Processing high-volume stream queries on a supercomputer

Erik Zeitler and Tore Risch
Department of Information Technology, Uppsala University
{erik.zeitler,tore.risch}@it.uu.se

Abstract

Scientific instruments, such as radio telescopes, colliders, sensor networks, and simulators generate very high volumes of data streams that scientists analyze to detect and understand physical phenomena. The high data volume and the need for advanced computations on the streams require substantial hardware resources and scalable stream processing. We address these challenges by developing data stream management technology to support high-volume stream queries utilizing massively parallel computer hardware. We have developed a data stream management system prototype for state-of-the-art parallel hardware. The performance evaluation uses real measurement data from LOFAR, a radio telescope antenna array being developed in the Netherlands.

1. Background

LOFAR [13] is building a radio telescope using an array of 25,000 omni directional antenna receivers whose signals are digitized. These digital data streams will be combined in software into streams of astronomical data that no conventional radio telescopes have been able to provide earlier. Scientists perform computations on these data streams to gain more scientific insight.

The data streams arrive at the central processing facilities at a rate of several terabits per second, which is too high for the data to be saved on disk. Furthermore, expensive numerical computations need to be performed on the streams in real time to detect events as they occur. For these data intensive computations, LOFAR utilizes an IBM BlueGene supercomputer and conventional clusters.

High-volume streaming data, together with the fact that several users wanting to perform analyses suggests the use of a data stream management system (DSMS) [9]. We are implementing such a DSMS called SCSQ (Super Computer Stream Query processor, pronounced *cis-queue*), running on the BlueGene computer. SCSQ scales by dynamically incorporating more computational resources as the amount of data grows. Once activated, continuous queries (CQs) filter and transform the streams to identify events and reduce data volumes of the result streams delivered in real time. The area of stream data management has gained a lot of interest from the database research community recently [1] [8] [14]. An important application area for stream-oriented databases is that of sensor networks where data from large numbers of small sensors are collected and queried in real time [21] [22]. The LOFAR antenna array will be the largest sensor network in the world. In difference to conventional sensor networks where each sensor produces a limited amount of very simple data, the data volume produced from each LOFAR receiver is very large.

Thus, DSMS technology needs to be improved to meet the demands of this environment and to utilize state-of-the-art hardware. Our application requires support for computationally expensive continuous queries over data streams of very high volumes. These queries need to execute efficiently on new types of hardware in a heterogeneous environment.

2. Research problem

A number of research issues are raised when investigating how new hardware developments like the BlueGene massively parallel computer can be optimally utilized for processing continuous queries over high-volume data streams. For example, we ask the following questions:

1. How is the scalability of the continuous query execution ensured for large stream data volumes and many stream sources? New query execution strategies need to be developed and evaluated.
2. How should expensive user-defined computations, and models to distribute these, be included without compromising the scalability? The query execution strategies need to include not only communication but also computation time.
3. How does the chosen hardware environment influence the DSMS architecture and its algorithms? The BlueGene CPUs are relatively slow while the

communication is fast. This influences query distribution.

4. How can the communication subsystems be utilized optimally? The communication between different CPUs depends on network topology and the load of each individual CPU. This also influences query distribution.

3. Our approach

To answer the above research questions we are developing a SCSQ prototype. We analyze the performance characteristics of the prototype system in the target hardware environment in order to make further design choices and modifications. The analyses are based on a benchmark using real and simulated LOFAR data, as well as test queries that reflect typical use scenarios. These experiments provide test cases for prototype implementation and system re-design. In particular, performance measurements provide a basis for designing a system that is more scalable than previous solutions on standard hardware.

The CQs are specified declaratively in a query language similar to SQL, extended with streaming and vector processing operators. Vector processing operators are needed in the query language since our application requires extensive numerical computations over high-volume streams of vectors of measurement data. The queries involve stream theta joins over vectors applying non-trivial numerical vector computations as join criteria. To filter and transform streams before merging and joining them, the system supports sub-queries parameterized by stream identifiers. These sub-queries execute in parallel on different nodes.

A particular problem is how to optimize high-volume stream queries in the target parallel and heterogeneous hardware environment, consisting of BlueGene compute nodes communicating with conventional shared-nothing Linux clusters. Pre- and post-processing computations are done on the Linux clusters, while parallelizable computations are likely to be more efficient on the BlueGene. The distribution of the processing should be automatically optimized over all available hardware resources. When several different nodes are involved in the execution of a stream query, properties of the different communication mechanisms (TCP, UDP, MPI) substantially influence the query execution performance.

4. The hardware environment

Figure 1 illustrates the stream dataflow in the target hardware environment. The users interact with SCSQ on a Linux *front cluster* where they specify CQs. The *input streams* from the antennas are first pre-processed according to the user CQs in the Linux *back-end cluster*. Next, *BlueGene* processes the CQs over these pre-processed streams. The output streams from BlueGene are then post-processed in the front cluster and the result stream is finally delivered to the user. Thus, three parallel computers are involved and it is up to SCSQ to transparently and optimally distribute the stream processing between these.

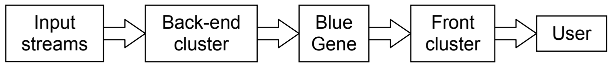

Figure 1. Stream data flow in the target hardware environment.

The hardware components have different architectures. The BlueGene features dual PowerPC 440d 700MHz (5.6 Gflops max) *compute nodes* connected by a 1.4 Gbps 3D torus network, and a 2.8 Gbps tree network [3]. Each compute node has a local 512 MB memory. The compute nodes run the *compute node kernel* (CNK) OS, a simple single-threaded operating system that provides a subset of UNIX functionality. Each compute node has two processors, of which normally one is used for computation and the other one for communication with other compute nodes. MPI is used for communication between BlueGene compute nodes, whereas communication with the Linux clusters utilizes *I/O nodes* that provide TCP or UDP. One important limitation of CNK is the lack of support for server functionality (no listen(), accept() or select() are implemented). Furthermore, two-way communication is expensive and should be avoided for time-critical code. Each I/O-node is equipped with a 1 Gbit/s network interface. In LOFAR's BlueGene, there are 6144 dual processor compute nodes, grouped in processing sets, or *psets*, consisting of 8 compute nodes and one I/O node. This I/O-rich configuration enables high volumes of incoming and outgoing data streams.

The Linux front and back-end clusters are IBM JS20 computers with dual PowerPC 970 2.2GHz processors.

5. The SCSQ system

Figure 2 illustrates the architecture of the SCSQ components running on the different clusters.

On the front cluster, the user application interacts with

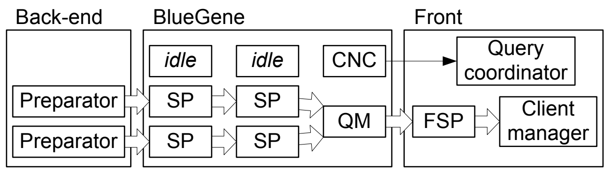

Figure 2. The SCSQ components. Double arrows indicate data streams.

a SCSQ *client manager*. The client manager is responsible for i) interacting with the user application, ii) sending CQs and meta-data, such as client manager identification, to the *query coordinator* for compilation.

The query coordinator is responsible for i) compiling incoming CQs from client managers, ii) starting one or more *front stream processors* (FSP) to do the post processing of the streams from the BlueGene, and iii) posting instructions to the BlueGene components for execution of CQs. When the query coordinator receives a new CQ from a client manager, the query coordinator initiates new FSPs for post-processing of that CQ. It also maintains a request queue of CQs and other instructions to be processed by the BlueGene. This queue is regularly polled by the BlueGene *compute node coordinator* (CNC) (single arrow in Figure 2).

The CNC is responsible for i) retrieving new CQs and instructions from the query coordinator, ii) assigning and coordinating *stream processors* on the compute nodes, and iii) monitoring the execution of all stream processors. The BlueGene processors to be used by SCSQ are initiated once when the system is set up. The CNC is always executing on a single node while all other nodes are stream processors waiting for instructions from the CNC. When the CNC retrieves a new CQ, it assigns one idle stream processor to be the new *query master* for that query.

A query master is responsible for i) compiling and executing its stream query, ii) delivering the result to an FSP on the front cluster previously initiated by the query coordinator, iii) starting new stream processors of subqueries if needed, iv) communicating with the back-end cluster to retrieve input data, and v) monitoring the execution of its stream query. When a query master receives a CQ it is compiled and then the execution is started. If the query master determines that additional stream processors are needed for some stream subqueries, it dynamically requests the CNC to assign new ones. The query master then sends the subqueries to the new stream processors for execution. Each stream processor may in turn start new subqueries when so required. Stream queries may be terminated either by explicit user intervention or by some stop condition in the query. Therefore, the stream processors also exchange control messages to initialize and terminate stream queries. Control messages are also used to regulate the stream flow between the processors.

The only difference between a stream processor and a query master is that the query master delivers its result to an FSP in the front cluster using TCP, while a stream processor delivers its result stream through MPI to the stream processor or query master that initiated it.

Nodes participating in the processing of a stream are called *working nodes*. Stream processors, query masters, and FSPs are all working nodes.

When a working node needs measurements from an input stream it initiates TCP communication for that stream through its *preparator*. A preparator is a working node running on the back-end cluster wrapping one or more input streams.

The set-up of a stream query generates a distributed query execution tree, as illustrated by the double arrows in Figure 2.

We have implemented the first SCSQ prototype and are evaluating it. All BlueGene and front node functionality for execution of single user queries have been implemented. We have used this implementation to analyze bandwidth properties of the I/O nodes and strategies for efficient buffering in the MPI and TCP communication subsystems.

The implementation of SCSQ nodes is based on Amos II (Active Mediator Object System) [18] [19], which is modified to allow execution of continuous queries over streams in the target hardware environments. The SCSQ modules are extensible by linking user-defined functions written in compiled C. On the front and back-end clusters, dynamic linking is allowed. However, only static linking is allowed on BlueGene. As a consequence, all user- defined stream operators written in C must be statically linked with the stream processor executable for the BlueGene. To configure dynamically the stream processors at run-time we utilize a built-in Lisp interpreter to communicate code between the front cluster and the BlueGene. All time-critical code running on the BlueGene is written in C and statically linked.

6. Related work

The SCSQ implementation is related to research in DSMSs, parallel and distributed databases, continuous query processing, and database technology for scientific applications.

A promising approach to achieve the high performance, flexibility, and expressiveness required is to develop a distributed DSMS running on highly connected clusters of main memory nodes [2] [7] [12], which is extensible through user-defined data representations and computational models [10]. Most of the DSMS, e.g. [6] [8] [14] [15] [20], are designed for rather small data items and a relatively small cost of the stream operators per item. In contrast, SCSQ is intended for a very high total stream volume, large data item sizes, and computationally expensive scientific operators and filters.

The use of extensible database technology where database queries call user-defined functions in the database engine have been shown very useful for astronomical applications [17]. Parallelization of user-defined functions has been studied in [16].

Distributed execution of expensive user-defined stream query functions has been studied in the recently proposed Grid Stream Data Manager (GSDM) [10] [11], an object-relational DSMS for scalable scientific stream query processing. GSDM features a framework for pre-defined and customized parallelization schemes, which distribute the execution of user-defined stream query functions over the Grid. Like SCSQ, GSDM is intended for scalable on-line analysis using expensive user-defined stream query functions over high-volume scientific data streams from instruments and simulations.

However, unlike all other DSMSs, SCSQ will be optimized for a heterogeneous target hardware environment including a BlueGene supercomputer.

7. Ongoing work

Query execution scalability is achieved by developing query processing strategies able to utilize an increasing number of compute nodes while optimally utilizing the communication facilities.

To generate local query execution plans on each stream processor we employ query optimization strategies based on heuristics and a simple cost model.

Queries are distributed based on the need to execute sub-queries in parallel. Currently, each stream processor can execute only one sub-query. Any stream processor can at run-time request idle stream processors from the CNC to execute sub-queries. This allows dynamic reconfiguration of the distributed query execution plan.

The performance monitoring subsystem in each stream processor measures the performance of different phases of stream query execution. It is currently used to evaluate the characteristics of different execution strategies. However, the same mechanism will also be used to optimize the stream query distribution itself. Since our system allows dynamic reassignment of stream processors we will use the performance monitoring subsystem for adaptive run-time query re-optimization. This is necessary since sudden bursts in the measured signals may require execution plans to be dynamically reconfigured.

To analyze the system and understand the issues that are relevant to the LOFAR application we are developing a benchmark. The benchmark includes real and simulated data as well as queries from the radio astronomy application domain. We are initially concentrating on queries that detect transients among a large number of incoming streams. We scale the number of incoming streams and optimize throughput and latency as the data volume grows. Therefore, we scale not only the data volume but also the computation time in our experiments.

A stream oriented communication protocol between stream processors is developed based on MPI. We measure the characteristics for different communication methods between the stream processors. The communication latency and bandwidth depend on the topology and the load of the nodes. For example, nodes far apart have long latency but may have a high bandwidth, since there are many communication links between them that can be used in parallel. On the other hand, highly loaded intermediate nodes slow down communication [5]. These characteristics will influence query decomposition and distribution.

The query execution performance depends on the utilization of each stream processor. The utilization of a stream processor depends on the relation between its stream rate and computational load. Each stream processor is buffering its incoming and outgoing streams. The buffer utilization of a stream processor indicates the load balance between communication and processing. Each stream processor monitors its buffer utilization and adapts the flow rate by sending control messages regularly. In an overflow situation, different policies can be devised, for example: load shedding by dropping incoming data [23], simplifying aggregation operators [4], sending control messages that slow down sub query stream processors, or asking CNC for more stream processors.

It is also important to analyze the performance of queries involving expensive operators. We investigate the scalability over large numbers of high-volume input streams that are merged by computationally expensive stream combination functions from the benchmark. The goal is to understand how to distribute the streams and computations optimally in the heterogeneous target hardware environment.

Acknowledgements

This work is supported by LOFAR.

References

[1] Daniel J. Abadi et al, “Aurora: a new model and architecture for data stream management”, *The VLDB Journal*, Springer, 12(2) 2003, pp 120–139.

[2] Daniel J. Abadi et al, “The Design of the Borealis Stream Processing Engine”, in *The Second Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA 2005, pp 277–289.

[3] George Almási et al, “Implementing MPI on the BlueGene/L Supercomputer”, *Lecture Notes in Computer Science, Volume 3149*, Jan 2004, pp 833–845.

[4] Brian Babcock, Mayur Datar, Rajeev Motwani, “Load Shedding for Aggregation Queries over Data Streams”, in *Proc. of the International Conference on Data*

Engineering (ICDE 2004), Boston, USA, pp 350–361.

[5] Gyan Bhanot et al, "Optimizing task layout on the Blue Gene/L supercomputer", *IBM Journal of Research and Development*, Volume 49, Number 2/3, 2005, pp 489–500.

[6] Donald Carney et al, "Monitoring Streams – A New Class of Data Management Applications", in *Proc. Of the 28th Int'l Conf. on Very Large Databases (VLDB'02)*, Hong Kong, China, 2002, pp 215–226.

[7] Mitch Cherniack et al, "Scalable distributed stream processing", in *The First Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA 2003.

[8] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk, "Gigascope: A Stream Database for Network Applications", in *Proc. Of the ACM SIGMOD Conference on Management of Data*, San Diego, CA 2003, pp 647–651.

[9] Lukasz Golab and M. Tamer Özsu, "Issues in data stream management", *SIGMOD Record*, 32(2), 2003, pp 5–14.

[10] Milena Ivanova and Tore Risch, "Customizable Parallel Execution of Scientific Stream Queries", in *Proc. Of the 31st Int'l Conf. on Very Large Databases (VLDB'05)*, Trondheim, Norway 2005, pp 157–168.

[11] Milena Ivanova, "Scalable Scientific Stream Query Processing", in *Uppsala Dissertations from the Faculty of Science and Technology 66*, Acta Universitatis Upsaliensis, Uppsala 2005, http://user.it.uu.se/~udbl/Theses/MilenaIvanovaPhD.pdf.

[12] Bin Liu et al, "A Dynamically Adaptive Distributed System for Processing Complex Continuous Queries", in *Proc. Of the 31st Int'l Conf. on Very Large Databases (VLDB'05)*, 2005, pp 1338–1341.

[13] LOFAR, http://www.lofar.nl/.

[14] Samuel Madden, Mehul A. Shah, Joseph M. Hellerstein, and Vijayshankar Raman, "Continuously adaptive continuous queries over streams", in *Proc. Of the ACM SIGMOD Conference on Management of Data*, Madison, Wisconsin 2002, pp 49–60.

[15] Rajeev Motwani et al, "Query processing, approximation, and resource management in a data stream management system", in *The First Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA 2003.

[16] Kenneth W. Ng and Richard R. Muntz, "Parallelizing user-defined functions in distributed object-relational DBMS", in *International Database Engineering and Applications Symposium (IDEAS)*, Montreal, Canada 1999, pp 442–450.

[17] María A. Nieto-Santisteban et al, "When Database Systems Meet the Grid", in *The Second Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA 2005, pp 154–161.

[18] Tore Risch and Vanja Josifovski, "Distributed Data Integration by Object-Oriented Mediator Servers", in *Concurrency and Computation: Practice and Experience J.* 13(11), John Wiley & Sons, September 2001, pp 933–953.

[19] Tore Risch, Vanja Josifovski, and Timour Katchaounov,

[20] Elke A. Rundensteiner et al, "CAPE: A Constraint-Aware Adaptive Stream Processing Engine", in Nauman Chaudhry, Kevin Shaw, and Mahdi Abdelguerfi (eds.): *Stream Data Management*, Advances in Database Systems Series, Springer 2005, pp 83–111.

[21] Special Section on Sensor Network Technology and Sensor Data Management, *SIGMOD Record*, 32(4), December 2003.

[22] Special Section on Sensor Network Technology and Sensor Data Management (Part II), *SIGMOD Record*, 31(1), March 2004.

[23] Nesime Tatbul, et al, "Load Shedding in a Data Stream Manager", in *Proc. Of the 29th Int'l Conf. on Very Large Databases (VLDB'03)*, Berlin, Germany, pp 309–320.