The Proactive Promotion Engine

Karen Works and Elke A. Rundensteiner

Worcester Polytechnic Institute
Worcester, MA USA
(kworks,rundenst)@cs.wpi.edu

***Abstract*— Given the nature of high volume streaming environments, not all tuples can be processed within the required response time. In such instances, it is crucial to dedicate resources to producing the most important results. We will demonstrate the *Proactive Promotion Engine* (*PP*) which employs a new preferential resource allocation methodology for priority processing of stream tuples. Our key contributions include: 1) our *promotion continuous query language* allows the specification of priorities within a query, 2) our *promotion query algebra* supports proactive promotion query processing, 3) our *promotion query optimization* locates an optimized PP query plan, and 4) our *adaptive promotion control* adapts online which subset of tuples are given priority online within a single physical query plan. Our "*Portland Home Arrest*" demonstration facilitates the capture of in-flight criminals using data generated by the Virginia Tech Network Dynamics and Simulation Science Laboratory via simulation-based modeling techniques.**

I. Introduction

A. Electronic Monitoring Applications

In electronic monitoring applications (or *EMA*s) the required response time (or *rrt*) for objects monitored varies depending on the associated risk of the objects and the current system load [4]. Consider an EMA tracking missiles. It may be critical to ensure that some objects are always monitored (e.g., nuclear missiles). While the monitoring of other objects (e.g., missiles bound for unpopulated areas) may depend upon whether or not processing resources remain after handling more important objects. In addition monitoring of certain objects may be temporarily skipped altogether (e.g., missiles sent by our country) until all more significant objects can be processed within the rrt. By not processing some less important objects, EMAs reduce the system load as needed and dedicate precious resources to the most important objects.

EMAs process queries online over large volumes of data arriving from high-speed data streams with fluctuating arrival rates for days, months, or even indefinitely. At times EMAs may not be able to process the sleuth of incoming data. During such times it is crucial for an EMA to consider the significance of tuples when allocating resources and *to proactively pull certain tuples forward ahead of others within the query pipeline*. A concept that we henceforth term *proactive promotion*.

B. Motivating Example

Consider a *Portland Home Arrest* EMA that reports the criminal's ID, locale, and officer's ID for all criminals at an improper location who are within 3 miles of an officer (see Home Arrest Query below). Patrol cars (via GPS) and criminals (via ankle bracelets) continuously submit their locale to the Home Arrest EMA. To catch criminals in improper locations, the current locale of each criminal is compared against a table of permitted locations.

Home Arrest Query:
SELECT PL.PrisonerID, PL.Locale, OL.OfficerID
FROM prisonerLocation PL, prisonerInfo PI, officerLocation OL
WHERE PL.Loacle != PI.ProperLocation **AND** PL.PrisonerID = PI.PrisonerID
AND Distance(PL.CurrentLocation, OL.CurrentLocation) $\leq$ 3 mi
WINDOW 30 *seconds*

It has been shown that monitoring requirements implemented in a centralized data driven technology increases the efficiency of police officers [8]. Such technology must incorporate a suitable monitoring order to handle which criminals should be monitored when the system is overloaded. First, violent criminals who are more than 3 miles from their proper location (i.e., *escaped and likely to cause harm*) should be monitored. Next criminals known to be at an improper location within the last 30 seconds (i.e., *likely to be in violation*) should be monitored. Finally criminals known to be flight risks should be monitored. Table I outlines the desired resource allocation order for the Portland Home Arrest EMA.

TABLE I
Portland Home Arrest EMA - Resource Allocation Order

System Load	Order of Resource Allocation
System not overloaded	all tuples processed in FIFO order
System mildly overloaded	tuples from prisoners 1) escaped and likely to cause harm 2) likely to be in violation 3) who are a known flight risk
System moderately overloaded	tuples from prisoners 1) escaped and likely to cause harm 2) likely to be in violation
System extremely overloaded	tuples from prisoners 1) escaped and likely to cause harm

The relevance criteria for each monitoring level defines how to determine which incoming tuples meet the requirements to be given preferential processing assigned to a given monitoring level. Current methods that define relevance criteria include *top k* [7] and *preference queries* [5]. Broadly, such methods return the most preferred results based on a ranking score computed by a possibly complex function over all relevance criteria. These methods always calculate all defined relevance criteria to establish the final (global) rank. If some criteria are complex or a significant number of criteria exist, this may result in a high overhead and delay the processing of some

results. In contrast, EMAs seek to reduce such overhead to preferentially allocate resources to specific significant tuples by adapting at runtime which criteria are evaluated and where in the pipeline each criteria is evaluated to the system load.

C. Shortcomings of State-Of-the-Art

State-of-the-art stream engines do not meet the needs of EMAs. The closest method to managing resource allocation based upon tuple precedence in streaming databases is shedding [2], [10]. Shedding improves the overall processing of *all* tuples by reducing the workload. Random shedding randomly drops a percentage of incoming tuples. Semantic shedding drops "less important" tuples based on data content or stream statistics. Shedding makes a major assumption that is contrary to the objectives of proactive promotion as explained below.

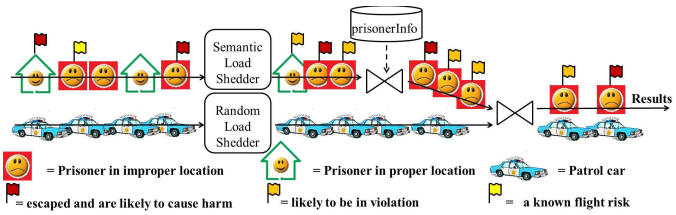


Fig. 1. State of the Art

Shedding considers all tuples not dropped to have equal importance. Consider the Home Arrest Query executed in moderately overloaded system utilizing shedding in Figure 1. In this case the *Semantic Load Shedder* would drop all tuples not from prisoners who have escaped and are likely to cause harm or likely to be in violation. Now consider that there is a period of time where many violent prisoners flee to improper locations (i.e., the workload quickly fluctuates). In this situation the most significant tuples (i.e., escaped violent prisoners) should swiftly be dedicated all the resources. However resources will still be dedicated to processing older in process tuples from prisoners likely to be in violation. This potentially may cause some significant tuples to not meet their required response time. This may cause some significant results to not be produced (i.e, the prisoner might get away).

D. Our Proactive Promotion Approach

Thus we now propose a new innovation to tackle the above shortcomings, namely, *Proactive Promotion* (or *PP*). PP adjusts resource allocation in accordance with the significance of tuples and the system load. PP's goal is to establish an adaptive processing ordering of tuples to ensure that at any given moment the most important tuples have the greatest chance at completing within the required response time (rrt). Some of our key innovations include:

1) We introduce the **promotion continuous query language** (P-CQL) that supports the specification of multi-tiered proactive promotion criteria as part of a query.

2) We design the **proactive promotion query algebra** that supports the assignment and propagation of tuple importance. **Priority classifier operator**s perform selective precedence evaluations. While all operators in the query pipeline employ a multi queue priority driven execution strategy.

3) We propose the **proactive promotion query optimizer** that locates an optimized PP query plan using a cost-based approach.

4) At runtime to promote specific tuples given the current runtime environmental conditions, the PP Adaptor efficiently adjusts which and where precedence criteria are used to assign preferential processing (Table I). The PP Adaptor instructs the query operators to adapt which, when, and where precedence evaluations are applied without requiring any changes to the query pipeline.

II. PP System Architecture

The PP architecture is composed of the PP Optimizer, PP Monitor, PP Adaptor, and the PP Executor (Figure 2) [1]. First the user specifies a query using P-CQL (Section III-A). Then the PP Optimizer produces an optimized PP query plan. The optimizer forwards the PP query plan to the PP Executor, which instantiates the PP runtime infrastructure.

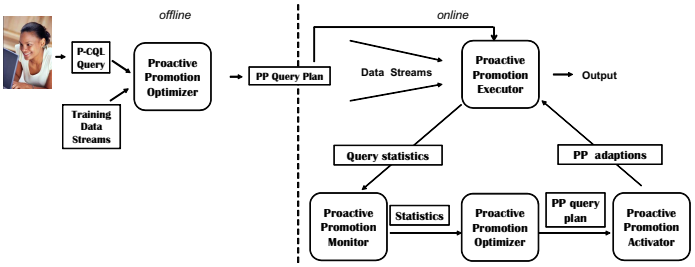


Fig. 2. PP Architecture

At runtime, the PP Monitor gathers statistics. Once changes in the number of significant tuples which complete processing within the rrt are identified, the PP Optimizer exploits these statistics to select an optimized PP query plan. The optimized PP query plan is forwarded to the PP Adaptor which effectively adapts which, when, and where precedence evaluations by instructing the query operators.

III. Technical Details

A. Precedence specification via Promotion-CQL

We designed the promotion continuous query language (P-CQL) as an extension of CQL [3] to allow the specification of multiple promotion levels and a *lifespan* for each query. For each *promotion level*, P-CQL supports the stipulation of *membership* and *rank*. Below is the P-CQL extension to the Home Arrest Query (Section I-B) using the resource allocation order in Table I.

The *lifespan*, or upper bound on required response time, is the tuple processing time limit for a query. Once a tuple has

[1]To reduce overhead, the PP Monitor, PP Optimizer, and PP Adaptor are run on a separate thread.

reached its lifespan limit, the tuple is expired (i.e., no longer processed).

The *rank* predicate specifies the significance of a promotion level in relation to the other promotion levels associated with the query. Each promotion level within a query has a unique *rank*. The most significant promotion level in the query has the *rank* value of 1.

Membership in a level is specified via *criteria* predicates. Each predicate is composed of 1) an attribute or function, 2) a comparison expression $(=, \leq, <, >, \geq)$, and 3) a data value or attribute. Membership may be defined by combining multiple *criteria* using conjunction and disjunction.

(Extension to Home Arrest Query)
LIFESPAN 1000 *milliseconds*
RANK 1
CRITERIA PI.CommittedViolentOffense = TRUE **AND**
Distance(PL.Locale, PI.ProperLocation) $\geq$ 3 mi
RANK 2
CRITERIA PI.RecentlyFoundInWrongLocation = TRUE
RANK 3
CRITERIA PI.KnownFlightRisk = TRUE

B. Promotion Query Algebra and Processing

We now extend the continuous query algebra [6] to design a promotion query algebra using two key features. First, we design novel query operators dedicated to significant tuple precedence evaluation. Second, we enhance existing CQL query operators to support promotion-aware processing.

Query operators could be augmented to compute the significance of tuples. However this would not easily allow resource allocation adjustments between query processing and precedence evaluations. Thus PP implements dedicated precedence evaluation operators referred to as *priority classifiers* or PCs.

After evaluating the precedence of a tuple, PCs associate promotion levels with tuples via *priority punctuations*. A priority punctuation states the rank and number of tuples at this rank. Using priority punctuations reduces resource overhead by sharing a promotion level across multiple tuples.

At runtime if a query operator receives a priority punctuation, the operator queues the associated tuples according to their rank. All other non-punctuated tuples are placed in the lowest priority queue. An operator starts processing tuples from the highest priority queue. When this queue is empty the operator moves to the next highest priority queue.

C. Promotion Query Optimization

At compile-time the PP optimizer, given a P-CQL query and a training data stream, locates an optimized PP query plan. The optimized PP query plan maximizes the number of the most significant tuples that met their required response time (i.e., dedicates resources to processing the most significant tuples) and minimizes the estimated tuple latency for the most significant tuples (i.e., reduces the processing overhead in terms of associated precedence evaluation). The PP optimizer first places a PC before each standard query operator in the regular CQL query plan. All possible PP query plans are created by adjusting in which PC each promotion level is evaluated. Then a cost-based search algorithm that considers the effectiveness of each possible PP query plan at producing the largest number of the most significant tuples that met the required response time with the minimum estimated tuple latency for the most significant tuples is applied.

D. Online Adaptive Promotion Control

Given that the system load can vary drastically over time, the PP Optimizer determines at run-time which and where in the query plan promotion levels are assigned preferential processing. To change the PP query plan, first the PP Optimizer selects an optimized PP query plan based upon statistics collected by the PP Monitor. The PP Adaptor reconfigures the PP query plan without requiring the physical query plan to change by adapting which PC evaluates each promotion level.

IV. Demonstration

Portland Home Arrest EMA: The real-life application we demonstrate is the Portland Home Arrest EMA which reports on criminals in improper locations who are in close proximity to police officers (Section I-B). PP addresses the problem of aligning resource allocation to the significance of tuples and the current system load. In our demonstration, we allow the audience to define promotion levels and adjust the load on the system. During execution, the audience can watch the movement of people (i.e., criminals and policemen) in real time. Then they can compare the results of the query ran using *PP* with several alternative systems, namely, random shedding, semantic shedding, and traditional query processing and assess which method caught the most significant criminals.

The promotion levels defined may lead to different PP query plans being executed. This scenario corresponds to the situation where the monitoring order of objects adapts based upon the characteristics of the objects monitored (i.e., there may be no violent criminal assigned to home arrest), the number of objects monitored (i.e., the number of officers available), and user's preferences.

Data and Queries: We use data streams from [1] that contain the daily movement of people in Portland, Oregon. The queries we will use include the CQL query in Section I-B with the addition of P-CQL specifications for the promotion levels selected (similar to the P-CQL query in Section III-A). As the audience can adjust the promotion levels, a plethora of P-CQL queries can be executed. In addition the audience can select the system load and observe how the system adaptively adjusts to the system load.

Plethora of Scenarios:

Distinct Data: The audience can adapt the distinct characteristics of people in the Portland, Oregon population. For example they can set the percentage of criminals (Figure 3).

System Load: The audience will be able to select the level of the system load between not overloaded, mildly overloaded, moderately overloaded and extremely overloaded (Figure 3).

Query Specification: The audience can specify promotion levels. Possible promotion level criteria include criminals likely to escape, criminals likely to flee, criminals who committed a violent offense, first time offenders, and criminals

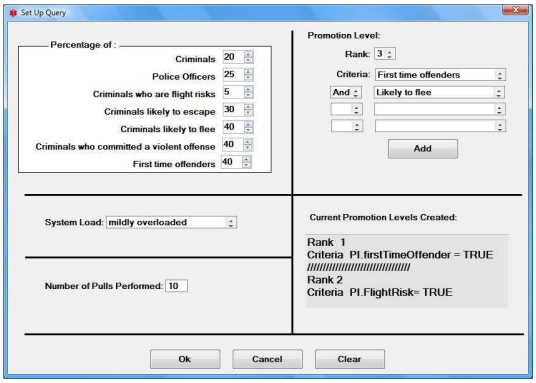


Fig. 3. Set Up Query

considered to be flight risks (Figure 3). Combining multiple criteria using conjunction and disjunction for promotion levels is also supported.

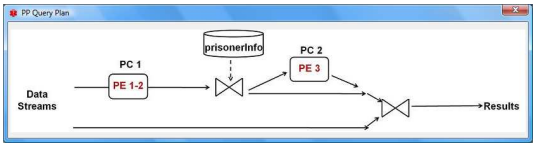
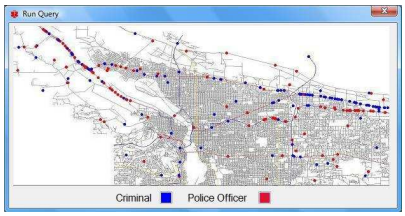
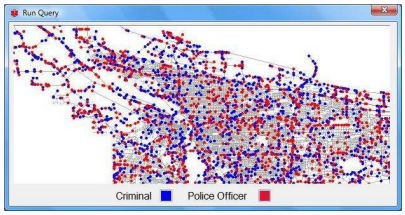


Fig. 4. PP Query Plan

PP Optimization: After a user submits a query, the PP optimizer will be invoked to compute the optimal PP query plan. Using the GUI, the audience will be able to see the generated PP query plan (Figure 4). For each priority classifier operator (*PC*), the system displays the PE flag which represents the promotion levels used to evaluate the precedence of tuples.



(a)



(b)

Fig. 5. Execution Runtime Output a) Not Overloaded b) Overloaded

PP Execution: During execution, the audience can watch the movement of criminals and policemen in real time. Criminals will be shown in blue. Police officers will be shown in red. In addition, the pixels that represent significant criminals as defined by the promotion levels used to proactively promote tuples will flash. Figure 5(a) shows an example from a not overloaded system. While Figure 5(b) shows an example from an overloaded system.

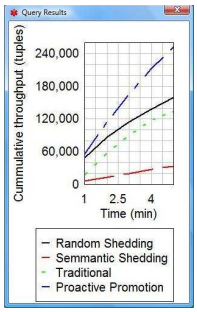


Fig. 6. Query Results

PP Performance Monitoring Showcase: Finally, our demonstration will graphically demonstrate the ability of PP to provide adequate resources to the most significant tuples (Figure 6) as compared to alternative systems. The audience can contrast the cumulative throughout of tuples at each defined promotion level of our *PP* system to random shedding, semantic shedding, and traditional query processing approaches.

V. Conclusion

Our proactive promotion engine (PP) takes an innovative approach towards addressing the problem of aligning resource allocation to the significance of tuples and the current system load. Our key contribution is to show that PP is a viable approach towards providing preferential resource allocation based upon user requirements and the current system load. This now opens research opportunities in this novel area of preferential query systems.

We thank our WPI peers for CAPE [9] and feedback. This work is supported by NSF grants IIS-1018443 & 0917017 & 0414567 & 0551584 (equipment), and GAANN.

References

[1] http://ndssl.vbi.vt.edu/opendata/index.php. *Virginia Tech:Network Dynamics and Simulation Science Laboratory*.
[2] D. Abadi and et. al. Aurora: a data stream management system. In *SIGMOD*, pages 666–666, 2003.
[3] A. Arasu and et. al. The cql continuous query language: Semantic foundations and query execution. Technical report, VLDB, 2003.
[4] D. Carney and et.al. Monitoring streams: a new class of data management applications. pages 215–226. VLDB, 2002.
[5] J. Chomicki. Semantic optimization techniques for preference queries. *Inf. Syst.*, pages 670–684, 2007.
[6] L. Golab and et.al. Update-pattern-aware modeling and processing of cont. queries. In *SIGMOD*, pages 658–669, 2005.
[7] I. F. Ilyas and et. al. A survey of top-k query processing techniques in relational database systems. *ACM*, pages 1–58, 2008.
[8] C. Lin. Technology implementation management in law enforcement. *Social Science Computer Review*, pages 24–36, 2004.
[9] E. A. Rundensteiner and et. al. Cape: Continuous query engine with heterogeneous-grained adaptivity. In *VLDB*, pages 1353–1356, 2004.
[10] M. Wei and et. al. Achieving high output quality under limited resources through structure-based spilling in xml streams. *PVLDB*, pages 1267–1278, 2010.