D-CAPE: Distributed and Self-Tuned Continuous Query Processing

Timothy M. Sutherland,Bin Liu, Mariana Jbantova, Elke A. Rundensteiner
Computer Science Department, Worcester Polytechnic Institute,Worcester, MA 01609
(tims | jbantova | binliu | rundenst)@cs.wpi.edu

Categories and Subject Descriptors

H.2 [**DATABASE MANAGEMENT Systems.**]: Distributed Databases

General Terms

Design, Performance

Keywords

Distributed Continuous Query Processing,
Distributed Stream Query Engine. Self-Tuning.

1. INTRODUCTION

Efficient continuous query processing is critical for many applications, including monitoring remote sensors network traffic management, and online transaction processing. To overcome resource limitations and achieve real-time responsiveness, continuous query systems research focuses on issues such as load shedding, punctuation-driven operator-state purging [4], operator scheduling [7] and plan optimization and on-line migration [9].

However, when real-time yet accurate results are critical such as in stock market analysis, then there is a limitation to what any of the above optimization techniques can accomplish. Hence, distributed processing must be applied to support required scalability. It has been already proven in traditional database systems [3] that distributed processing results in high scale-up and speed-up capabilities due to aggregated resources. The design of a distributed continuous query system is characterized by an extra complexity, considered in the design of our D-Cape engine. In such a system, data streams may be infinite and initial cost statistics about the data streams are typically unknown. Moreover, cost statistics continue to change over time.

Our research on distributed stream processing addresses two critical questions: (1) How to initially distribute query plans given little or possibly no cost information, and (2) How to efficiently adapt the query distribution corresponding to runtime environmental changes. Even though research is now under way in designing distributed continuous query systems [2], [1], results to date, as far as we know, are based on simulations. Our work offers empirical results of distributed continuous query processing using an actual software system [5], [6].

D-Cape, a distributed continuous query processing architecture, employs stream query engines over a cluster of shared-nothing processors. We employ a dedicated *distribution manager* to manage the distribution, monitoring, and runtime redistribution of query plans, thus separating the control responsibility from the actual query processing task. Unlike Aurora*/Medusa [2], [1] which focus on research issues of a large area network, D-Cape targets a local cluster environment connected with a high speed network. However, D-Cape's controllers can be multi-tiered such that we can have multiple controllers, each controlling a cluster.Our experiments illustrate that D-Cape's design is light-weight, yet effective.

Copyright is held by the author/owner.
CIKM'05, October 31–November 5, 2005,Bremen,Germany.
ACM 1-59593-140-6/05/0010.

Contributions. Contributions of this work include:

- A well-designed distributed architecture D-CAPE for effective query distribution, light-weight monitoring, and efficient run time redistribution of continuous queries.
- Assuming no initial cost statistics, a *balanced network-aware* algorithm is introduced that incorporates query plan topology and workload concerns into the initial distribution decision.
- In addition, D-CAPE offers a *degradation-based* redistribution policy that is effective at run-time redistribution based on observed throughput of query operators and machine loads.
- Our work is among the first to report experimental measurements on an actual distributed continuous query processing software system (not a simulation).

2. SOFTWARE ARCHITECTURE

Each machine in D-CAPE has a CAPE query processor [6] installed, which performs the query processing tasks (Figure 1). A set of query processors is managed by a dedicated *distribution manager* (Figure 2).

Each **CAPE Query Processor** is composed of seven modules. The *execution engine* oversees the execution of query plans based on information from *statistics gatherer*and decisions by the scheduler. The *stream feeder* is a separate thread responsible for taking tuples received by the *stream receiver* and placing them in the proper input queues of operators. The *stream distributor* sends tuples to the next query processor or to an end-user application. The *connection manager*, the interface between the *query processor* and the *distribution manager*, handles requests such as activating operators on a processor, or sending the current status of the processor to the distribution manager.

The Distribution Manager synchronizes the management of the installation and execution of query plans across the computing cluster. The *runtime monitor* listens for statistical updates from each query processor. The *connection manager* executes the connection protocol to establish remote connections between every pair of

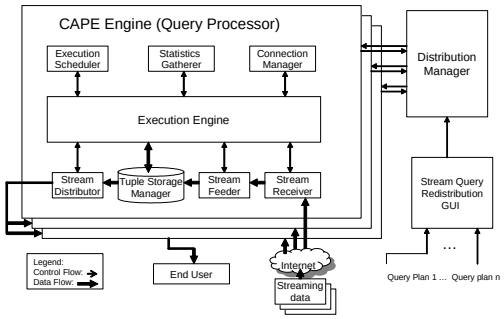

Figure 1: Query Processor Architecture

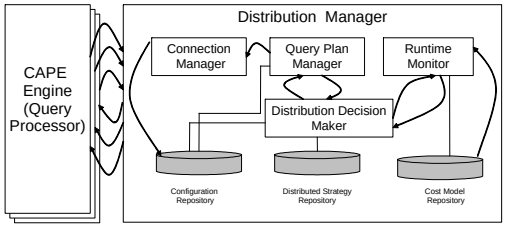

Figure 2: Distribution Manager Architecture

operators assigned to different query processors. The *distribution decision maker* decides how to distribute the query plans. There are two phases to this decision. First, an initial distribution is created at startup when only limited cost statistics about the query plans are known. Then, at run-time, query operators are redistributed to other query processors.

3. INITIAL QUERY PLAN DISTRIBUTION

Query plan distribution in D-CAPE is defined as the initial deployment of query plans across a set of query processors. Since little or even no cost statistics about data streams and query operators can be assumed, we make use only of the definition of queries to be processed and the number of available query processors. As our experiments show, the initial distribution significantly influences the overall performance [8]. Some distributions, if not carefully designed, will not always increase performance beyond that of a single query processor or even worse, they may degrade performance.

In D-CAPE, we introduce a *balanced network-aware* (BNA) distribution algorithm. This distribution reduces network connections by keeping adjacent operators on the same processor while at the same time it balances the load per machine measured in number of operators allocated on a query processor.

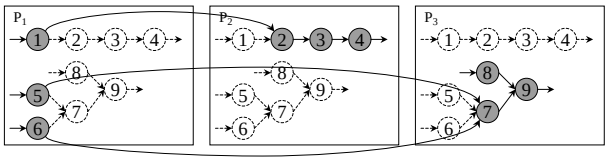

Figure 3: Example of Connection

Figure 4 shows how the choice of an initial distribution can significantly influence query plan performance. For this experiment a medium workload (a query plan with 40 operators) was distributed over 2 query processors using the BNA and the Random initial distribution. For 30 minutes Q_2 produces only 50,000 tuples if the Random distribution is used to deploy operators across query processors. Whereas, the same query plan performs about 10 times

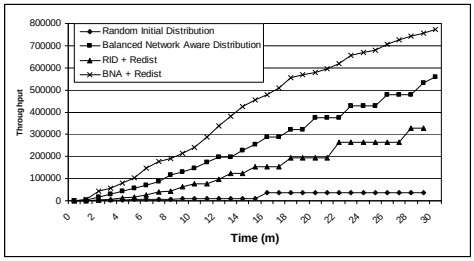

Figure 4: BNA and Random Distribution and Redistribution, 2 QPs (Medium Workload).

better in terms of the number of tuples produced if the BNA distribution is used for initial query plan deployment.

4. SELF-TUNING VIA REDISTRIBUTION

D-CAPE is designed to have the capabilities of monitoring processing performance in a non-obtrusive manner and of seamlessly redistributing query operators during runtime even under fluctuating network conditions [8]. One of the policies implemented in D-Cape is the *degradation-based* redistribution policy. This policy alleviates load on machines that have shown a degradation in output rate since the last time operators were allocated to the machine by moving the most costly operators to other query processors. In this, we give highest preference to operators that will reduce the number of network connections from the overall distribution if more than one operator is available to be moved. As Figure 4 shows, our runtime redistribution algorithm can improve the performance of a query plan even if a good initial distribution was used.

The protocol of moving query operators from one query processor to another seamlessly deactivates the operators to be moved in the original processor and reactivates them in the new processor. We package our protocol into a six-step protocol [8].

5. REFERENCES

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

[2] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-based load management in federated distributed systems. In *1st Symposium on NSDI*, March 2004.

[3] D. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[4] N. M. L.Ding, E.Rundensteiner, and G. Heineman. Joining punctuated streams. In *EDBT*, pages 587–604, 2004.

[5] B. Liu, Y. Zhu, M. Jbantova, and E. Rundensteiner. DAX: A Dynamically Adaptive Distributed System for Processing Complex Continuous Queries. In *VLDB Demo*, 2005.

[6] E. Rundensteiner, L. Ding, T. Sutherland, Y. Zhu, B. Pielech, and N. Mehta. Cape: Continuous query engine with heterogeneous-grained adaptivity. demonstration paper. In *VLDB Demonstration*, pages 1353–1356, 2004.

[7] T. Sutherland, B. Pielech, Y. Zhu, L. Ding, and E. A. Rundensteiner. An adaptive multi-objective scheduling selection framework. In *IDEAS*, 2005.

[8] T. Sutherland and E. Rundensteiner. D–cape: A self-tunung continuous query plan distribution architecture. Technical Report WPI-CS-TR-04-18, WPI, CS Dept., 2004.

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