence are indexed and can be randomly accessed based on their temporal order (time hashing).

For an example, let us assume a *CAR* ST-Cuboid with schema *CAR* (<u>*Car_ID*</u>: int, *SPEED*: *AST*(6)), with *Car_ID* being the unique identification of each car entity, and *SPEED* being an attribute of type atomic stream and historical span six. The space hash table (Figure 4-2) indexes the identifications, enabling random access of each car entity (architectural principle (i)). For example, in order to access a car entity with *Car_ID*=100 in the database we can randomly access the temporal sequence of its speed values through the expression *CAR*[100].*SPEED*(*).

At the same time we can access any data element within the temporal sequence of the *SPEED* atomic stream through a time hash table (architectural principle (ii)). For example we can access the latest speed data of the car with *Car_ID*=100 through the expression *CAR*[100].*SPEED*(0), or perform a historical reference, e.g. access the sixth latest speed data element through the expression *CAR*[100].*SPEED*(-5).

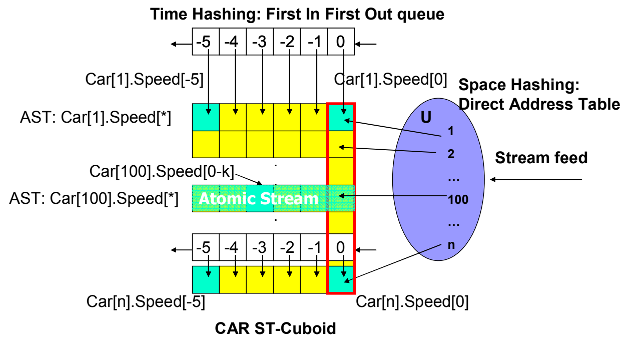


Figure 4-2. An ST-Cuboid example diagram

These object-oriented expressions can randomly access the whole volume of information within the ST-Cuboid, with excellent response times on the order of μsec. These rapid response times provide the real-time performance characteristics of Streamonas.

Stream Semantic Decomposition

Stream Semantic Decomposition (Figure 4-1) is a unique framework and sub-system of Streamonas. Its role is to decompose an incoming serial stream into temporal sequences to be stored in the atomic streams of the ST-Cuboid database.

Stream Semantic Decomposition reads the serial incoming stream of the ST-Cuboid and routes its data (based on their unique key and the hash table) to the respective entities/atomic streams in the order to be stored (Figure 4-2). The process is a transformation of the serial data model of a stream into the parallel data model of the ST-Cuboid object. During the transformation no data are dropped.

Stream Semantic Decomposition is a permanent sub-system of the DSMS, inaccessible to queries, having the fundamental role of decomposing an incoming serial stream into respective atomic streams. The Stream Semantic Decomposition engine maintains the ST-Cuboids database consistent in real-time guided by the Normalized Schema of the Database of ST-Cuboids. The decomposition of the incoming data stream into atomic streams is performed through an append operation of each tuple from the serial incoming stream on each respective atomic stream. Stream Semantic Decomposition (StSD) as published in [23, 24] is different from partitioned sliding windows [5], as it is not query specific and it is not defined within a query. Because StSD supports a database model, it can support different applications that are using the same data.

Thus StSD is not application-specific in the same way as the stream partitioning/splitting subsequently introduced in [33]. Stream Semantic Decomposition is transparent to query expressions such as (Expr. 1), and it supports the DSMS in offering data independence in a streaming environment as we discuss next.

Data Independence and Re-usability of information

Spatio-Temporal Cuboids have a role for the DSMS as fundamental as the role of relations in a Relational DBMS. Spatio-Temporal Cuboids offer *data independence,* enabling also the *re-usability of information*, properties which can be challenging to achieve in a streaming environment.

Database Consistency in Real-Time (Real-Time Consistency)

Streamonas updates the Spatio-Temporal Cuboids database upon arrival of each new tuple in the system, enforcing database consistency in real-time (Real-Time Consistency), as extensively presented in Chapter Five.

No reliance on Windows

CQL [5] incorporated Windows in its semantically rich model, extensively analyzed in [19]. Windows slide over a serial stream. As we have mentioned, Streamonas does not access the serial stream directly. Rather, its ST-Cuboid data abstraction permits parallel access to data through object-oriented queries, and consequently the system does not explicitly rely on windows.

Maintenance of temporal Order

As explained below, Streamonas consistently maintains the temporal order of data elements streamed into the system, and these are stored in respective temporal sequences (atomic streams). Complete information on the semantic space-time model used by the system can be found in [24].

4. Distributed Streamonas Architecture

4.1. Challenges of scaling-up the single-cpu / single-core Streamonas architecture

A single-cpu system has limitations in both memory and cpu capacities. Cluster systems offer a scalable platform that can host semantic spaces (a number of evolving entities) of very large cardinalities, and longer historical spans, thus supporting enterprise-level applications. Despite these benefits, scaling up an ST-Cuboid for a cluster system introduces specific challenges, which require special design. These challenges are:

(i) The use of a Distributed Memory model with Message Passing vs. the Shared Memory Model used in the single-cpu architecture.

Despite the fact that the memory in a cluster system can scale up with the number of nodes, the single-cpu system has the advantage of having a shared memory model. This allows direct access to objects in memory of the centrally designed database through object-oriented expressions.

We need to take into consideration that the development of distributed schemes for memory access through message passing can have a significant impact on query latency. Accessing different distributed memory segments through messages is a much slower process, posing a significant challenge to the real-time performance of the system. Despite these challenges, in this work our experimental results show an average query latency on the order of a millisecond (0.37 x 10^{-3} sec), over 13,000 times faster than the constraint set by the benchmark, which is 5 seconds.

(ii) Enterprise level information is streamed into the cluster system at much higher data rates and volume than the information streamed into the single-cpu architecture.

In order for the spatio-temporal cuboid database to handle streams of higher bandwidth and volume, special consideration must be given to the stability of the system. The scaled-up object-oriented structures of the spatio-temporal cuboid database are required to process the streaming information in a stable and efficient manner.

As an illustration, in our experiments for the LRB, Distributed Streamonas implemented a 65 GByte spatio-temporal cuboid database in distributed memory, and streamed 6.6 billion tuples (or equivalently 357.5 GBytes of information) into the system within 2hrs and 30 minutes. The simulation captured a large percentage of the interconnection network's capacity of 1Gbit/sec with an average rate of 589MBits/sec.

(iii) Multicore efficiency

One of the major advantages of a cluster system is the availability of a large number of cores. Efficient usage of multiple cores requires parallel architectures and algorithms, taking into consideration problems such as off-chip memory contention, and making provisions for avoiding conflicts between transactions.

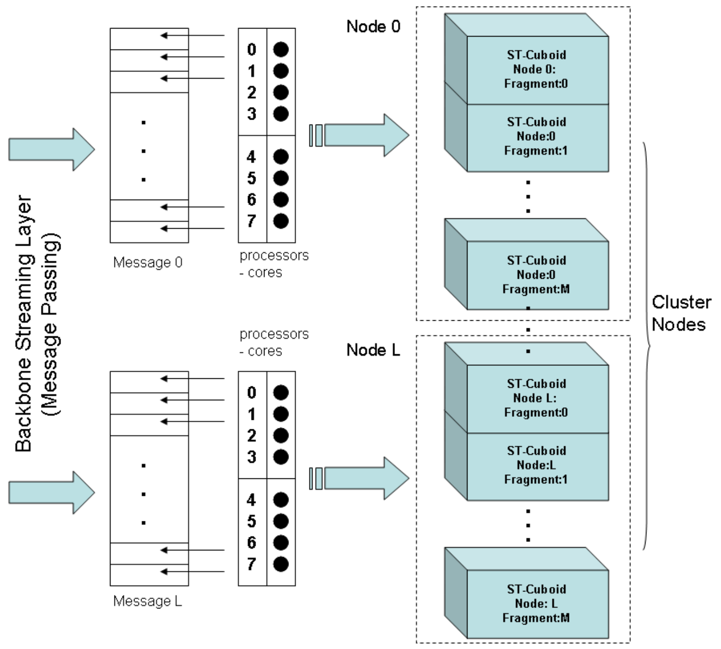

Backbone Streaming Layer (Message Passing)
Message 0
processors - cores
0 1 2 3 4 5 6 7
Node 0
ST-Cuboid Node 0: Fragment:0
ST-Cuboid Node:0 Fragment:1
ST-Cuboid Node:0 Fragment:M
Cluster Nodes
Node L
ST-Cuboid Node L: Fragment:0
ST-Cuboid Node:L Fragment:1
ST-Cuboid Node: L Fragment:M
Message L

Figure 4-3. Distributed Spatio-Temporal Cuboid Database

To illustrate we have used the very useful tabulation of performance results published in [34]. All reported systems in this tabulation had maximum efficiency (2.5 Xways/core) with cluster systems having a maximum efficiency of only 1.33 Xways/core. Distributed Streamonas – through parallel access by multiple cores of a single distributed spatio-temporal cuboid database object storing 55 database fragments – achieved the highest efficiency of all reported systems with 13.75 Xways/core.

4.2. Novel Multicore Fragmentation Interface for the Streaming Layer

The implementation in Streamonas of a distributed spatio-temporal cuboids database allows use of Fragmentation [26], which is a common technique for distribution of data across a network. Specifically, the parallel, object oriented data model of ST-Cuboids enables the definition of *a fragmented ST-Cuboids data model* guided by a normalized schema.

The fragmentation approach has differences with other excellent mature frameworks that directly access the serial incoming stream for query processing. As an example, in many data-flow centric architectures data fragmentation is query dependent (i.e. data follows different paths of a query graph depending on query semantics). Similarly, partitioned sliding windows [5] and stream partitioning are query specific. The work [34] refers to fragmentation as applied on routing and broadcast functions.

Fragmenting the distributed spatio-temporal cuboids database into mutually exclusive fragments is significant, as fragmentation enables the *locality of accesses* for both the appending of the streaming information and the read-only querying of the database. Locality of accesses presumes the application of data independence.

At the same time, the execution of a single query can be parallelized by dividing it into a set of sub-queries that operate on fragments. In this manner, the level of concurrency increases. By using locality of accesses and division of the same query into parallel sub-queries on the fragments, Distributed Streamonas enables significantly lower query latency (on the order of a millisecond for the LRB).

By adopting the fragmentation framework [26] in a DSMS, we need to take into consideration that sub-optimal use of the interconnection network, cpu processing power, and memory, due to the large-volume high-bandwidth data flows may cause much more severe problems than those experienced in relational DBMSs, leading to bottlenecks, overflows and in general to the instability of the DSMS. For this reason, in order to enable parallelism and avoid bottlenecks in the distributed streaming environment, Streamonas includes a novel *Multicore Fragmentation Interface for the Streaming Layer*, illustrated in Figure 4-3.

With this interface, after a sequence of tuples in a packet (message) is received by a node, each tuple in the sequence is read and associated with the ST-Cuboid fragment it belongs to (Figure 4-3). As there was a risk of a bottleneck (i.e., a single core of the node to read and decompose the whole volume of information for all fragments), we have parallelized the process by allowing each of the cores associated with a specific fragment to access the packet of information provided by the sending node with a hash table. This table allows parallel access to the sequences of tuples in predetermined locations within the packet array. The packet array establishes a 1-1 relationship of each packet segment (a specific sequence of tuples) to a respective spatio-temporal cuboid fragment. With this interface the original temporal sequence of tuples is maintained per fragment.

As we will analyze below, Stream Semantic Decomposition forwards each tuple through the network just once, and stores each tuple in the database just once, avoiding unnecessary propagation of data elements through the network and enabling *re-usability of the information* by multiple parallel queries. Furthermore, as query processing is not data-flow centric, intermediate results from queries are not in the form of streams. Through the ST-Cuboid indexing scheme, direct access of any element in the database is feasible, eliminating access of unnecessary elements. The overhead of accessing unnecessary data elements has been analyzed in [23]. As Distributed Streamonas computes intermediate results with parallel sub-queries over ST-Cuboid fragments and forwards distributed results for processing to a central node, it reduces the transmission of data elements to answer a specific query to only a few tuples, with great savings of bandwidth.

4.2.1. Example Fragmentation based on the Linear Road Benchmark

Figure 4-3 demonstrates the Distributed Design of the ST-Cuboids database on *L* nodes, each one serving *M* fragments. For the LRB application, our running example, an equivalent design was made for 5 nodes, each one hosting 11 fragments, serving 10 Xways each (i.e., 550 Xways).

In the following paragraphs we will describe the fragmentation methodology of the ST-Cuboids followed for the Linear Road Benchmark. The methodology is very similar to the one followed in relational DMBSs. This is another benefit of the ST-Cuboids data model that makes application development familiar, keeping transparent to the user the fact that data streams flow within the system at rates exceeding 700,000 tuples/second and response times are on the order of a millisecond.

For the ST-Cuboids fragmentation in the LRB, the global normalized conceptual schema of the spatio-temporal cuboids includes two fundamental ST-Cuboids storing evolution information of the entities in atomic streams as required by the benchmark, with *AST* denoting the type of Atomic Stream. The attributes serve the requirements of the benchmark as analyzed in [23, 24, 25]:

Global Normalized Conceptual Schema :
CAR (Car_Id:int, Speed(t):*AST*, Segment(t):*AST*)
XWAY (Xway:int, Segment:int, Direction:int, Lane:int,
AVGS(t):*AST*, *LAV*(t):*AST*)

The size and number of fragments to be allocated on each node were determined based on the semantics of the benchmark. Because the maximum number of Xways the benchmark supports is 10, we designed each fragment to host 10 Xways, as this would maintain the semantics of the application. The fragmentation logic also results in a design that distributes expected load over the fragments uniformly. The multiple fragments hosted on each node are served by multiple cores.

The number of fragments to be hosted by each node was determined by using the memory capacity of the node as a constraint [26]. More formally, using the LRB as an example, having a set N = {H_1,…,H_5} of 5 nodes available in the cluster, each one having 16GB or 14GB of RAM available, we were able to develop a set F {F_1,…,F_{11}} of 11 fragments in each node, each one capable of hosting 10Xways, satisfying the memory capacity constraint at node H_k, $\forall H_k \in H$:

$$\sum_{\forall F_j \in F} STCuboid\_MemSize_{jk} \leq Node\_Memory\_Capacity$$

We have applied primary horizontal fragmentation over both *XWAY* and *CAR* spatio-temporal cuboids by applying selection predicates over their primary keys. The *XWAY* spatio-temporal cuboid was fragmented in 55 fragments, allocated in 5 nodes, by storing 11 fragments per node. The decomposition of the spatio-temporal cuboid *XWAY* into the horizontal fragments $XWAY_i$, is defined by the respective selection operations as follows:

$XWAY_i = \sigma$ (Xway $\geq h$*FS*10 + i*10 AND

Xway $\leq h$*FS*10 + (i+1)*10 − 1)

with h = 0,…, 4 being the node identification , i=0, …, FS-1 being the fragment identification within each node, and FS being the maximum number of fragments a node can support.

As an example, node h=4 serves data for a maximum of FS=11 fragments. The ST-Cuboid of the node's fragment with identification i=10 stores data for expressways with unique identification Xway within the range [540,549].

In a similar manner with the assumption that cars in each 10Xways within the 3 hr simulation remain into the same system of expressways as generated by the benchmark (global queries process to a central node subquery results from each fragment), and knowing that each fragment of 10 Xways serves at most 1.4 million cars (actual number 1,373,327 cars), horizontal fragmentation has been applied on the *CAR* ST-Cuboid, resulting in the following fragments as follows:

$$CAR_i = \sigma\ (\ \text{Car\_Id} \geq h*\text{FS}*1.4*10^6 + i*1.4*10^6 \text{ AND}$$
$$\text{Car\_Id} \ \leq\ h*\text{FS}*1.4*10^6 \ + (i+1)*1.4*10^6 - 1)$$

with h = 0,…, 4 being the node identification , i=0, …, *FS-1* being the fragment identification within each node, and *FS* being the maximum number of fragments a node can support.

As an example, the ST-Cuboid of fragment i=10 of node h=4, stores data of cars with keys in the range [75600000,76999999]. Based on this theoretical framework, in our LRB experiments we implemented a distributed ST-Cuboid database of 65 GBytes in the distributed memory of five nodes of the cluster. The capacity of the database was able to host data for 550 Xways and 77 million cars (approximately 75.5 million active cars). Each node in the cluster has 8 cores available for the processing needs of the fragments it hosts.

Within each node, different multicore architectures were considered, and their performance was tested as presented in the sections below. Defining as $Q=\{Q_1,\ldots,Q_n\}$ the set of continuous queries running in the system, Distributed Streamonas should also satisfy the processing constraint:

$$\sum_{\forall q_j \in Q} \text{(processing load of } q_j \text{ at node } H_k) \leq \text{ (processing capacity of node } H_k\text{)}, \quad \forall H_k \in H$$

– linked to the query latency hard real-time constraint:

execution of $q_i \leq$ max query latency allowed for q_i, $\forall q_i \in Q$.

4.3. Derivation of Stability Condition for Distributed Streamonas

While the memory capacity constraint and processing constraint expressions presented in the previous section provide a solid background to our research based on fragmentation theory, especially for the data stream domain, there is one more constraint that needs to be met, which provides both stability and Quality of Service:

$$\lambda_{in}=\lambda_{out}$$

Here λ_{in} is the data flow arrival rate, requiring service, and λ_{out} is the total data flow rate served by the system.

For the Distributed Streamonas Architecture, where the number of fragments is kept the same for each node, the constraint $\lambda_{in}=\lambda_{out}$ can be expressed as:

$$\lambda_{in} = \sum_{1}^{h=nodes} \lambda_{h,messagepassing} = \sum_{1}^{h=nodes} \sum_{1}^{f=fragments_h} \lambda in_{hf} = \sum_{1}^{h=nodes} \sum_{1}^{f=fragments_h} \lambda p_{hf} =$$

$$= \sum_{1}^{j=GlobalSemanticSpace} \lambda_{ASTj} = \lambda_{out}$$

(Expr. 2 : Distributed Streamonas Stability Constraint)

Here λ_{in} is the total incoming flow rate arriving at the system; *nodes* is the number of nodes hosting the database; *fragments$_h$* is the number of fragments per node managed by the system; $\lambda_{h,messagepassing}$ is the data flow rate transmitted to each one of the nodes through message passing; λin_{hf}, is the data flow rate arriving with append data or query requests to each one of the fragments of the system; λp_{hf} is the data flow rate processed in relation to each one of the fragments and λ_{ASTj} is the data flow rate flowing within each of the atomic streams in the ST-Cuboid database in a FIFO manner. The number *Global Semantic Space* denotes the total number of atomic streams managed by the database.

The formula for λ_{in} above provides a stability constraint having three levels associated with the architecture of the system. At a first level, the incoming flow should equal the sum of data flows transmitted by the interconnection network via message passing; the second level expresses that the previous two expressions should also be equal to the sum of data flow arrival rates at each fragment of a node, for all nodes; and finally at a third level, as the processing rate of each fragment at a node should match the data flow rate arriving at each fragment, the sum of all processing rates of fragments at each node should provide an outflow equal to the inflow.

The last term denotes the fact that eventually information is streamed outside the system based on the FIFO data structure of the atomic streams of the ST-Cuboids data model.

As we will explain in section 4.6, there can be either a 1-1 allocation of processing cores to fragments, or processing can be performed in an *M-N* manner, where *M* fragments are served by a pool of *N* processing cores. In both architectures the stability constraint is maintained throughout the system.

4.4. Novel Hybrid Message Passing/Shared-Memory Interface between the Streaming and Querying Layers

As in Streamonas, querying is decoupled from streaming, the main task of the streaming layer is to transfer data to the fragments of the distributed database of ST-Cuboids in order to be appended. The streaming layer is the backbone infrastructure of the Distributed DSMS. Its stable operation is critical for the system's overall stability, having also an impact on query latency.

In the distributed design of Streamonas, streaming is performed in a synchronous manner in order to maintain database consistency in real-time, while querying is performed in an asynchronous parallel manner over a shared memory model.

In order to implement the model we have designed a novel *Hybrid Message Passing/Shared-Memory Interface between the Streaming and Querying layers* of the system. Based on the interface, MPI packets form a data stream backbone over the interconnection network operated through MPI processes following a message-passing framework. The multiple cores run a multi-threaded environment, allowing flexible parallel access of shared memory. Interfacing and synchronization between the message passing and the multi-threaded environment is performed through a system of locks. The architecture decouples the Streaming Layer from the Query Layer, allowing parallel query processing.

While the two layers can be totally decoupled in an asynchronous manner, in order to enforce database consistency in real-time we have synchronized the two layers. The synchronization performed establishes a hybrid Message Passing–Shared Memory model that preserves independence between the data stream channel of the backbone network and data accesses of the query layer.

Specifically, once a batch of data reaches the node through MPI, the locks are released by the backbone streaming layer enabling the parallel access of the batch by the cores of the query layer. Once parallel processing commits for all fragments, locks are acquired again by the streaming layer, which continues with the next batch of tuples.

As explained in the experimental section later (section 6), message passing in the streaming layer adapts to the pattern of latency at the query layer through the synchronization scheme, creating a control system for the stability of the system.

Latency at the streaming layer starts from 330 μsec for one core and converges to approximately 220 μsec for 4 to 7 cores (Figure 4-8 – Overall Latency). The horizontal line throughout the simulation in Figure 4-14 shows stable operation of the system.

4.4.1. HIERARCHICAL STREAM SEMANTIC DECOMPOSITION AS DATA STREAM FRAGMENTATION

In [23, 24] we presented fundamentals for the Streamonas process of Stream Semantic Decomposition (StSD). This process decomposes the incoming data stream into individual temporal sequences to be stored in respective atomic streams of the ST-Cuboid database. Hierarchical Stream Semantic Decomposition is a permanent sub-system of Distributed

Streamonas related to the data model of the DSMS (ST-Cuboids data model), and is inaccessible to queries (Figure 4-5).

The fragmentation scheme for the ST-Cuboids data model presented in section 4.2 would not be feasible in a streaming environment if it were not supported by hierarchical stream semantic decomposition in an integrated manner. Thus the StSD sub-system enables data independence, reusability of information, and locality of accesses in a distributed environment. These are established and well-understood properties in the relational DBMS literature.

Hierarchical Stream Semantic Decomposition is also guided by the Global Normalized Conceptual Schema and the Fragmentation and Allocation Scheme mentioned earlier, in harmony with ST-Cuboid fragmentation.

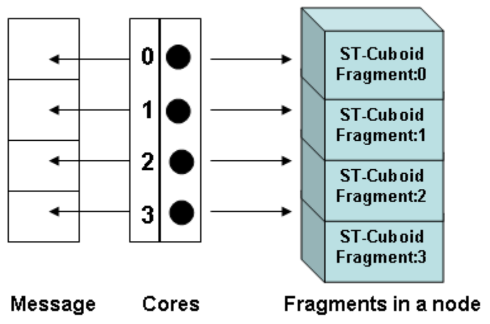


Figure 4-4. The 1-1 Fragment-to-Core node Architecture

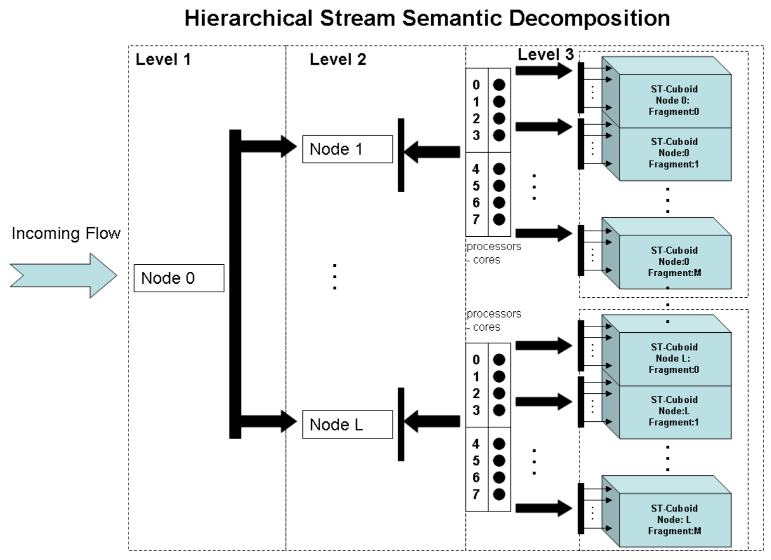

Hierarchical Stream Semantic Decomposition
Level 1
Level 2
Level 3
Incoming Flow
Node 0
Node 1
Node L
0 1 2 3 4 5 6 7
processors - cores
ST-Cuboid Node 0: Fragment:0
ST-Cuboid Node:0 Fragment:1
ST-Cuboid Node:0 Fragment:M
ST-Cuboid Node L: Fragment:0
ST-Cuboid Node:L Fragment:1
ST-Cuboid Node: L Fragment:M

Figure 4-5. Backbone Streaming Layer Architecture implementing Hierarchical Stream Semantic Decomposition

As mentioned earlier, Streamonas maintains the temporal order of data elements streamed into the system as atomic streams, and in this way maintains temporal semantics. Streamonas maintains the temporal order of arrival as it assumes synchronous and reliable communication of the streaming layer (such as tcp/ip) with the data source. The system timestamps each arriving tuple from the serial incoming stream based on the time of arrival of each tuple at the central process. This order is then maintained throughout the appending process in the respective atomic streams. The fact that Streamonas does not use a query graph eliminates clock skew and synchronization requirements.

4.5. Multicore Parallel Querying

Providing consistency in streaming environments (in real-time) is a challenge that has been considered by researchers as a tradeoff between performance and correctness [7]. However, it is common for queries to have hard real-time constraints, including having the "latest available" information "as soon as possible". By implementing a database of ST-Cuboids, single-cpu and Distributed Streamonas provide consistent database instances for parallel querying in a streaming environment.

In Distributed Streamonas, providing consistency in real-time through serialization and exclusive processing of each tuple (as applied in single-cpu/single-core Streamonas) would create a bottleneck in the parallel processing capabilities of the cluster. For this reason consistency in real-time (extensively presented in Chapter Five) is enforced in batches of data, and parallelism is applied at fragment level through architectures analyzed in the following section. In this way, append transactions are not conflicting since either (i) tuples in a batch belong to different fragments, or (ii) tuples of the same fragment are serialized by the streaming layer (based on their temporal order maintained within the same batch buffer, as explained in previous sections).

4.6. Multicore Architectures for ST-Cuboid Database Access and Processing

For parallel access of the database of ST-Cuboids by multiple cores, two multicore architectures have been developed. The *1-1 ST-Cuboid Fragment-to-Core architecture* assigns a dedicated core to each fragment, and the *M-N ST-Cuboid Fragment-to-Core architecture* assigns a pool of N cores to a number of M fragments. The 1-1 ST-Cuboid Fragment-to-Core architecture is used in cases where applications demand high processing capacity and very low query latency, while the resource-sharing M-N ST-Cuboid Fragment-to-Core architecture is appropriate for more generic applications, maximizing efficiency and creating economies of scale.

4.6.1. 1-1 Fragment-to-Core node Architecture

In the 1-1 ST-Cuboid Fragment-to-Core node architecture, shown in Figure 4-4, we assign one core per ST-Cuboid fragment with each node that hosts a number of fragments of the ST-Cuboid database based on the fragmentation scheme. Each core is dedicated to serve its fragment providing adequate processing power to serve the real-time requirements.

Figure 4-6 plots the performance characteristics of the 1-1 Fragment to Core Architecture for the cases 1 Fragment – 1 Core, …, 7 Fragments – 7 Cores. Off-chip memory contention has been observed in our experiments. Large-scale multicore systems are constrained by off-chip memory bandwidth as described in [18]. Off-chip memory contention has also been studied in [30], which emphasizes that memory contention is an important performance issue in multicore architectures, focusing on how off-chip memory contention affects the performance of parallel applications. It also mentions that contention for off-chip memory grows exponentially with the number of active cores. Specifically for Distributed Streamonas, by increasing the number of fragments in the 1-1 Fragment-to-Core architecture, we observed an increase in query latency from 27 to 69 μsec.

This degradation in behavior is attributed to off-chip memory contention of parallel programs in multicore systems. By correlating the behavior of our results with the contention results in [30], for the first 7 cores, we found a correlation of 0.94, which strongly supports our explanation of the result. We expect that newer hardware multicore architectures as described in [14] will also have a positive impact on the performance of Distributed Streamonas.

Figure 4-6 graphs the overall latency (average), which includes latency from both streaming and query processing. In all cases the graph of overall latency remains flat around the average value of 140 μsec, where delay from the streaming process synchronization dominates.

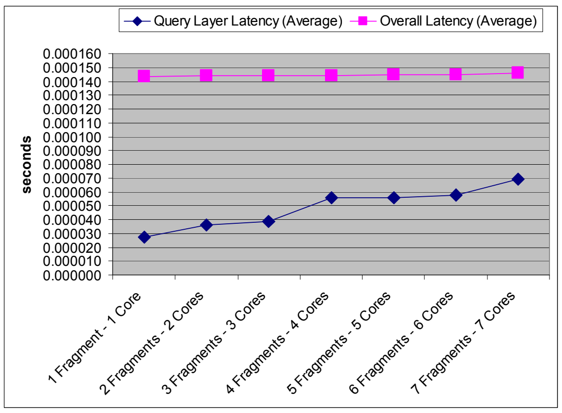


Fig. 4-6. Latency of a 1-1 Fragment-to-Core Architecture

The architecture implements also a second-level stream semantic decomposition role (Figure 4-5), where incoming data flow is decomposed into streams relevant only to the respective fragment. A naïve approach would be to assign a single core per node to the decomposition process.

Having in mind scalability requirements (and avoiding bottlenecks from assigning a single core for stream decomposition), we parallelized the process by allowing each core to access in parallel the MPI packet in the shared memory model. The packet is also fragmented, allowing each core to directly access data relevant to its fragment through a hash table that maps the memory space occupied by the MPI packet as described in section 4.2 and Figure 4-3.

4.6.2. M-N Fragment-to-Core node Architecture

The 1-1 Fragment architecture allows for the direct assignment of a single core to a single fragment, supplying adequate processing resources in a scalable manner within a specific node. Still, it has a serious limitation: the number of fragments cannot exceed the number of cores within the same node. This level of flexibility is required as RAM per node may be easily expanded, allowing the design of a spatio-temporal database with a larger number of fragments. As an example, it is natural for each fragment in the Linear Road Benchmark to host 10 Xways, the maximum number supported in the LRB design. Simultaneously, experiments on the single-cpu Streamonas architecture allow us to predict that only a single core will be more than enough to support a 10 Xway ST-Cuboid database fragment.

Thus, due to the object-oriented design of the database and the object-oriented logic of the queries, each core has sufficient processing capacity to serve more fragments.

For this reason, the *M*-*N* architecture creates a pool of *N* cores − pooling available capacity as a single processing source. In the architecture the *N* cores compete to serve *M* fragments (Figure 4-7). The number of cores *N* can be less or more than the number of fragments *M*. The case where *N* is greater than *M* (i.e., multiple cores can be assigned on a single fragment) enables intra-query parallelism, breaking up a single query into a number of subqueries [26]. Intra-query parallelism adds to the performance of the system, which already exploits inter-query parallelism performance through the ability of executing multiple queries at the same time, based on the fragmentation scheme [26].

For the case where the number of cores *N* is less than the number of fragments *M*, a locking scheme ensures that only a single core at each moment serves a single fragment. As the cores scan each one of the *M* fragments, the pool of cores virtually generates *M* different 1-1 processing modules. Once the scan is complete, through a second locking scheme, the pool of cores allows the backbone network to stream the next packet of tuples in a synchronous manner.

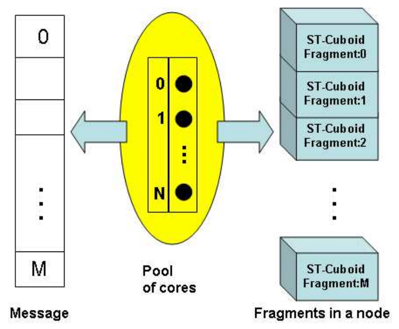


Figure 4-7. *M-N* Fragment-to-Core node Architecture

The performance of the *M-N* Architecture is presented in Figure 4-8. The diagram shows the overall latency and the query layer latency of incremental multicore deployment from 1 to 7 cores over a constant number of 11 fragments (in a topology of 3 nodes for this set of simulations). Based on the results of the simulations, we confirmed the efficiency of the object-oriented spatio-temporal cuboid design and the object-oriented query logic. A single core (a pool of a single core) is adequate to serve the requirements of 11 fragments storing 110 Xways of the benchmark, having a significantly low average query layer latency of 284 μsec, with the overall latency being 330 μsec.

By including additional cores in the pool, query layer latency can be further improved; the cores are ready to receive additional load both in terms of the number of queries but also in terms of query complexity (query processing time).

In our experiments (Figure 4-8), we have observed an upward step when adding the fourth core. We believe that the reason for this behavior is again the off-chip memory bandwidth constraints. As each node has two processors of 4 cores each, with the first core of the first processor dedicated to the streaming layer, the fourth core in the diagram is the first core of the second processor. The upward step appears when this second processor becomes active. Our explanation is that the activity of both processors on the same set of 11 ST-Cuboid database fragments creates a higher level of off-chip memory contention.

In order to experimentally support our explanation we again ran experiments for 1,2,3 and 4 cores, using only the cores of the second processor. The step upward was eliminated with latency results 289, 202, 131 and 119 µsec respectively, following a downward trend as expected. From the same results we can observe that both off-chip contention and the locking system reduce the marginal benefit of adding a new core. Our future research will elaborate on these first results through the deployment of a larger number of processors and cores per processor.

As mentioned earlier, the top line in figure 4-8 presents overall latency behavior while the bottom line presents query layer latency. As seen from the graph the synchronization of message passing at the streaming layer adapts to the pattern of latency at the query layer, creating a control system for the stability of the system. The overall latency of the system starts at approximately 330 µsec for one core and converges to approximately 220 µsec for 4 to 7 cores.

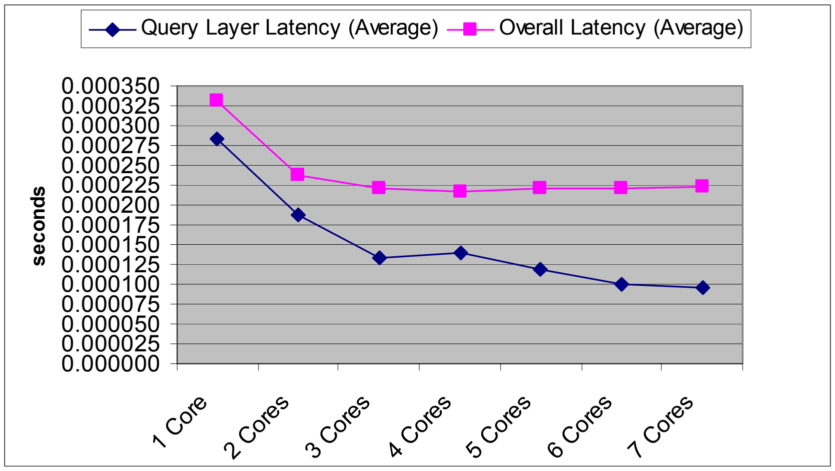
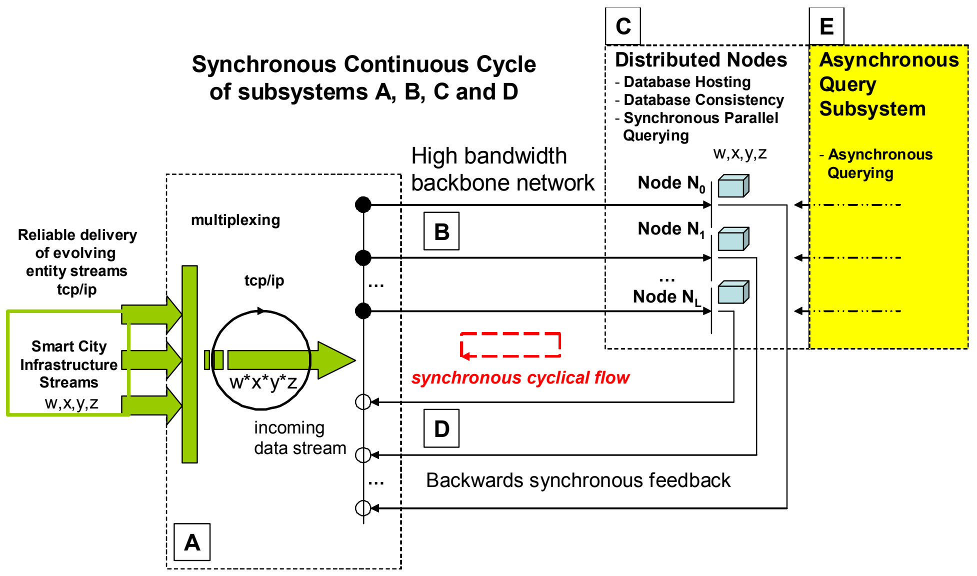


Figure 4-8. Latency of an *M-N* Fragment-to-Core Architecture with Incremental Core Deployment over 11 Fragments (*M*=11, *N*=1, …, 7)



Synchronous Continuous Cycle of subsystems A, B, C and D
C
Distributed Nodes
- Database Hosting
- Database Consistency
- Synchronous Parallel Querying
w,x,y,z
E
Asynchronous Query Subsystem
- Asynchronous Querying
High bandwidth backbone network
multiplexing
Reliable delivery of evolving entity streams tcp/ip
tcp/ip
Smart City Infrastructure Streams w,x,y,z
w*x*y*z
Node N_0
Node N_1
...
Node N_L
B
synchronous cyclical flow
incoming data stream
D
Backwards synchronous feedback
A

Figure 4-9. Synchronous Distributed Architecture extended with Asynchronous processing

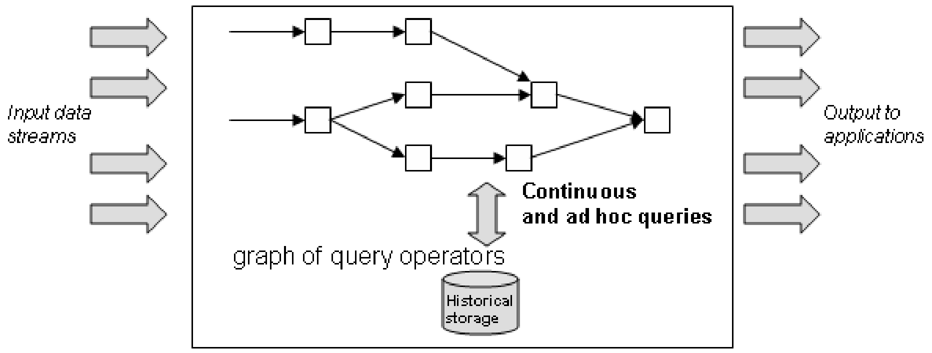


Figure 4-10. A reference sketch, for comparison purposes of a data-flow centric architecture based on [1]

5.1. Outline

In the following paragraphs we shall present an outline of the novel characteristics of the Streamonas Distributed Architecture, highlighting both its synchronous and asynchronous subsystems. A diagram of the architecture is presented in figure 4.9. For contrast, figure 4.10 sketches the mature, state-of-the-art, data-flow centric architecture of Aurora [1] in which data flows through a graph of query operators.

The large-scale application framework we have used for applying the Distributed Streamonas Architecture, is the framework of "smart city" infrastructures. Smart Cities have captured the imagination of many leaders as a path to restructuring the whole planet on the basis of knowledge. Multinational corporations, governments and academic institutions have offered their own perspectives of developing Smart Cities. This vision has been enabled by the ubiquitous presence of computing within an augmented physical environment, with sensors and mobile devices generating large volumes of streaming information in real-time, creating new dynamics. One of the core aspects of Smart Cities are Intelligent Infrastructures. Intelligent Transportation Systems, Smart Metering of Renewable Energy and Water and smart monitoring of assets such as homes and cars through mobile devices are only a few examples of the structure and magnitude of the expected economic change. The expected information is massive and continuous, and deviates from the traditional computational paradigms.

The dotted line in the center of figure 4.9 demonstrates the synchronous cyclical flow within our system. Synchronization is performed along the cycle of subsystems A, B, C and D while subsystem E enables, independently and in parallel, *asynchronous* query processing based on the underlying synchronous infrastructure. The cycle over subsystems A, B, C and D is analyzed in the following paragraph.

Subsystem A processes a high-bandwidth incoming stream of data elements which are sorted following a strict temporal order based on the time of data generation. In the example presented in figure 4.9, a single sensor sends consecutive readings w,x,y,z of the energy generated from a single home photovoltaic system (evolving entity) reliably through tcp/ip.

These readings are merged with data flows from additional individual evolving entities into the high bandwidth network, reliably (multiplexed in a FIFO manner) and thus respecting the temporal sequence of each entity. In our example, information in the high bandwidth network is expressed as w*x*y*z, with the symbol "*" denoting additional data elements due to the merging process.

- Subsystem A sends to distributed nodes relevant information based on a database fragmentation scheme as presented in section 4.2 of this Chapter. Received information is sent over the high bandwidth backbone network, through message passing - using an interface such as MPI.

- Subsystem B is responsible for the high-bandwidth, reliable, synchronous delivery of the data traffic to the nodes. As the data are forwarded into the network based on a database fragmentation scheme, the resulting data streams are also fragmented based on the same fragmentation scheme.

- Subsystem C refers to the distributed nodes. Each node is responsible for the hosting of a fragment (or a number of fragments) of the database and the enforcement of Database Consistency within each synchronous cycle based on the new batch of data elements received. Based on the unique identification of the entity the data described (e.g. unique identification of the home photovoltaic system), data are stored in their respective evolving entity in the database. In our example data elements w,x,y,z are forwarded to the entity object they describe in the database and they are stored in it, as a temporal sequence in its native form. Description of the Database Consistency framework ensuring accuracy (which we name real-time consistency) is presented in Chapter Five.

Distributed nodes are also responsible for query processing over their local fragments. They are also responsible for the evaluation of local sub-queries of a global query and the delivery of local results to a central location for final processing. In our first prototype system, the central location collecting all sub-queries was chosen to be the same as the node of subsystem A, as shown in figure 4.9.

- Subsystem D is responsible for the reliable delivery of the results of the sub-queries to the central location through the high-bandwidth interconnecting network (message passing). Subsystem D closes the loop defined by subsystems A, B, C and D, establishing synchronization. Subsystem A, based on sub-query results, evaluates global results and proceeds to the next cycle with the next batch of data.

- Subsystem E enables asynchronous querying over the synchronous system. In addition to the synchronous cycle over subsystems A, B, C and D, the independent subsystem E acts in a parallel manner, accessing the database asynchronously from the underlying synchronous system, potentially serving millions of external query requests per second.

While the asynchronous system cannot guarantee consistent results for complex query logic with interdependent intermediate results (in contrast to synchronous querying, a consistent database snapshot cannot be guaranteed for the whole duration of the evaluation of a complex asynchronous query), as we shall analyze in the following sections, asynchronous querying can be very useful in providing results for simple queries which request temporal information about single entities (e.g., current speed or current emissions of a vehicle).

5.2. Comparison and novel characteristics of the Distributed Streamonas Architecture

The Distributed Streamonas Architecture has the following novel characteristics:

(a) Real-Time Consistency: Enforcing Consistency in real-time in a streaming environment has been recognized by researchers as a key challenge, due to the high dataflow rates and real-time requirements. Within this context, advanced consistency trade-off frameworks such as load shedding and application-dependent approximate query processing have been developed by research works.

The Distributed Architecture outlined in the previous subsection enables the enforcement of database consistency which guarantees accurate results. As we extensively analyze in the experimental section of this Chapter (section 6) as also in Chapter Five, database consistency is enforced with low latency in real-time, and for this reason we refer to it as Real-Time Consistency.

A Synchronous Distributed System: In order to enforce database consistency in real-time, we have designed the system to work in a synchronous mode. By comparing our novel architecture of figure 4.9 to the state-of-the-art architecture of Aurora [1] which follows a data-flow centric philosophy (figure 4.10), we can see that the data-flow centric architecture is not synchronous and it does not provide any sort of feedback mechanism (or other synchronization mechanism) in order to close the loop (as subsystem D of Distributed Streamonas does in the loop of subsystems A,B,C and D).

(b) Synchronous Querying: Synchronous querying is performed within the synchronous mode of operation of the system. Query in synchronous mode guarantees accurate results and, at the same time, imposes the tight limits of synchronization mode, requiring efficient algorithms and underlying data structures.

(c) Asynchronous Querying over a Synchronous system:

The Synchronous system (subsystems A, B, C and D of figure 4.9) provides the necessary synchronization methods for the enforcement of database consistency, as also for consistency over application specific data having interdependencies. The Synchronous system also performs Synchronous Query processing.

It can be seen from the above that the synchronous system also keeps the database consistent at a single entity level. Consistency at this fine granularity enables consistent real-time results to be accessed by an Asynchronous subsystem (subsystem E – figure 4.9). Providing accurate results from data streams in an asynchronous manner (for limited scope queries), through the synergistic operation between synchronous and asynchronous subsystems, is both novel and provides an additional level of power and extensibility.

(d) Each tuple arriving to the system, is forwarded within the backbone network just once during a Streaming Process (Streaming Layer), and it is stored in the database just once, avoiding unnecessary propagation of data elements through the network, enabling re-usability of the information by multiple parallel queries (Query Layer). This "one-time-propagation" of data into the network limits network use and avoids bottlenecks or flooding due to generation of intermediate data from query operators and subsequent performance degradation. This novel characteristic of the Distributed Streamonas architecture is different from architectures that propagate data within a graph of query operators (figure 4.10).

6. Experimental Results based on the Linear Road Benchmark Application

For our experiments we used standard software and hardware on a Linux cluster. The Linux version was Red Hat Linux release 5.6 – CentOS. The respective Linux kernel was 2.6.18-238.19.1.e15. The cluster had 5 active nodes. Each one of the nodes had two quad-core processors providing 8 cores per node (Intel 3.1 GHz Xeon - X5460 with 6144 KB L2 cache). Each one of the nodes had 16 GBytes of RAM with one of them having 14 GBytes of RAM available. The cluster had an 1 Gbit/sec interconnecting network. The topology and configuration used for our experiments is presented in figure 4-11.

The cluster had shared disk storage which we used to store the historical data generated by the Linear Road Benchmark simulator, required by the type "d" historical query, as described in the benchmark's specifications [3, 20]. As the historical data were the same for each fragment of 10 Xways, we stored on disk a single copy to be accessed by all 55 fragments in the cluster. Access at thread level was serialized with the use of locks.

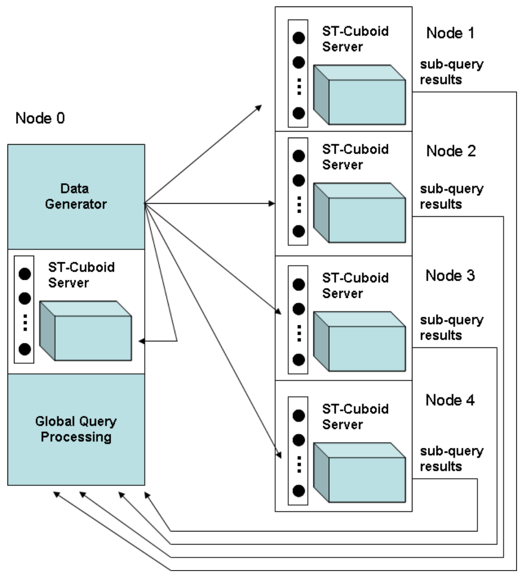


Figure 4-11. Topology of the Distributed Streamonas DSMS running the Linear Road Benchmark with a load of 550Xways

A database of Spatio-Temporal Cuboids managing 550 Xways was implemented and distributed on the 5 nodes of the cluster in a symmetric way. Each node hosted 11 fragments supporting data flows from 10 Xways each. The fragments on each node used 13 GBytes of memory each and the whole distributed ST-Cuboid database used 65 GBytes of distributed memory. The database was able to support 75.5 million evolving entities (cars).

The benchmark was designed to support up to 10 Xways, so we had to multiply the generation of data streams in order to feed the fragments of the spatio-temporal cuboid database distributed on the fragments of the cluster.

Node-0 served as the stream data generator (Figure 4-11). The generator read each single tuple from the original 10 Xways benchmark data-file directly from disk, and multiplied it internally for the number of fragments. Spatio-Temporal Cuboid database servers were hosted by all five available Nodes (including Node-0, data generator, as a separate process) in order to exploit all available resources in the cluster. Each one of the nodes had two processors of 4 cores each, i.e. 8 cores were available per node. Core with id=0 on each node (id=1 for Node-0), was assigned to the streaming layer server running on each Node. The rest of the cores for Nodes 0-4 were configured in an *M-N* Fragment-to-Core node architecture as presented in the previous sections (11 Fragments-5 Cores for Node-0, and 11 Fragments – 7 Cores for all other nodes). Node-0 also ran a Global Query Processing server on core with id 2, which continuously collected all results from distributed, parallel sub-queries to be processed centrally and gave the final result. The same process through a second set of MPI calls also received all output results from continuous queries running on the Nodes.

Assignment of cores to MPI ranks was performed through a respective rankfile, while threads were bound to cores with the *pthread_setaffinity_np* instruction. In our experiments we saw that having not done core binding to MPI processes, the generating and receiving process on Node-0 competed for resources, leading to starvation of one of the two, resulting in overflows or a system halt. The topology design and binding of resources resulted in a stable backbone of the streaming layer, adequately serving the needs of the expected load and database size. As mentioned earlier the linear road benchmark has a maximum number of 10 Xways. Thus in order to perform our experiments with the distributed Streamonas DSMS, we synthesized data from the initial file of 10 Xways into multiples of 10 Xways, generating adequate load to serve 550 Xways. Global identifications for both the *XWAY* and *CAR* entities were generated on the fly. Once received, the message was accessed by threads running on individual cores of the node.

The message was accessed in read-only mode in parallel by the threads. At this critical point a strict synchronization was maintained between the receiving process and the threads accessing received messages in parallel. Messages were kept in a buffer and locks are used for the synchronization between the process and the parallel threads.

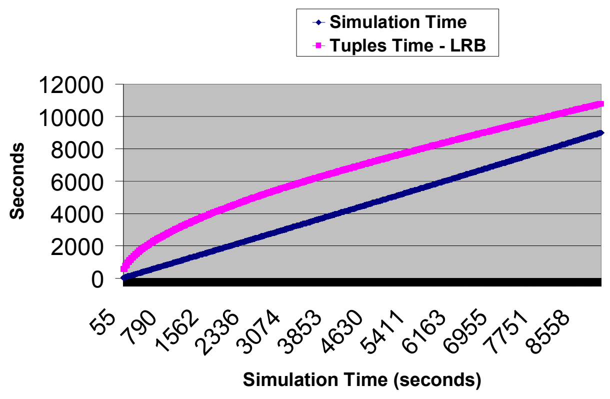


Figure 4-12. Simulation time comparison with tuples time

Extensive tests have been performed to demonstrate the ability of the system to (i) maintain a high data flow rate of information and (ii) enable through strict synchronization consistent database instances. Figure 4-14 presents a graph of the Overall Average latency of the system, showing an average of 0.000373 seconds (instances of maximum latency 0.399520 seconds). Overall latency was measured as the time needed for a batch of tuples to be processed until the next one was forwarded to the system for processing, i.e. as the period of a single processing cycle. The matching between MPI_Send and MPI_Recv instructions was used to synchronize the data flow rates at the streaming layer, while the system of locks synchronizes the streaming layer batch interarrival events with the query layer processing delay.

On the same graph of Figure 4-14, the average latency at the query layer is presented as 0.000097 seconds (maximum query layer latency 0.019466 seconds). The simulation completed within 8,891 seconds, which is faster than expected by the benchmark, having a 3 hr duration (10,800 seconds).

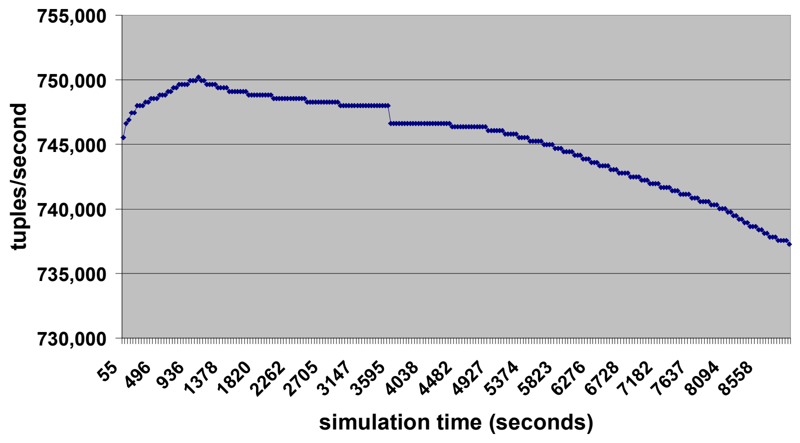


Figure 4-13. Average data flow rate of the Backbone Streaming Layer

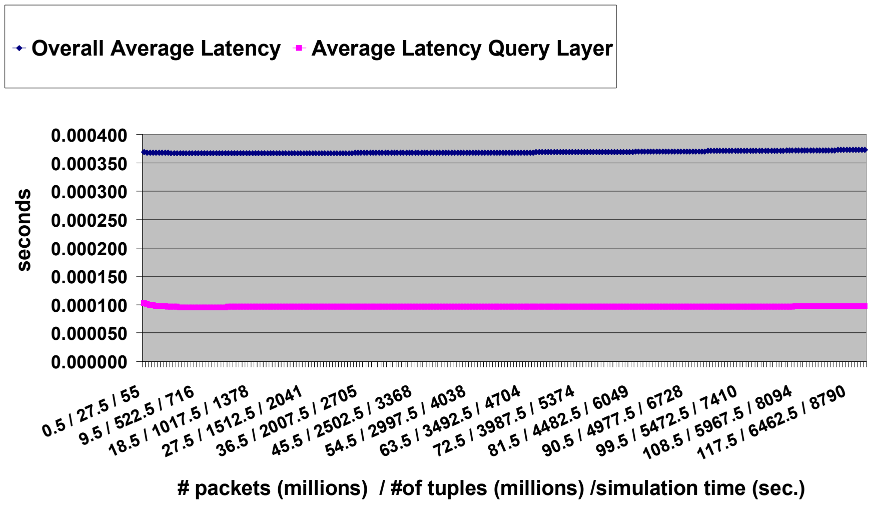


Figure 4-14. Overall Latency and Query Layer Latency

During the whole simulation, the time included in the tuples was greater than the actual simulation time, ensuring that the system would meet the Linear Road Benchmark requirements (Figure 4-12). The average dataflow rate at the backbone streaming layer was 737,275 tuples/sec (Figure 4-13). By the time the simulation was completed Distributed Streamonas had streamed/processed 6.6 billion tuples in 24 million packets – a volume of 357.5 GBytes of information.

7. Conclusion

In this work we have introduced the novel architecture of Distributed Streamonas. The architecture addresses the challenges of the large-scale distributed environment by successfully implementing a fragmented database model of Spatio-Temporal Cuboids. The fragmented data model is supported by the permanent sub-system of Hierarchical Stream Semantic Decomposition. The work introduces two novel interfaces for Distributed Streamonas: (i) the *Multicore Fragmentation Interface for the Streaming Layer* and (ii) the *Hybrid Message Passing/Shared-Memory Interface between the Streaming and Querying Layers*. It also presents two novel multicore architectures for parallel query processing: (a) the *1-1 ST-Cuboid Fragment-to-Core node Architecture* and (b) the *M-N ST-Cuboid Fragment-to-Core node Architecture*.

Distributed Streamonas managed to create a new record on the Linear Road Benchmark by running a load of 550 Xways, reaching also a new record in efficiency by supporting 13.75 Xways/core. In our experiments, Distributed Streamonas demonstrated robustness and suitability for large-scale, enterprise-level applications, as it managed a volume of 357.5 GBytes in less than 3 hours at an average backbone network data rate of 589 Mbits/sec, monitoring approximately 75 million cars and having an average query latency below 400 μsec.

8. REFERENCES

1. D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. "Aurora: a new model and architecture for data stream management," *The VLDB Journal*, 2003.

2. D J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. "The design of the Borealis stream processing engine," *in Proc. of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR 2005),* Asilomar, CA, 2005.

3. A. Arasu, M.Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbets. "Linear Road: A Stream Data Management Benchmark," *in Proc. of the 30th Intl. Conference on Very Large Data Bases*, 2004.

4. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein and J. Widom. "STREAM: The Stanford Stream Data Manager," *in Proc.of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, 2003.

5. A. Arasu, S. Babu and J. Widom. "CQL: A language for continuous queries over streams and relations," *in 9th Intl. Workshop on Database Programming Languages*, 2003.

6. M. Balazinska, H. Balakrishnan and M. Stonebraker. "Load Management and High Availability in the Medusa Distributed Stream Processing System," *in Proc. of the 2004 ACM SIGMOD Intl. Conference on Management of Data*.

7. R. Barga, J. Goldstein, M. Ali and M. Hong. "Consistent Streaming Through Time: A Vision for Event Stream Processing," *in Proc. of the third Biennial Conference on Innovative Data Systems Research,* 2007, pages 363-374.

8. I. Botan, P. M. Fischer, D. Florescu, D. Kossmann, T. Kraska, R. Tamosevicius. "Extending Xquery with Window Functions," *in Proc. of the 33rd Intl. Conference on Very Large Data Bases*, 2007.

9. S. Chandrasekaran, O. Cooper, A. Seshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurhty, S. Madden, V. Raman, F. Reiss and M. Shah. "TelegraphCQ: Continuous dataflow processing for an uncertain world," *in Proc. of the 2005 Conference on Innovative Data Systems Research.*

10. C. Cranor, T. Johnson, O. Spataschek, V. Shkapenyuk. "Gigascope: A Stream Database for Network Applications," *in Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, pages 647-651, 2003.

11. Q. Chen, M. Hsu and H. Zeller. "Experience in Continuous analytics as a Service (CaaaS)," *in Proc. EDBT 2011*, pages 509-514.

12. J. Dean, S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters," *in Proc. OSDI, pages 137-150*, 2004.

13. P. M. Fischer, K. S. Esmaili and R. J. Miller. "Stream schema: providing and exploiting static metadata for data stream processing," *in Proc. EDBT 2010*, pages 207-218.

14. N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki. "Reactive NUCA : near-optimal block placement and replication in distributed caches," *in Proc. of the 36th annual international symposium on Computer architecture*, pages 184-195, 2009.

15. (2013) IBM – Energy, the environment and IBM. [Online]. Available: http://www.ibm.com/ibm/green/data_center.html

16. N. Jain, L. Amini, H. Andrade, R. King, Y. park, P. Selo, C. Venkatramani. "Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing

Core," *in Proc. of the 2006 ACM SIGMOD Intl. Conference on Management of Data*, 2006.

17. V. Kumar, H. Andrade, B. Gedik, K.-L. Wu. "DEDUCE: At the Intersection of MapReduce and Stream Processing," *in Proc. EDBT 2010*, pages 657-662.

18. O. Khan, M. Lis, S. Devadas. "EM2: A Scalable Shared-Memory Multicore Architecture," *MIT Computer Science and Artificial Intelligence Laboratory Technical Report*. June 12, 2010. MIT-CSAIL-TR-2010-030.

19. Y. Law, H. Wang, C. Zaniolo. "Relational Languages and Data Models for Continuous Queries on Sequences and Data Streams," *ACM Transactions on Embedded Computing Systems*. Vol. 36, No. 3, Article 39, March 2011.

20. (2013) Linear Road – Home. [Online]. Available: http://www.cs.brandeis.edu/~linearroad/

21. J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos, T. Johnson, and D. Maier. "Out-of-Order Processing: A New Architecture for High-Performance Stream Systems," *in Proc. of the 34th VLDB Conference*, Auckland, New Zealand, 2008.

22. S. R. Madden, M. A. Shah, J. M. Hellerstein and V. Raman. "Continuously adaptive continuous queries over streams," *in Proc. of the 2002 ACM SIGMOD Intl. Conference on Management of Data*, 2003.

23. P. A. Michael, D. S. Parker. "Architectural Principles of the Streamonas Data Stream Management System and Performance Evaluation based on the Linear Road Benchmark," *in Proc. of the 2008 International Conference on Computer Science and Software Engineering (2008 CSSE)*, IEEE, Wuhan, China, 2008.

24. P. A. Michael, D. S. Parker. "The Semantic Space-time models of the Streamonas Data Stream Management System," *in Proc. of the 2009 World Congress on Computer Science and Information Engineering (2009 CSIE)*, IEEE, Los Angeles / Anaheim, USA, 2009.

25. P. A. Michael, D. S. Parker. "Real-Time spatio-temporal data mining with the Streamonas Data Stream Management System," *in Proc. of the Tenth International Conference on Data Mining, Detection, Protection and Security*, Wessex Institute of Technology, Crete, 2009.

26. M. T. Özsu, P. Valduriez. *Principles of Distributed Database Systems*. Springer, third edition, 2011.

27. D. S. Parker, R. R. Muntz, H. L. Chau. “The Tangram stream query processing system,” *in Proc. of the Fifth International Conference on Data Engineering*, 1989.

28. (2013) SCSQ-LR home page. [Online]. Available: http://www.it.uu.se/research/group/udbl/lr.html

29. H. Thakkar, B. Mozafari, and C. Zaniolo. “Designing an Inductive Data Stream Management System: the Stream Mill Experience,” *in Proc. of the Second International Workshop on Scalable Stream Processing Systems*, Nantes, France, 2008.

30. B. M. Tudor, Y. M. Teo, S. See. “Understanding Off-Chip Memory Contention of Parallel Programs in Multicore Systems,” *in Proc. of the 2011 International Conference on Parallel Processing,* IEEE, pages 602-611, 2011.

31. C. Zaniolo, S.Ceri, C.Faloutsos, R.T. Snodgrass, V. S. Subrahmanian and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann, 1997.

32. H. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker. “Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters,” *in Proc. 2007 ACM SIGMOD Intl. Conf. on Management of Data*, 2007.

33. E. Zeitler, T. Risch. “Scalable Splitting of Massive Data Streams,” *in Proc. DASFAA (2) 2010*, pages 184-198.

34. E. Zeitler and T. Risch, “Massive scale-out of expensive continuous queries,” *in Proc. VLDB Endow.* 4: 1181-1188, 2011.

CHAPTER FIVE

EMBARRASSINGLY PARALLEL QUERY PROCESSING WITH THE REAL-TIME CONSISTENCY FRAMEWORK OF THE DISTRIBUTED STREAMONAS DSMS

ABSTRACT

The challenging nature of the streaming environment has led researchers to develop advanced consistency trade-off frameworks such as load shedding and application-dependent approximate query processing. In this work we introduce the novel concept of enforcing consistency over a spatio-temporal generalization of a stream, and doing this in real time (i.e., providing Real-Time Consistency). This gets at basic consistency issues for Data Stream Management Systems (DSMSs) that have limited their scalability to large-scale, parallel systems.

Specifically, we study consistency over a ST-Cuboid, a parallel object-oriented data structure that implements slices of a multi-dimensional space. Accomplishing this permits development of new kinds of DSMS that have potential for significant performance improvements. We have implemented this kind of consistency framework for the ST-Cuboid database in the Streamonas DSMS.

In some cases, performance improvements can sharply reduce the need for consistency trade-offs. In fact, our implementation has been able to achieve 100% accurate results with excellent performance characteristics, without trade-offs, so that consistent database states are materialized within time intervals of less than 400 μsec. These results have been achieved for large distributed enterprise-like applications where the in-memory distributed database reaches 65 GBytes and throughput exceeds 730,000 tuples/sec.

We also introduce a novel asynchronous sub-system having a multicore architecture. The asynchronous sub-system re-uses the data stored in the ST-Cuboid database by the Linear Road Benchmark having a load of 550 Xways, in an embarrassingly parallel manner. The asynchronous sub-system can facilitate more than 45 million 'read' operations per second, performing a whole database scan of 75 million cars in 1.44 seconds, more than twice the number of registered cars in California in 2011.

1. INTRODUCTION

While Database Consistency has been a cornerstone for relational database systems, and has become synonymous with accuracy and semantic correctness, enforcing Consistency in real-time in a streaming environment has been recognized by researchers as a key challenge, due to the high dataflow rates and real-time requirements.

Within this context, researchers in [7] define consistency trade-offs, with weak, middle and strong consistency depending on application requirements. The pioneering work [1], having proven that relational DBMSs are not suitable for data stream applications (relational DBMSs apply database consistency over a database of relations), went on to introduce advanced frameworks such as semantic load shedding and other approximate query techniques such as data reduction, and summary techniques. These frameworks trade off result accuracy for efficiency.

In this work we introduce the concept of applying consistency over parallel, object-oriented data structures we call *Spatio-Temporal Cuboids* (ST-Cuboids), and doing this in real-time. ST-Cuboids are slices of a multidimensional space [18, 19, 20], and can be viewed as natural object-oriented generalizations of data streams; slice-like objects are popular in data analysis. An overview of the ST-Cuboid data model is provided in section 3 of this paper.

We believe the technique of imposing a new object-oriented data model on streams is both novel and significant for DSMS. ST-Cuboids can play a role for DSMS that is as fundamental as the role of a relation for a relational DBMS. They provide data abstraction and data independence, permitting many performance improvements. They also permit a variety of interesting DSMS extensions: indexing for random access, generalization of stream operations for spatio-temporal query, and parallel access and consistency. Because this consistency is most important in a real-time streaming environment, we refer to the framework here as *Real-Time Consistency* (RTC).

Improved performance can have great consequences for consistency. Based on both theoretical analysis and extensive experimental work, Real-Time Consistency in Distributed Streamonas has been able to provide 100% accurate results, and do this with impressive performance: consistent states have been materialized within time intervals of less than 400 μsec on the linear road benchmark (LRB) [3, 15] with a load of 550 Xways.

In section 4 we present the theoretical framework of Real-Time Consistency. Section 5 describes a novel asynchronous sub-system that implements RTC. The asynchronous sub-system re-uses the data stored in the ST-Cuboid database in an embarrassingly parallel manner, permitting significant performance improvements. Section 6 gives experimental results.

2. Previous Work

The issue of consistency in Data Stream Management Systems has a long history, with many important advances. For example the Tapestry system [23] developed the notion of Continuous Queries. In Tapestry, user-provided queries are re-written as Incremental Queries, which are executed periodically based on a time interval. Time intervals are also used by NiagaraCQ [9], which grouped together Internet-scale continuous queries based on the observation that many web queries share similar structures. Recently the work [7] introduced an event streaming system that embraces a temporal stream model to unify and further enrich query language features, handle imperfections in event delivery, define correctness guarantees, and define operator semantics. The system defines weak, middle and strong consistency while providing consistency guarantees as well as the ability to retract incorrect output and add the correct revised output.

In Aurora [1] tuples flow through a loop-free, directed graph of processing operations and the work introduced a fundamental classification for QoS graph types as Delay-based, Drop-based and Value-based. The work tries to maximize QoS towards 100% accuracy by applying advanced techniques such as semantic load shedding. Medusa [6] is a distributed version of Aurora intended to be used by multiple enterprises that operate in different administrate domains. Borealis [2] is a distributed system that inherits core stream processing functionality from Aurora and distribution functionality from Medusa.

The data abstractions used can have a significant effect on data consistency. An important aspect of data-flow systems is their ability to support continuous queries, and this property has a long history. For example, Transducers were used throughout the Tangram [22] stream processor, which provides a database flow computation capability. An adaptive data-flow architecture is adopted by TelegraphCQ [8], supporting a wide variety of intensive networked applications. CACQ [17] implements continuously adaptive, continuous queries. The Stream Processing Core [13], also a data-flow system, enables the execution of multiple stream processing applications simultaneously on a large cluster of machines. Each application is expressed in terms of a data-flow graph consisting of processing elements (PEs) that consume and produce streams of data. The STREAM project [4] Data Stream Management System prototype supports a large class of declarative continuous queries over continuous and traditionally stored data sets.

Query languages correspondingly influence consistency. Continuous Query Language (CQL) [5] reuses the formal foundations and large body of implementation techniques for relation-to-relation languages such as relational algebra and SQL by adding a small set of stream-to-relation and relation-to-stream operators. The Stream Mill system [24], through its Expressive Stream Language (ESL), defines user-defined aggregates (UDAs) on logical and physical windows and optimizes their computation. ESL treats data streams as unbounded ordered sequences of tuples. The Stream Mill Miner Data Stream Management System for knowledge discovery [24], provides a data stream mining workbench that combines the ease of specifying high level mining tasks with the performance and QoS guarantees of a DSMS. GSQL in Gigascope [10] is a pure stream query language with SQL-like syntax. All inputs in GSQL are streams, and the output is a data stream. Out-of-order processing [16] in NiagaraST and Gigascope provides an architecture that avoids ordering constraints by using explicit stream progress indicators, which simplifies the management of consistency.

In more recent work, cluster-based Data Stream Management System architectures apply the Map-Reduce [12] and Map-Reduce-Merge [26] programming models in data stream processing. Map-Reduce [12] is a programming model that enables highly-scalable (many terabytes of data on thousands of machines) and parallel data processing on large clusters of commodity machines. Map-Reduce-Merge [26] adds a Merge phase to Map-Reduce that can efficiently merge data already partitioned and sorted by map and reduce modules. The work demonstrates that Map-Reduce-Merge can express relational algebra operators especially supporting joins of heterogeneous datasets.

There have been many further extensions to the Map-Reduce model. The SCSQ DSMS in [28] exploited application-specific stream splitting (splitstream) functions where the user specifies non-procedural stream partitioning and replication. The work [29] then followed a split, map, merge paradigm (splitstream, mapstream, mergestream) in which an incoming stream is split into parallel sub-streams, with expensive query operators applied on these sub-streams in parallel. The work reached an impressive, breakthrough-level of 512 Xways on the Linear Road Benchmark. Other adaptations of map-reduce have been proposed recently. CaaaS [11] allows a long-standing SQL query instance to run cycle by cycle, each cycle for a chunk of data from an infinite data stream. The project scales-out analytics computation by implementing map-reduce during each cycle. The Project DEDUCE [14] combines the benefits of Map-Reduce and stream processing by providing periodic analysis of large amounts of stored data to generate a model using Map-Reduce, which is then used to process a stream of incident updates using a stream processing system.

By contrast with this work, the Streamonas system described here does not have a data-flow architecture or use Map-Reduce. Instead, it uses the parallel, object-oriented data model of ST-Cuboids [18, 19, 20] as a generalization of streams. The result has significant potential for DSMS. This data abstraction permits querying to be decoupled from the data streaming process, and performed independently and in parallel over the distributed database. As we will show, this decoupling also permits DSMS extensions that can improve performance and consistency.

3. The ST-Cuboid Data Model

While most data stream management systems process information directly from the incoming serial data stream, Streamonas processes its information after the serial stream is restructured into its own object-oriented data model called the Spatio-Temporal Cuboid (ST-Cuboid) [2].

As its name suggests, the ST-Cuboid models a slice of data in a multidimensional space [18, 19, 20]. Slices are increasingly common substream-like abstractions in data analysis systems. The ST-Cuboid permits restructuring and query of slice objects, gaining data independence and providing a foundation for performance and consistency improvements. The use of an object-oriented data abstraction for streams is a unique feature of Streamonas.

ST-Cuboids provide (i) random access to any temporal sequence of any entity in the database and (ii) random access to any data element within the whole spatio-temporal volume of the ST-Cuboid. These two fundamental principles of Streamonas are novel for data-stream systems and require an innovative, technical design, illustrated in Figure 5-1.

ST-Cuboids can model entities as evolving by storing their respective temporal sequences. An entity's attribute, instead of having a flat value of a standard type, has a temporal sequence called an atomic stream. The elements within atomic streams are indexed and can be randomly accessed based on their temporal order (time hashing).

For an example, let us assume a *CAR* ST-Cuboid with schema *CAR* (*Car_ID*: int, *SPEED*: *AST*(6)), with *Car_ID* being the unique identification of each car entity, and *SPEED* being an attribute of type atomic stream and historical span six. The space hash table (Figure 5-1) indexes the identifications, enabling random access of each car entity (architectural principle (i)). For

[2] All concepts of spatio-temporal cuboids used also throughout this paper, are defined along with extensive descriptions and diagrams in our previous publications [22, 23] and can be accessed through the IEEE Xplore digital library at http://ieeexplore.ieee.org/Xplore/home.jsp or through the project's website at http://www.Streamonas.org

example, in order to access a car entity with *Car_ID*=100 in the database we can randomly access the temporal sequence of its speed values through the expression *CAR*[100].*SPEED*(*).

At the same time we can access any data element within the temporal sequence of the *SPEED* atomic stream through a time hash table (architectural principle (ii)). For example we can access the latest speed data of the car with *Car_ID*=100 through the expression *CAR*[100].*SPEED*(0), or perform a historical reference, e.g. access the sixth latest speed data element through the expression *CAR*[100].*SPEED*(-5).

These object-oriented expressions can randomly access the whole volume of information within the ST-Cuboid, with excellent response times on the order of μsec. These rapid response times provide the real-time performance characteristics of Streamonas.

3.1. Stream Semantic Decomposition

Population of the ST-Cuboids with data elements is performed through a process that we call Stream Semantic Decomposition (StSD) [18, 19, 20]. StSD reads a serial incoming stream and routes its data (based on their unique keys and the hash table of the ST-Cuboid) to the respective entities/atomic streams in order to be stored. The process creates a transformation of the serial data model of a stream to the parallel data model of the ST-Cuboid data structure. During the transformation no data are dropped.

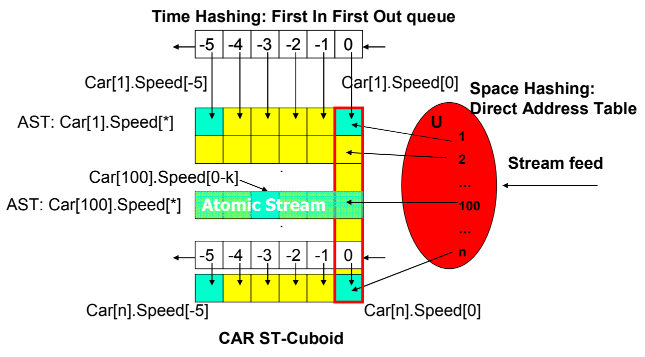


Figure 5-1. Diagram of a sample ST-Cuboid for Car speed in the LRB

Stream Semantic Decomposition is a permanent sub-system of the DSMS inaccessible to queries. The Stream Semantic Decomposition engine maintains the ST-Cuboid database consistent in real-time guided by the Normalized Schema of the Database of ST-Cuboids.

Stream Semantic Decomposition (StSD) as published in [18, 19] is different from partitioned sliding windows [5], as it is not query specific and it is not defined within a query. Because StSD supports a database model, it can support different applications that are using the same data.

Thus StSD is not application-specific in the same way as the stream partitioning/splitting that was subsequently introduced in [28]. Stream Semantic Decomposition is transparent to query expressions and it supports the DSMS in offering data independence in a streaming environment, as we discuss next.

4. The Real-Time Consistency Framework

While Database Consistency has been synonymous to accuracy and semantic correctness for relational DBMSs, enforcing consistency in real-time is so difficult that researchers often view the concept as a trade-off of accuracy for efficiency. Within this context, pioneering frameworks have been developed, including semantic load shedding and other approximate query techniques such as data reduction and application-specific summary techniques [1]. Similarly researchers in [7] define weak, middle and strong consistency.

Real-Time Consistency over the ST-Cuboid data model is a novel concept that has proven to be extremely effective in our experiments, both in terms of real-time performance as also in terms of accuracy. Performance improvements made possible by the ST-Cuboid model permit improved consistency, and in fact our implementation has achieved 100% accuracy without trade-offs. At the same time the existence of a consistent database instance enables reusability of information, and this permits embarrassingly parallel queries.

The mathematical framework presented in this section is feasible because of the ST-Cuboid data model. The framework provides necessary conditions for stability and semantic correctness for real-time consistency. The in depth technical analysis of the framework, methodologically organized in Real-Time Consistency for Streaming (sub-sections 4.1 and 4.2) and Real-Time Consistency for Querying (sub-sections 4.3 and 4.4) provides a robust foundation. This permits understanding of its synchronous processing principles and guides development of both real-time-critical and accuracy-critical applications.

The term "real-time" here is used in an abstract sense, many times just implying "fast". The question is how "fast" real-time should be, or can be, and what are its formal semantics.
In [19] we provide perspective on this question, treating "real-time" as a theoretical limit that current processing systems cannot achieve (as this could imply infinite processing capability, with speeds exceeding the speed of light). The word "fast" makes most sense when related to clock cycle times T or CPU speed (frequency in Hz). With a definition like this, the theoretical limit of real-time often can be reasonably replaced with the expression "as fast as possible". Many real-time applications require the latest available information as fast as possible.
Based on the above, Real-Time Consistency (RTC) enforces the maintenance of the latest available consistent database, thus enabling provision of the latest consistent information as fast as possible. While $T \rightarrow 0$ (i.e., CPU speed $\rightarrow \infty$) is a theoretical limit that cannot be achieved, it is useful to define the minimal real-time processing interval as the limit

$$T_o \rightarrow (1 / \text{CPU speed})$$

Equivalently the maximal technologically-feasible speed (frequency) can be defined as:

$$\Phi_o = 1 / T_o$$

Providing real-time continuous querying in Streamonas is based on a two phase process as follows:

(i) *Real-Time Consistency phase* (for streaming): the system evaluates the *latest* consistent database instance *as fast as possible*, and

(ii) *Query phase*: the system enables querying of the consistent database instance as fast as possible before the latest consistent database instance becomes invalid (i.e. before the next tuple arrives in the system).

We examine two types of queries: (1) Queries that store persistent information in the database that is required to be updated based on new data, and (2) Queries that are read-only and do not require query specific information to be stored and maintained.

For queries that require storage of persistent information, another phase called *Real-Time Consistency for querying* is required. The phases RTC for Streaming and RTC for Querying are serialized as RTC for Streaming provides the latest information in the database independently of query logic, as it is specialized in appending the streaming tuples based on the normalized schema. The requirement for existence of RTC for Querying depends on the semantics of the application. As an example our Linear Road Benchmark implementation [18, 19] relies heavily on RTC for Querying. The above can be expressed as below:

RTC for Streaming ; (RTC for Querying) ; Querying*

(Expr. 1)

Figure 5-2 illustrates the concept. In this diagram, incoming data flow from a data stream source with rate λ is received by the system.

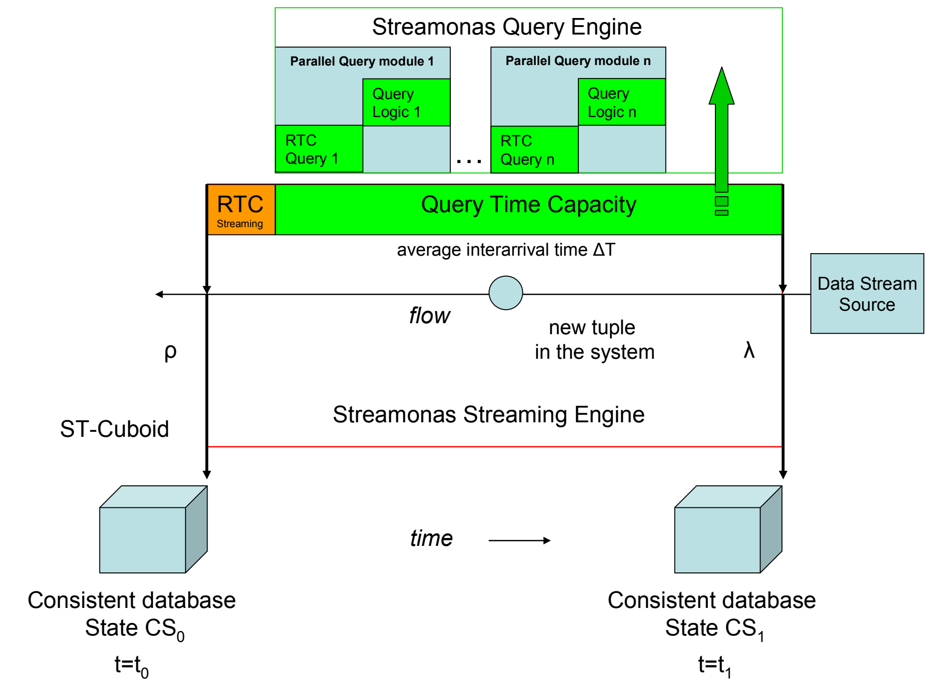


Figure 5-2. The Real-Time Consistency Framework

The diagram models the time vector with a direction from left to right, by modeling the data flow from right to left (i.e., older information is stored in ST-Cuboids at the beginning of the time axis). In the diagram we assume that at time t_0 the ST-Cuboid database is in a consistent state CS_0. Before a new tuple is appended (append commit) there is a time interval that is on average $\Delta T = 1/\lambda$ (average interarrival time). During this period the ST-Cuboid is static and consistent, until time t_1 when a new consistent database snapshot CS_1 is instantiated.

The diagram also demonstrates the two phases presented earlier for the period of time $[t_0, t_1)$: The Real-Time Consistency for streaming phase captures a percentage of the available time ΔT, while the remaining time is made available for querying. This available time is named Query Time Capacity (QTC). During QTC a number of parallel query modules run over the database which can have a respective RTC for Querying phase.

Having decoupled the streaming process from the querying process, as we explain in publications [18, 19], we can now independently address two "real-time" specific questions in the following sections:

1. How fast can we map a consistent database instance based on the newly arrived fact (tuple) to a new database instance (streaming layer)?
2. How fast can we answer continuous queries that require the latest available information, (i.e. based on the latest consistent database instance (query layer))?

In order to maintain stability, operations described in (Expr.1) should be on average completed within a time period that is less than $\Delta T = 1/\lambda$.

More formally, from (Expr.1) we can derive the *stability condition*:

$$\textit{duration } \{\textit{RTC for Streaming} + (\textit{ RTC for Querying })^* + \textit{Querying }\} \; < \; 1/\lambda$$

(Expr. 2)

In sections below devoted to Real-Time Consistency for Streaming (sections 4.1 and 4.2), we describe how the Real-Time Consistency framework is applied over the streaming process for both the single-cpu/single-core Streamonas architecture and the Distributed Streamonas architecture, fine-tuning Expr.2 for each case. Also, in sections devoted to Real-Time Consistency for Querying (sections 4.3 and 4.4) the concept is analyzed both for the single-cpu/single-core Streamonas architecture and for the Distributed Streamonas architecture.

4.1. Real Time Consistency for Streaming on single-CPU/single-core Streamonas DSMS

Let us assume that an initially consistent database instance of Spatio-Temporal Cuboids, which follow a normalized schema, is being populated by a serial stream of tuples. The serial stream of tuples through respective append transactions uniquely defines a sequence of consistent database instances, each one generated by the previous one.

Furthermore let us assume a consistent database instance $CS_{[t0,t1)}$ is valid for the time interval $[t_0,t_1)$ and a unique series of tuples is streamed into a DSMS system over time:

$$e_{[t1,t2)},\ e_{2[t2,t3)},\ \dots,\ e_{[tcurrent\text{-}1,tcurrent)} \qquad \textbf{(Expr. 3)}$$

The interarrival times and durations, $[t_1,t_2)$, $[t_2,t_3)$, … , $[t_{current\text{-}1},t_{current})$ are statistically described by the dataflow rate λ_{in} and the respective average interarrival time $1/\lambda_{in}$. We also assume that these durations are independent of other parameters of the DSMS which are kept constant.

Starting from an initial consistent database instance $CS_{[t0,\ t1)}$, that is valid during the interval $[t_0, t_1)$, we obtain the following unique sequence S of consistent database instances:

$$S\colon CS_{[t0,\ t1)} \xrightarrow{e_{[t1,t2)}} CS_{[t1+\delta 12,\ t2)} \xrightarrow{e_{[t2,t3)}} CS_{[t2+\delta 23,\ t3)} \xrightarrow{e_{[t3,t4)}}$$

$$\dots \xrightarrow{e_{[tcurrent-1,tcurrent)}} CS_{[tcurrent\text{-}1+\delta\ current\text{-}1\ current,\ tcurrent)}$$

(Expr. 4)

The arrows here denote the transformation applied on each consistent database instance, yielding the next consistent database instance based on a new tuple. The duration $\delta_{i(i+1)}$ denotes the time needed for the append transactions to commit in order to bring the database of spatio-temporal cuboids to a consistent state. We call the average δ of $\delta_{i(i+1)}$ the *Average RTC Commit Time*.

Respecting the stability condition (Expr.2), the Average RTC Commit Time δ should be less than the average interarrival time $1/\lambda_{in}$. More formally:

$$\delta < \textit{average interarrival time } 1/\lambda_{in} \qquad \textbf{(Expr. 5)}$$

In addition to Expr.5, in order to enforce accurate results based on the latest valid consistent database instance, we need to enforce the following strict *real-time consistency condition* on a per-tuple basis:

$$\delta_{i,i+1} \textit{ does not exceed } [t_i,t_{i+1}) \textit{ for all } i \geq 0. \qquad \textbf{(Expr. 6)}$$

In case the above strict expression is not respected, i.e. $\delta_{i,i+1} > [t_i,t_{i+1})$ takes time from the next interarrival time duration, the consistent state $CS_{[ti,\ ti+1)}$ once materialized, will not be reflecting the latest knowledge base, as a new data element $e_{[ti+1,ti+2)}$ during this next period $[t_{i+1},t_{i+2})$ will have arrived.

The synchronous manner of RTC for Streaming (per tuple in single-cpu/single-core Streamonas and per-batch in Distributed Streamonas) enforces this strict expression even in the case where spikes are observed in the average behavior of the system. The bottom line of figure 5-3 (marked as (2)), presents the Average RTC Commit Time δ during our experiments with the Linear Road Benchmark application (10 Xways single-cpu/single-core) which converges to value 7 μsec. Average interarrival time $1/\lambda_{in}$ during the same experiments (top line of figure 5-3 – marked as (1)) was 49 μsec.

Equivalently by moving in the frequency domain we can express that the average Real Time Consistency for streaming frequency $\Phi_{o,RTC,S} = 1/\delta$ should be higher than the average interarrival time between facts (tuples). Based on the above we receive the following expression:

$$\Phi_{CPU} > \Phi_{o,RTC,S} > \lambda_{in} = \textit{constant}. \qquad \textbf{(Expr. 7)}$$

The above condition recognizes for $\Phi_{o,RTC,S}$ the CPU frequency as the least upper bound due to technology limitations, while setting λ_{in} as the greatest lower bound as explained by the stability condition (Expr.2). In our single-cpu/single-core experimental results, the above expression was satisfied with Φ_{CPU} = 3GHz and $\Phi_{o,RTC,S}$ > 66,226 tuples/sec with an arrival rate λ_{in}= 20,368 tuples/sec.

Expr.4 leads to the following recursive expression, where $\Phi_{CPU} > \Phi_{o,RTC,S} > \lambda_{in}$ = constant:

$$CS_{[tcurrent+(1/\Phi o,RTC,S),\ tcurrent+1)} \leftarrow$$

$$CS_{[tcurrent-1+\ (1/\ \Phi o,RTC,S),\ tcurrent)}\ |\ e_{[tcurrent,\ tcurrent+1)}$$

(RTC Model 1)

providing the Real-Time Consistency model connecting semantic validity and stability of the system with frequency of execution. In other words, by following RTC Model 1, the system is guaranteed to (i) operate in a stable manner, (ii) maintain database and transaction consistency while *providing the latest consistent database state available,* valid during the time period in the immediate future: $[t_{current+(1/\Phi o,RTC)}, t_{current+1})$.

RTC Model 1 also links the operation of Real-Time Consistency to the series of events (signal) $e_{[tcurrent,\ tcurrent+1)}$ of Expr.3, signaling the arrival of a new tuple. In RTC Model 1, the recursive indexing has advanced by one as it is more natural to describe the transition from the current consistent state to the next. Conceptually, there are two dynamic processes: the signal $e_{[tcurrent,\ tcurrent+1)}$ generates a forward-looking temporal series $[t_{current}, t_{current+1}, \ldots)$, while the consistent state evaluation based on frequency parameter $1/\Phi_{o,RTC,S}$ is a backward looking process (recursive operator "$\leftarrow$"), where each consistent state relies on the evaluation of the previous one in a recursive manner.

It is clear that all systems implementing higher frequencies than $\Phi_{o,RTC,S} = \lambda_{in}$ provide equivalent results in terms of consistency, as higher frequency does not carry with it any additional information than the information streamed in the system through λ_{in}. Thus, by applying RTC Model 1 where the frequency $\Phi_{o,RTC,S}$ is the greatest lower bound for semantic validity and stability λ_{in} , we obtain Model 2, which describes Real-Time Consistency in a "push" streaming environment. In this environment, with initial state $CS_{[t0,\ t1)}$ and synchronous arrivals of a new tuples in an incoming data stream with data flow rate λ_{in} and $\Phi_{CPU} > \Phi_{o,RTC} = \lambda_{in}$:

$$CS_{[tcurrent+(1/\lambda in),\ tcurrent+1)} \leftarrow$$

$$CS_{[tcurrent-1+\ (1/\lambda in),\ tcurrent)}\ |\ e_{[tcurrent,\ tcurrent+1)}\ ,$$

(RTC Model 2)

Again $\delta_{i,i+1}$ must not exceed $[t_i,t_{i+1})$ for all $i \geq 0$. We need to emphasize that losing the semantics of a single tuple for a single consistent state destroys the validity of all future consistent states. Re-building consistency once it is lost is definitely not an easy task in a streaming environment with high throughput [3, 15].

4.2. Real Time Consistency for Streaming on Distributed Streamonas DSMS

In Distributed Streamonas, enforcing RTC on a per-tuple basis would create a bottleneck, diminishing any benefits of the scalable environment of a cluster. Furthermore, the overhead of message passing for processing a single tuple would be high. For this reason, processing is performed on batches of tuples. To enforce RTC, we adapt Expr.3 to define the following model of the input stream:

$$b_{[t1,t2)}, e^*_{[t1,t2)}, b_{2[t2,t3)},\ e^*_{[t2,t3)},\ \ldots\ , b_{[current-1,current)},\ e^*_{[current-1,current)}$$

Here $b_{[ti,ti+1)}$ denotes the arrival of a batch of tuples which includes $e^*_{[ti,ti+1)}$ elements (tuples). These batches of tuples are processed in parallel in a non-deterministic manner within the interval the batch is valid (i.e., within $[t_i, t_{i+1})$).

Like Expr.4, for Distributed Streamonas we have:

$$S\colon CS_{[t0,\ t1)} \xrightarrow{b_{[t1,t2)}} CS_{[t1+\psi12,\ t2)} \xrightarrow{b_{[t2,t3)}} CS_{[t2+\psi23,\ t3)} \xrightarrow{b_{[t3,t4)}}$$

$$\ldots \xrightarrow{b_{[tcurrent-1,tcurrent)}} CS_{[tcurrent-1+\psi\ current-1\ current,\ tcurrent)}$$

(Expr. 8)

In other words, Distributed Streamonas guarantees consistent database instances after processing all tuples within each batch. During the parallel processing of the tuples, no guarantees are made for the consistency of the database; it reaches its consistent state when all append transactions of the tuples in the batch are committed.

In Expr.8, the Average RTC Commit Time $\psi_{(i,i+1)}$ of Distributed Streamonas denotes the time ψ needed for all tuples in batch $b_{(i,i+1)}$ to be processed in parallel, and for the respective append transactions to commit. This brings the database of spatio-temporal cuboids to a consistent state $CS_{[ti+\psi(i,i+1),\ t(i+1))}$. As the parallel processing of the tuples is non-deterministic, we cannot define consistent database instances within the time duration $[t_i,\ t_i+\psi_{i(i+1)})$ while the append transactions take place. Only after time $t_i+\psi_{i(i+1)}$ when all transactions have committed does the consistent database instance become available for querying. While performance considerations do not allow us to apply Real-Time Consistency on a per-tuple basis, the temporal sequence of consistent database instances generated from batches of tuples in order to guarantee correctness should provide the same correct results as if the tuples in each batch were serialized (theoretical serialization based on their original timestamps).

In our design, through fragmentation, we ensure that parallel execution of the append transactions for the tuples within each batch by the system is performed over different fragments without conflicts. Thus the order of execution of the append transactions is not significant. The Stream Semantic Decomposition (StSD) of Distributed Streamonas described in section 3.1 enables embarrassingly parallel processing as it decomposes streams into individual streams (called atomic streams) describing the evolving behavior of each entity in the database. Assuming no rule procedures or any transaction dependencies, the ACID properties of transactions, Atomicity, Consistency, and Isolation [21] are preserved, as append-only operations are performed in parallel on independently modeled and stored atomic streams. Due to the streaming nature of the data and FIFO architecture of ST-Cuboids, Durability is maintained for a limited time, depending on application semantics.

Within each batch of tuples there are two cases (i) each tuple is aimed to be appended to a different atomic stream (ii) multiple tuples for each atomic stream may be included in the same batch.

Streamonas makes provisions that the temporal order of tuples is maintained within each batch. As the size of the semantic space i.e. the number of evolving entities stored in the system is much larger than the number of tuples in each batch, the first case described above, is expected to be frequently applied in various applications.

As an example, in our experiments for the Linear Road Benchmark, we were able to model and store 75.5 million evolving car entities. For this application, Distributed Streamonas uses 5 batches (each one aimed to a different node), hence processing in parallel 55 tuples in each

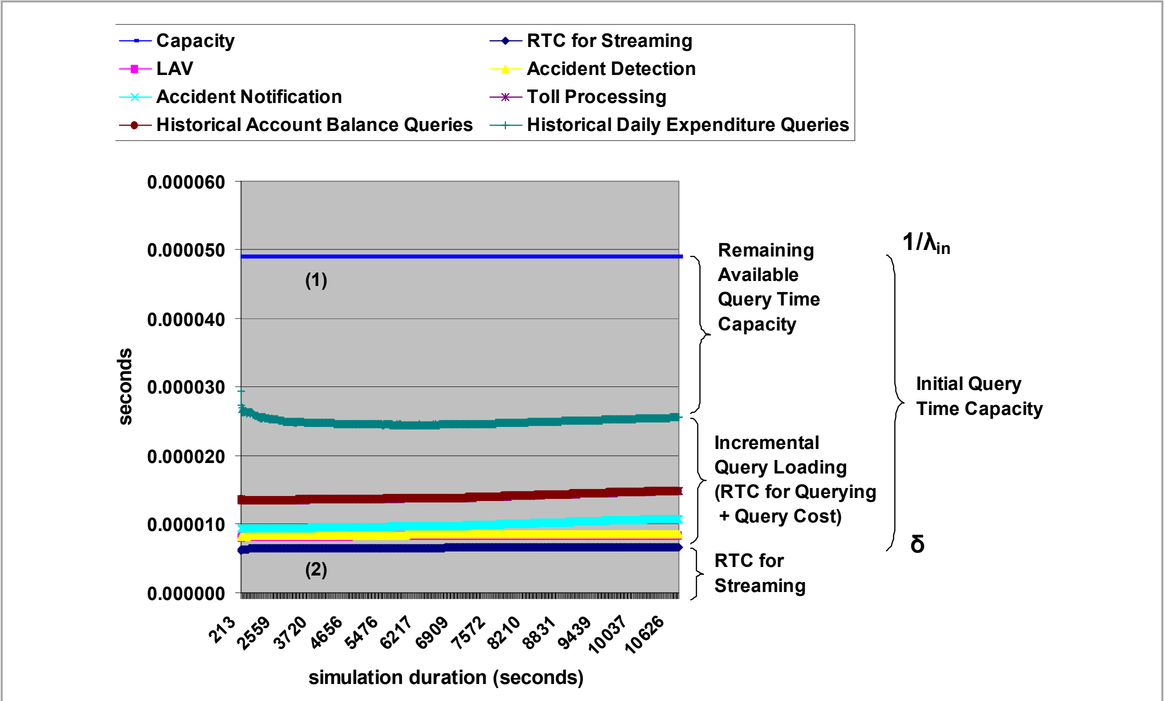


Figure 5-3. Quality of Service map for single-cpu/single-core Streamonas (load of 10 XWays)

batch. In the ST-Cuboid database fragments at each node, 15.1 million evolving entities are managed, a number of entities much larger than the 55 tuples within each batch. Figure 5-4 (bottom line marked as (2)) presents the Average RTC Commit Time ψ for 55 parallel ST-Cuboid fragments in a symmetric topology converged to 69 μsec (for batches of 5 tuples) after a 3 hour simulation.

If there is a high frequency of tuples that need to be appended to a specific atomic stream (i.e. the distribution of tuples over the database is not uniform), case (ii) may apply and the temporal order originally maintained within each batch will also be preserved. That is, during the decomposition of the data that updates a specific atomic stream, data elements are serialized per atomic stream based on their original temporal order, thus maintaining temporal semantics.

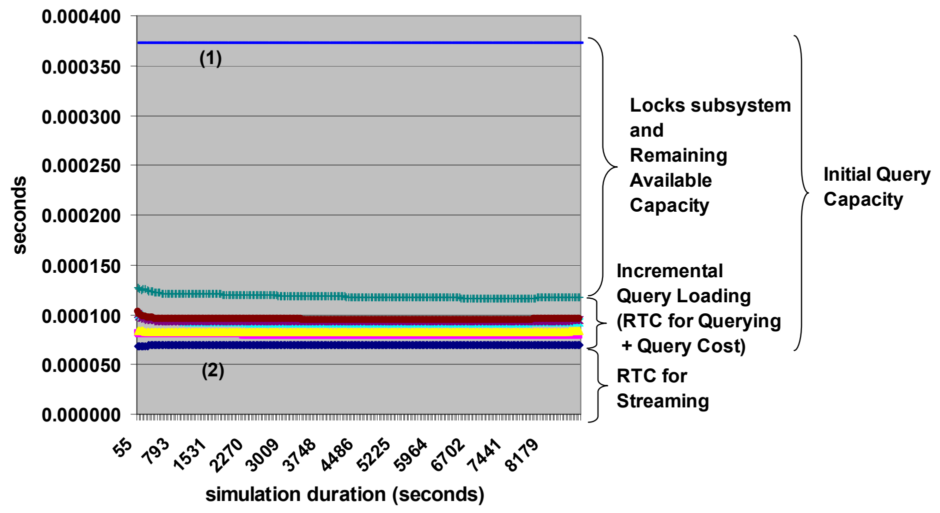


Figure 5-4. Quality of Service map for Distributed Streamonas (load of 550 XWays)

4.3. Real-Time Consistency for Querying and Query Evaluation on single-CPU/single-core Streamonas

As explained in the previous sections, by managing append transactions over ST-Cuboids of the database, Real-Time Consistency for streaming ensures that the database is kept consistent upon each arrival of a new tuple. In order to maintain the *stability* of the system the Average RTC Commit Time δ required for RTC for streaming ($RTC_{Streaming}$) should be less than the average interarrival time of the tuples from the incoming stream (Expr.5). Also in order to maintain RTC on a per-tuple basis, the duration of each append transaction needs to be lower than the interarrival time (Expr. 6).

The stability condition and real-time consistency condition formalizing the above concept analyzed in the previous sections were:

$$\delta < \textit{average interarrival time } 1/\lambda_{in}$$

$$\delta_{i,i+1} \textit{ does not exceed } [t_i, t_{i+1}) \textit{ for all } i \geq 0.$$

Once the duration $\delta_{i,i+1}$ for $RTC_{Streaming}$ is completed, and the consistent database instance $CS_{[ti+\delta i(i+1),\ ti+1)}$ is available, the remaining available time duration $[t_i+\delta_{i(i+1)},\ t_{i+1})$ can be used for query operations. As we have already mentioned in section 4 this duration is the Query Time Capacity (QTC). Query Time Capacity is demonstrated theoretically in figure 5-2, and presented experimentally in Figure 5-3 as the zone between average values δ and $1/\lambda_{in}$. As δ in our Linear Road Benchmark experiments (10 Xways single-cpu/single-core) was 7 μsec, and $1/\lambda_{in}$ was 49 μsec, and the initial Query Time Capacity was 42 μsec.

Within this interval, the Query Layer of Streamonas accesses a consistent database instance, which has been maintained by the Streaming Layer having available time equal to QTC in order to apply query logic.

As we have already mentioned in previous sections, some queries may need to store persistent information in the ST-Cuboid database. For this reason the database schema is extended to support query-specific information that needs to be maintained in a continuous manner by the Real-Time Consistency framework. This real-time maintenance is performed by the R*eal-Time Consistency for Querying* module ($RTC_{Querying}$). $RTC_{Querying}$ is performed in a synchronous manner upon the arrival of each new tuple.

$RTC_{Querying}$ semantics are maintained based on (i) query semantics (ii) the new tuple and (iii) the existing consistent database instance. The RTC requirements of some queries may be very expensive, depending on application requirements and the level of linkage with the signal of the arrival of a new tuple. The overhead required by $RTC_{Querying}$ can be measured by the average processing time required for RTC per query q ($RTC_{Query\ q}$) as presented next.

4.3.1. RTC for Querying

For the case of a single-cpu/single-core system each one of the $RTC_{Query\ q}$ periods incurs overhead. During each interarrival period $[t_i,t_{i+1})$ for $i \geq 0$ the overall average Real-Time Consistency overhead Θ_{RTC} is:

$$\Theta_{RTC} = RTC_{Streaming} + RTC_{Querying} = RTC_{Streaming} + \sum_{q=1}^{n} RTC_{Query\ q} \qquad \textbf{\textit{(Expr. 9)}}$$

Each $RTC_{Query\ q}$ term is subtracted from the Query Time Capacity (QTC), thus constraining the number of additional queries to be added to the system. This is shown in Figure 5-3 as Incremental Query Loading. In order to ensure a stable system the following inequality must hold (on average):

$$\sum_{q=1}^{n} RTC_{Query\ q} < QTC = 1/\lambda_{in} - \delta. \qquad \textbf{(Expr. 10)}$$

4.3.2. Overhead of Query Evaluation

Once $RTC_{Streaming}$ and $RTC_{Querying}$ leave the database in a consistent state, Query evaluation can be performed.

A continuous query *q* re-materialized on a per-tuple basis creates an additional overhead L_q. Extending Expr.10 to include also these terms we obtain:

$$\sum_{q=1}^{n} RTC_{Query\ q} + \sum_{q=1}^{n} L_q < QTC. \qquad \textbf{(Expr. 11)}$$

From expressions 10 and 11 we receive the following *QTC expression*, which estimates the available Query Time Capacity left in the system, when incremental loading of queries is performed:

$$1/\lambda_{in} - (\delta + \sum_{q=1}^{n} \mathrm{RTC}_{\mathrm{Query\ q}} + \sum_{q=1}^{n} \mathrm{L}_q\ ) > 0 \qquad \textbf{(Expr. 12: QTC)}$$

The left side of the inequality expresses the Query Time Capacity *ΔQTC* available in the system when gradual loading of queries is performed. Thus Expr.12 can be equivalently expressed as:

$$\Delta QTC > 0. \qquad \textbf{(Expr. 13)}$$

The Quality of Service map of Figure 5-3 shows the incremental loading of $\mathrm{RTC}_{\mathrm{Query\ q}}$ and terms L_q, gradually filling the capacity of the system. Specifically the term $(\delta + \sum_{q=1}^{n} \mathrm{RTC}_{\mathrm{Query\ q}} + \sum_{q=1}^{n} \mathrm{L}_q\ )$ at the end of the incremental loading has an average value of 26 μsec. The QoS map presents the behavior of the Linear Road Benchmark queries (10 Xways single-cpu/single-core). As shown in the QoS map, even after the deployment of the whole LRB application, the system remains within the stable region (Expr. 12), while there is still available Query Time Capacity in the system ($1/\lambda_{in}$ – 26μsec) = (49-26) μsec = 23 μsec. This means that the system can support additional query logic over the same database.

QTC guarantees that 46.9% (0.000023/0.000049) of the system's capacity is available to be loaded with additional query logic, assuming that other streaming layer parameters (such as memory management) remain constant. As continuous queries may not be necessary to be re-materialized on a per-tuple basis, the term $\sum_{q=1}^{n} L_q$ for Expr.12 may be inactive for a large volume of data, and thus Expr.12 reflects a worst-case scenario in terms of overhead.

Also for an application where a single read only query request is requested by an individual tuple having cost *L*, Real-Time Consistency is not performed. Consequently the Real-Time Consistency terms are inactive, and Expr.12 becomes

$$L < 1 / \lambda_{in} \qquad \textbf{(Expr. 14)}$$

Like the real-time consistency condition of Expr.6, the following strict per-tuple condition is maintained synchronously with the incoming tuples (Expr. 3):

$$\delta\,|_{[ti,ti+1)} + \sum_{q=1}^{n} \mathrm{RTC}_{\mathrm{Query}\ q}\,|_{[ti,ti+1)} + \sum_{q=1}^{n} L_{\mathrm{q}}\,|_{[ti,ti+1)} < [t_i,\ t_{i+1}) \qquad \textbf{(Expr. 15)}$$

For the case where hard real time constraints are imposed by the application itself (e.g. query latency constraints) the inequality constraint expressed by expressions 12 and 15 should be valid, guaranteeing the average *stability* and *per-tuple strict Real-Time Consistency* respectively, along with an additional constraint on a per-tuple basis:

$$\delta\,|_{[ti,ti+1)} + \sum_{q=1}^{n} RTC_{Query\ q}\,|_{[ti,ti+1)} + \sum_{q=1}^{n} L_q\,|_{[ti,ti+1)} < \textit{temporal constraint, for every tuple arriving in the system}$$

(Expr. 16)

This constraint enforces consistency within hard-real time performance constraints, based on *application requirements on a per-tuple basis*, i.e. not on average terms.

In our experiments we verified that the constraint of Expr.16 was satisfied by evaluating and reporting the maximum value of the left side of the expression throughout the simulation. For the LRB application this maximum value (due to spikes) was 0.14 seconds, which is much lower than the 5 seconds set by the benchmark.

While Expr.15 is enforced by the synchronous sub-system, we make sure that average λ_{in} levels are maintained based on Expr.12. In summary, while Expr.12 guarantees the stability of the system by evaluating average characteristics during the operation of the system, Expr.15 is evaluated on a per-tuple basis, guaranteeing strict real-time consistency. The synchronous manner of RTC for Streaming enforces both expressions even in cases where spikes are observed. Finally Expr.16 ensures application developers that the system conforms to application-specific hard real-time constraints.

4.4. Real-Time Consistency for Querying on Distributed Streamonas DSMS

In the previous section, we have presented the real-time consistency for Querying for a single-cpu/single-core case. In this section we present the same framework for Distributed Streamonas DSMS, highlighting the many benefits in terms of query efficiency the distributed architecture provides.

Most data-flow centric systems apply querying in series with the streaming flow performed through query graphs. For these systems, the ability to scale up depends highly on network bandwidth constraints and query complexity.

The distributed architecture of Streamonas reuses the already stored data in the database of ST-Cuboids, permitting a large number of parallel queries to access the same spatio-temporal cuboid database object, avoiding the transfer of large volumes of data.

For these queries Streamonas is <u>not</u> constrained by (i) large network latencies caused by query complexity or (ii) the available bandwidth on the streaming layer or (iii) the completion of transfers of streamed data.

As mentioned in section 4 (Expr. 1), the single-cpu/single-core architecture performs cycles over the sequence of operations:

RTC for Streaming Transactions ;

*(RTC for Querying Transactions/Logic) * ;*

Querying Logic

Here the symbol " * " denoting that RTC for Querying is applicable only for queries keeping query-specific persistent data in the ST-Cuboid database that require being kept consistent.

Distributed Streamonas DSMS scales-up by parallelizing the sequence of Expr.1 as follows:

(|| RTC for Streaming Transactions) **;**
(|| RTC for Querying Transactions/Logic) * **;**
(|| Querying Logic) **(Expr. 17)**

The above expression formally describes the process of a set of parallel RTC for streaming transactions followed by a set of RTC for querying transactions and logic. Once completed the database is available for query.

RTC for Querying has also the role of an interfacing module linking the Streaming layer with the Querying layer of the system. This enforcement of RTC synchronizes the two layers as it registers queries for RTC in a serialized manner with append transactions. Later in this section we will discuss how the Streaming layer and the Querying layer can be totally decoupled, working in parallel modes.

In our research an 1-1 fragment to core architecture was developed. Based on this architecture, let us assume a node serving *N* fragments by assigning *N* cores to them in an 1-1 architecture. Let's also assume for simplicity that a batch includes one tuple per fragment (even in the case of each batch sending a sequence of tuples for each fragment, only a single tuple can enter each fragment at each time after read from the batch). Assuming that λ_{node} is the rate of the dataflow arriving at the node (tuples/sec), as atomic streams are independent from each other, append

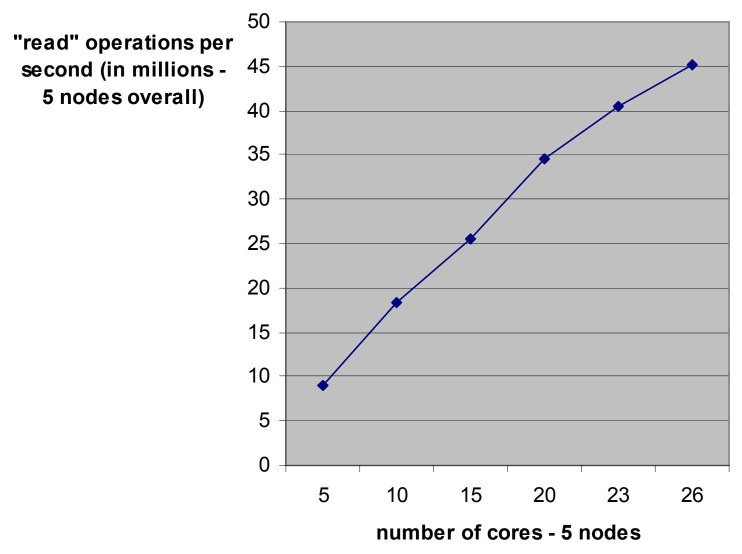


Figure 5-5. Asynchronous query layer performance through incremental deployment of cores

transactions can be performed in isolation in a parallel manner. Multiple cores access each batch in parallel. In an extreme case, each core could be serving a single atomic stream, applying an embarrassingly parallel framework at hardware level. As each RTC for streaming operation is performed in parallel, the cost for *N* parallel operations remains the same as for the single-cpu/single-core case and on average we can express:

$$\text{duration } \|_N \text{ RTC}_{\text{Streaming}} = \delta = \text{duration RTC}_{\text{Streaming}}$$

Thus Expr.17 becomes:

((embarrassingly ||) RTC for Streaming Transactions) **;**

(|| RTC for Querying Transactions/Logic) * **;**

(|| Querying Logic) **(Expr. 18)**

To maintain persistent information in a continuous manner, a number U of parallel queries can host their $RTC_{Querying}$ data in different spatio-temporal cuboids. $RTC_{Querying}$ can be parallelized, with its worst case duration being equal to the maximum duration of $RTC_{Querying}$ of each one of the parallel queries/query applications:

$$duration \ ||_U \ RTC_{Querying} = max \{ RTC_{Query\ p} \}, p=1,\dots,U$$

In a symmetric problem with on average equal durations $RTC_{Query\ p}$, as it is our implementation of the Linear Road Benchmark on multiple fragments and in average terms, the above expression becomes:

$$duration \ ||_U \ RTC_{Querying} = RTC_{Query\ p}$$

As $RTC_{Querying}$ is by itself a program it can be internally parallelized reducing the maximum duration at a lower level than the single-cpu/single-core case. Such intra-parallelism requires provisions for maintaining consistency when conflicts between operations occur, a direction which is within the scope of our future research.

Thus Expr.18 becomes:

((embarrassingly ||) RTC for Streaming Transactions) ;

(|| (intra ||) RTC for Querying Transactions/Logic) ;*

(|| Querying Logic) **(Expr. 19)**

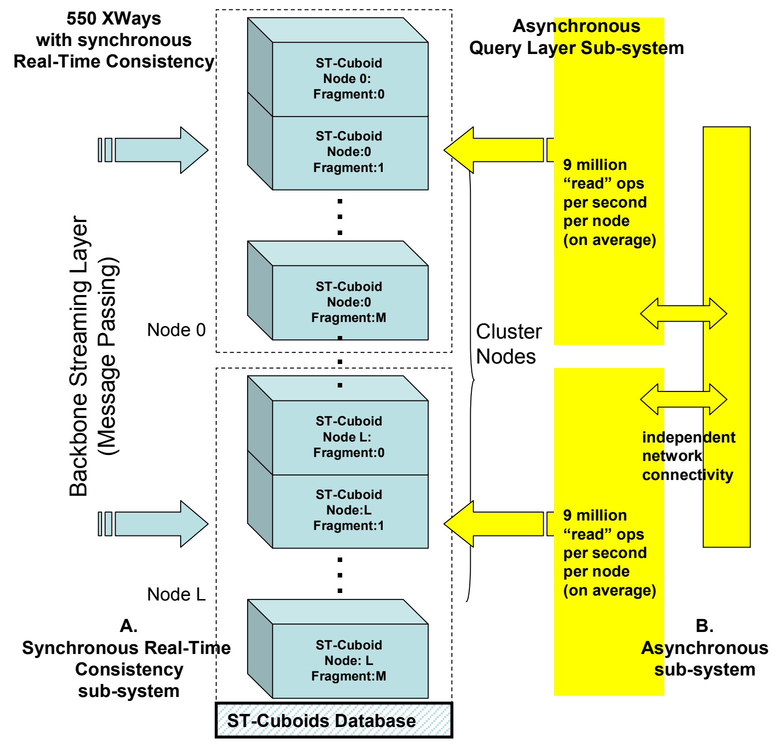


Figure 5-6 Distributed Streamonas Architecture with both Synchronous Real-Time Consistency sub-system and Asynchronous sub-system

This expresses that a module of embarrassingly parallel append transactions is followed by a module of parallel Real-Time Consistency Transactions which may also be internally parallelized (intra-transaction parallelism), followed by a module of parallel Query Transactions. After $\text{RTC}_{\text{Querying}}$ is performed, a number of *Q* queries having durations *Dq* are applied which access in a read-only manner the fragments of the database of spatio-temporal cuboids in an embarrassingly parallel manner (please note that not all queries require to have $\text{RTC}_{\text{Querying}}$ program modules). A specific class of these queries may also be able to be internally parallelized (intra-parallelism) and thus be able to be deployed on a large number of cores.

The duration of the execution of the parallel queries is:

$$duration\ ||_{Q}\ Querying\ Logic = max\ \{\ D_q\ \},\ q=1,\ldots,Q$$

(Expr. 20)

In a symmetric problem with on average equal durations D_q, as it is the case of our implementation of the Linear Road Benchmark on multiple fragments, the above term becomes:

$duration\ ||_{Q}\ Querying\ Logic = D_q$

Based on the above Expr.19 becomes:

((embarrassingly ||) RTC for Streaming Transactions) **;**

*(|| (intra ||) RTC for Querying Transactions/Logic) ** **;**

((embarrassingly ||) (intra ||) Querying Logic)

(Expr. 21 : Streamonas Embarrassingly Parallel Model)

Distributed Streamonas also supports global queries over the distributed spatio-temporal cuboid database, which can be broken to individual subqueries per node. This case can also be described by Expr.20, where D_q describes the duration of each sub-query. The Streamonas Embarrassingly Parallel Model, includes the very powerful expression:

((embarrassingly ||) (intra ||) Querying Logic)

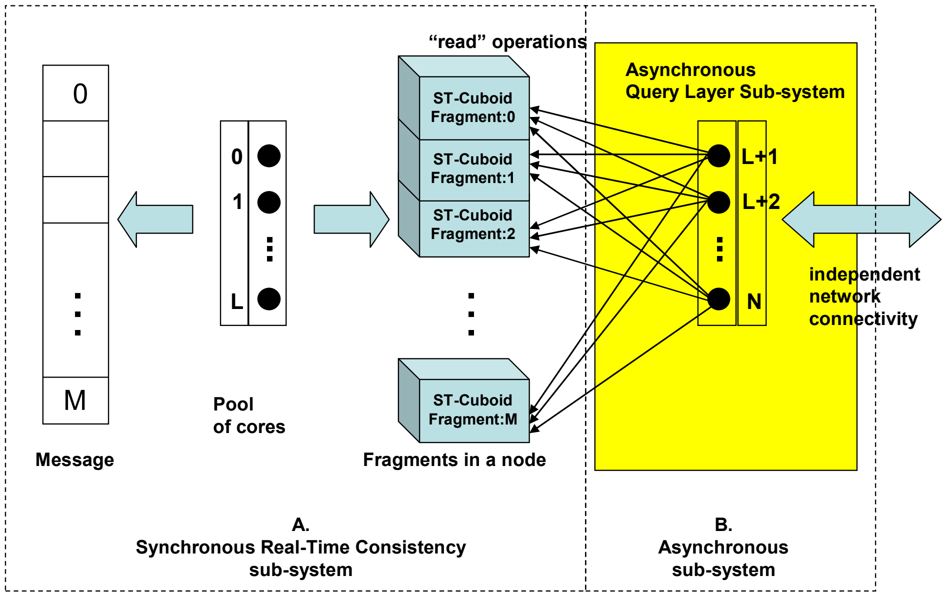


Fig. 5-7. Asynchronous access of Spatio-Temporal Cuboid database fragments by a pool of cores at a Node

which allows for complex query applications to run in parallel in a streaming environment at large scale, accessing through object-oriented expressions, the same spatio-temporal cuboid database object in a distributed manner, re-using the stored information, a paradigm not followed up to the moment by DSMSs as described in the introductory section of this paper and also analyzed in the following section.

5. ASYNCHRONOUS, EMBARRASSINGLY PARALLEL QUERYING OF THE ST-CUBOID DATABASE

Streamonas through the existence of the database of the object-oriented ST-Cuboid data structures, enables the access of its database by a large number of external queries in an embarrassingly parallel, asynchronous manner. The combination of the synchronous real-time consistency framework with asynchronous processing is a novel concept in the Data Streams domain.

At the same time the reusability of information, while natural for the ST-Cuboid model following a normalized schema, is more challenging for dataflow-centric architectures, as in these architectures, tuples are diffused through a network of operations and thus sets of these tuples become query-specific, making their re-usability to depend on query semantics.

Based on Streamonas Embarrassingly Parallel Model (Expr. 21), we have developed an additional sub-system that decouples querying from streaming, operating in an asynchronous manner. The following model, describes *subsystem A* running the synchronous Real-Time Consistency framework, while *subsystem B* performs querying in a very powerful asynchronous manner.

Subsystem A. synchronously at frequency Φ_{RTC} :
{ Streaming Layer Sub-system:
((embarrassingly ||) RTC for Streaming Transactions) ***;***
Querying Layer Sub-system:
*(|| (intra ||) RTC for Querying Transactions/Logic)** ***;***
*(|| (intra ||) Synchronous Querying Logic)**
}
Subsystem B. asynchronously at own frequency:
{ Querying Layer Sub-system:
((embarrassingly ||) (intra ||) Querying Logic)
}

(Expr. 22: Streamonas Synchronous/Asynchronous Model)

As expressed by the Synchronous/Asynchronous model, the asynchronous querying sub-system is totally decoupled, running independently from the streaming process. As mentioned earlier in this section, this highly parallel process in the streaming environment is <u>not</u> constrained by large network latencies caused by query complexity, or available bandwidth at the streaming layer, or the requirement for completion of transfers of streamed data, as it accesses main memory communicating through a different network channel. Figure 5-7 presents the architecture where a pool of cores accesses asynchronously fragments of the database of spatio-temporal cuboids at a Node. Figure 5-6 demonstrates the overall cluster architecture with the synchronous real-time consistency sub-system running the Linear Road Benchmark with load of 550 Xways, while the asynchronous subsystem accesses the same database in a distributed manner, with an overall average rate of 45 million "read" operations per second.

6. Experimental Results

We have initially deployed the experimental testbed for both single-cpu/single-core (load of 10 Xways) and Distributed Streamonas (load of 550 Xways). For our experiments we used standard software and hardware on a Linux cluster. The Linux version was Red Hat Linux release 5.6 – CentOS. The respective Linux kernel was 2.6.18-238.19.1.e15. The cluster had 5 active nodes. Each one of the nodes had two quad-core processors providing 8 cores per node (Intel 3.1 GHz Xeon - X5460 with 6144 KB L2 cache). Each one of the nodes had 16 GBytes of RAM with one of them having 14 GBytes of RAM available. The cluster had a 1 Gbit/sec interconnecting network.

By gradually loading query modules, we have developed the QoS maps of figures 5-3 and 5-4. The indexes at the top of the figures provide abbreviations of the names of the queries. Detailed specifications of these queries can be found in the Linear Road Benchmark references [3, 15].

While running a load of 550 Xways, Distributed Streamonas demonstrated robustness in supporting large-scale enterprise level applications, by managing a data volume above 350 GBytes in less than 3 hours and an average backbone network data rate of almost 600 Mbits/sec, monitoring approximately 75 million evolving entities (cars) with an average query latency below 400 μsec (Figure 5-4; top line marked as (1)). More complete description of these results are being published separately with specifics about the architecture, but for this paper the important result is that Distributed Streamonas, with the synchronous Real-Time Consistency sub-system, was supporting 550 Xways on the Linear Road Benchmark.

We have performed lengthy experiments in order to evaluate the performance of the asynchronous subsystem for embarrassingly parallel querying. For this reason we developed a simple scenario where the Authorities of California through an independent secure network, query the database of ST-Cuboids, which collects information from 75 million cars from sensors on 550 expressways. The independent secure network presented in Figure 5-6 while assumed, was not implemented in this set of experiments. While the synchronous sub-system was running the benchmark, the asynchronous subsystem was performing queries in parallel. Each query was a double "read" request from the database of ST-Cuboids. Each double "read" request was retrieving latest as also historical information.

By using the architecture presented in Figures 5-6 and 5-7, 26 cores deployed over 5 nodes were devoted to the asynchronous query layer. Query requests were distributed to the cores that accessed in parallel the whole volume of the Distributed ST-Cuboid database supporting more than 75 million evolving entities, enabling embarrassingly parallel database query. The asynchronous sub-system, as shown in Figure 5-5, was able to serve on average more than 45 million "read" requests per second over the whole cluster. The figure also shows the linear scalability of the sub-system to the number of cores. A complete scan in a distributed manner of the whole database of 75 million cars took 1.44 seconds. Taking into consideration that the registered cars in year 2011 in California were 31 million, the Authorities of California would be able to receive a complete scan, picturing the status of the highway system, in approximately every 0.62 seconds.

7. Conclusion

The parallel, object-oriented ST-Cuboids data model of Streamonas, successfully managed to support the novel framework of Real-Time Consistency for data streams which enables 100% accurate results within time intervals of less than 400 µsec, even for large distributed enterprise applications where the in-memory distributed database reaches 65 GBytes. The framework achieves surprisingly good performance in the domain of data stream management, as it actually provides consistency rather than just trade-off accuracy to efficiency.

The innovative theoretical framework of Real-Time Consistency presented in this work is supported by Quality of Service maps that monitor the stability and consistency of the system. Through a multicore architecture, a novel asynchronous sub-system implements Real-Time Consistency. This asynchronous sub-system re-uses the data stored in the ST-Cuboid database in an embarrassingly parallel querying manner. As verified by extended experimentation with the LRB benchmark, this architecture permits it to attain extremely high levels of performance.

8. Conclusion of the Dissertation

In this dissertation we presented our novel Data Stream Management System with name "Streamonas". Through the design and development of the system, we have made significant contributions to the scientific area of data streams, as we developed novel internal database structures and indexing schemes suitable for the data stream environment; novel formal models for spatio-temporal logic; novel single-cpu architectural designs which guided the implementation and benchmarking of the system; as also novel distributed, multicore architectural designs which guided the implementation and benchmarking of the cluster system.

Both single-cpu Streamonas as also Distributed Streamonas architectures, have reached record-level performance when compared with state-of-the-art Data Stream Management Systems on the Linear Road Benchmark, a benchmark developed by M.I.T. , mainly in cooperation with Brandeis University, Brown University and Stanford University.

The contributions of the dissertation are presented below: (also listed for convenience in the respective section of Chapter One) :

1. The design, implementation and performance evaluation, of a novel, parallel, object-oriented data model with name Spatio-Temporal Cuboid, which transforms the serial incoming stream, arriving to a Data Stream Management System. Data are temporarily stored in the First-In-First-Out Spatio-Temporal Cuboid data structure, and are made available for query processing. A Spatio-Temporal Cuboid as its name suggests, represents a multi-dimensional slice of data in a spatio-temporal space.

   Through object-oriented expressions, Spatio-Temporal cuboids, enable the constant time complexity O(1) access of temporal sequences in their native form (named as atomic streams), as also the constant time complexity O(1) access of any element streamed into the Data Stream Management System (named as spatio-temporal elements), providing extremely fast performance to the system. This approach is very novel in introducing a layer of abstraction above streams, and this is unique among Data Stream Management Systems.

2. Novel formal models defining the space-time semantics upon which the Streamonas architecture is based, as also the fundamental temporal semantics for application development on the prototype system. These models are:
    a. The Fact Generation Model which models the continuous fact (tuple) generation by a system of sensors, streamed to the Data Stream Management System.
    b. The Query Result Generation Model, describing continuous result generation by the Data Stream Management System from continuous queries.
    c. The Semantic Space-Time model for Streaming, which decomposes streams based on a fundamental process for the system with the name Stream Semantic Decomposition.
    d. The Semantic Space-Time model for Querying, allowing multiple queries to execute in parallel.

3. The Data Stream Management System architecture design for a single-cpu system with the name "Streamonas", which decouples streaming from querying in two independent layers and uses Spatio-Temporal cuboid data structures following a database schema, to model the underlying database. The architecture (i) enables the reusability of the streamed information stored in the Spatio-Temporal 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.

4. A novel Distributed Data Stream Management System architecture for large-scale, enterprise level applications, designed and implemented on a multicore cluster environment, based on the principles of the single-cpu Streamonas. The Distributed Architecture has a Synchronous data-stream processing engine and enables both synchronous as also asynchronous query processing.

5. The enforcement of database consistency in real-time based on the Spatio-Temporal Cuboids data model providing 100% accuracy in data stream processing, along with in depth technical analysis.

An interesting recent focus for this work has been on its application in "Smart City" infrastructures, which automate key services in an integrated way. Recently Smart Cities have captured the interest of many leaders and multinational corporations as a path to restructuring the planet. We believe that the dissertation will influence the next generation of data stream management systems, especially those which will be designed to serve "Smart Cities" applications.

REFERENCES

1. D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. "Aurora: a new model and architecture for data stream management," *The VLDB Journal*, 2003.

2. D J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. "The design of the Borealis stream processing engine," *in Proc. of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, 2005.

3. A. Arasu, M.Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbets. "Linear Road: A Stream Data Management Benchmark," *in Proc. of the 30th Intl. Conference on Very Large Data Bases*, 2004.

4. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein and J. Widom. "STREAM: The Stanford Stream Data Manager," *in Proc.of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, 2003.

5. A. Arasu, S. Babu and J. Widom. "CQL: A language for continuous queries over streams and relations," *in 9th Intl. Workshop on Database Programming Languages*, 2003.

6. M. Balazinska, H. Balakrishnan and M. Stonebraker. "Load Management and High Availability in the Medusa Distributed Stream Processing System," *in Proc. of the 2004 ACM SIGMOD Intl. Conference on Management of Data*.

7. R. Barga, J. Goldstein, M. Ali and M. Hong. "Consistent Streaming Through Time: A Vision for Event Stream Processing," *in Proc. of the third Biennial Conference on Innovative Data Systems Research,* 2007, pages 363-374.

8. S. Chandrasekaran, O. Cooper, A. Seshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurhty, S. Madden, V. Raman, F. Reiss and M. Shah. "TelegraphCQ: Continuous dataflow processing for an uncertain world," *in Proc. of the 2005 Conference on Innovative Data Systems Research.*

9. J. Chen, D. DeWitt, F. Tian, Y. Wang. "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," *in Proc. Of the 2000 ACM SIGMOD Intl. Conference on Management of Data*, 2000.

10. C. Cranor, T. Johnson, O. Spataschek, V. Shkapenyuk. "Gigascope: A Stream Database for Network Applications," *in Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, pages 647-651, 2003.

11. Q. Chen, M. Hsu and H. Zeller."Experience in Continuous analytics as a Service (CaaaS)," *in Proc. EDBT 2011,* pages 509-514.

12. J. Dean, S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters," *in Proc. OSDI,* pages 137-150, 2004.

13. N. Jain, L. Amini, H. Andrade, R. King, Y. park, P. Selo, C. Venkatramani. "Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core," *in Proc. of the 2006 ACM SIGMOD Intl. Conference on Management of Data*, 2006.

14. V. Kumar, H. Andrade, B. Gedik, K.-L. Wu. "DEDUCE: At the Intersection of MapReduce and Stream Processing," *in Proc. EDBT 2010,* pages 657-662.

15. (2013) Linear Road – Home. [Online]. Available: http://www.cs.brandeis.edu/~linearroad/

16. J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos, T. Johnson, and D. Maier. "Out-of-Order Processing: A New Architecture for High-Performance Stream Systems," *in Proc. of the 34th VLDB Conference*, Auckland, New Zealand, 2008.

17. S. R. Madden, M. A. Shah, J. M. Hellerstein and V. Raman. "Continuously adaptive continuous queries over streams," *in Proc. of the 2002 ACM SIGMOD Intl. Conference on Management of Data*, 2003.

18. P. A. Michael, D. S. Parker. "Architectural Principles of the Streamonas Data Stream Management System and Performance Evaluation based on the Linear Road Benchmark," *in Proc. of the 2008 International Conference on Computer Science and Software Engineering (2008 CSSE)*, IEEE, Wuhan, China, 2008.

19. P. A. Michael, D. S. Parker. "The Semantic Space-time models of the Streamonas Data Stream Management System," *in Proc. of the 2009 World Congress on Computer Science and Information Engineering (2009 CSIE),* IEEE, Los Angeles / Anaheim, USA, 2009.

20. P. A. Michael, D. S. Parker. "Real-Time spatio-temporal data mining with the Streamonas Data Stream Management System," *in Proc. of the Tenth International Conference on Data Mining, Detection, Protection and Security,* Wessex Institute of Technology, Crete, Greece, 2009.

21. M. T. Özsu, P. Valduriez. *Principles of Distributed Database Systems*. Springer, third edition, 2011.

22. D. S. Parker, R. R. Muntz, H. L. Chau. "The Tangram stream query processing system," *in Proc. of the Fifth International Conference on Data Engineering*, 1989.

23. D. Terry, D. Goldberg, D. Nichols and B. Oki. "Continuous queries over append-only databases," *in Proc. of the 1992 ACM SIGMOD Intl. Conf. on Management of Data*, 1992.

24. H. Thakkar, B. Mozafari, and C. Zaniolo. "Designing an Inductive Data Stream Management System: the Stream Mill Experience," *in Proc. of the Second International Workshop on Scalable Stream Processing Systems*, Nantes, France, 2008.

25. H. Thakkar, N. Laptev, H. Mousavi, B. Mozafari, V. Russo, C. Zaniolo. "SMM: A Data Stream Management System for Knowledge Discovery," *in Proc of the 2011 IEEE 27th International Conference on Data Engineering, 2011.*

26. H. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker. "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters," *in Proc.of the 2007 ACM SIGMOD Intl. Conf. on Management of Data*, 2007.

27. C. Zaniolo, S.Ceri, C.Faloutsos, R.T. Snodgrass, V. S. Subrahmanian and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann, 1997.

28. E. Zeitler, T. Risch. “Scalable Splitting of Massive Data Streams,” *in Proc. DASFAA (2) 2010,* pages 184-198.

29. E. Zeitler and T. Risch. “Massive scale-out of expensive continuous queries,” *in Proc. VLDB Endow.,* 4: 1181-1188, 2011.