# Towards Hybrid Online On-Demand Querying of Realtime Data with Stateful Complex Event Processing

Qunzhi Zhou
*Department of Computer Science*
*University of Southern California*
*Los Angeles, USA*
*qunzhizh@usc.edu*

Yogesh Simmhan, Viktor Prasanna
*Ming Hsieh Department of Electrical Engineering*
*University of Southern California*
*Los Angeles, USA*
{*simmhan, prasanna*}*@usc.edu*

***Abstract*—Emerging Big Data applications in areas like e-commerce and energy industry require both online and on-demand queries to be performed over vast and fast data arriving as streams. These present novel challenges to Big Data management systems. Complex Event Processing (CEP) is recognized as a high performance online query scheme which in particular deals with the *velocity* aspect of the 3-V's of Big Data. However, traditional CEP systems do not consider data *variety* and lack the capability to embed *ad hoc* queries over the *volume* of data streams. In this paper, we propose $H2O$, a stateful complex event processing framework, to support hybrid online and on-demand queries over realtime data. We propose a semantically enriched event and query model to address data *variety*. A formal query algebra is developed to precisely capture the stateful and containment semantics of online and on-demand queries. We describe techniques to achieve the interactive query processing over realtime data featured by efficient online querying, dynamic stream data persistence and on-demand access. The system architecture is presented and the current implementation status reported.**

***Keywords*-complex event processing; stream processing; in-memory database; big data; semantic web;**

## I. Introduction

Existing Big Data solutions primarily focus on addressing the *volume* aspect of the 3-V's of Big Data [16]. As a result, specialized technologies including distributed databases [10], Hadoop [19] and NoSQL [12] have been developed to support scalable data storage and on-demand querying over large volumes of data. These systems usually provide high read performance for data that has been persisted and properly indexed.

Recently, the *velocity* and *variety* sides of Big Data are attracting increasing attention [1]. Organizations such as social media, utility companies and financial institutions often face scenarios where they need to analyze heterogeneous "in-flight" data for businesses analytics. For example, applications including realtime bidding (RTB), digital shopping, dynamic demand response in Smart Grid [24] and algorithmic trading need to process realtime data such as advertisements, shopping activities, sensor readings and stock ticks, respectively, originating from various sources to make timely decisions. Data for these applications arrive at high-rate streams and may vary in terms of structure and meanings.

One challenge of these emerging realtime Big Data applications is the need to concurrently support both online and on-demand queries over data streams. Online queries represent persistent user interests that need to be evaluated continuously as new data appears in order to trigger realtime actions. On the other hand, on-demand queries model transient user interests which need to be answered in an *ad hoc* manner, usually using stored history data. Traditional Big Data solutions often adopt a "multi-temperature" architecture where data that is frequently accessed (hot data) is on fast storage compared to less-frequently accessed data (warm and cold data) placed on slower storage. Realtime data appears at the top of hot storage requiring on-the-fly write and read processing.

The unique characteristics of realtime data offers the opportunity for on-the-fly querying with dynamic data lifecycle management, that obviates the need to persistently store everything. One key notion of realtime data is its transient property: (1) new data arrives on streams at high velocity; and (2) old data leaves streams or the values of data fade away at potentially high velocity. For example, consider a digital shopping scenario where a large number of web orders are placed and processed concurrently. Activities such as order creation, payment and delivery need to be monitored continuously to detect service level agreement (SLA) violations and offer on-demand status check. However after an order is completely fulfilled, it is no longer "alive" or of value for SLA detection and status check queries.

Complex Event Processing (CEP) is a popular technique for querying realtime data. CEP deals with detecting realtime situations, represented as event patterns, from among many event streams. CEP is commonly used for operational intelligence, where online query pattern detection drives realtime responses, and is used in domains overlapping with Big Data applications, ranging from e-commerce to energy industry to financial services [2].

However, traditional CEP systems do not consider data *variety* and only support online queries. Most existing CEP

systems such as [11], [3] adopt a SQL-like query language and evaluate event data at the syntactic level. These CEP queries are pre-defined, listen to new events and trigger outputs when the patterns are matched completely. Temporary states observed by the system are not managed for external access and hence not available for on-demand querying.

In this paper, we propose $H2O$, a stateful Complex Event Processing framework to support hybrid online and on-demand querying over realtime data. Specifically, our contributions include:

- We introduce a unified event and query model (§ III) for online and on-demand query specification over heterogeneous realtime data streams.
- We develop a formal query algebra to capture the stateful and containment semantics of queries, which forms the basis for dynamic life-cycle management and on-demand access of realtime data (§ IV).
- We discuss processing techniques of this hybrid model for efficient online and on-demand query evaluation over data streams (§ V).

## II. Motivating Applications

Our work is motivated by applications from domains as diverse as e-commerce and Smart Grid to provide users with both *happened-after* (online) and *happened-before* (on-demand) forms of situation awareness. Online queries detect immediate occurrences of pre-defined situations to drive timely actions. On-demand queries, on the other hand, extract post-defined situations from observed data for analysis. In the following we describe a few compelling scenarios from the e-commerce and the energy industry which demonstrate the need for both hybrid online and on-demand query capabilities over realtime data across diverse domains.

Digital shopping plays a vital role in today's retail industry, and allows consumers to directly buy (physical or virtual) goods and services from vendors over the Internet. Consider a simplified digital shopping process which consists of a sequential set of activities: creating web order, checking inventory, processing payment, processing delivery, and email notification. An order has to be fulfilled within a limited time window which may range from a few minutes to days. Online queries can be used to monitor the order completeness or report SLA violations in realtime. During the intermediate processing steps, a customer may issue on-demand queries to check the order status, and vendors may need to determine how many orders are accumulated in, say, the order creation step in case the inventory is low and action needs to be taken to avoid an SLA violation.

Dynamic demand response (DR) is a promising technique in Smart Grid for identifying demand-side energy curtailment opportunities through the analysis of high volume sensor readings that are delivered over the network [20]. In a campus Micro Grid, for example, online queries may correlate meter readings, occupancy sensor readings and room schedules to detect fine-grained power curtailment opportunities both in temporal and spatial dimensions [24]. However infrequent events such as a change in room schedules or sporting events may also impact DR decisions. This requires correlation of partially matched online queries with unexpected events, on-demand, for operational intelligence.

Other applications that motivate hybrid querying of data streams include realtime bidding (RTB) – the next breakthrough of mobile advertising, algorithmic financial trading and so on. In general, we notice that from a *user's perspective*, a query of interest in these applications can be either persistent, which requires continuous evaluation over new data, or it can be transient, which require one-time evaluation over "live" data observed on streams. On the other hand, from the *data's perspective*, all the applications deal with data that arrives and leaves on streams dynamically over time. Users are only interested in data items within a certain period of time, or in other word, within the data's life-cycle. This life-cycle can be a simple time window, or a complex set of correlation constraints.

## III. Event and Query Model

We propose a Complex Event Processing based approach for hybrid querying of realtime data. CEP systems are originally designed to process online queries over high-rate event streams. Since CEP queries persist intermediate matching states for incremental evaluation, we propose to leverage CEP query states for data life-cycle management and expose the dynamically persisted streams for on-demand querying. In addition, we also attempt to address data variety by providing semantically enriched queries.

In this section we present the event and unified query model, extended from traditional CEP models, for both online and on-demand query specification over heterogeneous data streams. In particular, we illustrate how this model can be used to support the Big Data application scenarios discussed in the previous section.

### A. Event Model

*1) Event Syntax:* Realtime data can be modeled as timestamped relational tuples or name-value pairs as below,

```
Event := {[name, value]*, timestamp}
```

To facilitate later discussions, we introduce an event operator $T(e)$ to get the timestamp property of an event $e$ represented using the above model. We introduce an event set operator $\varepsilon$ to get the set of temporally ordered events which exist in an event set or a set of event sets. In addition, event stream is formally defined as,

***Definition 1:*** A temporally ordered event set $S$ is said to be an **event stream**. A stream $S$ of events with attribute names $a_1, a_2...$ is denoted as $S(a_1, a_2, ...)$.

A few useful relations between event streams are identified as follows:

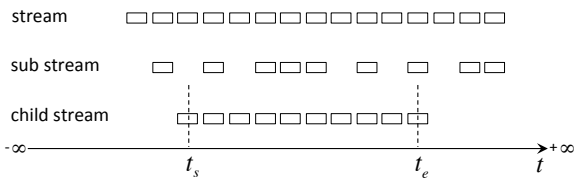

Figure 1: Stream Containment

***Definition** 2:* A stream $S'$ is said to be **contained by** stream $S$, if $\forall e \in S'$ we have $e \in S$ in the same temporal order. $S$ is said to be the **super stream** of $S'$ and $S'$ is the **sub stream** of $S$, denoted as $S \supseteq S'$ or $S' \subseteq S$.

***Definition** 3:* A temporarily ordered subset of stream $S$ from start time $t_s$ to end time $t_e$, denoted as $S_{t_s}^{t_e}$, is said to be a **child stream** of $S$, denoted as $S_{t_s}^{t_e} < S$, and $S$ is the **parent stream** of $S_{t_s}^{t_e}$, denoted as $S > S_{t_s}^{t_e}$. Particularly, $S_t^{+\infty}$ is said to be the **upper stream** of $S$ and $S_{-\infty}^{t}$ is said to be the **down stream** of $S$, at time $t$.

As shown in Figure 1, a child stream $S_{t_s}^{t_e}$ of $S$ is a special sub stream of $S$ and $S = S_{-\infty}^{+\infty}$. In later discussions, symbol $-\infty$ and $+\infty$ will be omitted.

*2) Event Semantics:* Realtime data may vary in both structural formats and semantic meanings. The tuple-based representation normalizes data formats. The semantic variety is reflected in the tuple representation by allowing names and values, which are either literals or domain concepts, to have related variants. Take realtime bidding as example, and assume a simplified RTB event captures *category*, *zip*, *price* attributes of an advertised item. Among these attributes, the value of *price* is a number, which intrinsically has arithmetic relations to other numbers. However the attribute name *price* itself can be equivalent to *cost* in the domain. On the other hand, the *category* attribute may have values *SUV* or *VAN*, which are conceptually subclasses of *vehicle* or *car*.

Capturing the backend knowledge of target domains is a prerequisite to query over data with semantic varieties. The semantics of numeric literals are built into computing systems. For events which compose domain-specific attributes, we have to provide an explicit knowledge model to capture data semantics. We have applied Semantic Web ontologies to model relevant concepts and their relationships in domains such as Smart Grid [23]. These domain ontologies are seamlessly integrated with the proposed event model and query model, as discussed later, to address data varieties.

### *B. Query Model*

We provide a unified model for both online queries which represent persistent user interests, and on-demand queries which represent transient user interests, over realtime events. Existing CEP query languages [3], [4], [18] extend relational query models with explicit temporal operators to facilitate query specification over event streams. We adopt the following query syntax with the key CEP query operators,

```
H2O Query :=
    SELECT(output definition)
    [FILTER(non-correlated constraints)]*
    [JOIN(attribute correlations)]*
    [SEQ(temporal order correlations)]*
    [WINDOW(temporal range correlations)]*
```

where the `SELECT` operator projects event attributes to query result; the `FILTER` operator specifies constraints over attributes of individual events; while `JOIN`, `SEQ` and `WINDOW` define attribute comparison, time ordering and bounding constraints across multiple events.

We formally define query constraints as,

***Definition** 4:* A **constraint** or **clause** $C$ specified in query $Q$ is an event evaluation expression defined using a query operator, denoted as $C \in Q$.

Here we assume each query is composed of a set of constraints with logic *AND* relations; otherwise the query is decomposed to multiple independent queries. We also note $R = Q \cup P$ if query $R$ contains and only contains constraints defined in query $Q$ and $P$.

We illustrate the various query constructs of the above model using online query examples from the motivating applications. Denote digital shopping stream as *DS(from, orderId, task, category)*, realtime bidding stream as *RTB(from, adId, action, category, zip, price)*, and meter reading stream as *MTR(from, buildingId, reading)*.

***Query** 1:* Simple Filtering. Notify whenever there is an automobile sale in the neighborhood area of zip code 90007 and with price less than $5,000.

```
SELECT ?a FILTER(?a.from=RTB)
FILTER(?a.category rdf:type auto:car)
FILTER(?a.zip geo:hasNeighbor geo:Z90007)
FILTER(?a.price<5000)
```

As shown in Query 1, *FILTER* constraints are represented as triple patterns which evaluate realtime events against literals such as *5000* or domain concepts such as *auto:car* and *geo:Z90007* using arithmatic or semantic predicates. The semantic predicates and concepts with namespaces are captured in domain ontology models, described previously.

The above query combined with the domain ontologies shield semantic varieties of underlying data from users. The ontology with name space "auto:" captures various concepts and their relations in the automobile domain. Events using any equivalent or subclass concept of *auto:car* can potentially match the high-level query.

***Query** 2:* Aggregation. Report when the last 10 minutes average power demand of building *EEB* exceeds 600KW.

```
SELECT AVG(?m.reading)>600
FILTER(?m.from=MTR) FILTER(?m.building=
EEB) WINDOW(?m,sliding,10min)
```

Query 2 shows the specification of an aggregation pattern and moving time window. A sliding time window is essentially a constraint which correlates event data with intrinsic world time events. Here, a meter reading event will fall out of the window and no longer be of concern when it was 10 minutes older than the current time.

***Query** 3:* Join. Report when an advertised item was purchased within one day.

```
SELECT ?p FILTER(?p.from,?s.from=RTB)
FILTER(?p.action=publish) FILTER(?s.action
=purchase) JOIN(?p.adId=?s.adId) WINDOW
(?w,sliding,1day) WINDOW(?e,sliding,10min)
```

Query 3 shows the specification of *JOIN* constraint which always appear with *WINDOW*s to correlate multiple events.

***Query** 4:* Sequence. Report orders which are successfully fulfilled within 2 hours, following a sequence of tasks including *order*, *deliver* and email *notice*.

```
SELECT ?w FILTER(?w.from,?d.from,?e.from=DS)
FILTER(?w.task=order) FILTER(?d.task=deliver
FILTER (?e.task=notice) SEQ(?w->?d(orderId=
?w.orderId)->?e(orderId=?w.orderId),2hour)
```

Query 4 shows the specification of a sequence data pattern on *DS* stream. The *SEQ* operator defines an ordered set of events. "−>" is a transitive predicate indicating the left-side event has a older timestamp than the right-side event.

On-demand queries have the same syntax as the online queries shown above. However on-demand queries summarize observed data rather than continuously evaluating new data arriving on streams. Due to the transient property of realtime data described in § I, it is not necessary to persist the entire data set observed on streams for on-demand querying. On the other hand, we propose a query-over-query scheme which allows perform on-demand queries over data of interest which is dynamically managed by online queries. To achieve this, next we introduce a formal query algebra to establish the relations between CEP queries.

## IV. Query Algebra

To support an interactive query processing model we study CEP query relations based on their intermediate and final matching results. We start with individual query operators and constraints, then extend discussions to full queries.

### A. Operator and Constraint Semantics

Between individual query constraints we have,

***Definition** 5:* A query constraint $C'$ is said to be **contained by** another constraint $C$ if $\forall E$, where $E$ is an event set, that satisfies $C'$, $E$ also satisfies $C$. $C$ is said to be the **super constraint** of $C'$, denoted as $C \supseteq C'$, and $C'$ be the **sub constraint** of $C$, denoted as $C' \subseteq C$.

Obviously super constraints are more relaxed than their sub constraints. For example, *FILTER(?a.price>5000)* is a super constraint of *FILTER(?w.price>1000)*; *FILTER(?a.category rdf:type auto:car)* is a super constraint of *FILTER(?a.type rdf:type auto:SUV)*; *WINDOW (?m,sliding,10min)* is a super constraint of *WINDOW(?m,sliding,5min)*; and *SEQ (?w−>?d)* is a super constraint of *SEQ (?w−>?d−>?e)*.

Online CEP queries are evaluated incrementally and events belonging to a result set do not necessarily arrive at the same time. This property is associated with query operators and we have the following notion,

***Definition** 6:* A CEP operator is **stateful** if evaluating constraints defined using it for a new event refers to any observed events. Otherwise it is said to be **stateless**.

Obviously, among operators described in § III *FILTER* is stateless, while *JOIN*, *SEQ* and *WINDOW* are stateful.

### B. Query Semantics

We extend the concept of statefulness and containment to full queries. Statefulness of an online CEP query indicates the necessity of the query to temporarily persist events during the matching process, while the containment property provides insights into the relations between partial and final matchings of different queries.

***Definition** 7:* A query is **stateful** if online evaluation of the query for a new event refers to previously observed events. Otherwise the query is said to be **stateless**.

Obviously an online query is stateful if it contains any stateful operators, otherwise it is stateless. Next we formally define query result sets before studying query containment.

***Definition** 8:* An event set $M$ is said to be a **match** of query $Q$ on stream $S$, denoted as $M \in Q(S)$, if $M$ fulfills all query constraints in $Q$, and if any event in $M$ was removed, $M$ no longer fulfills $Q$. An event set $M$ is said to be a **prior-match** of query $Q$ on stream $S$ at $t$, denoted as $M \in \overleftarrow{Q}_t(S)$, if $M \in Q(S)$, and $\forall e \in M$ we have $T(e) \leq t$. Similarly an event set $M$ is said to be a **post-match** of query $Q$ on stream $S$ at $t$, denoted as $M \in \overrightarrow{Q}_t(S)$, if $M \in Q(S)$, and $\forall e \in M$ we have $T(e) > t$. An event set $M$ is said to be a **cross-match** or **eventual-match** of query $Q$ on stream $S$ at $t$, denoted as $M \in \overleftrightarrow{Q}_t(S)$, if $M \in Q(S)$, and $\exists e \in M$ having $T(e) > t$, and $\exists e' \in M$ having $T(e') < t$.

Obviously we have

$$Q(S) = \overleftarrow{Q}_t(S) \cup \overrightarrow{Q}_t(S) \cup \overleftrightarrow{Q}_t(S) \tag{1}$$

To detect *eventual-matches*, at time $t$ an online query $Q$ has to manage a temporary storage of events which consists of all $e$ which satisfies $e \in \varepsilon(\overleftrightarrow{Q}_t(S))$ and $T(e) \leq t$. Since $\overleftrightarrow{Q}_t(S)$ is unknown at $t$, the storage condition has to be relaxed and $Q$ persists event sets which potentially match $Q$ after $t$. The potential matches are the *partial matches* or *states* of the query at $t$. The formal definition is below.

***Definition** 9:* An event set $N$ is said to be a **partial match** of query $Q$ on stream $S$ at time $t$, denoted as $N \in \bar{Q}_t(S)$, if $N \notin Q(S)$, $N \neq \varnothing$, $N$ does not violate any constraints of $Q$, removing an event from $N$ violates satisfied constraints, and adding any event $e$ to $N$ where $e \in S^t$ does not fulfill new constraints in $Q$. The set of all partial matches $\bar{Q}_t(S)$ is said to be the **state** of $Q$ at $t$.

To justify that the query states defined above contain all maximum subsets of *eventual-matches* of $Q$ at $t$, we have the following theorem, and its proof by contradiction,

***Theorem** 1:* For all $M \in \overleftrightarrow{Q}_t(S)$ we have $M - S_t \in \bar{Q}_t(S)$.

*Proof:* Suppose not. Assume, on the contrary, $\exists M \in \overleftrightarrow{Q}_t(S)$ we have $M - S_t \notin \bar{Q}_t(S)$. Obviously we have $M \in Q(S)$. Denote $M_t = M \cap S_t$ and $M^t = M - S_t$, we have $M = M_t \cup M^t$. From the supposition, we have $M^t \notin \bar{Q}_t(S)$, i.e., $M^t$ is not a partial match of $Q$ at $t$. Based on the definition of *cross-match*, we already have $M^t \neq \varnothing$, $M^t \notin Q(S)$, and $M^t$ does not violate any constraints of $Q$. Based on the definition of *partial match*, either removing an event $e$ from $M^t$ does not violate satisfied constraints or adding an event $e$ where $e \in S^t$ to $M^t$ will fulfill additional query constraints. If it is the former case, we have $(M^t - e) \cup M_t = M - e$ is a match of $Q$ which contradicts with $M \in Q(S)$. If it is the later case, we have $(M^t + e) \cup M_t = M + e$ is a match of $Q$ which also contradicts with $M \in Q(S)$. Hence the supposition $M - S_t \notin \bar{Q}_t(S)$ is false and the theorem is true. ■

From Theorem 1, we in addition have,

***Theorem** 2:* Let $\bar{Q}^t(S)$ be the state of query $Q$ at $t$, we have $\varepsilon(Q(S) - \overleftarrow{Q}(S)) \subseteq \varepsilon(\bar{Q}_t(S)) \cup S_t$.

Theorem 2 indicates all unknown matches of $Q$ at $t$ can be derived from the query state at $t$ and events arriving on stream after $t$. In other words, it is sufficient for an online query $Q$ to only persist states given by Definition 9 to detect matches after $t$. The proof of Theorem 2 is straightforward.

Existing CEP systems implement online pattern matching based on the fact formally described in Theorem 2, while query states have been isolate and hidden. As we described in § II, additional insights to realtime data can be extracted or queried over intermediate query states on-demand. To formalize such query-over-query scheme, we study the relations between CEP queries.

***Definition** 10:* A query $Q'$ is said to be **contained by** another query $Q$, if $\forall$ stream $S'$, $\exists S \supseteq S'$ having $\forall M' \in Q'(S')$, $\exists M \in Q(S)$ and $M' \subseteq M$. $Q$ is said to be the **super query** of $Q'$, denoted as $Q \supseteq Q'$, and $Q'$ be the **sub query** of $Q$, denoted as $Q' \subseteq Q$.

The definition indicates for any match event set of a sub query $Q'$, we can find a complimentary event set which makes the union of the two sets a match of its super query $Q$. Obviously the containment relation is transitive and symmetric. In addition, it has the following property,

***Theorem** 3:* Let $Q' \subseteq Q$, $\forall$ query $D$ we have $D(\varepsilon(\bar{Q'}_t(S))) \subseteq D(\varepsilon(\bar{Q}_t(S)))$.

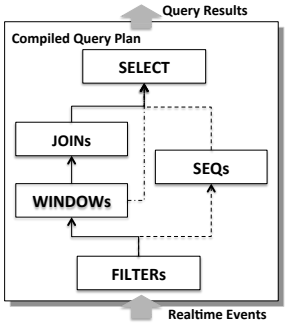


Figure 2: Pipelined Query Plan

*Proof:* We only need to prove $\varepsilon(\bar{Q'}_t(S)) \subseteq \varepsilon(\bar{Q}_t(S))$. Suppose not. Assume, on the contrary, $\exists e \in \varepsilon(\bar{Q'}_t(S))$ and $e \notin \varepsilon(\bar{Q}_t(S))$. Denote $M' \in \bar{Q'}_t(S)$ as one *partial match* of $Q'$ at $t$ with $e \in M'$. Based on Definition 10, we have an event set $N$ which makes $M' \cup N \in Q(S)$. From Theorem 1, $M' \cup N - S_t \in \bar{Q}_t(S)$. Since $e \notin S_t$ we have $e \in M' \cup N - S_t$ and $e \in \varepsilon(\bar{Q}_t(S))$ which contradicts with the supposition. Hence the supposition is false and the theorem is true. ■

Theorem 3 indicates given any on-demand query $D$ at time $t$, one can retrieve the patterns which satisfy $D$ and potentially match $Q'$ from the result event sets of applying $D$ over the state of a super query of $Q'$ at $t$. This allows use a single query persisting dynamic matching states for its sub queries for on-demand querying.

To determine containment relations we obviously have,

***Theorem** 4:* Query $Q$ is a super query of $Q'$ if and only if any constraint of $Q'$ has a super constraint in $Q$.

## V. Processing Model

Having described the unified query model and algebra, in this section we present approaches to process online and on-demand queries respectively.

### A. Online Query Processing

In the *H2O* model, online queries evaluate new data continuously arriving on streams and persist states dynamically.

*1) Query Evaluation:* Online query evaluation implements a data-driven paradigm with pipelined operators as shown in Figure 2. We classify online queries as queries with *WINDOW* operators and queries with *SEQ* operators. *WINDOW* queries may also have optional *JOIN* operators.

For a given query, stateless *FILTERs* are firstly evaluated following an evaluation path. As depicted in Figure 3, filters associated with all queries form a filtering graph, where the root nodes are filters evaluating stream associations such as *FILTER(?a.from=RTB)* in Query 1 and leaf nodes are filtering outputs associated with particular event variables. The inner filters evaluate event attributes against literals or semantic concepts which may be shared across queries.

Depending on window types, a stateful *WINDOW* constraint is compiled to an event queue persisting observed events and evaluated on every new arriving event, including

system time events, to dynamically expire events that fall out of the window. *JOIN* constraints correlate events persisted in windows. *SEQ* constraints, similar to most existing CEP systems, can be processed with a Non-deterministic Finite Automata (NFA) based algorithm for incremental evaluation.

*2) Data Life-cycle Management:* One main objective of *H2O* is to achieve dynamic data life-cycle management to avoid storing the entire stream data for on-demand querying. The life-cycle of realtime data is managed based on online query state persistence. Since we adopt an in-memory storage model, although we use the word "state persistence", individual online queries do not physically store "live" events but only references to them. Physically, events for each stream are stored in separated event lists. The number of query states which reference a particular event is updated as the metadata of the events in realtime. When an event is evicted from the last referencing query state due to a violation of query constraints or match of a query, the event is removed from the in-memory storage. As such the system keeps at most one copy of an event and each online query can be considered as a *view* over the actual data store.

### B. On-demand Query Processing

On-demand queries are applied over online query views. We adopt an event replay approach for on-demand query evaluation since on-demand queries adopt the same temporal query models. Here, on-demand queries are compiled, similar to online queries, to pipelined query plans, with the difference that they do not monitor input data streams. When an on-demand query is issued over a base online query, an event stream which consists of events persisted in the current online query states will be reconstructed and fed to the on-demand query plan to generate results. An alternative approach worth mentioning is query rewriting. In this approach, query constraints defined using temporal operators including *WINDOW* and *SEQ* are rewritten into equivalent *FILTER* constraints so that the new query can be performed over persisted data directly.

### C. System Architecture

The proposed architecture of *H2O* is shown in Figure 3. It consists of three main components – the *domain ontology models*, the *query publisher* module and the core *query processing engine*.

The *domain ontology model*, represented in Semantic Web OWL ontology language, forms a knowledge-base of domain concepts and their relations to support semantic-level query specification and evaluation. Particularly for the Smart Grid DR application, we organize ontologies in a modular fashion for extensibility [23].

The *query publisher* module creates and publishes query plans for both online and on-demand queries. For an online query, the module compiles and deploys the query to the processing engine for continuously evaluation. For an on-demand query, this module compiles the query as query plan and replay corresponding state event sets for results.

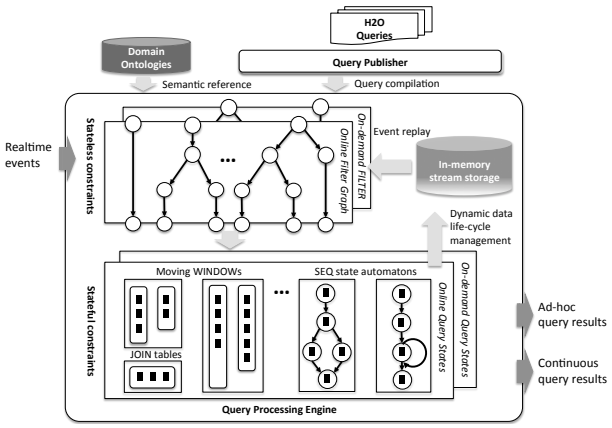


Figure 3: System Architecture.

The *query processing engine* implements the core algorithms and data structures for online query evaluation, query state persistence and on-demand query answering. On-line and on-demand queries are processed independently. The query engine consists of operator processors executing *constraints* in pipeline. These include the stateless filter graph processor, stateful window, sequence and join processors using techniques described in § V. A global dynamic stream storage is used to hold "alive" events for on-demand access. The data life-cycle is dynamically controlled by online query states as described previously.

We build the hybrid online on-demand query engine by extending an existing CEP engine Siddhi [18]. We are currently working on extending stateful processors for state persistence and access. Most of the other system building blocks are available. In our prior work [24], we have extended Siddhi to support ontology integration and semantic filter evaluation. The on-demand query processing is mostly a reuse of our event replay implementation described in [25].

## VI. Related Work

Our work falls in the space of stream computing and Complex Event Processing. These systems offer intuitive languages to query and react to realtime data with low latency. Stream computing systems such as Aurora [8], InfoSphere [6] and Storm [13] support filtering and aggregation of data streams using a workflow based paradigm. On the other hand, CEP systems like SASE [11], Cayuga [3] and Siddhi [18] typically follow a SQL-like query syntax, offering native temporal operators in addition to the relational operators for temporal pattern specification and matching. Both types of systems have been focusing on scalability and high availability for online query evaluation [5], [17]. *H2O* is however the first Big Data streaming system which aims to deal with all the 3-V's aspects. *H2O* incorporates

semantic domain knowledge to address data *variety*. We not only consider online querying over data streams but also from a completely new perspective, use CEP as a dynamic stream storage for on-demand querying.

Database systems which may seem appealing to implement the hybrid query model is also relevant. Distributed databases and in-memory databases [14], [7] support efficient on-demand queries at high reading rates. Realtime databases and active databases [15], [21] developed schedule or trigger mechanisms to perform online queries over dynamic data. We extend CEP to support hybrid online and on-demand queries with a query-over-query paradigm. This allows flexible stream storage provisioning which is extremely important for applications supporting heterogeneous data and users. In addition, our approach inherits intuitive temporal query specification and efficient online query processing capabilities from CEP systems. Different to relational and NoSQL query models, CEP provides explicit temporal operators such as time window and temporal order to correlate realtime data. Designated algorithms such as NFA are used for high-throughput query matching over data streams. Realtime and active databases can perform online relational queries, but do not scale to such high-rate inputs or queries. Stream databases such as [9], [22] offer powerful query languages with sliding windows and sequence operators but also do not scale to high data rate.

## VII. Conclusions

In this paper, we introduce a stateful Complex Event Processing framework to meet the needs of emerging realtime Big Data applications across all 3-V's. The proposed system supports semantic-level query specification over realtime data to deal with data *variety*. It inherits CEP systems' high performance online querying capabilities to deal with data *velocity*. Furthermore, we extend traditional CEP systems to support on-demand querying to deal with data *volume* on streams. The formal query algebra we proposed captures the stateful and containment semantics of realtime data queries, which form the basis for the hybrid query paradigm. Various query processing techniques we introduce work toward efficient execution of this hybrid model, including online query evaluation, query-based stream storage provisioning, dynamic data life-cycle management, and on-demand query evaluation.

There are several implementation issues to be considered further as we complete our prototype based on these formal abstractions. One of them is the scalability of the in-memory state persistence model. When the stateful CEP system processes a large number of long running online queries, the system memory may be drained out. To address this issue, we are exploring scale-out opportunities by distributing the *H2O* query model over the network. The detailed performance evaluation will further validate our proposed models and optimizations.

## References


[1] Big data gets real-time. *Oracle White Paper*, 2013.

[2] A. Adi and et al. Complex event processing for financial services. In *IEEE Services Computing Workshops*, 2006.

[3] A. Demers, J. Gehrke and et al. Cayuga: A general purpose event monitoring system. In *CIDR*, 2007.

[4] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. Stream reasoning and complex event processing in etalis. *Semantic Web Journal*, 2012.

[5] N. Backman, R. Fonseca, and U. Çetintemel. Managing parallelism for stream processing in the cloud. In *International Workshop on Hot Topics in Cloud Data Processing*, 2012.

[6] A. Biem and et al. IBM infosphere streams for scalable, real-time intelligent transportation services. In *SIGMOD*, 2010.

[7] R. Cattell. Scalable sql and nosql data stores. *SIGMOD Record*, 39(4), 2011.

[8] D. Abadi and D. Carney et al. Aurora: a data stream management system. In *SIGMOD*, 2003.

[9] D. Carney, U. Cetintemel and et al. Monitoring streams a new class of data management applications. In *VLDB*, 2002.

[10] H. Garcia-Molina, J. D. Ullman, and J. Widom. Database systems: The complete book. *Prentice Hall Press*, 2008.

[11] D. Gyllstrom, E. Wu, and et. al. SASE: Complex event processing over streams. In *the 3rd Conference on Innovative Data Systems Research*, 2007.

[12] J. Han, E. Haihong, G. Le, and J. Du. Survey on nosql database. In *International Conference on Pervasive Computing and Applications (ICPCA)*, 2011.

[13] M. T. Jones. Process real-time big data with twitter storm. *IBM Technical Library*, 2013.

[14] H. Joshi and G. R. Bamnote. Distributed database: A survey. In *International Journal Of Computer Science And Applications*, 2013.

[15] B. Kao and H. Garcia-molina. An overview of real-time database systems. In *Advances in Real-Time Systems*, 1994.

[16] D. Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, 2001.

[17] S. S. Loesing, M. Hentschel, T. Kraska, and D. Kossmann. Stormy: an elastic and highly available streaming service in the cloud. In *Joint EDBT/ICDT Workshops*, 2012.

[18] S. Suhothayan, K. Gajasinghe and et al. Siddhi: A second look at complex event processing architectures. In *ACM GCE Workshop*, 2011. http://siddhi.sourceforge.net.

[19] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *IEEE Symposium on Mass Storage Systems and Technologies*, 2010.

[20] Y. Simmhan, S. Aman, A. Kumbhare, R. Liu, S. Stevens, Q. Zhou, and V. Prasanna. Cloud-based software platform for data-driven smart grid management. *CiSE*, 2013.

[21] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *SIGMOD*, 1992.

[22] Y. Law and et al. Query languages and data models for database sequences and data streams. In *VLDB*, 2004.

[23] Q. Zhou, S. Natarajan, Y. Simmhan, and V. Prasanna. Semantic information modeling for emerging applications in smart grid. In *ITNG*, 2012.

[24] Q. Zhou, Y. Simmhan, and V. Prasanna. Incorporating semantic knowledge into dynamic data processing for smart power grids. In *International Semantic Web Conference*, 2012.

[25] Q. Zhou, Y. Simmhan, and V. Prasanna. Scepter: Semantic complex event processing across realtime and persistent streams. *IEEE Transactions on Computers*, 2013. Under review.