A Dynamically Adaptive Distributed System for Processing Complex Continuous Queries

Bin Liu, Yali Zhu, Mariana Jbantova, Bradley Momberger and Elke A. Rundensteiner

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280, USA
{(binliu | yaliz | jbantova | bmombe1 | rundenst)}@cs.wpi.edu

1 Introduction

Recent years have witnessed rapidly growing research attention on continuous query processing over streams [2, 3]. A continuous query system can easily run out of resources in case of large amount of input stream data. Distributed continuous query processing over a shared nothing architecture, i.e., a cluster of machines, has been recognized as a scalable method to solve this problem [2, 8, 9]. Due to the lack of initial cost information and the fluctuating nature of the streaming data, uneven workload among machines may occur and this may impair the benefits of distributed processing. Thus *dynamic adaptation* techniques are crucial for a distributed continuous query system.

Dynamic adaptation in a distributed system usually corresponds to load balancing at run time by moving certain workload across machines. In existing distributed continuous query systems such as [2, 8], the basic unit being moved during the adaptation is one whole operator, assuming that each operator is small enough to fit on one machine. We refer to this adaptation as *operator-level adaptation*. Many operators in continuous queries need *states* to keep tuples they have received so far for future processing. In case of high stream workloads, the states in one operator can grow too large to fit in the main memory of a single machine. Moreover, moving around large amounts of states at run time can be inefficient.

The Flux approach [9] addresses this problem by proposing strategies to divide one large operator state into many smaller partitions. One partition can then be treated as one moving unit during runtime adaptation. We refer to this type of adaptation as *state-level adaptation*. However, the Flux approach so far has only dealt with the relatively simple cases of stateful operators with one input, namely, aggregate operators. It has not considered the more complex cases of stateful operators with multiple inputs and multiple states, such as binary or multi-way joins, which are more likely to have bigger operator states.

The *D-CAPE* system to be demonstrated is a general-purpose distributed system for continuous query processing. In particular, the system is able to apply dynamic load balancing to the processing of complex continuous queries that contain stateful operators at both the operator-level and the state-level, with no restrictions on the size of operator states. To the best of our knowledge, the D-CAPE system is the first distributed continuous query system to manage state-level adaptation for complex multiple input operators, such as joins. Various join predicates among the input streams make this adaptation a rather challenging problem. We have developed several adaptation strategies to address this problem.

Dynamic load balancing, however, can be applied only when spare processing resources exist in the distributed system. Although a distributed system has higher scalability than a centralized system, the overall system resource summed over all participating machines is still limited. To further increase the robustness of the distributed system, D-CAPE explores two other adaptation techniques, namely data spilling and query plan shape changing across multiple machines, to reduce the resources required by query processing at run time.

The *data spilling technique* can dynamically choose certain partitions of the operator states to push to disk to temporarily lessen the burden of the query processing. This is especially useful when the system encounters sudden increase of workload and quickly runs out of processing resources. Accordingly, data recovering techniques also need to be applied in a timely manner to recover missing results due to such data spilling.

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

The *query plan shape changing across multiple machines* can also decrease the overall query processing cost. As a unique feature in the D-CAPE system, we are the first to explore such cross-machine plan-level adaptation opportunities to increase the efficiency of the distributed system. The shape of the query plans distributed across a cluster is usually assumed to stay unchanged during its execution. However, as the stream characteristics may fluctuate over time, sometimes it is beneficial to modify the shape of the query plan across a cluster. For example, if multiple joins in a query plan are being executed on more than one machine in the cluster, dynamically changing the order of these joins based on their selectivities can reduce the runtime processing cost.

Our demo uses a unique application domain of *Fire Protection Engineering* (FPE) that engages in fire modeling, monitoring, and prediction. We have an on-going collaboration with the FPE Department at Worcester Polytechnic Institute (WPI) [1]. To solve firesafety problems in our modern age, FPE researchers construct sophisticated computer models about complex phenomena of fire spread in structural systems. Empirical measurements are also being made by executing and monitoring live experiments, in either laboratory facilities, such as the WPI Fire Science Laboratory, or in rare times in controlled real structures. Such empirical measurements are essential to ensuring accurate models. During live experiments, sensors are placed at key locations in controlled structures and stream data back to the computer host. Such data often has highly unpredictable characteristics. It can include information such as temperature, humidity, smoke dynamics and water suppression. Our distributed query engine allows the FPE researchers to submit online continuous queries and analyze the experimental stream data at runtime, comparing actual phenomena against the predicted simulation model even under very high stream workload.

2 System Architecture

The overall architecture of the *D-CAPE* system is described in Figure 2. We refer to each machine in the cluster as one *CAPE processor* [8], or one *processor* for short. A dedicated *distribution manager* is deployed to control a set of *CAPE* processors.

Each *CAPE* processor has a central *continuous query processing engine* responsible for executing operators that are activated in this processor. The *data receiver* receives tuples from stream sources or other processors, feeding them into the right operators in the processor. The *data distributor*, on the other hand, sends output tuples to operators in different processors. The *local statistics gatherer* continuously collects statistics for the current processor. This information will be used to make local adaptation decisions as well as reported to the *distribution manager* for the adaptation across processors.

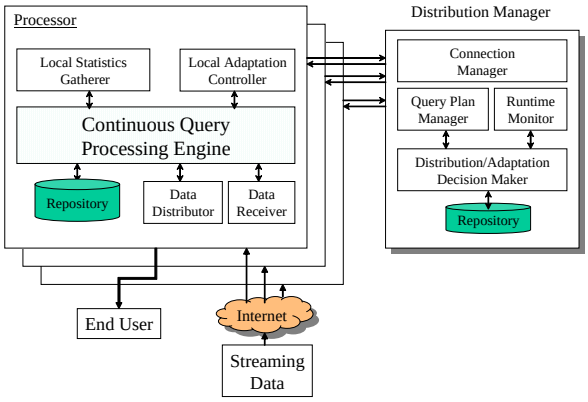

Figure 1: D-CAPE Framework Architecture

The *distribution manager* controls a set of CAPE processors. It is in charge of 1) the initial distribution, 2) connecting query plans that are distributed over multiple processors, 3) collecting statistics about each processor, and 4) making adaptation decisions based on adaptation policies. The distribution manager is designed to be light-weight. Thus one distribution manager is able to manage a large number of processors.

3 System Highlights

3.1 Initial Distribution

An effective initial distribution can set a good starting point and have positive influence on the characteristics of the adaptation techniques to be applied at runtime. However, at this stage no runtime statistics regarding the state of the distributed system and the cost of operators are available. In the D-CAPE system, to have a comprehensive estimation of the statistics, the initial distribution utilizes many cost factors including the number and the type of query operators, the size of the window constraints, and the number of available processors. The initial distribution is a dual-level algorithm that operates at both the operator level and the state level. It divides operators into groups and distribute these groups across the cluster. An operator with large windows (states) is divided and placed into multiple groups, and distributed to multiple processors. The initial distribution algorithms aim to be both *workload-aware*, to equalize the number of operators and states across processors, and *network-aware*, to reduce the number of connections across processors.

3.2 Operator-level Adaptation

When the query operators have small or no states, the runtime adaptation can be applied at the operator level. That is, an operator as a whole or even multiple operators are moved across processors.

Several cost measurements are used to determine overloaded and underloaded processors. For example, the load can be measured based on the output rate at which an operator sends tuples across the network. The load can also be estimated by the sum of the costs of all active operators on that processor. The memory consumption can be a good indicator of load as well. Experiments in our prototype system show that the number of network connections (network costs) in a processor is often a non-trivial cost factor. Figure 2 illustrates the CPU costs incurred when the number of network connections increases. As we can see, the network connections play an important role in the overall performance of the system. In the D-CAPE system, this cost factor affects the design of both initial distribution algorithms and runtime adaptation policies.

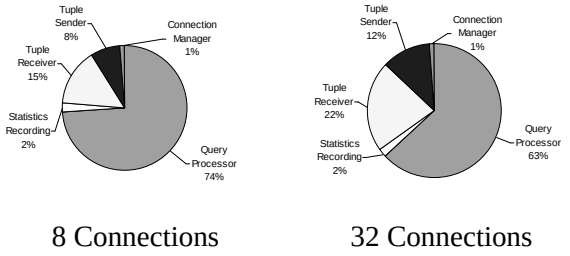

Figure 2: CPU Utilization/Network Connections

Redistribution policies are designed based on the combinations of these cost measurements. For example, the *degradation based* policy, one of the redistribution policies in the system, aims to alleviate the loads on processors that have shown an increase in the cost. If the increase is beyond a certain threshold, the cost is lessened by moving the most costly operators to other processors. During the redistribution decision phase, higher preference is given to operators that by moving them can reduce the overall number of network connections in the system.

3.3 State-level Adaptation

Partitioned parallelism [5, 7] is applied to query operators with large states accumulated at run-time. By using this strategy, each operator will be run on multiple processors with each operator instance working only on a portion of whole data streams. We here use an m-way symmetric hash join as an example to illustrate the basic ideas of state-level adaptation.

Adapting States Across Processors. Our system utilizes the partitioned parallel processing as used in Volcano [4] and Flux [9]. Input streams of operators are partitioned into many smaller partitions and a number of partitions are assigned to each operator instance. At runtime, the adaptation algorithms can choose certain partitions to move across the machines if necessary. For example, Figure 3 illustrates a 3-way join that is processed over three *CAPE* processors. In this case, each input stream is partitioned into 500 partitions, which then distributed to 3 processors. Here, each processor has join states from different input streams with the same partition ID. At runtime, our mechanism is to choose all the states with the same partition ID as a whole unit to move. This avoids joins across multiple processors. For example, if we only move A_1 from m_1 to m_2, then the newly incoming tuples to partition A_1 would have to probe B_1 which is now located on another machine.

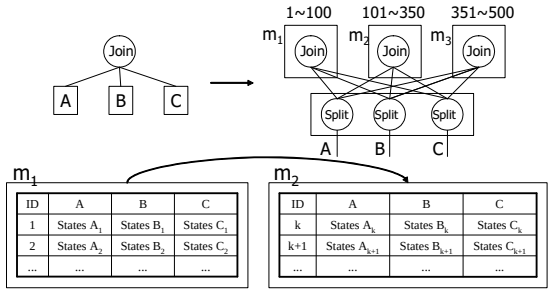

Figure 3: Partitioned Mway-Join Example

State Spilling and Recovering. However, the overall resources of a distributed system are still limited. Adapting states across processors by itself may not solve the problem of overall system resource shortage. Thus, the D-CAPE system is also designed to be able to temporarily spill state partitions into the local disk of the processor. This allows the system to react to overall resource shortage immediately. States that have been spilled become inactive. Thus, new inputs to the operator are processed based on the partial states (the main memory resident part) only. Given that, a state recovering process is necessary to merge the memory resident and the disk resident states when resources become available. In the recovery process, all the missing results are generated. For example, as shown in Figure 4, processor m_1 pushes state partitions into the local disk when main memory overflows. These states will be recovered later when free memory on m_1 becomes available.

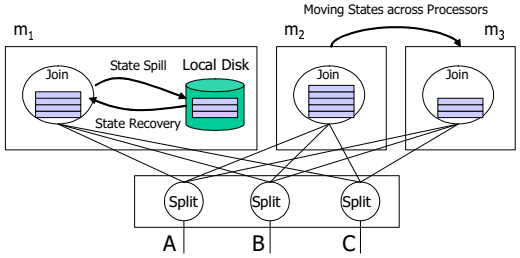

Figure 4: State Spill and Recovery

Our D-CAPE employs both state adapting across machines and state spilling into local disks, as illus-

trated in Figure 4. We also design multiple strategies for integrating both types of adaptations to maximally utilize the overall resources in the distributed system.

3.4 Run-time Query Plan Adaptation

Query plan shapes may also need to be adapted due to variations in input streams. In addition to traditional query plan adaptation [6] which reorders operators in query plans, we explore the adaption of merging and breaking operators so to switch between binary joins (trees) and m-way joins (trees). Figure 5(a) shows a binary join tree with each join allocated to one machine. Here, the letters AB, I_1C, and I_2D represent the operator states that have to be stored in the join operators, while I_1 and I_2 denote the intermediate results, and m_1, m_2, and m_3 represent available machines. As shown in Figure 5(b), we would merge $Join_1$ and $Join_2$ into a 3-way join $Join_{12}$ if we observe that large intermediate results are transferred from m_1 to m_2 and then stored in m_2. After that, state ABC becomes the only state that needs to be stored. However, the state ABC may no longer fit into one machine. Thus, we need to partition the state into multiple machines as shown in Figure 5(c) [1].

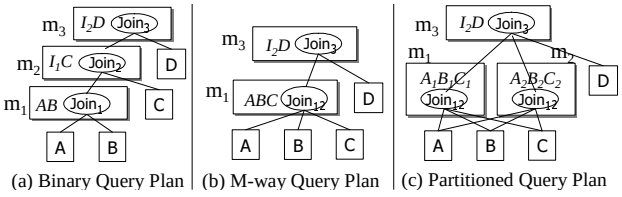

Figure 5: Adapt Partitioned Query Plan

4 Demonstration Focus

In this demonstration, we focus on showing the following aspects of the D-CAPE system:

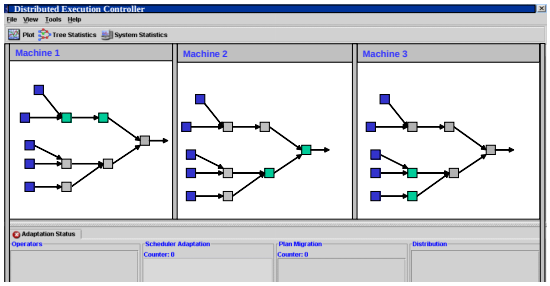

Figure 6: GUI for the Distribution Manager

Overall System. We have developed two separate GUIs to show the status of the processors and the distribution manager respectively. The processor GUI displays the operators that are currently running on this processor and the processor's cost. The distribution manager GUI, as partly shown in Figure 6, illustrates the overall framework. It shows the processors in the cluster and their runtime statistics.

Initial Distribution. We will show the impact of different initial distribution algorithms, such as workload aware and network-aware distributions, on the overall performance.

Dual-level Adaptations. We will show the performances of the run-time adaptations at the operator-level, the state-level and hybrid adaptation at both levels. For the operator-level adaption, we show the performance impact of different redistribution policies, such as the degradation-based policy and the balanced redistribution policy. As for the state-level adaptation, we show the tradeoffs between two adaptation methods, and how these two adaptations can be integrated to improve the query processing performance.

Runtime Query Plan Adaptation. We will show that adapting the shape of the query plan boosts the system performance. For example, in some cases breaking one m-way join into several smaller m-way or binary join operators can add flexibility to the redistribution procedure. In other cases several binary joins can be merged to one m-way join so to decrease the overall memory consumptions, as shown in Figure 5.

[1] Query plan migration techniques [10] are applied to switch plans without missing or corrupted query results.

References

[1] WPI Department of Fire Protection Engineering. http://www.wpi.edu/Academics/Depts/Fire/.

[2] D. Abadi, Y. Ahmad, and et. al. The design of the borealis stream processing engine. In *CIDR*, 2005.

[3] B. Babcock, S. Babu, and et. al. Models and issues in data stream systems. In *ACM PODS*, pages 1–16, 2002.

[4] G. Graefe. Encapsulation of parallelism in the volcano query processing system. In *ACM SIGMOD*, pages 102–111, 1990.

[5] W. Hasan. *Optimization of SQL Queries for Parallel Machines*. PhD thesis, Stanford University, Dec 1995.

[6] J. M. Hellerstein, M. J. Franklin, and et. al. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2):7–18, 2000.

[7] B. Liu and E. A. Rundensteiner. Revisiting pipelined parallelism in multi-join query processing. In *VLDB*, page to appear, 2005.

[8] E. A. Rundensteiner, L. Ding, and et. al. Continuous query engine with heterogeneous-grained adaptivity. In *VLDB Demo Session*, pages 1353–1356, 2004.

[9] M. A. Shah, J. M. Hellerstein, and et. al. Flux: An adaptive partitioning operator for continuous query systmes. In *ICDE*, pages 25–36, 2003.

[10] Y. Zhu, E. A. Rundensteiner, and G. T. Heineman. Dynamic plan migration for continuous queries over data streams. In *ACM SIGMOD*, pages 431–442, 2004.