Microsoft CEP Server and Online Behavioral Targeting

M. H. Ali[1], C. Gerea[1], B. S. Raman[1], B. Sezgin[1], T. Tarnavski[1], T. Verona[1], P. Wang[1], P. Zabback[1], A. Ananthanarayan[1], A. Kirilov[1], M. Lu[1], A. Raizman[1], R. Krishnan[1], R. Schindlauer[1], T. Grabs[1], S. Bjeletich[1], B. Chandramouli[2], J. Goldstein[2], S. Bhat[3], Ying Li[3], V. Di Nicola[3], X. Wang[3], David Maier[4], S. Grell[5], O. Nano[5], I. Santos[5]

[1]Microsoft SQL Server, {mali, cgerea, sethur, beysims, tihot, tomerv, pingwang, pzabback, asvina, antonk, milu, alexr, ramkri, romans, torsteng, sharonbj}@microsoft.com

[2]Microsoft Research, {badrishc, jongold}@ microsoft.com

[3] Microsoft Audience Intelligence, {sudinb, yingli, vidini, xfwang}@ microsoft.com

[4]Department of CS, Portland State University, {maier@cs.pdx.edu}

[5] European Microsoft Innovation Center, {stgrell, onano, ivosan}@ microsoft.com

ABSTRACT

In this demo, we present the *Microsoft Complex Event Processing* (CEP) Server, *Microsoft CEP* for short. *Microsoft CEP* is an event stream processing system featured by its declarative query language and its multiple consistency levels of stream query processing. Query composability, query fusing, and operator sharing are key features in the *Microsoft CEP* query processor. Moreover, the debugging and supportability tools of *Microsoft CEP* provide visibility of system internals to users.

Web click analysis has been crucial to behavior-based online marketing. Streams of web click events provide a typical workload for a CEP server. Meanwhile, a *CEP* server with its processing capabilities plays a key role in web click analysis. This demo highlights the features of *Microsoft CEP* under a workload of web click events.

1. INTRODUCTION

The tremendous growth in event streaming applications, coupled with the level of maturity achieved by research efforts in data stream systems, have significantly influenced the vision and the strategies of commercial database systems. *Microsoft SQL Server* has been leveraging its query processing expertise to handle streaming-oriented workloads. As a result, the design of *Microsoft Complex Event Processing* (CEP) server has incorporated state of the art research to meet the demands of commercial workloads.

Microsoft CEP is based on the CEDR [1, 2] research project. In CEDR, a data stream is modeled as a time-varying relation, motivated by early work on temporal databases by Snodgrass et al. [3]. In such a relation, events are represented as tuples f gaf g a bc f c c c c g a timestamps, encoding a validity interval, or *lifetime*. The lifetime indicates the range of time when the tuple is valid from f c c c gbc c ca g c *Microsoft CEP* has several features that include: automatic handling of *compensations* for out-of-order events, speculative execution, support for modifying the lifetimes of earlier events, and the ability to operate over a wide range of consistency levels (as defined by tradeoffs between output blocking and memory usage). *Microsoft CEP* uses application time (as opposed to system time [4]) for specifying and manipulating lifetimes, which contribute to precise semantics and well-defined deterministic operator behavior.

A prominent domain with very challenging requirements is Click Stream Analysis. Here, the online behavior of users in terms of page visits is analyzed and processed. The insight gained g f c c actions can then be leveraged to adjust the online experience accordingly, e.g., by tailoring the navigational structure of the web site according to the anticipated click path or by displaying targeted ads. This is what we call *behavioral targeting*. For such dynamic experience to be meaningful and efficient, the analysis of the user behavior has to be done in real-time. Considering this low-latency demand and the expected data rates, an in-memory CEP engine is far more suited to accomplish this task than a traditional transactional database.

This paper demonstrates the basic features of *Microsoft CEP* in the context of online behavioral targeting. We overview the features of the *Microsoft CEP* server in Section 2, highlight the online behavioral targeting scenarios and use cases in Section 3, and describe the demo scenario in Section 4.

2. Features of Microsoft CEP Server

2.1 Declarative Query Language

The complex event processing paradigm departs from the principles of traditional relational database systems with its transient nature of event data. However, by creating a sound and consistent query algebra for CEP, we are able to provide a fully declarative query experience. The query algebra contains operators from the relational world, such as select, project, and join, with their semantics adapted to the processing of transient time series. Yet, the new semantics never compromise the full composability of the proposed operators: Each operator receives as well as produces a stream of events.

On top of the streaming algebra, a suitable query language exposes the operator functionality to the user. A good declarative query language should be a concise, yet intuitive interface to the underlying algebra. Several attempts have been undertaken to adapt SQL to the CEP domain, enriching its syntax by constructs to specify windows in time. We chose Language Integrated Query (*LINQ*) [5] as our approach to

VLDB '09, August 24-28, 2009, Lyon, France.

express CEP queries. LINQ is a uniform programming model for any kind of data that introduces queries as first class citizens in the *Microsoft .NET* framework.

2.2 Consistent Streaming through Time

In this subsection, we discuss some details about the various unique properties of *Microsoft CEP*. The interested reader is referred to [2] for more details.

2.2.1 Canonical History Table (CHT)

This is the logical representation of a stream. Each entry in a CHT consists of a start time (V_s), an end time (V_e), and the payload. All times are application times, as opposed to system times. Table 1 shows an example CHT. This CHT could be derived by actual physical events which could be either new inserts or lifetime modifications of older events (e.g., *retractions*, where lifetime is shortened). For example, Table 2 shows one possible physical stream with an associated CHT shown in Table 1. Note that a retraction event includes the new valid end time of the modified event (V_{newe}). *Microsoft CEP* operators are well-behaved in terms of their effect on the CHT. This makes our streaming algebra well-defined and deterministic, even when data arrives out of order.

Table 1 – Example of a CHT

ID	V_s	V_e	Payload
E0	1	5	P1
E1	4	9	P2

Table 2 – Example of a physical stream

ID	Type	V_s	V_e	V_{newe}	Payload
E0	Insertion	1		-	P1
E0	Retraction	1		10	P1
E0	Retraction	1	10	5	P1
E1	Insertion	4	9	-	P2

2.2.2 Windowing Semantics

Microsoft CEP borrows heavily from SQL to define semantics for each supported operator. One important difference from existing streaming systems is that windows in *Microsoft CEP* are associated with individual events instead of streams or operators. This approach removes the constraint that all events in a stream have the same lifetime. Each operator produces events with a span-based representation that indicates its lifetime. Windows are specified by a special stateless operator called *AlterLifetime*, that (for a time-based window of *w* time units) simply sets event end time (V_e) to be $V_s + w$.

2.2.3 Navigating Consistency using Speculation

Two interesting aspects of processing can be varied:

- How long do we wait before providing a result based on an incoming event (blocking)?
- How long do we remember input state both for blocking and for providing necessary compensations once we unblock?

These variables lead to the spectrum of consistency levels described in Figure 1. Briefly, the diagonal corresponds to the line of zero speculation, which is supported by existing streaming systems. CEDR allows us to navigate to the lower part of the triangle, where we can reduce blocking by emitting speculative output and potentially correcting wrong answers later using lifetime changes and insertions.

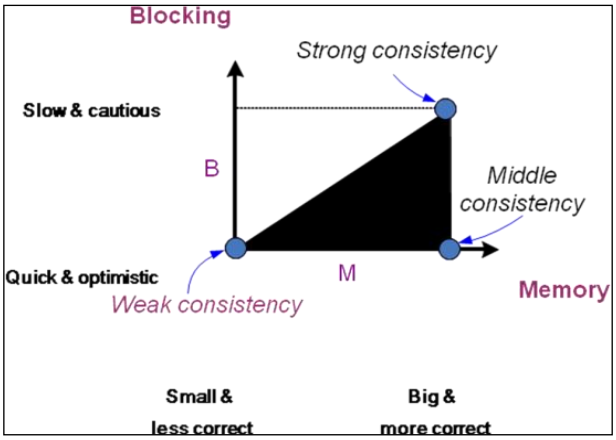

Figure 1. Spectrum of possible consistency levels.

2.3 Query Reusability and Composability

In *Microsoft CEP*, queries are declared through query templates. A query template is an XML representation of a query operator tree with generic endpoints for the input and the output streams. This XML representation is an intermediate format generated by the LINQ language interface and is consumed by the CEP query compiler. Query templates allow for flexible re-usability through their generic endpoints. Endpoints get bound to input/output adapters to connect event producers/consumers to the CEP query. The same query template can be used to instantiate multiple query instances. Each query instance has the same operator tree as in the template, yet, is bound to different streaming producers/consumers through different adapters.

Besides this re-use of templates, it is also possible to produce more complex queries by composing query templates. This is best explained using the intermediate XML representation of a CEP query template. This intermediate format represents a tree of individual operators (filters, projects, joins, etc) as nodes and streams as links. One of these nodes can be a query template reference. The CEP compiler expands this template reference and replaces it with the XML representation of the referenced query template (which in turn may contain other query template references). This is a very powerful way to produce complex data flow queries out of individually maintained (and testable) components. A query template reference can appear anywhere in the operator tree. The only requirement is that the input and output schemas d f c cdc c acb c c c c bpoints match the ones expected by the referencing template.

Query template reference is a compile time query composability feature. However, the data-flow paradigm of CEP queries allows it very naturally to compose (connect) queries during run-time. The output stream of one query can serve as input to another query. If multiple queries are consuming the output from one producer query, the system will introduce a broadcast operator to replicate an output event to all consuming inputs.

2.4 Query Fusing and Operator Sharing

By default, each operator in a query tree is implemented as an independent task by the *Microsoft CEP* runtime. Tasks exchange data (events) through streams (i.e., queues of events) between tasks. This fine grained task model allows very flexible assignment of execution units (tasks) to CPU cores. The data-stream model also enables distributed parallel execution of queries.

With an increasing number of operators and their intermediate streams, it becomes prohibitive to deploy every operator in a separate task. The overhead of task communication through streams can easily dominate the cost of evaluating individual operators. *Microsoft CEP* eliminates the communication overhead by fusing operators together. Two types of operator fusing are provided: vertical fusing and horizontal fusing.

Vertical operator fusing transparently replaces the stream between operators by direct function calls. If the optimizer decides to fuse a filter operator that is on top of another operator, the stream enqueue operation is transformed into a direct function call that evaluates the filter predicate directly. Therefore, both operators will execute in the same task and intermediate streams between the two operators are eliminated. Since, this type of fusing allows operators that are on top of each other to share execution tasks, it is referred to as *vertical fusing*.

Horizontal operator fusing is also possible if there are multiple queries that are based on the same query template. In this case, *Microsoft CEP* can decide to map identical operators from different queries to the same execution task. Whenever an operator of a horizontally fused query is scheduled to execute, the correct local operator state is associated with the task. Horizontal fusing is a form of virtualization that is similar to lightweight (fiber mode) scheduling.

Horizontal fusing can also be applied in cases where data streams are partitioned and for each partition the same operators are executed. In case of thousands or even millions of partitions it would be prohibitive to create this large a number of execution tasks. Instead, the compiler decides the number of execution tasks based on hardware properties (number of cores) and other cost-based criteria. Then, the compiler horizontally fuses multiple partitions.

Operator fusing (both vertical and horizontal fusing) is a cost-based compiler decision. It is even conceivable to dynamically fuse/un-fuse operators during runtime when run-time statistics and the number of outstanding queries suggest this is advantageous.

2.5 Scalability

In *Microsoft CEP*, scalability is a two-fold problem: (1) partitioning the incoming event streams as well as the standing queries across multiple CEP instances and (2) managing these instances. Partitioning introduces the opposite and complementary direction of the query fusing approach (presented in Section 2.4). While query fusing gathers many light weight queries into a single query, partitioning divides heavy workloads into smaller workloads for deployment on several CEP instances. *Microsoft CEP* provides two approaches for partitioning: *stream partitioning* and *query partitioning*.

Stream partitioning is achieved in *Microsoft CEP* through the e -and- c f f fc gg a c c into multiple queries (of the same operator tree) such that each query operates on a portion of the stream. *Query partitioning* divides a query into multiple subqueries. Each subquery is deployed on an instance of *Microsoft CEP*. Operators across different CEP instances have the ability to communicate through streams that span machine boundaries.

Managing multiple *CEP* instances addresses the creation and the tear-down of an instance. Data replication and fail-over between instances are critical instance manageability features that trade off hardware cost versus high availability. Continuous load balancing across instances is crucial to the c c d ance and is achieved through stream repartitioning (in case of partitioned streams) or operator migration across instances (in case of partitioned queries).

2.6 Debuggability and Supportability

The nature of CEP queries brings new challenges to the debugging and supportability of live production systems with continuous queries. The uptime of continuous queries is very long compared to queries in traditional database systems. Richer support is needed to troubleshoot live systems where shutting down a query may be expensive. Also, stream producers and consumers are independent and work asynchronously. This asynchronous behavior results in non-determinism in the order of processing events. In such environments, traditional step-by-step debugging is less useful and often not practical. The extended uptime and the output non-determinism of CEP queries call for non-intrusive automated monitoring and diagnostics.

Microsoft CEP server exposes the state of the system through both point in time snapshots and streams of manageability events. Snapshots contain the per-query memory and CPU usage, latency, throughput, and other runtime statistics. They are used for ad-hoc troubleshooting and to gain instant visibility into the system. Meanwhile, the engine generates manageability streams of all noteworthy actions taken by the system e.g., a query start, a query failure, an event entering the system, an event exiting from the system, stream overflow, etc. Reusing its own infrastructure, the system issues g g e c gc eainst manageability streams. These monitoring queries run continuously at the background of the CEP engine or probably on another CEP instance. The monitoring queries consume the manageability streams to continuously monitor user-issued queries for failures, performance problems, and other critical conditions. Building on this infrastructure, *Microsoft CEP* server provides a graphical tool for the inspection of event flow in a query as a means of debugging and performing root cause analysis of problems.

3. Online Behavioral Targeting

Behavioral Targeting allows advertisers to deliver highly *relevant* ads to predefined sets of audience, resulting in enhanced advertising effect and optimized return on

investment. It employs various user and content intelligence technologies to classify online users into audience segments according to their online activities. A critical factor for the relevance of an ad is *timeliness*, and ideally, being in *real-time* i.e., presenting the ad while the user is *in the process* of a specific web experience (e.g., searching for a place to eat, booking a ticket, looking for a gift to purchase, and so on). The round-trip duration for this classification varies in most ad platforms - from hours to days. The CEP engine has helped the Microsoft Advertising platform to reduce this time to seconds, enabling advertisements to be displayed in the same user session.

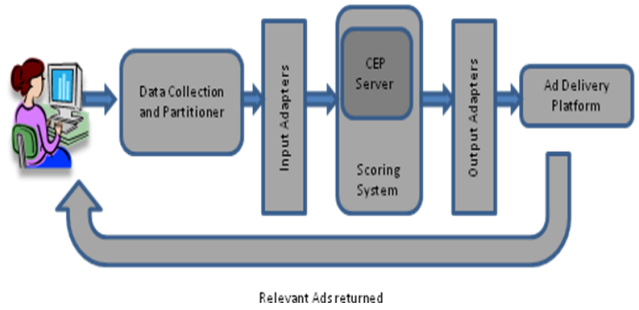

Figure 2. The Microsoft Advertisement Platform.

The ad platform consists of the following parts (Figure 2). A data collection system collects user's web clicks from various forms of interactions (e.g., search URLs and page clicks), formulates them as events, and pushes them into the *Microsoft CEP* c c g ddc . These tuples are streamed into a real time scoring system, of which the CEP engine is the central piece. The engine runs scoring queries against the tuples, allowing different scoring algorithms to be integrated. Factors that go into the scoring algorithm include user click rates at sites of interest, "memory" of the user's interactions, and other business logic embedded in the scoring model. The scoring query classifies the user to belong in one or more ce c e.g., auto researcher, Europe traveler, fast food customer). Based on this real-time insight, advertisements pertaining to that segment are immediately narrowcasted to the c g e c g bc g c d

This immediacy of results provided by CEP has demonstrably proved (in a live Web deployment) to be highly successful at improving the efficiency of campaigns, ad delivery and the overall (marketing) lift from the application. The scoring system handles 100s of millions of events per day, the system has variable memory ranging from seconds to days, and the queries have stringent requirements to return scoring results within seconds or minutes.

4. Demo Description

The proposed demo scenario includes three parts. The first part is an interactive user session with the www.msn.com website. A user signs in with a dedicated demo account. The user navigates through the *MSN* website searching for travel packages, car dealers, fast food restaurants, etc. As the user builds up a search context, targeted ads appear on the top banner. The choice of the targeted ad is the outcome of a *segmentation* query (as described in Section 3) running inside the CEP server to categorize f c c c a ga events into segments of interest.

The second part of the demo exposes the internals of the CEP engine. Using the *Microsoft CEP* debugger (Figure 3), the user traces an event as it passes through the query pipeline, and inspects output events from each operator as time advances.

The third part of the demo focuses on CEP adapters. The CEP server receives events at its input buffers from various types of input adapters, pushes these events through the query pipeline and streams the resultant output events to output adapters. Input and output adapter are the interfaces of the CEP server to the outside world. The third part of this demo focuses on two aspects of the adapters: (1) the presentation capabilities that translate high rate input/output streams into visually and programmatically inspectable information, and (2) the ability to control the behavior of input/output streams through various control knobs.

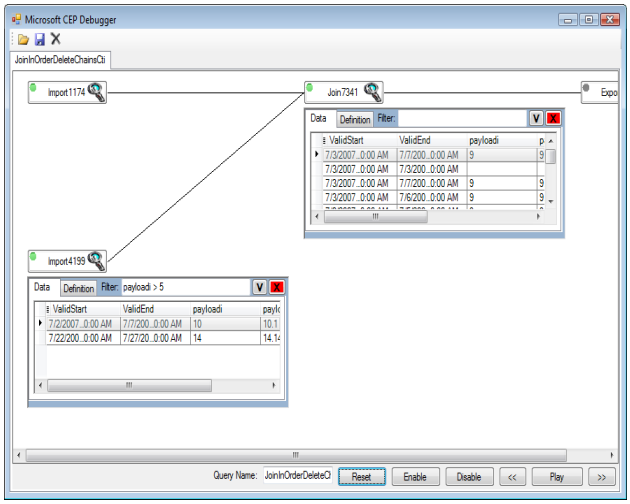

Figure 3. The Microsoft CEP Server debugging tool.

5. REFERENCES

[1] Roger S. Barga, Jonathan Goldstein, Mohamed H. Ali, and Mingsheng Hong. *Consistent Streaming Through Time: A Vision for Event Stream Processing*. In Proceedings of CIDR, 412-422, 2007.

[2] Jonathan Goldstein, Mingsheng Hong, Mohamed Ali, and Roger Barga. *Consistency Sensitive Streaming Operators in CEDR*. Technical Report, MSR-TR-2007-158, Microsoft Research, Dec 2007.

[3] C. Jensen and R. Snodgrass. *Temporal Specialization*. In proceedings of ICDE, 594-603, 1992.

[4] Utkarsh Srivastava, Jennifer Widom. *Flexible Time Management in Data Stream Systems.* In PODS, 263-274, 2004

[5] Paolo Pialorsi, Marco Russo. *Programming Microsoft LINQ*, Microsoft Press, May 2008.