The VLDB Journal (2013) 22:421–446
DOI 10.1007/s00778-012-0297-3

REGULAR PAPER

Modeling the execution semantics of stream processing engines with SECRET

Nihal Dindar · Nesime Tatbul · Renée J. Miller · Laura M. Haas · Irina Botan

Received: 23 December 2011 / Revised: 20 September 2012 / Accepted: 24 September 2012 / Published online: 20 December 2012

Abstract There are many academic and commercial stream processing engines (SPEs) today, each of them with its own execution semantics. This variation may lead to seemingly inexplicable differences in query results. In this paper, we present SECRET, a model of the behavior of SPEs. SECRET is a descriptive model that allows users to analyze the behavior of systems and understand the results of window-based queries (with time- and tuple-based windows) for a broad range of heterogeneous SPEs. The model is the result of extensive analysis and experimentation with several commercial and academic engines. In the paper, we describe the types of heterogeneity found in existing engines and show with experiments on real systems that our model can explain the key differences in windowing behavior.

Keywords Data streams · Continuous queries · Stream processing engines · Semantic heterogeneity

N. Dindar (✉)· N. Tatbul · I. Botan
ETH Zurich, Zurich, Switzerland
e-mail: dindarn@inf.ethz.ch

N. Tatbul
e-mail: tatbul@inf.ethz.ch

I. Botan
e-mail: irina.botan@inf.ethz.ch

R. J. Miller
University of Toronto, Toronto, Canada
e-mail: miller@cs.toronto.edu

L. M. Haas
IBM Almaden Research Center, San Jose, CA, USA
e-mail: lmhaas@us.ibm.com

1 Introduction

Stream computing is passing from the domain of pure research into the real world of commercial systems. Many research projects (e.g., [1,6,17], and others) have shown how data can be processed as it pours into a system from a diversity of sources such as sensors, online transactions, and other feeds. Each system proposed its own set of operators, windowing constructs, and, in some cases, whole new query languages (e.g., [2,10]). As these systems have been commercialized [7,25,26], they have added features to meet the needs of their own customers. There are no standards today for querying streams; each system has its own semantics and syntax. For the purchaser or user of an SPE, the choices are confusing. Without a clear understanding of features and semantics, applications are not portable and can be hard to build, even on a given SPE.

Even common capabilities may be expressed differently by different SPEs. For example, both StreamBase [25] and Coral8 [7] allow time-based windows where a window is defined by an interval size (in units of time) and where different windows are separated by a slide value that specifies how many units of time separate the start of different consecutive windows. To specify such a window in StreamBase, the user has to write "`[SIZE x ADVANCE y TIME]`". In Coral8, the same function is requested with the "`KEEP x SECONDS`" clause. StreamBase allows an arbitrary slide value for a window (specified by the `ADVANCE` clause); Coral8 only permits two values: 1 time unit or a slide that is equal to the window size. Worse yet, the underlying semantics of such common features as windows is often radically different. Even if we set window size and slide to the same values in Coral8 and StreamBase, we can get different query results due to hidden differences in their query execution models.

Recently, a few abstract models for streams and windows have been proposed [13,14,20], for the most part not associated with any existing system. These models define only a portion of the behavior expected of an SPE. While they are useful as guides to future SPE developers, they do little to help users understand existing SPEs, and even less for comparing or explaining the behaviors of different SPEs.

We have proposed a general model, SECRET [5], for describing and predicting the behavior of these diverse systems. Our model is a descriptive model, not another execution model. It strives to explain, and to allow the comparison of, the differing behaviors found in existing SPEs. The model is the result of detailed analysis and experimentation with a set of real commercial and academic systems. We believe that our unique approach of creating a descriptive and explanatory model offers significant benefits to potential users of stream systems, both before and after they choose an engine for building their applications.

In this paper, we extend SECRET with tuple-based windows and add a new commercial system, Oracle CEP [19] to the experiments. These extensions are added with careful and minimal changes to the base model, staying loyal to its core design principles. We preserve the four basic dimensions of SECRET, but reformulate their definitions to capture a new behavior of Oracle CEP, and to support tuple-based windows, taking into account the change in windowing domain (from time to tuple-id's) as well as the "evaporating tuples" [12] behavior of time-driven SPEs. Last but not least, we present a wide range of experiments systematically showing that SECRET can correctly predict SPEs' time- and tuple-based window execution semantics under various input and query settings. These changes validate SECRETs extensibility, expressivity, and simplicity.

The next section illustrates the differences in features and semantics of several SPEs. With these differences as motivation, Sect. 3 presents the basics of our proposed model, SECRET. We detail SECRET for time-based windows in Sect. 4 and for tuple-based windows in Sect. 5. Section 6 demonstrates, through an extensive set of examples run on different engines, how our model predicts the results that similar queries will generate for the different systems. In Sect. 7, we provide an assessment of SECRET's design principles as well as discussing its potential extensions and uses. Related work is covered in Sect. 8. Finally, we conclude in Sect. 9 with a discussion of future work.

2 Motivation

Heterogeneity across SPEs comes in three forms:

1. Syntax heterogeneity: Since there is no standard language for stream processing, different SPEs use different language clauses (keywords) to define common constructs (e.g., windows).

2. Capability heterogeneity: SPEs vary in their support for different query types. This variance is also exposed at the language syntax level. For example, Coral8 offers a clause that controls how often a query result should be emitted, a feature we have not encountered in any other system.

3. Execution model heterogeneity: Below the language level, hidden from application developers, each SPE executes queries based on an underlying query execution model. Differences in these models are subtle and hence may be especially confusing. As a result, we focus on analyzing the execution semantics of SPEs in this paper.

To motivate the type of descriptive model we propose, consider the following three examples, defined on a simple input stream `InStream(Time, Val)` of tuples. `Time` represents the application timestamp of the tuple in seconds, and `Val`, an integer value, represents the content of the tuple. Our queries compute an average over `Val`, and `OutStream(Avg)` is the output stream containing the results of the query.

Example 1 Differences in window construction

Consider a query which continuously computes the average value of the tuples in the input stream using a time-based tumbling window of size 3 s.[1] We ran this query in two different SPEs: Oracle CEP [19] and StreamBase [25], with the following results:

```
InStream(Time, Val) = {(10,10),(11,20),(12,30),(13,40),
                       (14,50),(15,60),(16,70),...}
Oracle CEP(Avg)     = {(20),(50),...}
StreamBase(Avg)     = {(15),(40),...}
```

Intuitively, we expected to see the result of Oracle CEP in both engines (i.e., the first three tuples belong to the first window, the next three to the second window, etc.). However, StreamBase produced a different result (i.e., the first two tuples belong to the first window, the next three to the second window, etc.). Given the simplicity of the input and the query, this points to an important difference in the way these engines construct their windows.

Example 2 Differences in window evaluation

Consider a query which continuously computes the average value of tuples over a time-based window of size 5 s that slides by 1 s. We ran this query in four different SPEs: Coral8 [7], Oracle CEP [19], STREAM [23], and StreamBase [25], with the following results:

```
InStream(Time, Val) = {(30,10),(31,20),(36,30),...}
Coral8 (Avg)        = {(10),(15),(20),...}
Oracle CEP(Avg)     = {(10),(15),(20),...}
STREAM(Avg)         = {(10),(15),(20),...}
StreamBase(Avg)     = {(10),(15),(15),(15),(15),(20),...}
```

[1] In a *tumbling* window, the size of the window is equal to its slide.

StreamBase produced a different result than the other SPEs. Why? In Coral8, Oracle CEP, and STREAM, the average operator is invoked on a window whenever the window's content changes (i.e., when a tuple is added to or expires from the window), whereas in StreamBase, the invocation happens every second, even if the tuple content of the window stays the same. As a result, the first two input tuples are aggregated multiple times in StreamBase as opposed to once in the other SPEs. Thus, the evaluation strategy used by an SPE is another important factor affecting the query results.

Example 3 Differences in processing granularity
Consider a query which computes the average value of tuples over a tuple-based tumbling window of size 1 tuple.We ran this query in the same four SPEs with the following results:

```
InStream(Time, Val) = {(10,10),(10,20),(11,30),(12,40),
                       (12,50),(12,60),(12,70),(13,80),
                       (14,90),(15,100),...}
Coral8(Avg)         = {(10),(20),(30),(40),(50),(60),
                       (70),(80),(90),...}
Oracle CEP(Avg)     = {(20),(30),(70),(80),(90),...}
STREAM(Avg)         = {(20),(30),(70),(80),(90),...}
StreamBase(Avg)     = {(10),(20),(30),(40),(50),(60),
                       (70),(80),(90),...}
```

We observed two different result sets. Coral8 and StreamBase produced a result for every tuple, while Oracle CEP and STREAM produced results for only a subset of the tuples. Why? Coral8 and StreamBase react to each tuple arrival separately and produce a result, whereas Oracle CEP and STREAM react to each application timestamp, choosing a single tuple with that timestamp in this example, since the window size is 1. Simply, the same query is executed as "compute an average value for every tuple" in Coral8 and StreamBase, and "compute an average value for the most recent tuple" in Oracle CEP and STREAM, leading to different results. Thus, the processing granularity used by an engine is another important factor affecting the query result.

The examples above (selected from many we have analyzed) show that we need a way to understand, express, and predict the query execution behaviors of different SPEs. Our model, SECRET, takes up this challenge.

3 SECRET model basics

This section introduces our model. We start by defining some basic terms and then give an overview of the model. Sections 4 and 5 will go into detail for time- and tuple-based windows, respectively.

We start with definitions for a set of basic stream processing concepts and constructs that we use in our model, together with any relevant assumptions we make.

Definition 1 (*Time Domain*) The time domain $\mathbb{T}$ is a discrete, linearly ordered, countably infinite set of time instants $t \in \mathbb{T}$. We assume that $\mathbb{T}$ is bounded in the past, but not necessarily in the future. In order to simplify time arithmetic, we will also assume that the time domain is the domain of integers ($\mathbb{T} = \mathbb{Z}$).

Definition 2 (*Stream*) A stream $\mathbb{S}$ is a countably infinite set of elements $s \in \mathbb{S}$. Each stream element $s : \langle v, t^{app}, t^{sys}, tid, bid\rangle$ consists of a relational tuple v conforming to a schema S, with an application time value $t^{app} \in \mathbb{T}$, a system time value $t^{sys} \in \mathbb{T}$, a tuple-id value $tid \in \mathbb{N}^+$, and a batch-id value $bid \in \mathbb{N}^+$ (see below for the definition of "batch"). We use the notation $s.t^{app}$, $s.t^{sys}$, $s.tid$, and $s.bid$ to denote the application time value, system time value, tuple-id value, and batch-id value of stream element s, respectively. We assume that elements of a stream $\mathbb{S}$ are totally ordered by their t^{sys} and tid values (where the two orderings should agree with each other), while they are partially ordered by their t^{app} and bid values.

In the above definition (as in related work [22]), we have used two different notions of time: "application time" (t^{app}) and "system time" (t^{sys}). These both take values from our time domain $\mathbb{T}$, but carry two different meanings, and therefore are used for two different purposes in our model.The value t^{app} captures the time information that is associated with the occurrence of the application event that a stream element represents (usually provided by the data source) and therefore will be used as the basis for query execution over the stream, whereas t^{sys} captures the time information that is associated with the occurrence of the related system event (arrival of the corresponding stream element at the system) and therefore will be used as the basis for reasoning about tuple arrival events in the system and how the system should react to them. Elements in a stream are assigned unique t^{sys} values, but multiple elements can share the same t^{app} value. Therefore, streams are totally ordered by the t^{sys} values of their elements, whereas they are partially ordered by their t^{app} values. Similarly, elements in a stream are assigned unique tid values, and therefore, streams are totally ordered by the tid values of their elements.

Definition 3 (*Batch*) A batch $\mathbb{B}$ of stream elements for a given stream $\mathbb{S}$ is a finite subset of $\mathbb{S}$, where all $b \in \mathbb{B}$ have an identical t^{app}. Each such batch is given a unique batch-id $bid \in \mathbb{N}$ such that, for all $b \in \mathbb{B}$, $b.bid = bid$, indicating that b belongs to the batch that is uniquely identified by bid. For tuples t_1 and t_2 where $t_1.t^{sys} < t_2.t^{sys}$, then $t_1.bid \leq t_2.bid$.

Batches are used to define a further ordering among simultaneous tuples [12]. By definition, all tuples in a given batch have the same t^{app} value, but that does not mean that *all* tuples with the same t^{app} value are in the same batch. For example, we can have four tuples with $t^{app} = 5$ in two consecutive batches of two tuples each. Therefore, a new batch can arrive without t^{app} advancing. This implies that streams are also partially ordered by their bid values.

Definition 4 (*Window*) A window W over a stream $\mathbb{S}$ is a finite subset of $\mathbb{S}$.

Windows can be defined in many ways. In this paper, we will mainly focus on "time-based windows" and "tuple-based windows". In time-based windows, stream elements whose t^{app} values fall into a certain t^{app} interval constitute a window. Likewise, in tuple-based windows, stream elements whose tid values fall into a certain tid interval constitute a window. More formally:

Definition 5 (*Time-based Window*) A *time-based window* $W = (o, c]$ over a stream $\mathbb{S}$ is a finite subset of $\mathbb{S}$ containing all data elements $s \in \mathbb{S}$ where $o, c \in \mathbb{T}$ and $o < s.t^{app} \le c$.

Definition 6 (*Tuple-based Window*) A *tuple-based window* $W = (o, c]$ over a stream $\mathbb{S}$ is a finite subset of $\mathbb{S}$ containing all data elements $s \in \mathbb{S}$ where $o, c \in \mathbb{Z}$ and $o < s.tid \le c$.

In general, systems do not process arbitrary sets of windows, but rather require the windows to have a specific relationship to each other defined by two parameters, *size* (ω) and *slide* (β). More formally:

Definition 7 (*Window Size and Slide*) The set $\mathbb{W}$ of all windows defined over a stream $\mathbb{S}$ must satisfy the following three constraints:

1. Size(ω): All windows must be the same size. That is: For time-based windows where $o, c, \omega \in \mathbb{T}$ and $\omega > 0$, $\forall\ W = (o, c] \in \mathbb{W}, c - o = \omega$. For tuple-based windows where $o, c, w \in \mathbb{Z}$ and $\omega > 0$, $\forall\ W = (o, c] \in \mathbb{W}$, $c - o = \omega$.
2. Slide(β): The distance between consecutive windows must be the same. For two windows $W_1 = (o_1, c_1]$ and $W_2 = (o_2, c_2]$, we require that $o_1 \neq o_2$. Furthermore, we say W_1 and W_2 are consecutive if $o_1 < o_2$ and there is no window $W' = (o', c']$ such that $o_1 < o' < o_2$. For all consecutive windows W_1 and W_2 in $\mathbb{W}$, we require that: For time-based windows where $o_2, o_1, \beta \in \mathbb{T}$ and $\beta > 0$, $o_2 - o_1 = \beta$. For tuple-based windows where $o_2, o_1, \beta \in \mathbb{Z}$ and $\beta > 0$, $o_2 - o_1 = \beta$.
3. Slide should not be greater than size ($\beta \le \omega$).

At $t^{app} = t$, we say a time-based window $W = (o, c]$ is *open*, if $o < t \le c$. A window is *closed*, if $c < t$. Similarly, at $tid = i$, we say a tuple-based window $W = (o, c]$ is *open*, if $o < i \le c$. A window is *closed*, if $c < i$.

We are now ready to describe our model. We named our model SECRET, as it captures window-based query execution semantics along four complementary dimensions: ScopE, Content, REport, and Tick. Given a query's window parameters, $\underline{S}$cop$\underline{E}$ provides information about potential window intervals. $\underline{C}$ontent then helps us map those intervals into actual window contents, for a given input stream. $\underline{RE}$port states under what conditions those window contents become visible to the query processor for evaluation. Finally, $\underline{T}$ick models what drives an SPE to take action on a given input stream. Tick is the actual entry point to the control loop of our model, creating a chain reaction by invoking Report, which in turn invokes Content, which builds on Scope (Tick → REport → Content → ScopE).

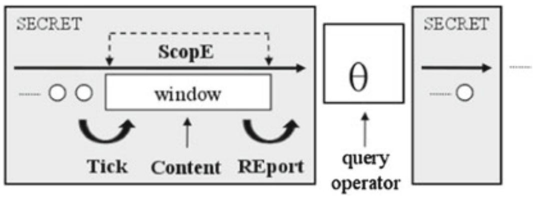

Fig. 1 SECRET of a query plan

We have designed SECRET based on a number of principles. First of all, SECRET must be **expressive** so that it can capture the key behaviors of a broad range of stream systems. Second, it should be **simple** (i.e., easy to understand and to apply, avoiding complicated and redundant features). Third, the features should be **orthogonal** to each other. Furthermore, SECRET should be **extensible**, offering the ability to add new features if necessary as new SPEs or query types are encountered. Finally, for **clarity**, we want our model to separate the operational aspects of *how* the SPE processes streams from the non-procedural effects of that processing. For example, we should be able to talk about how windows are formed independently of their content. By contrast, when the system chooses to evaluate results depends heavily on its processing model. The model should also make a clear separation between data-level issues (e.g., values in a stream), query-level issues (e.g., window size) and system-level issues (e.g., when the engine takes an action).

Figure 1 illustrates how we use SECRET to explain the semantics of a given query plan. SECRET is compositional in the same way a query plan is composed of a sequence of operators. We next define each of the SECRET parameters in detail, first for time-based windows (Sect. 4) and then for tuple-based windows (Sect. 5).

4 SECRET for time-based windows

We define the SECRET parameters from Scope to Tick.

4.1 Scope

For a query q, the function *Scope* maps an application time value t to an interval over which q should be evaluated. We define the *active window* at time t as the open window at t with the earliest start time.

We assume a value $t_0 \in \mathbb{T}$ that denotes the application time instant of the start of the very first window in a given system. Its value is system specific, since different systems use a different starting point for their application time line.

Hence, the initial window (W_0) starts at time t_0, the next one (W_1) starts at time $t_0 + \beta$, and window i (W_i) starts at time $t_0 + i\beta$. Let $W_i = (o_i, c_i]$ be the i^{th} window in $\mathbb{W}$.

The index of the active window at time t, n, is given by:

$$n = max(0, \lceil \frac{t - t_0 - \omega}{\beta} \rceil)$$

This formula is obtained as follows: W_0 closes at $t_0 + \omega$; W_1 closes at $t_0 + \omega + \beta$; and W_n closes at time $t_0 + \omega + n\beta$. At time t, we are interested in the earliest open window, which is the smallest n that satisfies $t \le t_0 + \omega + n\beta$ (i.e., $n > (t - t_0 - \omega)/\beta$). Intuitively, $n = 0$ if the first window that opened at t_0 (and to be closed at $t_0 + \omega$) is still open. Otherwise, a new window has been opened every β time units. Then to find n for the earliest open window at t, we need to divide the total elapsed time since the close of the first window (i.e., $t - (t_0 + \omega)$) by β (and round it up to get a whole number).

Hence, the start time of $W_n = (o_n, c_n]$ is $o_n = t_0 + n\beta$, and $Scope$ at time t ($Scope : \mathbb{T} \rightarrow (\mathbb{T}, \mathbb{T}]$) is defined as follows:

$$Scope(t) = \begin{cases} \emptyset & \text{if } t < t_0 \\ (o_n, t] & \text{otherwise} \end{cases}$$

Figure 2 illustrates our $Scope$ formulation for time-based windows. As a simple example, assume we have a query q with a window of size 5 s and of slide 2 s, to be run on a system with t_0 of 30 s. Then the window scope at $t = 34$ s is $Scope(34) = (30, 34]$, since $n = 0$ and $o_0 = 30$.

There are a few important points to note about $Scope$:

1. The scope of a window for a given application time solely depends on an SPE's t_0 parameter and the query's window parameters (which define $\mathbb{W}$). All these parameters are non-operational.

2. During our analysis, we observed that systems may interpret the window slide value in two different ways, leading to two different window construction mechanisms. Some construct their windows at every slide in the backward direction, i.e., every new slide signals the end of a window which started ω time units ago [2,20]. Others construct their windows at every slide in the forward direction, so every new slide signals the beginning of a new window [1,25]. The window scopes produced for these two alternative interpretations differ only by a fixed amount δ, and therefore, one can choose one of these models and calibrate the starting time of the very first window t_0 by δ in case the other model's behavior is desired. As a result, t_0 is allowed to take negative values.

3. Our Scope formula focuses on the time interval for the active window. This is one of many ways one could define Scope. Some previous work defines Scope to be the time interval for the most recently closed window [2,20]. One could also define $Scope$ as the set of intervals for all open windows. We need a general and flexible $Scope$ definition that could be used to explain the behavior of systems that report their results on partial windows as well as those that do so on closed windows only.

In SECRET, we use the forward interpretation of slide, since it is finer grained than its backward counterpart and therefore enables reporting of partial as well as full windows.

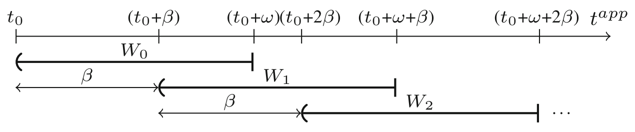

Fig. 2 Scope of a time-based window

4.2 Content

$Scope$ defines the interval for query evaluation at application time t. $Content$ specifies the set of elements of stream $\mathbb{S}$ that are in this scope. As such, $Content$ makes the mapping from the application time interval of a window to a set of data elements. We can formally define the content of a time-based window ($Content : \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{S}$) at application time instant t and system time instant τ as follows:

$$Content(t, \tau) = \{s \in \mathbb{S} : s.t^{app} \in Scope(t) \wedge s.t^{sys} < \tau\}$$

Note that unlike $Scope$, the result of $Content$ depends on actual contents of the input stream, which only become available at run time. Therefore, $Content(t, \tau)$ may return different results for the same t value, depending on how much of the input stream is already available (determined by τ) when it is invoked.

4.3 Report

The Report dimension in our model defines the conditions under which the window contents become visible for further query evaluation and result reporting. SPEs use different reporting strategies as illustrated in Example 2 of Sect. 2. We have identified four basic reporting strategies.

1. *Content change* (R_{cc}): reporting is done for application time t, only if the content has changed since last reporting. Given a specific system time instant τ, we represent last reporting as of τ with a pair $< t^{app}, t^{sys} >$ that corresponds to the most recent Report invocation that the reporting condition was satisfied. More formally:

$$last\_rep(\tau) = max\_pair\{< x, y > | \\ y \le \tau \wedge Report(x, y).second = 1\}$$

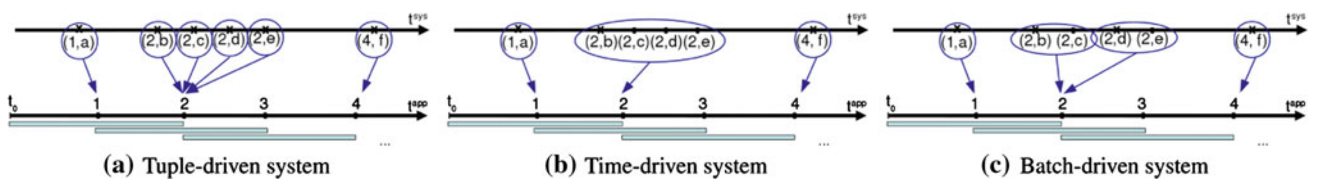

Fig. 3 Tick models for time-based windows

where, given a set P of unique $< t^{app}, t^{sys} >$ pairs, max_pair returns the most recent of them based on the following:

$$max\_pair(P) = \begin{cases} max\_pair\_ne(P) & \text{if} P \neq \emptyset \\ < t_0, \tau_0 > & \text{otherwise} \end{cases}$$

$$\begin{aligned} max\_pair\_ne(P) = \{p \in P | \ \ \forall p' \in P, p' \neq p \wedge \\ (p'.t^{app} < p.t^{app} \vee \\ (p'.t^{app} = p.t^{app} \wedge p'.t^{sys} < p.t^{sys}))\} \end{aligned}$$

Above, τ_0 represents an initial system time value that is smaller than the t^{sys} of the very first tuple in the stream.

2. *Window close* (R_{wc}): reporting is done for application time t, only when the active window closes.
3. *Non-empty content* (R_{ne}): reporting is done for application time t, only if the content at t is not empty.
4. *Periodic* (R_{pr}): reporting is done for application time t, only if it is a multiple of a given reporting frequency, λ.

Furthermore, some systems use multiple strategies (e.g., the content must have changed and be non-empty). Hence, we will use four boolean variables (R_{cc}, R_{wc}, R_{ne}, R_{pr}), each of which can be set to true or false by a system. When all four variables are false, it is interpreted as there is no condition on reporting; therefore reporting of $Content(t, \tau)$ takes place every time it is triggered by the previous step of the model. This is the default behavior in our SECRET model.

Given the above, we define Report for time-based windows ($Report : \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{S} \times \mathbb{N}$) as follows:

$$Report(t, \tau) = \begin{cases} (Content(t, \tau), 1) & \text{if } (\neg R_{cc} \vee Content(t, \tau) \neq Content(\\ & \quad last\_rep(\tau).t^{app}, last\_rep(\tau).t^{sys})) \\ & \wedge(\neg R_{wc} \vee (|Scope(t)| = \omega \wedge \\ & \quad t < max\{s.t^{app} | s \in S \wedge s.t^{sys} \leq \tau\})) \\ & \wedge(\neg R_{ne} \vee Content(t, \tau) \neq \emptyset) \\ & \wedge(\neg R_{pr} \vee mod(t, \lambda) = 0) \\ (\emptyset, 0) & \text{otherwise} \end{cases}$$

Report returns a pair of values (a,b), (a) content of the window or empty set and (b) whether the reporting condition is satisfied (1) or not (0). Please note that the latter values is required in the last rep formula of the content-change reporting strategy (R_{cc}).

4.4 Tick

Tick defines the condition which drives an SPE to take action on its input (also referred to as "window state change" or "window re-evaluation" [12]). Like $Report$, $Tick$ is part of a system's internal execution model. While some systems react to individual tuples as they arrive, others collectively react to all or subsets of tuples with the same t^{app} value. During our analysis, we have identified three main ways that systems "tick": (a) tuple-driven, where each tuple arrival causes a system to react; (b) time-driven, where the progress of t^{app} causes a system to react; (c) batch-driven, where either a new batch arrival or the progress of t^{app} causes a system to react.[2] These different Tick behaviors for time-based windows are illustrated in Fig. 3. We show two time lines for t^{sys} and t^{app}. Tuple arrivals are shown on the time line for t^{sys}, and window scopes are shown underneath, on the time line for t^{app}. Circles around the tuples show the units of tuples that the system will react to at one time, whereas the arrows show to which application time instant those units belong. Note that the tuples are the same in all three figures and that the four tuples in the middle have the same t^{app} value.

The tick models described above are based on the detection of three events: new tuple arrival, the progress of application time, and new batch arrival. The detection of each of these events is really based on the detection of new tuple arrival, since both application time information as well as batch-id information are carried in the tuples. Every new tuple arrival can only be uniquely detected if we check whether the stream has a tuple corresponding to every system time instant (for which there can be either one or none).

Two key ideas helped us structure our formulation:

1. At every tick, SECRET needs to check the reporting condition. However, since $Tick$ operates on system time units and $Report$ for time-based windows operates also on application time units, we need a mapping between them. We achieve this mapping with five mapping functions (app, $prev\_app$, $batch$, $prev\_batch$, and $prev\_tick$). The mapping is purely based on what is observed in the input stream and not on any synchronization assumption between t^{sys} and t^{app} time lines.

[2] Remember from Definition 3 that a new batch can arrive without t^{app} advancing.

2. Since tick events can only be detected at new tuple arrivals, this will be a basic condition in our formulation. We must be able to account for irregularities in tuple arrival such as simultaneous tuples (i.e., multiple tuples with a common t^{app}) and gaps (i.e., absence of tuples at certain t^{app}). To detect simultaneous tuples, we need to be able to compare the current t^{app} with the previous tick time. To handle gaps, the arrival of a new tuple with t^{app} causes all application time instants between t^{app} and the previous tick time to invoke *Report* so that we do not miss any important application time instants. This is why we need mapping functions that map current instants as well as previous ones.

First, we define $S(\tau)$ and $S_I(\tau)$ as follows:

$S(\tau)$ denotes the set of tuples in stream S that has arrived through time instant τ.

$$S(\tau) = \{s \in S | s.t^{sys} \leq \tau\}$$

$S_I(\tau)$ denotes the set of tuples in stream S that has arrived at time instant τ. There can be at most one such tuple.

$$S_I(\tau) = \left\{s \in S | s.t^{sys} = \tau\right\}$$

We use the following mapping functions to define $Tick$ for time-based windows:

$app(\tau)$: Given a system time instant τ, returns the application time value of the tuple that has arrived at τ.

$$app(\tau) = \{s.t^{app} | s \in S_I(\tau) \wedge S_I(\tau) \neq \emptyset\}$$

$prev\_app(\tau)$: Given a system time instant τ, returns the application time value of the most recent tuple that has arrived before τ. If no such tuple exists, it returns t_0.

$$prev\_app(\tau) = max(max\{t_0, s.t^{app} | s \in S(\tau - 1)\})$$

$batch(\tau)$: Given a system time instant τ, returns the batch-id value of the tuple that has arrived at τ.

$$batch(\tau) = max\{s.bid | s \in S_I(\tau)\}$$

$prev\_batch(\tau)$: Given a system time instant τ, returns the batch-id value of the most recent tuple that has arrived before τ. If no such tuple exists, it returns b_0 (where b_0 represents an initial batch-id value that is smaller than the bid of the very first tuple in the stream).

$$prev\_batch(\tau) = max(b_0, max\{s.bid | s \in S(\tau - 1)\})$$

$prev\_tick(\tau)$: Given a system time instant τ, returns the application time value of the most recent tuple that has arrived before τ for which the result of the tick was non-empty. If no such tuple exists, it returns t_0.

$$prev\_tick(\tau) = \{max(t_0, app(max(x | x < \tau \wedge Tick(x) \neq \emptyset)))\}$$

Based on the above, we will now formulate the $Tick : \mathbb{T} \rightarrow \{\mathbb{S}\}$ of time-based windows for each tick model. All formulas follow a similar structure.

In a tuple-driven system, Tick is triggered under two conditions: (i) if a tuple arrives whose t^{app} is the same as the previous tick time, or (ii) if a tuple arrives whose t^{app} is greater than the previous tick time. The former ensures that the system reacts to each tuple in a simultaneous sequence, whereas the latter ensures that the system also reacts to the application time instants where there might be a gap.

$$Tick(\tau) = \begin{cases} \{Report(app(\tau), \tau).first\} & \text{if } S_I(\tau) \neq \emptyset \wedge \\ & prev\_tick(\tau) = app(\tau) \\ \bigcup_{x=prev\_tick(\tau)}^{x<app(\tau)} Report(x, \tau).first & \text{if } S_I(\tau) \neq \emptyset \wedge \\ & app(\tau) > prev\_tick(\tau) \\ \emptyset & \text{otherwise} \end{cases}$$

In a time-driven system, there is no need to react to each tuple in a simultaneous sequence separately, and therefore, the first condition in the tuple-driven case is skipped. On the other hand, the second condition needs to be triggered if a tuple with a new t^{app} arrives (which means that t^{app} has advanced, to which the system must react).

$$Tick(\tau) = \begin{cases} \bigcup_{x=prev\_tick(\tau)}^{x<app(\tau)} Report(x, \tau).first & \text{if } S_I(\tau) \neq \emptyset \wedge \\ & app(\tau) > prev\_app(\tau) \\ \emptyset & \text{otherwise} \end{cases}$$

Finally, a batch-driven system acts like a modified tuple-driven system. We need to check both the condition for simultaneous tuples as well as for a tuple with a new t^{app} arriving. The only difference is that we need to additionally check if the new tuple arrival initiates a new batch by checking if the new batch-id is greater than the previous one.

$$Tick(\tau) = \begin{cases} \{Report(app(\tau), \tau).first\} & \text{if } S_I(\tau) \neq \emptyset \wedge \\ & prev\_tick(\tau) = app(\tau) \wedge \\ & batch(\tau) > prev\_batch(\tau) \\ \bigcup_{x=prev\_tick(\tau)}^{x<app(\tau)} Report(x, \tau).first & \text{if } S_I(\tau) \neq \emptyset \wedge \\ & batch(\tau) > prev\_batch(\tau) \\ \emptyset & \text{otherwise} \end{cases}$$

In Table 1, we present a sample trace of the model for the scenario shown in Fig. 3.

The table shows when each of the three models triggers *Report* and with which time values. The time-driven model invokes *Report* only when time advances, once for each time point including the gap time ($t^{app} = 3$). The tuple-driven model invokes *Report* at every new tuple arrival. Finally, the batch-driven model invokes *Report* at every new batch arrival as well as time advance ($t^{app} = 3$).

5 SECRET for tuple-based windows

In this section, we will formally define the four SECRET dimensions for tuple-based windows. The core structure of our model stays the same, but the change in window domain

Table 1 Tick example for time-based windows

τ	S(v, t^{app}, t^{sys}, tid, bid)	tuple-driven	time-driven	batch-driven
$\tau = 30$	$(a, 1, 30, 1, 1)$	Report(x, 30), $\forall x \in [t_0, 1)$	Report(x, 30), $\forall x \in [t_0, 1)$	Report(x, 30), $\forall x \in [t_0, 1)$
$\tau = 40$	$(b, 2, 40, 2, 2)$	Report(x, 40), $\forall x \in [1, 2)$	Report(x, 40), $\forall x \in [1, 2)$	Report(x, 40), $\forall x \in [1, 2)$
$\tau = 45$	$(c, 2, 45, 3, 2)$	Report(2, 45)	-	-
$\tau = 60$	$(d, 2, 60, 4, 3)$	Report(2, 60)	-	Report(2, 60)
$\tau = 80$	$(e, 2, 80, 5, 3)$	Report(2, 80)	-	-
$\tau = 90$	$(f, 4, 90, 6, 4)$	Report(x, 90), $\forall x \in [2, 4)$	Report(x, 90), $\forall x \in [2, 4)$	Report(x, 90), $\forall x \in [2, 4)$

Table 2 Time-based versus tuple-based: number and granularity of report invocations per window domain value

	tuple-driven tick	time-driven tick	batch-driven tick
time-based window	multiple times per t^{app}	once per t^{app}	multiple times per t^{app}
tuple-based window	once per tid	**once per multiple** tid	**once per multiple** tid

requires us to formalize the dimensions slightly differently. We first discuss the conceptual differences and then present the formulas.

5.1 From time-based to tuple-based

The main structural difference between time- and tuple-based windows is the change in window domains. For time-based windows, windows are defined in terms of application time units, and therefore, window size and slide take values from the t^{app} domain. For tuple-based windows, windows are defined in terms of number of tuples, and therefore, window size and slides take values from the tid domain. This change will directly affect the Scope formulation and indirectly the Content that builds on Scope. The Tick and Report dimensions in SECRET are affected in more subtle ways due to the interplay between window domains (time-based vs. tuple-based) and tick domains (time-driven vs. tuple-driven).[3]

Through Tick, SECRET maps tuples from the domain of tuple arrival events (i.e., the t^{sys} domain) to the window domain over which the query operates (i.e., the t^{app} domain for time-based windows and the tid domain for tuple-based windows). This mapping defines the number and granularity of Report invocations per window domain value. This is where the two window domains lead to an important difference (Table 2). For time-based windows, tuple-driven Tick may invoke Report one or more times per t^{app} value, including once for each simultaneous tuple as well as once for every gap; time-driven Tick invokes Report exactly once per t^{app} value, including once for every gap as well; and batch-driven Tick may invoke Report one or more times per t^{app} value, including once for each batch as well as once for every gap. On the other hand, for tuple-based windows, time-driven and batch-driven Tick may invoke Report with multiple tid values at a time (when there are simultaneous tuples or batches, respectively). This was certainly not the case in time-based windows, where Tick always invoked Report with a single t^{app} value at a time. When this happens, time- and batch-driven systems make a choice across those multiple tid values.[4] SECRET can easily capture this choice as part of its Report dimension (Sect. 5.4).

These differences can be better understood by examining the input streams.

Regular input: For regular input (i.e., no simultaneity or gaps), all tick types behave similarly: t^{app} and tid both increment by one, and therefore, Report is invoked for all tick types exactly once per t^{app} and tid for time- and tuple-based windows, respectively.

Input with simultaneity: When there are simultaneous tuples in the input, and the tick types and window domains do not match, we see a major difference in tick behavior. Tuple-driven tick + time-based window needs to tick once for every tuple arrival event, and therefore, it causes multiple reactions for the same t^{app} value. Time-driven tick + tuple-based window needs to tick once for every time-passing event, and therefore, it causes a single reaction for multiple tid values. On the other hand, when tick types and window domains match, ticking once for every window domain value is sufficient, as simultaneity either does not mean anything in this case (i.e., tuple-driven tick + tuple-based window) or is naturally captured in the tick behavior already (i.e., time-driven tick + time-based window).

Input with gaps: While time gaps in the input directly affect time-based windows (as both gaps and windows refer to the t^{app} domain), they can be completely ignored for tuple-based windows, as gaps in the t^{app} domain do not mean anything in the tid domain over which the windows are constructed.

5.2 Scope

For tuple-based windows, Scope maps a tuple-id value i (instead of an application time value t) to an interval over which a query q should be evaluated (Definition 7). The active window is defined as the earliest open window as of a given tid value i (instead of a given t^{app} value t).

[3] We will exclude the batch-driven tick from the discussion for now, since it has similar implications as time-driven tick.

[4] This is also the explanation for the "evaporating tuples" situation described by Jain et al. [12], which can happen when a system ignores some of the simultanenous tuples while building a tuple-based window whose size is not large enough to accomodate all such tuples.

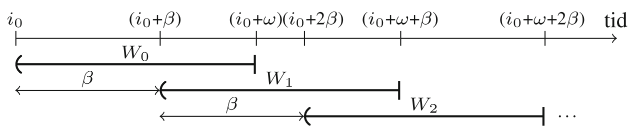

Fig. 4 Scope of a tuple-based window

We assume a value $i_0 \in \mathbb{Z}$ that denotes the first tuple-id of the first tuple-based window in a given system.[5] Like t_0 for time-based windows, the value of i_0 is system specific. Hence, the initial window (W_0) starts at tuple-id i_0, the next one (W_1) starts at tuple-id $i_0 + \beta$, window 2 (W_2) starts at tuple-id $i_0 + 2\beta$, and so forth.

Based on the above, the following formula computes n, the index of the active window as of tuple-id i:

$$n = max\left(0, \left\lceil \frac{i - i_0 - \omega}{\beta} \right\rceil\right)$$

Hence, the start tid of the n^{th} window $W_n = (o_n, c_n]$ is $o_n = i_0 + n\beta$, and the Scope as of tuple-id value i ($Scope : \mathbb{Z} \rightarrow (\mathbb{Z}, \mathbb{Z}]$) is defined as follows:

$$Scope(i) = \begin{cases} \emptyset & \text{if } i < i_0 \\ (o_n, i] & \text{otherwise} \end{cases}$$

Figure 4 illustrates our *Scope* formulation for tuple-based windows. As a simple example, assume that we have a query q with a window of size 5 tuples and of slide 2 tuples, to be run on a system with i_0 of 1. Then the window scope as of tuple-id $i = 8$ is $Scope(8) = (3, 8]$, since $n = 1$ and $o_1 = 3$.

5.3 Content

We can formally define the content of a tuple-based window ($Content : \mathbb{Z} \rightarrow \mathbb{S}$) as of tuple-id i as follows:

$$Content(i) = \{s \in \mathbb{S} : s.tid \in Scope(i)\}$$

Unlike its time-based counterpart presented in Sect. 4.2, the tuple-based Content formula does not require a system time as an input parameter. This is because $Content(i)$ always returns the same result when called with the same i value, since tuples have unique tid values.

5.4 Report

For tuple-based windows, as for time-based, Report has four basic strategies which can be combined: content change (R_{cc}), window close (R_{wc}), non-empty content (R_{ne}), and periodic (R_{pr}).

1. *Content change* (R_{cc}): reporting is done for tuple-id i, only if the content has changed since last reporting.
2. *Window close* (R_{wc}): reporting is done for tuple-id i, only when the active window closes.
3. *Non-empty content* (R_{ne}): reporting is done for tuple-id i, only if the content as of tuple-id value i is not empty.
4. *Periodic* (R_{pr}): reporting is done for tuple-id i, only if it is a multiple of a given reporting frequency, λ.

Note that reporting conditions for R_{cc} and R_{ne} are always satisfied for tuple-based windows, since for this type of windows, a system ticks and calls reporting only if a tuple has arrived (i.e., not for time gaps as in time-based windows), which makes these conditions true by default. While this simplifies our Report formulation to some extent, we have to extend our previous formula to model a new reporting behavior peculiar to time- and batch-driven systems with tuple-based windows, in the presence of simultaneous tuples or batches. Let us illustrate this behavior with an example.

Consider a tumbling tuple-based window with size and slide of 2 tuples each and the input stream shown in the second column of Table 1. Assume that we have a time-driven system for which the reporting strategy is window close (R_{wc}). In this case, one would expect that tuples with tid values 2, 4, and 6 should close windows, thus leading to reporting. However, in our time-driven system, tuples 2 through 5 which all have the same t^{app} value (2), will be treated as simultanenous tuples, and therefore, Tick will invoke Report only once for all of them. As a result, both tuple 2 and tuple 4 can potentially lead to reporting at the same time. All existing time-driven systems that we have observed make a choice between such potential reports, instead of returning all of them. Thus, our Report formula should account for this choice.

To model this choice, we define a $pick$ function, which can be customized for different systems. Furthermore, we define when possible reporting conditions (i.e., window close or periodic reporting) should fire, based on the set of tid values received as input.

The window close condition for tuple-based windows checks if the number of tuples inside a window is equal to its size. Thus, the wc function returns all tuple-ids in a set I that satisfy this condition:

$$wc(I) = \{i \in I \,|\, |Scope(i)| = \omega\}$$

The periodic reporting condition for tuple-based windows checks if a tuple's tid is a multiple of a given reporting frequency λ. Thus, the per function returns all tuple-ids in a set I that satisfy this condition:

$$per(I) = \{i \in I | mod(i, \lambda) = 0\}$$

[5] We allow the value of i_0 to be negative so that we can flexibly adjust it to model forward and backward windows in a uniform way as in the case of time-based windows.

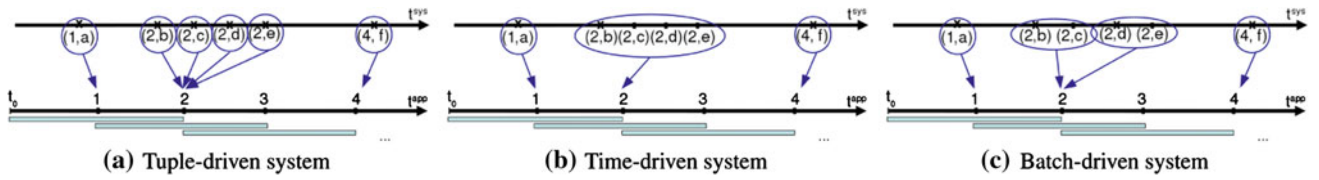

Fig. 5 Tick models for tuple-based windows

All the time-driven systems we have seen so far choose the most recent event. For them, $pick$ can be defined as follows:

$$pick(i) = \begin{cases} max(I) & \text{if } I \neq \emptyset \\ i_0 - 1 & \text{otherwise} \end{cases}$$

The second case in the above formula is to handle the degenerate case where the tuple-id set to choose from (I) is empty. In this case, we return a tuple-id value that is smaller than i_0 so that when Scope is invoked, it will return $\emptyset$.

Since some systems use multiple strategies for report, our formula should allow any logical combination of R_{cc}, R_{wc}, R_{ne}, and R_{pr}, with a call to $Content(pick(I))$ being the default. Note that, when multiple strategies are set to true, then $pick$ will choose from the tuple-id's for which all reporting conditions are satisfied. For example, if both R_{wc} and R_{pr} are set, then all tuple-id's that survive both the wc and the per function (i.e., the intersection of their output) will be the possible candidates for $pick$. Also note that, since R_{cc} and R_{ne} are always satisfied for tuple-based windows, they will not have any effect on $pick$'s input. Lastly, in our formula, we consider each possible logical combination of R_{wc} and R_{pr} separately, as each of them requires a different reporting condition to be applied on I.

Given the above, we define Report for tuple-based windows ($Report : \{\mathbb{Z}\} \rightarrow \mathbb{S}$) as follows:

$$Report(I) = \begin{cases} Content(pick(wc(I) \cap per(I))) & \text{if}(R_{wc} \wedge R_{pr}) \\ Content(pick(wc(I))) & \text{if}(R_{wc} \wedge \neg R_{pr}) \\ Content(pick(per(I))) & \text{if}(\neg R_{wc} \wedge R_{pr}) \\ Content(pick(I)) & \text{if}(\neg R_{wc} \wedge \neg R_{pr}) \end{cases}$$

5.5 Tick

The basic meaning of Tick and the possible values that it can take (i.e., tuple-driven, time-driven, batch-driven) are the same for tuple-based windows as for time-based windows. These three different Tick behaviors for tuple-based windows are illustrated in Fig. 5. We show two lines for t^{sys} and tid. Tuple arrivals are shown on the time line for t^{sys}, and window scopes are shown underneath, on the line for tid. Circles around the tuples show the units of tuples that the system will react to at one time, whereas the arrows show to which tuple-id values those units belong. Note that the tuples are the same in all three figures and that the four tuples in the middle have the same t^{app} value. If we contrast this example with that of Fig. 3 where we used an identical input stream but a time-based window, we see that the circles around the input tuples on the t^{sys} time line are unchanged, since this is purely determined by the input stream and the tick value of the system that is reacting to this input. On the other hand, the mapping of these circles to the window domain is different, since this mapping also involves the window domain which is now tid instead of t^{app}. As a result, the different tick mechanisms all lead to mapping the incoming tuples to individual tid values.

Before we present the formulas, we would like to highlight a few important points. First, since simultaneous or batch sequences still determine tick units for time- and batch-driven systems, we will reuse the corresponding mapping functions from Sect. 4.4 (i.e., $prev\_app$ and $prev\_batch$, respectively). However, we need two additional mapping functions to take us from the t^{app} and the bid domains into the tid domain, over which the $Report$ for tuple-based windows operates. Hence, we introduce app_tid and $batch\_tid$, respectively. Furthermore, for tuple-driven systems, although gaps and simultaneity need not be detected any more, we still need a new mapping function, $prev\_tid$, to take us from the t^{sys} domain into the tid domain. Lastly, in all tick formulas, $Report$ needs to be invoked with a set of tuple-id values (and a t^{sys} value), instead of the singular t^{app} value (and a t^{sys} value) that was sufficient in the time-based case.

Thus we define these additional mapping functions:

$app\_tid(t)$: Given an application time value t, returns the tuple-id values of all the tuples that have $t^{app} = t$.

$$app\_tid(t) = \{s.tid | s \in S \wedge s.t^{app} = t\}$$

$batch\_tid(b)$: Given a batch-id value b, returns the tuple-id values of all the tuples that have $bid = b$.

$$batch\_tid(b) = \{s.tid | s \in S \wedge s.bid = b\}$$

$prev\_tid(\tau)$: Given a system time instant τ, returns the tuple-id value of the most recent tuple that has arrived before τ. If no such tuple exists, it returns i_0.

$$prev\_tid(\tau) = max(max\{i_0, s.tid | s \in S(\tau - 1)\})$$

Table 3 Tick example for tuple-based windows

τ	S(v, t^{app}, t^{sys}, tid, bid)	tuple-driven	time-driven	batch-driven
$\tau = 30$	$(a, 1, 30, 1, 1)$	-	-	-
$\tau = 40$	$(b, 2, 40, 2, 2)$	Report($\{1\}$)	Report($\{1\}$)	Report($\{1\}$)
$\tau = 45$	$(c, 2, 45, 3, 2)$	Report($\{2\}$)	-	-
$\tau = 60$	$(d, 2, 60, 4, 3)$	Report($\{3\}$)	-	Report($\{2, 3\}$)
$\tau = 80$	$(e, 2, 80, 5, 3)$	Report($\{4\}$)	-	-
$\tau = 90$	$(f, 4, 90, 6, 4)$	Report($\{5\}$)	Report($\{2, 3, 4, 5\}$)	Report($\{4, 5\}$)

Using these functions, we now define $Tick : \mathbb{T} \rightarrow \{\mathbb{S}\}$ for tuple-based windows, for each type of tick. Again, all formulas follow a similar structure.

In a tuple-driven system, Tick is triggered with each tuple arrival, which in turn should invoke Report with the tuple-id of the newly arrived tuple. Since there are neither gaps nor a notion of simultaneity in the tid domain, Report is simply invoked at all system time instants for which there is a tuple arrival, leading to the following simpler formula:

$$Tick(\tau) = \begin{cases} \{Report(prev\_tid(\tau))\} & \text{if } S_I(\tau) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

In a time-driven system, there is no need to react to each tuple in a simultaneous sequence separately, and therefore, Tick should be triggered every time a tuple with a new t^{app} arrives. Then, Tick will invoke Report once with the set of tuple-id values of all the tuples inside this sequence. While in the time-based window case, Report was invoked once per t^{app} value, here we need to include all tid values in Report invocation. Furthermore, since time gaps are not important for tuple-based windows, there is no need to invoke Report multiple times to account for the missing t^{app} values. Thus, we again end up with a simpler formula:

$$Tick(\tau) = \begin{cases} \{Report(app\_tid(prev\_app(\tau)))\} & \text{if} S_I(\tau) \neq \emptyset \wedge \\ & app(\tau) > prev\_app(\tau) \\ \emptyset & \text{otherwise} \end{cases}$$

Finally, a batch-driven system acts like a time-driven system. For a batch-driven system, Tick should be triggered every time a tuple with a new bid arrives. Tick will invoke Report once with the set of all the tuple-id values of that batch. Again, neither gaps nor simultaneity requires a special treatment. Thus, we end up with the following simpler formula:

$$Tick(\tau) = \begin{cases} \{Report(batch\_tid(prev\_batch(\tau)))\} & \text{if} S_I(\tau) \neq \emptyset \wedge \\ & batch(\tau) > \\ & prev\_batch(\tau) \\ \emptyset & \text{otherwise} \end{cases}$$

In Table 3, we present a sample trace of the model for the scenario shown in Fig. 5. The table shows when each of the three models triggers $Report$ and with which tuple-id set and system time value. The time-driven model invokes $Report$ only once when time advances with the set of tuple-id values observed for the previous application time step. The tuple-driven model, on the other hand, invokes $Report$ at every new tuple arrival. Finally, the batch-driven model invokes $Report$ at every new batch arrival. Note that, unlike the example shown in Table 1, none of the tick models invokes $Report$ for gaps in application time.

6 Experiments

The main goal of this section is to show that our proposed SECRET model can be used to predict and analyze the behavior of a representative set of academic and commercial SPEs. Additionally, we will also show how differences in individual SECRET dimensions and their combinations can affect the query results.

6.1 Setup and methodology

We have tested our model with four different SPEs: the Coral8 Version 5.5 [7], Oracle CEP Version 11.1 [19], STREAM open-source academic prototype [23], and StreamBase Version 6.4 [25]. StreamBase commercialized the Aurora/Borealis academic prototypes, and therefore, its basic execution model descends from these two systems. Similarly, the Oracle CEP engine descends from STREAM's query execution model [12], but its implementation is more complete and it supports a larger set of query types. Lastly, Coral8 is a widely used commercial system (now owned by SAP), with a substantially different execution model. We have focused to explain default behaviors of the engines to a given declarative windowed query. Thus, we cover a significant variety of SPE models in this study.

SECRET will be successful if it is possible to set the four SECRET parameters so that they explain the execution behavior of each system for a common set of windowed queries and input configurations. We have done extensive experiments with these systems to find their respective parameter values.

In these experiments, we varied the input data and queries carefully. More specifically, we explored cases where the input stream had irregularities due to gaps in application time or due to simultaneous tuples with common application times. We examined queries with windows (time- or tuple-based) in three categories: sliding windows with $\beta = 1$, sliding windows with $1 < \beta < \omega$, and tumbling windows with $\beta = \omega$.

Not all systems support all possible input/query configurations (due to capability differences or incomplete

Table 4 SECRET parameters of Coral8, Oracle CEP, STREAM, and StreamBase

SPE	**Scope** (t_0, i_0)	**Report**	**Tick**
Coral8 [7]	$(\lceil \frac{t_{t1}-\omega}{\beta} \rceil \beta - 1), 0$	content change & non-empty & λ=1	batch-driven
Oracle CEP [19]	$(\lceil \frac{t_{t1}}{\beta} \rceil \beta - \omega)\ , (\beta - \omega)$	window close & content change & λ=1	time-driven
STREAM [23]	$(t_{t1} - \omega), (\beta - \omega)$	window close & content change & non-empty & λ=1	time-driven
StreamBase [25]	$(\lceil \frac{t_{t1}-\omega}{\beta} \rceil \beta - 1), 0$	window close & non-empty & λ=1	tuple-driven

implementation). For example, the available release of STREAM [23] only supports sliding windows with $\beta = 1$. Therefore, we can only test it with this type of query. However, the same system has an associated research paper that describes the more general theoretical model underneath its implementation [2], which we used as a reference where applicable.

After analyzing the execution of all supported configurations on each system, we obtained the SECRET parameters for these systems as shown in Table 4. Next, we describe how we arrived at these values.

For Scope, the system only influences the choice of the t_0 or the i_0 parameter (i.e., the application time instant or the tuple-id for the start of the first window, respectively). We obtained the t_0 and i_0 formulas in Table 4 by running different queries with various size and slide value combinations on regular input streams multiple times. Running window queries with $\beta > 1$ over regular input streams were very useful to identify how SPEs construct their windows. While tuple groupings are easy to observe on regular input streams, queries having $\beta > 1$ help us vary these groupings. On the other hand, in order to determine Report values of SPEs, analyzing the results of queries having $\beta = 1$ over input streams with large gaps were helpful. In this case, observation of repetitive and/or missing results gave us hints about SPEs' reporting strategies. Related work has already revealed the tick models for StreamBase (tuple-driven), Oracle CEP Engine (time-driven), and STREAM (time-driven) [12]. We verified these based on experimenting with various input and query settings. In addition, we found that Coral8 uses a batch-driven model, based on our own experiments, personal communication with members of the Coral8 support team, as well as a blog discussion provided at Coral8's website [18]. In Coral8, batches form automatically depending on how the input adapter feeds tuples into the Coral8 server. Lastly, we note that all tick models behave similarly when the input is regular. To identify the differences, we needed to feed irregular input (with simultaneity in particular) to our sample queries.

We now apply our model on various input and query settings (including the examples from Sect. 2) that we ran on the SPEs and show how SECRET can explain the differences in their answers. Details of the experiments can be found in SECRET web site [21]. In these experiments, we compare the query results produced by the SPEs with those predicted by our SECRET model simulator.

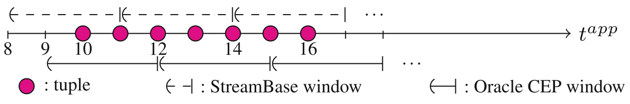

Fig. 6 Window scopes & contents for Experiment 6.2.1

Throughout our experiments, we run a given query on a given input stream in all engines which support that query type. Then, we explain the results by using SECRET. For each experiment, the discussion proceeds in three steps: (i) Tick, (ii) Scope and Content, and (iii) Report. We explain Tick with a table such as Table 7, where the first column describes the arriving tuple with its necessary properties and the subsequent columns show whether a given SPE or system type ticks at that point, and if so, how Report is invoked. We explain Scope and Content with a figure such as Fig. 6, where tuple arrivals and window intervals are depicted. Finally, we explain the Report setting for time-based windowed queries with a table such as shown in Table 8, where t^{app} denotes the application time with which Report is invoked. For rows in which the reporting condition is met for a given SPE, the corresponding window scopes and contents are shown. For tuple-based window queries, we use a modified table (e.g., Table 16), where I denotes the tuple-id set with which Report is invoked. For rows in which the reporting condition is met for a given SPE, the tuple-id that is chosen and the corresponding window scopes and contents are shown.

We present our results first for time-based windows and then for tuple-based windows.

6.2 Experiments with time-based windows

In this first series of experiments, we show that SECRET can be used to analyze and predict time-based windowed query execution behavior in SPEs. We ran an exhaustive range of experiments verifying our purpose; here we present results of six selected experiments, in increasing order of complexity.

Note that, per Table 4, Coral8 and StreamBase have identical Scope, and so do Oracle CEP and STREAM for sliding time-based window queries with $\beta = 1$, which is the only type of queries that STREAM prototype supports. As such, these pairs of engines will always yield the same Scopes and Contents for a given query and input. Although our SPEs have different reporting strategies, this difference may not be visible, depending on the input and query. For instance

Table 5 Road-map for Time-based Window Experiments

Experiment	**Input**	**Query** (ω, β)	**SPEs**	**Goal**	**Result summary**
6.2.1	regular	3, 3	StreamBase Oracle CEP	window construction	Same Tick Different t_0 (Fig. 6) Same Report (Table 6) Difference due to Window Construction
6.2.2	gap	5, 1	Coral8 Oracle CEP STREAM StreamBase	report	Same Tick (Table 7) Similar t_0 (Fig. 7) Different Report (Table 8) Difference due to Report

6.2.3	simultaneous	4, 1	Coral8(tuple)
			Coral8(time)
			Coral8(batch)
				tick	Different Tick (Table 9)
					Same t_0 (Fig. 8)
					Same Report (Table 9)
					Difference due to Tick

6.2.4	simultaneous	4, 1	Coral8(all) Oracle CEP STREAM StreamBase	report + tick	Different Tick (Table 9) Similar t_0 (Fig. 8) Different Report (Table 9 and 10) Difference due to Tick & Report
6.2.5	regular	3, 3	Coral8 Oracle CEP STREAM StreamBase	window construction + report	Same Tick Different t_0 (Fig. 9) Different Report (Table 11) Difference due to Window Construction & Report
6.2.6	simultaneous	3, 3	Coral8(all) Oracle CEP StreamBase	window construction + report + tick	Different Tick (Table 12) Different t_0 (Fig. 10) Different Report(Table 13) Difference due to Tick & Window Construction & Report

Table 6 Report for Experiment 6.2.1

	Oracle CEP			StreamBase		
t^{app}	Report?	Scope	Content	Report?	Scope	Content
$[t_0, 10)$	No	-	-	No	-	-
10	No	-	-	No	-	-
11	No	-	-	Yes	(8 11]	{10,20}
12	Yes	(9 12]	{10,20,30}	No	-	-
13	No	-	-	No	-	-
14	No	-	-	Yes	(11 14]	{30,40,50}
15	Yes	(12 15]	{40,50,60}	No	-	-
...	...	...	...	...	...	...

sliding window queries with $\beta = 1$ over regular input streams make the difference in reporting invisible. In our experiments, we exercise both cases. Finally, our SPEs include all types of Tick values, where the batch-driven Coral8 can also be configured as time- or tuple-driven by adjusting its input.

As a road-map for the reader, we provide a summary of the setup, goals, and results of our experiments in Table 5.

6.2.1 Difference in window construction

Example 1 of Sect. 2 shows that even a simple query over a regular input stream might produce completely different results on different engines, e.g., the result produced by Oracle CEP does not have a single common value with StreamBase. Next, we discuss how this example can be explained by SECRET.

Step 1. Tick: Since the input stream does not contain any simultaneous tuples, the tick difference between the time-driven Oracle CEP and the tuple-driven StreamBase will not be visible in this experiment (i.e., the arrival of a new tuple corresponds to the arrival of a new t^{app} value).

Step 2. Scope and Content: Given t_{t1}=10, ω=3, and β=3, t_0 is calculated as 9 and 8 s for Oracle CEP and StreamBase, respectively. Figure 6 depicts their corresponding window scopes and contents. Oracle CEP's windows are shifted by 1 s compared to those of StreamBase.

Step 3. Report: Table 6 illustrates the execution trace of our Report formula for the experiment. Although Oracle CEP and StreamBase have different reporting strategies, they seem to report similarly, since in this experiment there are no empty windows, and therefore, content change and window close conditions happen to coincide. Despite this, the two query results still differ due to these engines' difference in window construction (see Fig. 6).

6.2.2 Difference in report

Example 2 of Sect. 2 shows that the same query on the same input stream might produce different results on different SPEs. While Coral8, Oracle CEP, and STREAM gave similar query results for this query, StreamBase gave a different one. We show how SECRET explains this situation.

Step 1. Tick: The four systems in Example 2 have three different tick values: Coral8 is batch-driven, Oracle CEP and STREAM are time-driven, and StreamBase is tuple-driven. However, differences in the ticks' effect can only be seen when there are simultaneous tuples in the input stream. Table 7 illustrates the execution trace of our Tick formula (see Sect. 4.4) on Example 2 for Coral8, Oracle CEP, STREAM,

Table 7 Tick for Experiment 6.2.2

tuple	tuple-driven systems (StreamBase)		time-driven systems (Oracle CEP, STREAM)		batch-driven systems (Coral8)	
(t^{app},t^{sys},tid,bid)	Tick?	Report(t,τ)	Tick?	Report(t,τ)	Tick?	Report(t,τ)
(30,τ_1,1,1)	(1>0)	Report(x, τ_1), $\forall x \in [t_0, 30)$	(30> t_0)	Report(x, τ_1), $\forall x \in [t_0, 30)$	(1>0)	Report(x, τ_1), $\forall x \in [t_0, 30)$
(31,τ_2,2,2)	(2>1)	Report(30, τ_2)	(31>30)	Report(30, τ_2)	(2>1)	Report(30, τ_2)
(36,τ_3,3,3)	(3>2)	Report(x, τ_3), $\forall x \in [31, 36)$	(36>31)	Report(x, τ_3), $\forall x \in [31, 36)$	(3>2)	Report(x, τ_3), $\forall x \in [31, 36)$
...	...	...	...	...	...	...

and StreamBase, respectively. One can quickly see that on that input, all three systems "tick" in exactly the same way.

Step 2. Scope and Content: Using Table 4, t_0 equals 24, 25, 24, and 25 for Coral8, Oracle CEP, STREAM and StreamBase, respectively. These values are obtained by plugging in the values for $t_{t1} = 30$, $\omega = 5$, and $\beta = 1$ in the t_0 formulas. To calculate the window scopes themselves, SECRET uses the *Scope* formula presented in Sect. 4.1. Figure 7 depicts the corresponding window scopes and contents. Due to their common t_0 values, Coral8 and StreamBase share the same window scopes and contents, while Oracle CEP and STREAM exclude the very first scope. However, since the very first scope is empty for the given input anyway, in practice, there is no difference in the scopes and contents of the four engines (thus, they have "similar" t_0).

Step 3. Report: Table 8 displays the execution trace for the report parameter in Experiment 6.2.2 at the application time instants when reporting is called. If the reporting condition is true for a window, the scope and the actual content of the window are calculated and the content then becomes visible to the aggregation operator. More specifically, StreamBase reports only the contents of non-empty windows when they close, which happens at every second until the last tuple expires. On the other hand, Coral8 reports only the contents of non-empty windows when their contents change, which happens at time 30, 31, 35, and so on. Finally, Oracle CEP and STREAM report only the contents of the windows when they close and their contents change, which, in this example, happens at exactly the same time points as for Coral8. Unlike Oracle CEP, STREAM reports only non-empty windows, but in this experiment since there is no empty window for Oracle CEP and STREAM, the difference is not present. The table shows that SECRET correctly models Coral8's, Oracle CEP's, STREAM's results as {10, 15, 20, ...} and StreamBase's results as {10, 15, 15, 15, 15, 20, ...} when

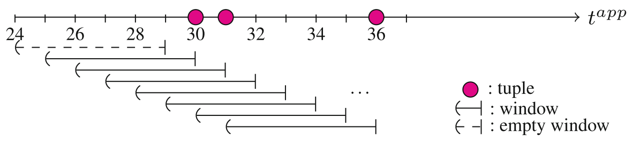

Fig. 7 Window scopes & contents for Experiment 6.2.2

an average is applied over their window contents. Hence, differences in their Report parameters explain the different results.

6.2.3 Difference in tick

In this experiment, we show that different tick models can yield different results for a given query on a given input stream. For this experiment, we chose a fixed input with simultaneous tuples and a batch-driven system, Coral8, but we configured the batches in three different ways to create the three different tick scenarios. We did this using Coral8's *atomic bundling* mechanism [7]. If each individual tuple is placed in a separate bundle, then Coral8 acts like a tuple-driven system, since it then reacts to every new input arrival (i.e., tuple $\sim$ batch). On the other hand, if we place all of the simultaneous tuples in a common bundle, Coral8 works like a time-driven system, since it then reacts to all such tuples collectively when the time advances (i.e., time unit $\sim$ batch). For all other configurations, Coral8 behaves like a normal batch-driven system.

More concretely, for a fixed input stream of tuples

```
InStream(Time, Val) = {(3,10),(5,20),(5,30),(5,40),
                       (5,50),(7,60), ...},
```

we obtained the following three configurations: STuple (one tuple per batch), STime (all simultaneous tuples in the same

Table 8 Report for Experiment 6.2.2

	Coral8			Oracle CEP			STREAM			StreamBase		
t^{app}	Report?	Scope	Content	Report?	Scope	Content	Report?	Scope	Content	Report?	Scope	Content
$[t_0, 29)$	No	-	-	No	-	-	No	-	-	No	-	-
29	No	-	-	No	-	-	No	-	-	No	-	-
30	Yes	(25 30]	{10}	Yes	(25 30]	{10}	Yes	(25 30]	{10}	Yes	(25 30]	{10}
31	Yes	(26 31]	{10, 20}	Yes	(26 31]	{10, 20}	Yes	(26 31]	{10, 20}	Yes	(26 31]	{10, 20}
32	No	-	-	No	-	-	No	-	-	Yes	(27 32]	{10, 20}
33	No	-	-	No	-	-	No	-	-	Yes	(28 33]	{10, 20}
34	No	-	-	No	-	-	No	-	-	Yes	(29 34]	{10, 20}
35	Yes	(30 35]	{20}	Yes	(30 35]	{20}	Yes	(30 35]	{20}	Yes	(30 35]	{20}
...	...	...	...	...	...	...	...	...	...	...	...	...

Table 9 Different input batch configurations (i.e., tick models) in Coral8 of Experiment 6.2.3 and 6.2.4

tuple	tuple-driven Coral8 (STuple)			time-driven Coral8 (STime)			batch-driven Coral8 (SBatch)		
(t^{app},t^{sys},tid,bid)	Tick?	Report	Content	Tick?	Report	Content	Tick?	Report	Content

$(3, \tau_1, 1, 1)$	(1>0)	Report(t_0, τ_1)
		Report(..., τ_1)
		Report(2, τ_1)
			-
			-
			-
				(3> t_0)	Report(t_0, τ_1)
					Report(..., τ_1)
					Report(2, τ_1)
						-
						-
						-
							(1>0)	Report(t_0, τ_1)
								Report(..., τ_1)
								Report(2, τ_1)
									-
									-
									-

$(5, \tau_2, 2, 2)$	(2>1)	Report(3,τ_2)
		Report(4,τ_2)
			{10}
			-
				(5>3)	Report(3,τ_2)
					Report(4,τ_2)
						{10}
						-
							(2>1)	Report(3,τ_2)
								Report(4,τ_2)
									{10}
									-

$(5, \tau_3, 3, 2)$	(3>2)	Report(5,τ_3)	{10,20}	(5>5)	-	-	(2>2)	-	-
$(5, \tau_4, 4, 3)$	(4>3)	Report(5,τ_4)	{10,20,30}	(5>5)	-	-	(3>2)	Report(5,τ_6)	{10,20,30}
$(5, \tau_5, 5, 3)$	(5>4)	Report(5,τ_5)	{10,20,30,40}	(5>5)	-	-	(3>3)	-	-

$(7, \tau_6, 6, 4)$	(6>5)	Report(5,τ_6)
		Report(6,τ_6)
			{10,20,30,40,50}
			-
				(7>5)	Report(5,τ_6)
					Report(6,τ_6)
						{10,20,30,40,50}
						-
							(4>3)	Report(5,τ_6)
								Report(6,τ_6)
									{10,20,30,40,50}
									-

...	...	...	...	...	...	...	...	...	...

batch), and SBatch (two simultaneous tuples per batch). With the batch-id (shown in bold), the input streams are:

```
STuple(Time, Val, bid)= {(3,10,1),(5,20,2),(5,30,3),
                         (5,40,4),(5,50,5),(7,60,6),...}
STime (Time, Val, bid)= {(3,10,1),(5,20,2),(5,30,2),
                         (5,40,2),(5,50,2),(7,60,3),...}
SBatch(Time, Val, bid)= {(3,10,1),(5,20,2),(5,30,2),
                         (5,40,3),(5,50,3),(7,60,4),...}
```

Then we ran a query on these three input streams, which continuously computes the sum of values over a sliding window of size 4 s and slide of 1 s, yielding the following results:

```
Coral8 Output(STuple) = {(10),(30),(60),(100),(150),...}
Coral8 Output(STime)  = {(10),(150),... }
Coral8 Output(SBatch) = {(10),(60),(150),...}
```

Step 1. Tick: Table 9 shows the execution trace of our batch-driven Tick formula for STuple, STime, and SBatch, each simulating a different Tick value. The tick condition returns true, if the batch-id of the newly arrived tuple is greater than the previous tuple's batch-id. STuple ticks at every new tuple arrival, STime ticks every time t^{app} changes, and SBatch ticks at every new batch arrival. The first column in the table refers to the arrival of the tuple. For instance, when the third tuple (t_3) arrived, STuple ticked since the batch-id of the tuple was greater than that of the previously seen tuple. Consequently, Report was called, and the content of the window (the table shows only the Val attribute of the tuples) became visible for the evaluation of the sum operator. On the other hand, STime and SBatch did not tick, since the batch-id of the tuple remained the same. As a result of different Tick models, the SPEs called Report at different time instants, which led to different actual window contents as shown in Table 9. This trace clearly shows how SECRET's Tick can capture the effect of the three tick behaviors on window content reporting. To see the complete picture, we need to also examine Scope, Content, and Report.

Step 2. Scope and Content: Given $t_{t1} = 3$, $\omega = 4$, and $\beta = 1$, t_0 can be calculated as -2 s for Coral8. Accordingly, Fig. 8 depicts the window scopes and contents based on the *Scope* formula of Sect. 4.1.

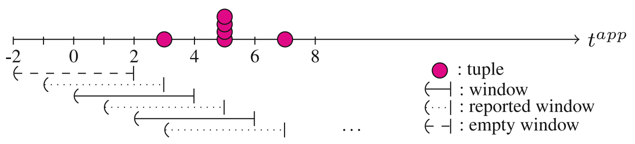

Fig. 8 Window scopes & contents for Experiments 6.2.3, 6.2.4

Step 3. Report: In Fig. 8, windows which have produced results because of changes in their contents are denoted with fine dots. SECRET correctly models the result for STuple as {10, 30, 60, 100, 150, ...}, for STime as {10, 150, ...}, and for SBatch as {10, 60, 150, ...}.

6.2.4 *Difference in report and tick*

In this experiment, we show the combined effect of different semantic decisions made by engines on the result of a given input and query. More specifically, we will show the effect of different report and tick strategies. For this experiment, we used the same input stream and query as for Experiment 6.2.3. Additionally, we included the rest of the engines (Oracle CEP, STREAM, and StreamBase). The following results were given by the engines:

```
Coral8(STuple) Output = {(10),(30),(60),(100),(150),...}
Coral8(STime)  Output = {(10),(150),...}
Coral8(SBatch) Output = {(10),(60),(150),...}
Oracle CEP Output     = {(10),(150),...}
STREAM Output         = {(10),(150),...}
StreamBase Output     = {(10),(10),(150),(150),...}
```

Step 1. Tick: In this experiment, we cover all tick values such that Coral8 is batch-driven; StreamBase is tuple-driven; and Oracle CEP, and STREAM are time-driven. As in Experiment 6.2.3, since the input stream contains simultaneous tuples, we expect to see differences in how these systems 'tick'. As expected, StreamBase ticks exactly like tuple-driven Coral8, and similarly, Oracle CEP and STREAM tick exactly like time-driven Coral8 (see Table 9 of Experiment 6.2.3).

Step 2. Scope and Content: For the given query ($\omega = 4$, $\beta = 1$ s) and $t_{t1} = 3$, t_0 is calculated as -2 s for Coral8

Table 10 Report of Oracle CEP, STREAM, and StreamBase in Experiment 6.2.4

	Oracle CEP			STREAM			StreamBase		
t^{app}	Report?	Scope	Content	Report?	Scope	Content	Report?	Scope	Content
$[t_0, 3)$	No	-	-	No	-	-	No	-	-
3	Yes	(-1 3]	{10}	Yes	(-1 3]	{10}	Yes	(-1 3]	{10}
4	No	-	-	No	-	-	Yes	(0 4]	{10}
5	-	-	-	-	-	-	No	-	-
5	-	-	-	-	-	-	No	-	-
5	-	-	-	-	-	-	No	-	-
5	Yes	(1 5]	{10,20,30,40,50}	Yes	(1 5]	{10,20,30,40,50}	Yes	(1 5]	{10,20,30,40,50}
6	No	-	-	No	-	-	Yes	(2 6]	{10,20,30,40,50}
...	...	...	...	...	...	...	...	...	...

and StreamBase, and −1 s for Oracle CEP and STREAM. Figure 8 depicts the window scopes and contents for this experiment. Coral8 and StreamBase share the same scope and contents, whereas Oracle CEP and STREAM exclude the very first window (thus, they have "similar" t_0).

Step 3. Report: Tables 9 and 10 together illustrate the execution trace of our Report formula for all engines involved in this experiment. Table 9 shows, as we discussed in the previous experiment, how Report was triggered differently for three different tick configurations of Coral8. In Table 10, we additionally show the execution trace of our Report formula for Oracle CEP, STREAM, and StreamBase, where it can be clearly seen that the former two engines report differently than the latter one. While StreamBase reports only the contents of the closed windows, both Oracle CEP and STREAM report every time the window content changes. Furthermore, we can also see the combined effect of both different ticking and reporting behaviors, if we compare the result of StreamBase (tuple-driven and window close) with that of time- or batch-driven configurations of Coral8 (time-driven and content change, batch-driven and content change, respectively). SECRET was able to predict both of these differences correctly.

6.2.5 *Difference in window construction and report*

In this experiment, we continue to show the combined effect of different semantic choices made by the engines. More specifically, we will show the simultaneous effect of different window construction and reporting strategies. For this experiment, we used the following regular input:

```
InStream(Time, Val) = {(11,10),(12,20),(13,30),(14,40),
                       (15,50),(16,60),(17,70),(18,80),
                       (19,90),(20,100), ...}
```

We ran a tumbling window of size 3 s in Coral8, Oracle CEP, and StreamBase, computing a sum over the values in each window. We obtained the following results:

```
Coral8 Output     = {(10),(20),(50),(90),(50),(110),
                     (180),(80),(170), ...}
Oracle CEP Output = {(30),(120),(210), ...}
StreamBase Output = {(10),(90),(180), ...}
```

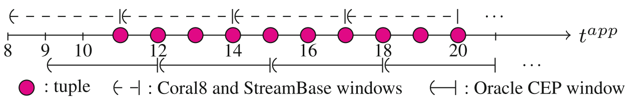

Fig. 9 Window scopes & contents for Experiment 6.2.5

A glance over the result sets shows that the result given by StreamBase is a subset of that given by Coral8. However, the result produced by Oracle CEP does not have a single common value with either Coral8 or StreamBase. Next, we discuss how these results can be explained by SECRET.

Step 1. Tick: Since the input stream does not contain any simultaneous tuples, tick differences will not be visible in this experiment, even though the engines have different ticks.

Step 2. Scope and Content: Given t_{t1}=11, ω=3, and β=3, t_0 is calculated as 8, 9, and 8 s for Coral8, Oracle CEP, and StreamBase, respectively. Figure 9 depicts their corresponding window scopes and contents: Coral8 and StreamBase share the same window scope and contents, whereas Oracle CEP's windows are shifted by 1 s compared to those of Coral8 and StreamBase.

Step 3. Report: Table 11 illustrates the execution trace of our Report formula for the experiment. As discussed before, all engines in this experiment have the same tick behavior. What causes differences in their results is different window construction and reporting strategies. If we compare the reporting behavior of Coral8 and StreamBase, we clearly see that different reporting strategies yield different results. On the other hand, if we compare Oracle CEP and StreamBase, although they have different reporting strategies, they seem to report similarly, since in this experiment each window has a unique content. Despite this, their results still differ due to their difference in window construction (see Fig. 9). Finally, when we compare the result of Coral8 with that of Oracle CEP, we see the combined effect of both window construction and reporting differences between these engines.

6.2.6 *Difference in window construction, report, and tick*

In this last experiment with time-based windows, our goal is to show the combined effect of different window

Table 11 Report for Experiment 6.2.5

	Coral8			Oracle CEP			StreamBase		
t^{app}	Report?	Scope	Content	Report?	Scope	Content	Report?	Scope	Content
$[t_0, 11)$	No	-	-	No	-	-	No	-	-
11	Yes	(8 11]	{10}	No	-	-	Yes	(8 11]	{10}
12	Yes	(11 12]	{20}	Yes	(9 12]	{10,20}	No	-	-
13	Yes	(11 13]	{20,30}	No	-	-	No	-	-
14	Yes	(11 14]	{20,30,40}	No	-	-	Yes	(11 14]	{20,30,40}
15	Yes	(14 15]	{50}	Yes	(12 15]	{30,40,50}	No	-	-
16	Yes	(14 16]	{50,60}	No	-	-	No	-	-
17	Yes	(14 17]	{50,60,70}	No	-	-	Yes	(14 17]	{50,60,70}
18	Yes	(17 18]	{80}	Yes	(15 18]	{60,70,80}	No	-	-
19	Yes	(17 19]	{80,90}	No	-	-	No	-	-
...	...	...	...	...	...	...	...	...	...

Table 12 Tick for Experiment 6.2.6

tuple	tuple-driven systems		time-driven systems		batch-driven systems	
(t^{app},t^{sys},tid,bid)	Tick?	Report(t,τ)	Tick?	Report(t,τ)	Tick?	Report(t,τ)
(3,τ_1,1,1)	(1>0)	Report(x, τ_1), $\forall x \in [t_0, 3)$	(3> t_0)	Report(x, τ_1), $\forall x \in [t_0, 3)$	(1>0)	Report(x, τ_1), $\forall x \in [t_0, 3)$
(3,τ_2,2,1)	(2>1)	Report(3, τ_2)	(3>3)	-	(1>1)	-
(5,τ_3,3,2)	(3>2)	Report(x, τ_3), $\forall x \in [3, 5)$	(5>3)	Report(x, τ_3), $\forall x \in [3, 5)$	(2>1)	Report(x, τ_3), $\forall x \in [3, 5)$
(7,τ_4,4,3)	(4>3)	Report(x, τ_4), $\forall x \in [5, 7)$	(7>5)	Report(x, τ_4), $\forall x \in [5, 7)$	(3>2)	Report(x, τ_4), $\forall x \in [5, 7)$
(7,τ_5,5,3)	(5>4)	Report(7, τ_5)	(7>7)	-	(3>3)	-
(7,τ_6,6,4)	(6>5)	Report(7, τ_6)	(7>7)	-	(4>3)	Report(7, τ_6)
(7,τ_7,7,4)	(7>6)	Report(7, τ_7)	(7>7)	-	(4>4)	-
(9,τ_8,8,5)	(8>7)	Report(x, τ_8), $\forall x \in [7, 9)$	(9>7)	Report(x, τ_8), $\forall x \in [7, 9)$	(5>4)	Report(x, τ_8), $\forall x \in [7, 9)$
(11,τ_9,9,6)	(9>8)	Report(x, τ_9), $\forall x \in [9, 11)$	(11>9)	Report(x, τ_9), $\forall x \in [9, 11)$	(6>5)	Report(x, τ_9), $\forall x \in [9, 11)$
(13,τ_{10},10,7)	(10>9)	Report(x, τ_{10}), $\forall x \in [11, 13)$	(13>11)	Report(x, τ_{10}), $\forall x \in [11, 13)$	(7>6)	Report(x, τ_{10}), $\forall x \in [11, 13)$
...	...	...	...	...	...	...

construction, reporting, and tick strategies. For this experiment, we used the following input streams with simultaneity:

```
STuple(Time, Val, Bid) = {(3,10,1),(3,20,2),(5,30,3),
                          (7,40,4),(7,50,5),(7,60,6),
                          (7,70,7),(9,80,8),(11,90,9),
                          (13,100,10), ...}
STime(Time, Val, Bid)  = {(3,10,1),(3,20,1),(5,30,2),
                          (7,40,3),(7,50,3),(7,60,3),
                          (7,70,3),(9,80,4),(11,90,5),
                          (13,100,6), ...}
SBatch(Time, Val, Bid) = {(3,10,1),(3,20,1),(5,30,2),
                          (7,40,3),(7,50,3),(7,60,4),
                          (7,70,4),(9,80,5),(11,90,6),
                          (13,100,7), ...}
```

We ran a tumbling window of size 3 s in Coral8, Oracle CEP, and Streambase, computing a sum over the values of each window. The following results were given by the engines:

```
Coral8(STuple) Output = {(10),(30),(60),(40),(90),(150),
                         (220),(80),(170),...}
Coral8(STime)  Output = {(30),(60),(220),(80),(170),...}
Coral8(SBatch) Output = {(30),(60),(90),(220),(80),
                         (170),...}
Oracle CEP Output     = {(30),(30),(300),(90),...}
StreamBase Output     = {(60),(220),(170),...}
```

Step 1. Tick: The engines included in this experiment all have different tick strategies: Coral8 is batch-driven, Oracle CEP is time-driven, and StreamBase is tuple-driven. With a simultaneous input stream, the effect of tick differences becomes visible in this experiment. Table 12 illustrates the execution of our tick formula for all engines, including three alternative configurations of Coral8.

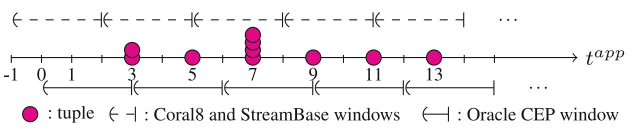

Fig. 10 Window scopes & contents for Experiment 6.2.6

Step 2. Scope and Content: Given $t_{t1} = 3$, $\beta = 3$, and $\omega = 3$ s, t_0 is calculated as -1, 0, and -1, for Coral8, Oracle CEP, and StreamBase, respectively. Due to their common t_0 value, Coral8 and StreamBase share the same window scopes and contents, whereas Oracle CEP's window scopes and contents are shifted by 1 s. Figure 10 depicts the corresponding window scopes and contents.

Step 3. Report: Table 13 illustrates how the engines in this experiment report their window contents. The difference in results of tuple-driven Coral8 and StreamBase originates from different reporting strategies. On the other hand, the difference between Oracle CEP and StreamBase originates from different window construction. The only difference among different Coral8 configurations is their different ticks. When we compare the result given by Oracle CEP with the result of tuple-driven Coral8, we see that the difference is caused by the combined effect of different ticks, reporting, and window construction strategies. As this experiment clearly shows, SECRET is able to systematically explain even complex behavioral differences across a wide variety of SPEs.

Table 13 Report for experiment 6.2.6

	Oracle CEP		StreamBase		Coral8(tuple)		Coral8(time)		Coral8(batch)	
t^{app}	Report?	Content	Report?	Content	Report?	Content	Report?	Content	Report?	Content
$[t_0, 3)$	No	-	No	-	No	-	No	-	No	-
3	-	-	No	-	Yes	{10}	-	-	-	-
3	Yes	{10,20}	No	-	Yes	{10,20}	Yes	{10,20}	Yes	{10,20}
4	No	-	No	-	No	-	No	-	No	-
5	No	-	Yes	{10,20,30}	Yes	{10,20,30}	Yes	{10,20,30}	Yes	{10,20,30}
6	Yes	{30}	No	-	No	-	No	-	No	-
7	-	-	No	-	Yes	{40}	-	-	-	-
7	-	-	No	-	Yes	{40,50}	-	-	Yes	{40,50}
7	-	-	No	-	Yes	{40,50,60}	-	-	-	-
7	No	-	No	-	Yes	{40,50,60,70}	Yes	{40,50,60,70}	Yes	{40,50,60,70}
8	No	-	Yes	{40,50,60,70}	No	-	No	-	No	-
9	Yes	{40,50,60,70,80}	No	-	Yes	{80}	Yes	{80}	Yes	{80}
10	No	-	No	-	No	-	No	-	No	-
11	No	-	Yes	{80,90}	Yes	{80,90}	Yes	{80,90}	Yes	{80,90}
12	Yes	{90}	No	-	No	-	No	-	No	-
...	...	...	...	...	...	...	...	...	...	...

Table 14 Road-map for Tuple-based Window Experiments

Experiment	**Input**	**Query** (ω, β)	**SPEs**	**Goal**	**Result summary**
6.3.1	regular	3, 2	Oracle CEP StreamBase	window construction	Same Tick (Table 15) Different i_0 (Fig.11) Same Report (Table 16) Difference due to Window Construction
6.3.2	regular	3, 3	Coral8 Oracle CEP StreamBase	report	Same Tick (Table 15) Same i_0 (Fig.12) Different Report (Table 17) Difference due to Report
6.3.3	simultaneous	1, 1	Coral8(tuple) Oracle CEP STREAM StreamBase	tick	Different Tick (Table 18) Same i_0 (Fig.13) Same Report (Table 19 and 20) Difference due to Tick
6.3.4	simultaneous	3, 2	Oracle CEP StreamBase	window construction + tick	Different Tick (Table 18) Different i_0 (Fig.11) Same Report (Table 22 and 21) Difference due to Tick & Window Construction

6.3.5	simultaneous	2, 2	Coral8(tuple)
			Coral8(time)
			Oracle CEP
			StreamBase
				report
				+ tick
					Different Tick (Table 18)
					Same i_0 (Fig.14)
					Different Report (Table 23 and 24)
					Difference due to Tick & Report

6.3 Experiments with tuple-based windows

In the next series of experiments, we show that SECRET can be used to analyze and predict tuple-based windowed query execution behavior in SPEs. We ran an exhaustive range of experiments; here we present results of five selected experiments, in increasing order of complexity.

Again, per Table 4, Oracle CEP and STREAM have identical Scopes ($i_0 = \beta - \omega$), and so do Coral8 and StreamBase ($i_0 = 0$). As such, these pairs of engines will always yield the same Scopes and Contents for a given query and input. Furthermore, as explained in Sect. 5.4, content change, non-empty, and $\lambda = 1$ conditions are always true for tuple-based windows, which makes Oracle CEP, STREAM, and StreamBase always identical in terms of their Report, while always satisfying the Report condition for Coral8. Finally, our SPEs cover all types of Tick values, where the batch-driven Coral8 can also be configured as time- or tuple-driven by adjusting its input. Remember that although Oracle CEP and STREAM seem to have exactly the same SECRET parameters for tuple-based windows, they differ in their query capabilities (i.e., STREAM only supports queries with $\beta = 1$).

Table 14 provides a summary of the setup, goals, and results of our experiments.

6.3.1 *Difference in window construction*

In this experiment, we show that different window construction strategies can yield different results for a given query and input stream. For this experiment, we chose a regular input stream, where there are no simultaneous tuples:

```
InStream(Time, Val) = {(10,10),(11,20),(12,30),(13,40),
                       (14,50),(15,60),(16,70),(17,80),
                       (18,90),(19,100), ...}
```

We ran a query on this input stream which computes the sum of the values over a window of size 3 and slide 2 tuples in Oracle CEP and StreamBase (STREAM can't support this query and Coral8 introduces a difference in Report, which we will analyze separately in Experiment 6.3.2). The following results were given by the engines:

Table 15 Ticks for Experiment 6.3.1 and 6.3.2

tuple	tuple-driven systems		time-driven systems	
(t^{app}, tid)	Tick?	Report(I)	Tick?	Report(I)
(10, 1)	(1> i_0)	Report({i_0})	(10> t_0)	Report({i_0})
(11, 2)	(2>1)	Report({1})	(11>10)	Report({1})
(12, 3)	(2>3)	Report({2})	(12>11)	Report({2})
(13, 4)	(4>3)	Report({3})	(13>12)	Report({3})
(14, 5)	(5>4)	Report({4})	(14>13)	Report({4})
(15, 6)	(6>5)	Report({5})	(15>14)	Report({5})
(16, 7)	(7>6)	Report({6})	(16>15)	Report({6})
(17, 8)	(8>7)	Report({7})	(17>16)	Report({7})
(18, 9)	(9>8)	Report({8})	(18>17)	Report({8})
(19, 10)	(10>9)	Report({9})	(19>18)	Report({9})
...	...	...	...	...

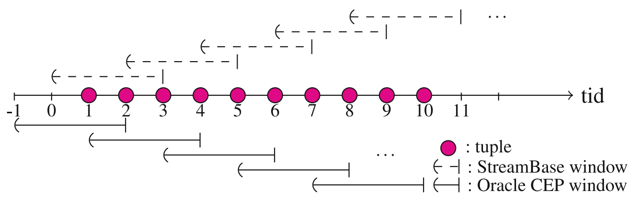

Fig. 11 Window scopes & contents for Experiment 6.3.1 and 6.3.4

```
Oracle CEP Output = {(30),(90),(150),(210), ...}
StreamBase Output = {(60),(120),(180),(240), ...}
```

Both engines gave the same number of results, but with completely different values. Next, we show how SECRET explains the situation.

Step 1. Tick: Although Oracle CEP and StreamBase have different ticks (see Table 4), we do not see the effect of tick differences in this experiment, since we have a regular input stream where each tuple has a unique application time. More specifically, arrival of a new tuple corresponds to the arrival of a new t^{app} value. Table 15 illustrates the execution trace of our Tick formula (see Sect. 5.5) on Experiment 6.3.1. As expected, both engines "tick" in exactly the same way.

Step 2. Scope and Content: Given that ω = 3 and β = 2, i_0 can be calculated as -1 for Oracle CEP and 0 for StreamBase by using the formulas in Table 4. Accordingly, Fig. 11 depicts the window scopes and contents for Oracle CEP and StreamBase for Experiment 6.3.1.

As can be seen from the figure, Oracle CEP and StreamBase construct their windows differently, even for the same query. More specifically, at every slide value (2nd, 4th, ... tuple-id), Oracle CEP engine constructs its windows in the *backward* direction, whereas StreamBase constructs them in the *forward* direction. SECRET captures this difference with the help of the i_0 parameter as explained in Sect. 5.2.

Step 3. Report: Table 16 illustrates the execution trace of our Report formula (see Sect. 5.4). Although both engines report full windows, we see a difference in their results, since they have different window scopes, and therefore different window contents. As Table 16 clearly shows,

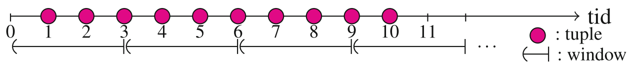

Fig. 12 Window scopes & contents for Experiment 6.3.2

SECRET correctly models the result for Oracle CEP as {30, 90, 150, 210, ...}, and for StreamBase as {60, 120, 180, 240, ...}, when a sum is applied over their window contents.

6.3.2 Difference in report

In this experiment, we show that different reporting strategies can yield different results for a given query and input stream. For this experiment, we used the same regular input stream as in Experiment 6.3.1, but we ran a tumbling window query which computes the sum of the values for every 3 tuples. We ran this query in all SPEs that can support it (Coral8, Oracle CEP, and StreamBase), with the following results:

```
Coral8 Output     = {(10),(30),(60),(40),(90),(150),
                     (70),(150),(240), ...}
Oracle CEP Output = {(60),(150),(240), ...}
StreamBase Output = {(60),(150),(240), ...}
```

We see two different result sets. Oracle CEP and StreamBase gave the same results, whereas Coral8 gave different results. Next, we show how SECRET explains the situation.

Step 1. Tick: Since our input is regular, as in the previous experiment, we do not see any difference in tick behaviors of the systems (see Table 15).

Step 2. Scope and Content: According to Table 4, i_0 is 0 for Coral8, Oracle CEP, and StreamBase in this experiment. Figure 12 depicts the corresponding window scopes and contents. Since i_0 values and the query parameters are the same for all engines in this experiment, they have exactly the same scopes and contents.

Step 3. Report: Since both the ticks and scopes of the engines are the same for this experiment, what causes the difference is the different reporting strategies. Table 17 illustrates the execution trace of Report for the engines (see Sect. 5.4). Coral8 reports partial windows due to its content change condition, whereas both Oracle CEP and StreamBase report full windows due to their window close condition. Therefore, the result produced by Coral8 is a superset of the result produced by Oracle CEP or StreamBase. As can be clearly seen from Table 17, SECRET correctly predicts the result for Coral8, Oracle CEP, and StreamBase, when a sum is applied over the indicated window contents.

Table 16 Report for Experiment 6.3.1

	Oracle CEP				StreamBase			
I	Report?	Pick	Scope	Content	Report?	Pick	Scope	Content
$\{i_o\}$	No	-	-	-	No	-	-	-
{1}	No	-	-	-	No	-	-	-
{2}	Yes	2	(-1 2]	{10,20}	No	-	-	-
{3}	No	-	-	-	Yes	3	(0 3]	{10,20,30}
{4}	Yes	4	(1 4]	{20,30,40}	No	-	-	-
{5}	No	-	-	-	Yes	5	(2 5]	{30,40,50}
{6}	Yes	6	(3 6]	{40,50,60}	No	-	-	-
{7}	No	-	-	-	Yes	7	(4 7]	{50,60,70}
{8}	Yes	8	(5 8]	{60,70,80}	No	-	-	-
{9}	No	-	-	-	Yes	9	(6 9]	{70,80,90}
...	...	...	...	...	...	...	...	...

Table 17 Report for Experiment 6.3.2

	Coral8				Oracle CEP				StreamBase			
I	Report?	Pick	Scope	Content	Report?	Pick	Scope	Content	Report?	Pick	Scope	Content
$\{i_o\}$	No	-	-	-	No	-	-	-	No	-	-	-
{1}	Yes	1	(0 1]	{10}	No	-	-	-	No	-	-	-
{2}	Yes	2	(0 2]	{10,20}	No	-	-	-	No	-	-	-
{3}	Yes	3	(0 3]	{10,20,30}	Yes	3	(0 3]	{10,20,30}	Yes	3	(0 3]	{10,20,30}
{4}	Yes	4	(3 4]	{40}	No	-	-	-	No	-	-	-
{5}	Yes	5	(3 5]	{40,50}	No	-	-	-	No	-	-	-
{6}	Yes	6	(3 6]	{40,50,60}	Yes	6	(3 6]	{40,50,60}	Yes	6	(3 6]	{40,50,60}
{7}	Yes	7	(6 7]	{70}	No	-	-	-	No	-	-	-
{8}	Yes	8	(6 8]	{70,80}	No	-	-	-	No	-	-	-
{9}	Yes	9	(6 9]	{70,80,90}	Yes	9	(6 9]	{70,80,90}	Yes	9	(6 9]	{70,80,90}
...	...	...	...	...	...	...	...	...	...	...	...	...

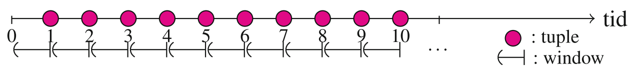

Fig. 13 Window scopes & contents for Experiment 6.3.3

Table 18 Ticks for Experiment 6.3.3, 6.3.4, and 6.3.5

tuple	tuple-driven systems		time-driven systems	
(t^{app},tid)	Tick?	Report(I)	Tick?	Report(I)
(10,1)	(1>0)	Report($\{i_0\}$)	(10> t_0)	Report($\{i_0\}$)
(10,2)	(2>1)	Report({1})	(10>10)	-
(11,3)	(3>2)	Report({2})	(11>10)	Report({1,2})
(12,4)	(4>3)	Report({3})	(12>11)	Report({3})
(12,5)	(5>4)	Report({4})	(12>12)	-
(12,6)	(6>5)	Report({5})	(12>12)	-
(12,7)	(7>6)	Report({6})	(12>12)	-
(13,8)	(8>7)	Report({7})	(13>12)	Report({4,5,6,7})
(14,9)	(9>8)	Report({8})	(14>13)	Report({8})
(15,10)	(10>9)	Report({9})	(15>14)	Report({9})
...	...	...	...	...

6.3.3 Difference in tick

In this experiment, we analyze Example 3 of Sect. 2. This example shows that different processing granularities can yield different results on a given input stream and query. The input stream that we used in this example had simultaneous tuples with the following values:

```
 InStream(Time, Val) = {(10,10),(10,20),(11,30),(12,40),
                        (12,50),(12,60),(12,70),(13,80),
                        (14,90),(15,100), ...}

Coral8 Output       = {(10),(20),(30),(40),(50),(60),(70),
                       (80),(90), ...}
Oracle CEP Output = {(20),(30),(70),(80),(90), ...}
STREAM              = {(20),(30),(70),(80),(90), ...}
StreamBase Output = {(10),(20),(30),(40),(50),(60),(70),
                       (80),(90), ...}
```

The query we used had a tuple-based tumbling window of size and slide of 1 tuple, and the following results were given by the engines:

Next, we analyze these results with SECRET.

Step 1. Tick: In this experiment, we used the default configuration of Coral8, which is tuple-driven. Table 18 illustrates the execution of our Tick formulas from Sect. 5.5 for tuple-driven (Coral8, StreamBase) and time-driven (Oracle CEP, STREAM) systems used in this experiment. As expected, time-driven systems invoke Report only once when time advances, with the set of tuple-id values observed for the previous t^{app} value. On the other hand, tuple-driven systems invoke Report at every new tuple arrival.

Step 2. Scope and Content: Given the query (ω=1, β=1), i_0 is calculated as 0 for Coral8, Oracle CEP, STREAM, and StreamBase per Table 4. Figure 13 depicts the corresponding scopes and contents. Since the i_0 value is the same for all engines for the chosen query, they share the same scopes for a input stream.

Step 3. Report: Tables 19 and 20 illustrate the execution trace of Report (Sect. 5.4) for Experiment 6.3.3 in tuple-driven and time-driven systems, respectively. Since window size and slide are both equal to 1, the reporting condition is always satisfied for all the engines. As these tables show, the different results for this experiment are due to the different ticks these systems have. In particular, time-driven systems miss some results due to choosing among simultaneous tuples. Again, SECRET can capture this difference correctly.

Table 19 Report of tuple-driven systems in Experiment 6.3.3

	Coral8				StreamBase			
I	Report?	Pick	Scope	Content	Report?	Pick	Scope	Content
$\{i_o\}$	No	-	-	-	No	-	-	-
{1}	Yes	1	(0 1]	{10}	Yes	1	(0 1]	{10}
{2}	Yes	2	(1 2]	{20}	Yes	2	(1 2]	{20}
{3}	Yes	3	(2 3]	{30}	Yes	3	(2 3]	{30}
{4}	Yes	4	(3 4]	{40}	Yes	4	(3 4]	{40}
{5}	Yes	5	(4 5]	{50}	Yes	5	(4 5]	{50}
{6}	Yes	6	(5 6]	{60}	Yes	6	(5 6]	{60}
{7}	Yes	7	(6 7]	{70}	Yes	7	(6 7]	{70}
{8}	Yes	8	(7 8]	{80}	Yes	8	(7 8]	{80}
{9}	Yes	9	(8 9]	{90}	Yes	9	(8 9]	{90}
...	...	...	...	...	...	...	...	...

Table 20 Report of time-driven systems in Experiment 6.3.3

	Oracle CEP				STREAM			
I	Report?	Pick	Scope	Content	Report?	Pick	Scope	Content
$\{i_o\}$	No	-	-	-	No	-	-	-
{1,2}	Yes	2	(1 2]	{20}	Yes	2	(1 2]	{20}
{3}	Yes	3	(2 3]	{30}	Yes	3	(2 3]	{30}
{4,5,6,7}	Yes	7	(6 7]	{70}	Yes	7	(6 7]	{70}
{8}	Yes	8	(7 8]	{80}	Yes	8	(7 8]	{80}
{9}	Yes	9	(8 9]	{90}	Yes	9	(8 9]	{90}
...	...	...	...	...	...	...	...	...

Table 21 Report of Oracle CEP for Experiment 6.3.4

	Oracle CEP			
I	Report?	Pick	Scope	Content
$\{i_0\}$	No	-	-	-
{1,2}	Yes	2	(-1 2]	{10,20}
{3}	No	-	-	-
{4,5,6,7}	Yes	6	(3 6]	{40,50,60}
{8}	Yes	8	(5 8]	{60,70,80}
{9}	No	-	-	-
...	...	...	...	...

Table 22 Report of StreamBase for Experiment 6.3.4

	StreamBase			
I	Report?	Pick	Scope	Content
$\{i_0\}$	No	-	-	-
{1}	No	-	-	-
{2}	No	-	-	-
{3}	Yes	3	(0 3]	{10,20,30}
{4}	No	-	-	-
{5}	Yes	5	(2 5]	{30,40,50}
{6}	No	-	-	-
{7}	Yes	7	(4 7]	{50,60,70}
{8}	No	-	-	-
{9}	Yes	9	(6 9]	{70,80,90}
...	...	...	...	...

6.3.4 Difference in window construction and tick

In this experiment, we show the combined effect of different execution semantics on the result of a given input and query. More specifically, we will show the effect of different window construction and tick strategies. For this experiment, we used the same, simultaneous input stream as in Experiment 6.3.3. We ran the same query as in Experiment 6.3.1, i.e., a tuple-based window query which computes the sum of the values over a window of size 3 and slide 2. The following results were given by Oracle CEP and StreamBase:

```
Oracle CEP Output = {(30),(150),(210), ...}
StreamBase Output = {(60),(120),(180),(240), ...}
```

The two engines gave not only different numbers of results, but also completely different values. Next, we show how SECRET explains these differences.

Step 1. Tick: Oracle CEP has a time-driven tick, whereas StreamBase has a tuple-driven tick. Since we used the same input stream as in Experiment 6.3.3, the systems will tick in the same way as in Table 18. Once again this table reveals that systems show different tick behaviors due to the simultaneous tuples in the input stream.

Step 2. Scope and Content: Using Table 4, i_0 is calculated as -1 for Oracle CEP and 0 for StreamBase, given that $\omega = 3$ and $\beta = 2$. Figure 11 depicts the corresponding window scopes and contents for Oracle CEP and StreamBase. Clearly, Oracle CEP and StreamBase have different scopes due to their different i_0 values.
Step 3. Report: Tables 21 and 22 illustrate the execution trace of Report (Sect. 5.4) for Oracle CEP and StreamBase, respectively. Although both engines report for full windows due to their window close condition, we see a difference in their results, since they *tick* and *close* their windows at different tuple-id values. This difference is a result of the combined effect of tick and window construction differences, as SECRET shows.

6.3.5 Difference in report and tick

For this experiment, we used the same, simultaneous input stream as for Experiment 6.3.4, but with a different query which computes the sum of the values over a tumbling window of size 2 tuples. We ran this query in all SPEs that

Table 23 Report of tuple-driven systems in Experiment 6.3.5

	tuple-driven Coral8(tuple)				StreamBase			
I	Report?	Pick	Scope	Content	Report?	Pick	Scope	Content
$\{i_o\}$	No	-	-	-	No	-	-	-
{1}	Yes	1	(0 1]	{10}	No	-	-	-
{2}	Yes	2	(0 2]	{10,20}	Yes	2	(0 2]	{10,20}
{3}	Yes	3	(2 3]	{30}	No	-	-	-
{4}	Yes	4	(2 4]	{30,40}	Yes	4	(2 4]	{30,40}
{5}	Yes	5	(4 5]	{50}	No	-	-	-
{6}	Yes	6	(4 6]	{50,60}	Yes	6	(4 6]	{50,60}
{7}	Yes	7	(6 7]	{70}	No	-	-	-
{8}	Yes	8	(6 8]	{70,80}	Yes	8	(6 8]	{70,80}
{9}	Yes	9	(8 9]	{90}	No	-	-	-
...	...	...	...	...	...	...	...	...

Table 24 Report of time-driven systems in Experiment 6.3.5

	time-driven Coral8(time)				Oracle CEP			
I	Report?	Pick	Scope	Content	Report?	Pick	Scope	Content
$\{i_o\}$	No	-	-	-	No	-	-	-
{1,2}	Yes	2	(0 2]	{10,20}	Yes	2	(0 2]	{10,20}
{3}	Yes	3	(2 3]	{30}	No	-	-	-
{4,5,6,7}	Yes	7	(6 7]	{70}	Yes	6	(4 6]	{50,60}
{8}	Yes	8	(6 8]	{70,80}	Yes	8	(6 8]	{70,80}
{9}	Yes	9	(8 9]	{90}	No	-	-	-
...	...	...	...	...	...	...	...	...

can support it (Coral8, Oracle CEP, and StreamBase), with the following results:

```
Coral8 Output(tuple) = {(10),(30),(30),(70),(50),(110),
                        (70),(150),(90), ...}
Coral8 Output(time)  = {(30),(30),(70),(150),(90), ...}
Oracle CEP Output    = {(30),(110),(150), ...}
StreamBase Output    = {(30),(70),(110),(150), ...}
```

Engines gave different numbers of results, with completely different values. Here we explain why.

Step 1. Tick: Since we used the same input stream as in Experiment 6.3.3, the systems will tick in the same way as in Table 18 and again show different behaviors because of the simultaneous tuples in the input stream.
Step 2. Scope and Content: Given the query (ω=2, β=2), i_0 is calculated as 0 for all engines using Table 4. Figure 14 depicts the corresponding window scopes and contents. Due to the common i_0 value, all engines have the same scopes and contents in this experiment.
Step 3. Report: Tables 23 and 24 illustrate the execution of Report (Sect. 5.4) for this experiment. The difference between the results of tuple-driven Coral8 and StreamBase is caused by their different reporting strategies. Coral8(tuple) reports partial windows due to its content change condition, whereas StreamBase only reports full windows due to its window close condition. Thus, tuple-driven Coral8's result is a superset of StreamBase's result. For the time-driven systems, we see the effect of different reporting strategies even more strongly.

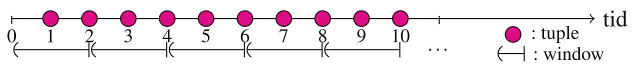

Fig. 14 Window scopes & contents for Experiment 6.3.5

Although both engines tick at the same time, different reporting strategies cause them to pick different tuple-ids, leading to different window scopes and contents. When we compare the results of the time-driven systems with the results of the tuple-driven systems, we see that the tuple-driven systems' results are a superset of the time-driven systems' results, since tuple-driven systems tick for every tuple, i.e., at the highest possible granularity. Overall, this experiment clearly shows the combined effect of report and tick differences across engines, even on simple queries with identical window scopes and contents.

7 Discussion

In this section, we briefly discuss: (i) why we believe that SECRET satisfies the design principles that we have set forth earlier in Sect. 3, (ii) how SECRET can be further extended, and (iii) potential uses of SECRET.

7.1 Design principles revisited

As shown in the previous section, SECRET can successfully explain the execution behavior of four real, representative SPEs which are quite different from each other. Its **expressivity** is the result of careful design choices. For example, Scope is defined for the earliest open window making it easy for SECRET to capture the behavior of systems that report partial window results (e.g., Coral8) as well as full window results (e.g., StreamBase). Our model embraces **simplicity** wherever possible. For example, Scope refers to a single active window and we use the t_0 parameters to adjust window intervals rather than distinguishing between forward/backward window construction, greatly simplifying our Scope formulation. We do not model real-time effects (e.g., timeouts, synchronizing application time with system time, etc.), as this would make it difficult to define a clean, predictable, and repeatable semantics. SECRET's features are **orthogonal** and **extensible**. Each behavior seen in our experiments is explained in a single way by the model, and the various aspects can be combined as needed. For example, different systems may use different combinations of the evaluation strategies for the Report dimension. Meanwhile, SECRET also is **clear**: it decouples the different levels of concerns from each other and treats them separately. For example, Scope handles query-level issues, Content handles data-level issues, and Report and Tick handle system-level issues. Likewise, Scope and Content capture non-operational effects of query processing, whereas Report and Tick capture the operational ones. Thus, SECRET embodies the characteristics we desired in our model.

7.2 Extending SECRET

In this section, we discuss how SECRET can be extended further to model SPEs that differ in input, query, or system aspects from those discussed so far. We also discuss the extensions we have made to the original model [5] in this paper.

7.2.1 Input aspects

Windows based on System time: In this paper, we have focused on windows which are based on application time. Time-based windows can also be defined using system time, the time at which they arrive at the system. In this case, $t^{app} = t^{sys}$. This case can also be handled by SECRET, since in practice, it does not matter whether the timestamps are assigned at the source or by the system at arrival. One might also imagine time-based windows that are constructed based on the system time at the point that the tuples hit each window-based operator. We have excluded this case from our model, since it has a non-repeatable semantics (e.g., the behavior would be sensitive to the operator scheduling policy in the system).

Synchronized Timestamps: In our current model, application time information can only be gathered through tuples. In practice, this strategy might delay the processing if there is a gap (in terms of system time) between tuple arrivals. In order to avoid delay, real systems use various mechanisms to synchronize the application time of tuples with the actual system time (e.g., heartbeats in STREAM [22], MAXDELAY in Coral8 [7], and TIMEOUT in StreamBase [25]). Extending our model to consider synchronized timestamps is straightforward: if a maximum delay threshold is known in advance, dummy tuples with punctuations can be injected into the input stream and the application time can be advanced without waiting for the delayed tuples to arrive.

Out-of-order Streams: SECRET makes certain ordering assumptions about the elements of a data stream (i.e., total ordering by t^{sys} and tid as well as partial ordering by t^{app} and bid). We use t^{sys} to reason about tuple arrival events in a system, and therefore, assume that the system uses a timer with high enough resolution to assign distinct and monotonically increasing t^{sys} values to tuples as they arrive. Similarly, we use tid to reason about tuple counts for tuple-based windows, and therefore, assume that the system assigns distinct and monotonically increasing tid values. We believe that these two ordering assumptions apply in practice without causing any limitations. On the other hand, the partial order assumption on t^{app} and bid (which are usually provided by the data source) might not be met in practice because of network latencies or distributed data sources. Most SPE query processors assume stream inputs obeying an ordering convention like in SECRET (e.g., STREAM). In case of out-of-order tuples, these SPEs buffer input tuples in order to put them into correct order before passing to the query processor [22]. While SECRET can explain the behavior of such SPEs, it has to be extended for other SPEs (e.g., Microsoft StreamInsight [16]), where the query processor has been designed to work with potentially out-of-order streams. These systems rely on stream punctuations and punctuation-aware query processing in order to handle the disorder. SECRET can also be extended in this direction.

7.2.2 Query aspects

Different Window Types: In a recent publication, we presented SECRET for *time-based windows* [5]. In this paper, we have extended that to include *tuple-based windows*. During this process, we have observed that the core structure of SECRET with its four basic dimensions stays the same, while new requirements for modeling tuple-based windows (i.e., change of window domains from application time to tuple-id together with their relevant mappings, the choice - or evaporation - of tuples behavior) could be added with minimal changes after a careful analysis that respects SECRET's design principles. As such, this extension has been a validation of our initial design choices. We have primarily focused on these two window types, since they not only are commonly used in many streaming applications [24], but also are implemented by almost all SPEs that we have encountered so far. One can also find more specialized windows in the literature such as *predicate-based windows* (e.g., [3]) or *semantic windows* (e.g., [8]). For these windows, there is no fixed windowing domain such as time or tuple-id (and therefore, no t_0 or i_0); instead, they require examining the tuple contents for determining the window intervals and/or contents. In SECRET terms, this means extending Scope to be predicate-based, which requires bringing Scope and Content closer. Furthermore, we have to redefine *active* window, since windows may slide in arbitrary ways leading to new situations such as a window being fully contained in another, multiple windows closing at the same tuple, or windows closing in a different order than their opening order. In other words, multiple active windows may need to be maintained at a time, and accordingly, their closing conditions and reporting orders should also be specified in SECRET. As a result, we expect that again the four basic dimensions of SECRET will stay the same, but their formulation and interaction may change due to the content-based and irregular sliding behavior of these windows.

Binary Operators: Our focus so far has been on unary operators, as they are the foundation for stream queries. We can also extend our model to handle binary operators (e.g., joins). Join operators are fundamentally different from unary sliding window operators, as they involve two inputs with two windows. Our model can directly explain how each of those windows are populated with input tuples (i.e., Tick, Scope, and Content can be used without any change). One additional issue to consider is when to make the windows of the two input streams visible to the join operator. The Report definition must be extended to address this issue.

7.2.3 System aspects

Although we cover a major subset of SPEs and their execution models, it would be interesting to expand our experimental set even further to include other SPEs as well. The first step in analyzing an SPE with our SECRET model is to find out what value each SECRET parameter should take for the given system. If the required knowledge about the system is not readily available, these values can be obtained experimentally by executing a set of queries against a range of inputs. The input stream should have irregularities due to gaps in application time and due to simultaneous tuples with common application times. Furthermore, queries should include some with windows that slide each time unit, windows with slide parameters (i.e., having a slide value greater than minimum window unit, but less than the window size), or tumbling windows (i.e., having a slide value of the same size as the window size). By executing different configurations of these input and query properties, the SECRET parameter values can be revealed, as discussed in Sect. 6.1.

In our initial study of SECRET [5], we analyzed three SPEs (Coral8, STREAM, StreamBase). In this paper, we have added Oracle CEP to this set. In analyzing Oracle CEP, we observed a new reporting behavior and revised one of our Report strategies (R_{cc}, content change) accordingly. Next, we will briefly discuss this revision, which illustrates once again that SECRET lends itself to extension without much change in its core structure.

Consider a time-based tumbling window of size and slide of 3 s each. Given an input stream `InStream(Time, Val) = {(1,10), (2,20), (4,30), (5,40), (7,50), ... }`, Oracle CEP reports the following window contents: `{10,20}` and `{30,40}`. If we applied $R_{wc} \wedge R_{cc}$ as they were defined in our original model [5], SECRET would not return any window contents. This is because we originally defined R_{cc} as "report for t only if the content has changed since $t-1$". At $t=3$, the window closes, but content does not change, as no new tuple has arrived since $t=2$. Likewise, at $t=6$, the window closes, but content does not change, as no new tuple has arrived since $t=5$.

However, in this paper we redefined R_{cc} as "report for t only if the content has changed since last reporting" (Sect. 4.3). This correctly captures Oracle CEPs behavior. This behavior could not be observed in STREAM, since STREAM supports only sliding window queries ($\beta = 1$). This change only affects SPEs with R_{cc} in their report condition and, while more general than the original condition, is precise enough for the systems we have tested. Finally, the distinction is irrelevant in tuple-based windows, since in this case, R_{cc} is always true anyway for both definitions (see Sect. 5.4).

7.3 Potential uses of SECRET

The original motivation behind SECRET was to analyze the query execution behavior of SPEs and their differences. However, the model can be used for other important purposes as well.

First, SECRET can also be useful for discovering equivalences among SPEs. These equivalences can then be used for rewrite-based query optimization in integrated stream processing settings (e.g., MaxStream [4]). Equivalences can also be used to semantically translate a query to enable porting of applications across SPEs. For instance, if it is known that no simultaneous tuples and gaps can occur in a given input stream, query translation is possible even among SPEs having different Tick values, as in this case, tuple-, time-, and batch-driven SPEs all behave similarly in terms of their Tick behavior. We are pursuing this research direction as part of our ongoing work [9].

SECRET parameters cover all the key aspects of query execution: input, query, and system. Based on different application input and query requirements, one SPE might be preferred over another for a given application. For example, if an application has simultaneous tuples (e.g., position reports of cars in dense areas of traffic) and there is no well-defined order among them, using an SPE following a time-driven Tick model might be a better option since such an engine would not impose any arrival-related ordering among simultaneous tuples. On the other hand, if it is possible to define batches among the simultaneous tuples (e.g., based on car types), an SPE following a batch-driven Tick model might produce the result faster than a time-driven SPE. Different reporting strategies may also be better suited for particular application needs. For instance, for slowly changing datasets and queries with small window sizes to process them, an SPE having window content change as its reporting strategy might be preferred over one with a window close strategy in order to avoid repeated results.

8 Related work

The earliest models for stream systems sought to provide a clean semantics for a single system. For example, STREAM's CQL provides a formal model based on the relational model [2]. In addition to the stream data type and mapping operations between streams and relations, CQL also introduced the notion of time into the relational model, essentially adding "time-driven" continuous query execution semantics.

In the aftermath of the early-generation systems, a few recent studies have tried to offer cleaner abstract models without necessarily being tied to a specific system implementation.

Maier et al. [15] generalizes the denotational semantics approach of STREAM CQL. They focus on defining the meaning of a stream itself rather than the complete query execution semantics. Li et al. [14] have proposed a framework for defining window semantics based on three functions: $windows$, $extent$, and $wids$, where the window semantics is described independent of the execution model. Our work differs from Li et al. [14], in that we not only consider window contents but also other operational issues that influence the query results.

Patroumpas and Sellis have also proposed a formal framework for expressing windows for a CQL-like model [20]. The model is based on a time-parameterized scope function that specifies a time-based window's size and progression in time. This work is not based on real system implementations, and the proposed formalism has not been tested to see whether it is powerful enough to capture the existing systems behaviors.

Most recently, Kramer and Seeger have proposed a pair of logical and physical operator algebras for stream operators, applying ideas from temporal databases [13]. Their approach is similar to that of the STREAM team, with a few differences. First, every tuple is assigned a time interval showing its validity period instead of a single timestamp. Second, the snapshot reducibility concept from temporal databases is used in finding equivalences for query optimization; however, this concept does not apply to window operators. Finally, the authors describe the physical implementation of their operators in their PIPES system. This paper covers similar semantic issues as in the CQL model and does not provide constructs to explain the operational aspects of other SPE systems.

It is hard to get information about underlying formal models used by current commercial systems [7,11,16,19,25,26]. Each system seems to use a different model, and the query results that they generate are not easy to compare. Jain et al. [12] tried to reconcile the differences across two of these commercial systems, Oracle CEP and StreamBase. They only consider the way that window execution is triggered. Though an important first step, this work focuses on only one aspect of execution behavior (i.e., Tick in SECRET), just one of the aspects our model captures and explains.

9 Conclusion and future work

The SECRET model describes important differences in the semantics underlying stream processing models. We developed SECRET by studying both academic and industrial SPEs. We have shown, through examples and experimentation, how SECRET can be used to understand, compare, and predict the behavior of diverse SPEs. Our model is unique to date in its comprehensive consideration of common differences in execution models, differences that can lead to surprisingly varied results when even simple stream queries are executed on different engines.

We have focused on queries with time- and tuple-based windows and unary operators only, excluding real-time effects, and analyzed four representative SPEs. We have additionally discussed some preliminary results on extending SECRET for other query types, SPEs, and system considerations. In addition to exploring and explaining the differences between different engines, SECRET can be used to find equivalences between them. Such equivalences can be used to devise query rewrite and transformation rules (e.g., providing a foundation for query optimization in a federation of SPEs [4]). We are currently pursuing this interesting research direction [9].

References

1. Abadi, D. et al.: Aurora: a new model and architecture for data stream management. VLDB J. **12**(2) (2003)
2. Arasu, A. et al.: The CQL continuous query language: semantic foundations and query execution. VLDB J. **15**(2) (2006)
3. Botan, I. et al.: Extending XQuery with window functions. In: VLDB Conference. Vienna, Austria (Sep. 2007)
4. Botan, I. et al.: Design and Implementation of the MaxStream Federated Stream Processing Architecture. Technical Report TR-632, ETH Zurich Department of Computer Science (June 2009)

5. Botan, I. et al.: SECRET: a model for analysis of the execution semantics of stream processing systems. In: VLDB Conference, Singapore (Sep. 2010)
6. Chandrasekaran, S. et al.: TelegraphCQ: continuous dataflow processing for an uncertain world. In: CIDR Conference (2003)
7. Coral8. http://www.coral8.com/
8. Dindar, N. et al.: DejaVu: declarative pattern matching over live and archived streams of events (demonstration). In: ACM SIGMOD Conference, Providence, RI (June 2009)
9. Dindar, N. et al.: Time-based window execution equivalence across heterogeneous stream processing engines (2012) (under conference submission)
10. Gedik, B. et al.: SPADE: the system S declarative stream processing engine. In: ACM SIGMOD Conference (2008)
11. IBM Info Sphere Streams http://www.ibm.com/software/data/infosphere/streams/
12. Jain, N. et al.: Towards a streaming SQL standard. In: VLDB Conference, Auckland, New Zealand (Aug. 2008)
13. Kramer, J., Seeger, B.: Semantics and implementation of continuous sliding window queries over data streams. ACM TODS **34**(1) (2009)
14. Li, L. et al.: Semantics and evaluation techniques for window aggregates in data streams. In: ACM SIGMOD Conference (2005)
15. Maier, D. et al.: Semantics of data streams and operators. In: ICDT Conference. Edinburgh, Scotland (Jan. 2005)
16. Microsoft StreamInsight. http://www.microsoft.com/sqlserver/2008/en/us/R2-complex-event.aspx
17. Motwani, R. et al.: Query Processing, approximation, and resource management in a data stream management system. In: CIDR Conference, Asilomar, CA, USA (Jan. 2003)
18. On Streaming SQL Standards. http://www.coral8.com/blogs/blog-entry/streaming-sql-standards
19. Oracle CEP. http://www.oracle.com/technetwork/middleware/complex-event-processing/
20. Patroumpas, K., Sellis, T.: Window specification over data streams. In: EDBT Workshops (2006)
21. SECRET Project. http://www.systems.ethz.ch/research/SECRET
22. Srivastava, U., Widom, J.: Flexible time management in data stream systems. In: ACM PODS Conference (2004)
23. STREAM. http://infolab.stanford.edu/stream/
24. Stream Query Repository. http://infolab.stanford.edu/stream/sqr/
25. StreamBase. http://www.streambase.com/
26. Truviso. http://www.truviso.com/