CAPE: Continuous Query Engine with Heterogeneous-Grained Adaptivity *

Elke A. Rundensteiner, Luping Ding, Timothy Sutherland, Yali Zhu, Brad Pielech, Nishant Mehta

Department of Computer Science, Worcester Polytechnic Institute
100 Institute Road, Worcester, MA 01609, U.S.A
{rundenst, lisading, tims, yaliz, winners, nishantm}@cs.wpi.edu

1 Introduction

We present CAPE, our **C**ontinuous **A**daptive Query **P**rocessing **E**ngine, that is designed to efficiently evaluate continuous queries in highly dynamic stream environments with the following characteristics: (1) the input data may stream into the query engine at widely-varying rates; (2) meta knowledge such as punctuations [10] may dynamically be embedded into data streams; (3) as queries are registered into or removed from the query engine, the computing resources available for processing an individual operator may vary greatly over time; (4) different users may impose different quality of service (QoS) requirements.

In view of these uncertainties, no one unique optimization technique can be expected to always succeed. Correspondingly, CAPE employs an optimization framework with heterogeneous-grained adaptivity for effectively coping with such dynamic variations.

In our demonstration, we will focus on the following novel features of the CAPE system:
1. Highly reactive query operators capable of exploiting metadata to reduce resource usage and to improve execution efficiency (*intra-operator adaptivity*).
2. Online query reoptimization and migration between sub-plans to continuously converge to the best possible plan given the current situation (*plan-level adaptivity*).
3. Adaptive operator scheduling and plan distribution among multiple machines (*system-wide adaptivity*).

Our system differs from prior continuous query systems in several ways. The STREAM system [7], focusing on formal continuous query semantics, also applies some adaptation strategies such as runtime modification of the resource allocation and scheduling policy [2]. In addition to this, our system enables finer-grained intra-operator adaptivity [4, 5] and distributes a query plan to run on multiple machines for scaling purposes [9].

TelegraphCQ [3] provides a very fine-grained level of flexibility by continuously routing each tuple individually through its network of operators. However, it only achieves lightweight online query rewriting such as changing the order of query operators. Furthermore, the Eddies approach has the inherent problem of recomputing all delta intermediate results in the case of multiple joins. This may incur a significant overhead given high stream rates and join selectivities. Our system instead takes the more established approach of working with pre-defined optimized query plans, such as query networks in Aurora [1] or STREAM, and then extends this approach with adaptive techniques.

Unlike the Aurora system, we focus on the design of the fine-tuned individual query operators [4, 5], such as the join operator with intra-operator scheduling of various related tasks including state purge and metadata propagation. Moreover, we support online migration of sub-plans not only composed of stateless operators, but also containing stateful operators, to allow for maximal online adjustments in plan execution performance [11].

In summary, CAPE aims to deliver exact query answers by making the best effort through heterogeneous-grained adaptations with the goal to meet users' QoS requirements.

2 Application Example

The applications we will use to demonstrate CAPE are online auction management and web log analysis. Figure 1 shows several streams and a sample query in an online auction application. For each person that has registered to the auction system, this query asks for the count of distinct categories that include the items

*The research was partly supported by the RDC grant 2003-04 on "On-line Stream Monitoring Systems: Untethered Healthcare, Intrusion Detection, and Beyond."

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

this person has bid on within 12 hours of her registration time. Since the exploitation of dynamic metadata, i.e., punctuations, is a novel feature in CAPE, we now explain our techniques for optimizing this query.

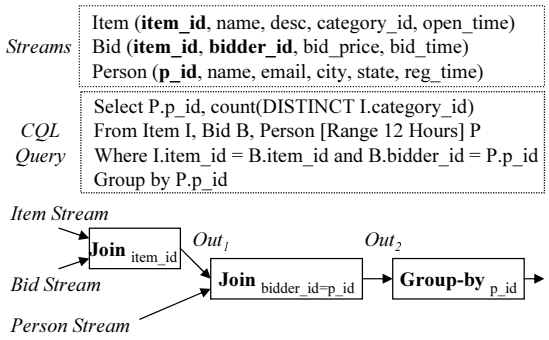

Figure 1: Sample Query in an Online Auction System.

Since the auction system knows the closing time for each auction, it can insert punctuations into the *Bid* stream to mark the end of bids for each specific item. This kind of dynamic metadata along with static metadata such as the unique key and window semantics can help the stateful operators to shrink the runtime state and the blocking operators to emit partial results regularly. Therefore, in the first join (Item $\bowtie$ Bid), when a punctuation is received from the *Bid* stream, the tuple with the same *item_id* from the *Item* stream can be purged. The second join ($Out_1 \bowtie$ Person) is a window join which applies a 12-hour window on *Person.reg_time*. Whenever a Person tuple drops out of the window, no more join results will be generated for that person. Thus, a punctuation for this *p_id* can be produced and placed into the Out_2 stream. This will trigger the *group_by* operator to emit a partial result for this person. Our experimental results [4, 5] show that by exploiting appropriate metadata, the join only requires near-constant memory overhead. The group-by operator is able to produce partial results at the earliest time possibly by exploiting the punctuations propagated by the upstream join.

3 System Overview

CAPE embeds novel adaptation techniques for tuning different levels of query evaluation, including intra-operator execution, operator scheduling, query plan structuring and plan distribution. Each level of adaptation yields maximally optimized performance by working on its own. However, none of them can possibly handle all variations that occur in a stream environment. In addition, the improper use of all levels of adaptations may cause either optimization counteraction or over-optimization. Hence, an important task is to coordinate different levels of adaptations, guiding them to function properly on their own and also to cooperate with each other in a well-regulated manner. CAPE not only incorporates novel adaptation strategies for all aspects of continuous query evaluation, but also employs a well-designed mechanism for coordinating different levels of adaptations.

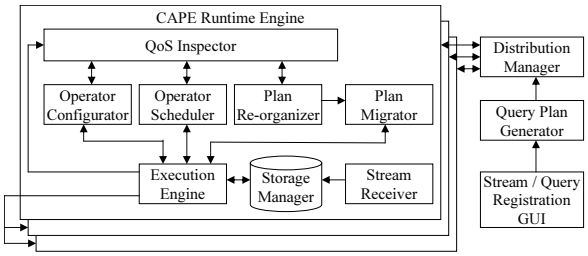

Figure 2: CAPE System Architecture.

In the system architecture shown in Figure 2, the key adaptive components are Operator Configurator, Operator Scheduler, Plan Reorganizer and Distribution Manager. Once the Execution Engine starts executing the query plan, the QoS Inspector will regularly collect statistics within each sampling interval from the Execution Engine. All of the four adaptive components use these statistics along with QoS specifications to determine if they need to adjust their behavior.

To synchronize adaptations at all levels, we have designed a heterogeneous-grained adaptation schema. Since these adaptations deal with dissimilar runtime scenarios and have different overheads, they are invoked in CAPE under different conditions.

The intra-operator adaptation incurs the lowest overhead so that it functions within an operator's execution time slot. Second, the Operator Scheduler adjusts the operator processing order after a run of a single operator or a group of operators, called a scheduling unit. After a scheduling unit finishes its work, the scheduler will check the QoS metrics for the operators and decide which operator to run next or even switch to a better scheduling strategy. This is a novel feature unique to our system. The Plan Reorganizer will wait for the completion of several scheduling units and then check the QoS metrics for the entire query plan residing on its local machine to decide whether to restructure the plan. The Distribution Manager, which likely incurs the highest cost, is assigned the longest decision making time interval. If a particular machine is detected to be overloaded, the Distribution Manager will redistribute one or multiple query plans among a cluster of machines. The Plan Migrator is invoked as necessary by the Plan Reorganizer to migrate from the old plan to the new plan, or by the Distribution Manager to migrate a query from one machine to another machine.

4 Adaptive Techniques in CAPE

4.1 Highly Reactive Query Operators

Our operators are able to adjust their behavior according to changes in the environment. We take our adaptive punctuation-exploiting join operator *PJoin* [4, 5]

as an example. PJoin is able to utilize punctuations to remove no-longer-needed data from its join state in a timely manner, thereby reducing the memory overhead and improving the probe efficiency. It can also propagate punctuations to help the blocking operators or stateful operators downstream.

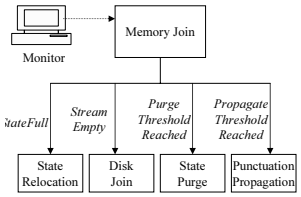

Figure 3: PJoin.

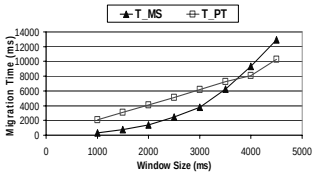

Figure 4: Migration Time.

The PJoin execution logic is composed of several tasks including *join*, *purge* and *propagation*. We have designed various state purge and punctuation propagation strategies, including eager and lazy purge, and active and passive propagation, to achieve different optimization goals. We also employ an event-driven intra-operator scheduling mechanism (Figure 3) to switch among individual tasks driven by the events modeling the punctuation arrivals, the memory threshold reached, etc. The events, listeners and thresholds are configured by the Operator Configurator and can be dynamically adjusted at runtime. This realizes an adaptive execution logic for PJoin.

All stateful operators in CAPE are equipped with a punctuation-exploiting adaptive execution logic. These operators can react to punctuation arrivals, resource (memory) modifications and punctuation requests from downstream operators.

4.2 Adaptive Operator Scheduling

The existing stream systems usually stick to an operator scheduling algorithm once it is determined to be the optimal one from the beginning of the query execution. However, as the stream environment experiences changes, this initially optimal scheduling algorithm may become sub-optimal. In response, in CAPE we employ an *adaptive scheduler-selection framework* [8]. Our Operator Scheduler can take any number of scheduling algorithms and a set of QoS requirements as input, and dynamically select the scheduling algorithm to use based on how well the system is performing. Thus, it leverages the strengths of multiple scheduling strategies and also works even under changing QoS requirements at runtime. No a priori knowledge about the scheduling algorithms in the system is required. Due to the lightweight nature of the framework, the overhead of using an adaptive strategy rather than a static algorithm has been shown experimentally to be minimal. Figure 5 shows how the adaptive framework reduces average tuple delay in the query plan.

CAPE currently incorporates a wide variety of scheduling algorithms, ranging from well established ones like Round Robin and FIFO, to those proposed recently for continuous query processing such as Train [1] and Chain [2].

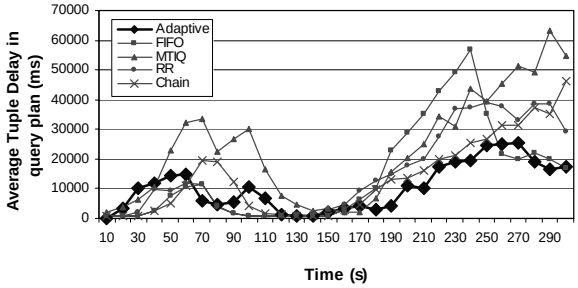

Figure 5: Improving QoS w/ Adaptive Scheduling.

4.3 Online Plan Migration

When a relatively dramatic change occurs in the stream environment, it may be more effective to optimize the query plan itself to improve the performance. We have developed a set of online plan reoptimization methods and corresponding rewriting rules. Depending on the user QoS requirements, such as output rates, the appropriate optimization method and rewriting rules are dynamically invoked.

Once the Plan Reorganizer has generated new query sub-plans, the existing plans can then be seamlessly migrated to the new ones on-the-fly [11]. This is a feature thus far unique to CAPE. We have developed two lightweight online plan migration strategies [11]: parallel track strategy and moving state strategy. Both strategies take special care to safely migrate the state of stateful operators, such as joins, to prevent any missing or duplicate results.

The *moving state strategy* first drains out tuples inside intermediate queues, and then carefully maps and moves over all relevant tuples in the states of the old query plan to their corresponding location in the new plan. In contrast, the *parallel track strategy* operates in a more gradual fashion by plugging in the new plan and starting to execute both plans in parallel. The old plan can be disconnected once the old tuples inside it are all purged. The cost of the two strategies are determined by several system parameters, such as operator selectivities, window sizes, and data arrival rates. As one of our experiment results on query plans with multiple joins, Figure 4 shows the relation between migration elapsed time and window size for both strategies (MS for moving state and PT for parallel track). The Plan Migrator dynamically evaluates the cost of these two strategies and chooses the more efficient one.

4.4 Self-Adjusting Plan Distribution

CAPE handles query plan distribution with a *Distribution Manager* which manages the tuple queues between machines [9]. A distributed query plan is created by connecting multiple machines over a network

connection to distribute operators. There are many distribution plans available for use in CAPE. Newly-developed distribution plans can also be plugged in easily. To maintain the integrity and validity of the data, especially the data in the state of the stateful operators, the Distribution Manager follows a strict set of rules to migrate an operator to another processing machine.

In addition, CAPE has the unique ability to redistribute a workload in both a local (decentralized) and a global (centralized) control mode at runtime. This enables ultimate flexibility in distribution based on the number of processing machines. The distribution is based on a cost model, which considers memory usage, network costs and overall processing costs.

5 Demonstration

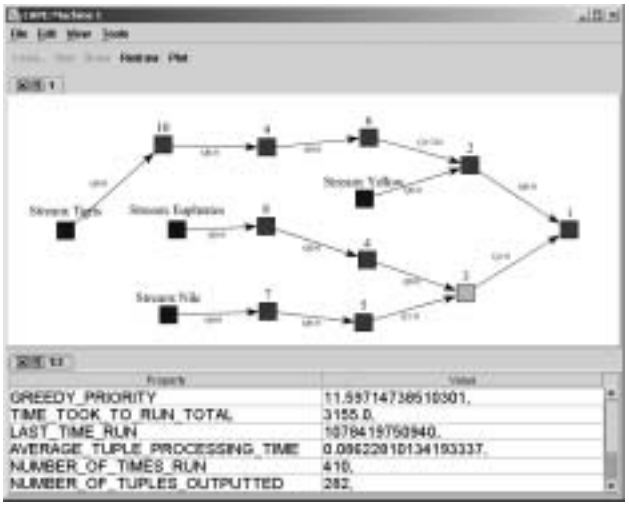

Figure 6: View of Query Plan During Execution.

In the demonstration, we will visually show the six aspects of the CAPE system listed below through a GUI (Figure 6). We employ a *stream feeder* to supply the application data at varying arrival rates, which can be as fast as 400 tuples/sec in our FIFA World Cup 1998 web log analysis application [6].

1. **General system features**. We will show the runtime status of query plan structures, inter-operator queues and storage manager. In particular, we have the ability to monitor the QoS statistics as they are updated at runtime (Figure 7).

2. **Reactive operators**. We will show the PJoin intra-operator scheduling switches from one task to another upon the arrival of punctuations, or when the allocated memory is used up. We will also show how the punctuations delivered by the upstream join operator unblock the downstream group-by operator.

3. **Adaptive operator scheduling.** We will show how the system switches to a different scheduling algorithm to yield better query performance.

4. **Runtime plan reoptimization and migration.** We will show how the *Plan Reorganizer* locates the sub-plan that needs to be restructured when it detects the performance decline of a query plan exceeds a pre-determined threshold. It then invokes the Plan Migrator to migrate from the old sub-plan to the new one at runtime.

5. **Plan distribution.** We will show that when one machine is overloaded, the *Distribution Manager* redistributes query plans to handle this situation.

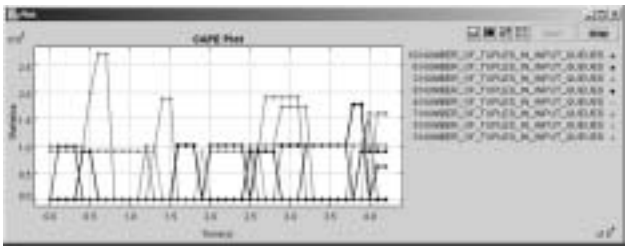

Figure 7: Statistics Monitor.

6 Acknowledgment

The authors wish to thank Maylene Natasha Waltz for the CAPE GUI development.

References

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, August 2003.

[2] B. Babcock, S. Babu, R. Motwani, and M. Datar. Chain: operator scheduling for memory minimization in data stream systems. In *ACM SIGMOD*, pages 253–264, 2003.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, pages 269–280, 2003.

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

[5] L. Ding, E. A. Rundensteiner, and G. T. Heineman. MJoin: A metadata-aware stream join operator. In *DEBS*, June 2003.

[6] Internet Traffic Archive. http://www.acm.org/sigcomm/ITA/, 2003.

[7] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR*, pages 245–256, Jan 2003.

[8] B. Pielech, T. Sutherland, and E. A. Rundensteiner. Adaptive scheduling framework for a continuous query system. Technical Report WPI-CS-TR-04-16, Worcester Polytechnic Institute, April 2004.

[9] T. Sutherland. D-cape: A self-tuning continuous query plan distribution architecture. WPI, MS Thesis, May 2004.

[10] P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting punctuation semantics in continuous data streams. *TKDE*, 15(3):555–568, May/June 2003.

[11] Y. Zhu, E. A. Rundensteiner, and G. T. Heineman. Dynamic plan migration for continuous queries over data streams. In *ACM SIGMOD*, June 2004, to appear.