Multi-granular Time-based Sliding Windows over Data Streams

Kostas Patroumpas †
† *School of Electrical & Computer Engineering*
National Technical University of Athens, Hellas
kpatro@dbnet.ece.ntua.gr

Timos Sellis †,§
§ *Institute for the Management of Information Systems*
Research Center "Athena", Hellas
timos@dbnet.ece.ntua.gr

***Abstract*—We introduce a multi-level window operator that concurrently spans temporal extents of increasing granularity over a streaming dataset. This windowing construct is inherently sliding with time, essentially providing at each granularity a varying, but always finite portion of the most recent stream items. After a careful algebraic formulation of its semantics, we investigate interesting properties and suggest a suitable data structure that can efficiently maintain tuples qualifying for each granular level. Moreover, we propose techniques for evaluating advanced continuous requests against multiple time horizons, achieving near real-time response at reduced overhead. Finally, this framework is empirically validated against streaming data, offering concrete evidence of its benefits to online stream processing.**

I. Introduction

Over the recent years, voluminous streaming information is being collected and analyzed by many monitoring applications used in telecommunications, financial tickers, traffic surveillance systems, web crawlers etc. Such time-varying, transient and possibly unbounded *data streams* [5] must be processed online, so as to incrementally provide timely response to many long-running queries. Towards this goal, evaluation is repeatedly performed against the most recent data portion, after specifying a suitable window of fixed size over the stream. Windows are not system-controlled optimizers imposed towards more efficient query processing; instead, it is users themselves who specify such constructs in their requests, thus stipulating their interest on periodically obtaining refreshed results over particular stream chunks. Most processing engines support definition of *sliding windows*, expressed either in time units (e.g., items received during past 10 minutes) or tuple counts (e.g., 1000 recent items), which get refreshed with the advancement of time or the arrival of new items, respectively.

In this paper, we extend the notion of sliding windows, suggesting a multi-level windowing construct that specifies a set of temporal scopes at diverse user-defined *granularities* over a data stream. By default, such a window is also sliding with time, but possibly at a diverse pace per level, depending on the chosen parametrization. Consider the following CQL-like [3] clause for a 3-granular window:

```
[RANGES 1 HOUR, 10 MINUTES, 1 MINUTE
 SLIDES 5 MINUTES, 1 MINUTE, 1 SECOND]
```

which is equivalent to three typical sliding windows:

–0^{th} *level*: `[RANGE 1 MINUTE SLIDE 1 SECOND]`

–1^{st} *level*: `[RANGE 10 MINUTES SLIDE 1 MINUTE]`

–2^{nd} *level*: `[RANGE 1 HOUR SLIDE 5 MINUTES]`

Essentially, each level prescribes a fixed time frame (*"range"*) covering different portions of the received items and moving forward periodically (*"slide"*). Although in terms of expressiveness both versions are comparable, this is not true from a semantics and evaluation perspective. In fact, the former version provides several stream chunks (i.e., data slices of diverse size) to *concurrently* evaluate a single continuous query over them, whereas the latter implies that three separate queries are used to obtain the same result. This processing paradigm can be proven valuable for many operations, such as:

- Detecting bursts of unusual activity in networks (e.g., denial-of-service attacks) over multiple time periods.
- Monitoring traffic streams (like network packets, moving vehicles, web clicks or RFID's) along diverse time intervals over their recent "historical" traces.
- Maintaining online aggregates against rapidly evolving time series (financial tickers, temperature measurements) to discover trends across multiple resolutions.
- In-network manipulation of sensor readings (for weather, pollution, etc.), assigning a diverse grade of detail at each level in the network hierarchy.

Multiple time frames have been utilized for summarization and clustering of data streams [2], [9], [11], [18]; to the best of our knowledge, ours is the first framework that associates multi-level extents to window specification, in order to concurrently provide several nested stream fragments. We think that a fixed one-dimensional sliding window may not easily capture essential features of the data and the underlying stream evolution. As we analyze and empirically verify in this paper, a multi-granular window offers advantages for "layered" evaluation, thus greatly reducing the amount of processing required at higher granular levels. To this end, we develop a flexible scheme for incremental maintenance of items in all window levels, which is also useful for providing timely response to multi-grained aggregates, online computation of linear fit coefficients and detection of recurring

items across multiple nested time horizons.

The remainder of this paper is organized as follows. In Section 2, we discuss fundamental notions concerning windows over data streams. In Section 3, we introduce multi-granular windows and develop a scheme for their efficient maintenance. Section 4 presents techniques for answering advanced continuous requests over nested time horizons. Results from an empirical validation are reported in Section 5. Related work is reviewed in Section 6, whereas Section 7 offers conclusions and directions for future research.

II. Preliminaries

Streaming items can be considered similar to relational tuples with one important distinction: prior to evaluation, *ordering* must be established among them. Typically, a *timestamp* value is assigned to each item, either at its source (e.g., valid time of sensor readings) or upon admission to the system (i.e., transaction time). This is often a time indication from a global chronometer (like a clock tick), so tuples generated simultaneously or arriving synchronously get identical timestamps. Hence, we specify *Time Domain* $\mathbb{T}$ as an infinite set of discrete instants $\tau \in \mathbb{T}$ with a total order $\leq$ [6], so timestamp values can be regarded as natural numbers from $\mathbb{N}$. Accordingly, a time interval $[\tau_1, \tau_2] = \{\tau \in \mathbb{T} : \tau_1 \leq \tau \leq \tau_2\}$.

At any $\tau \in \mathbb{T}$, a possibly large, but always finite number of data elements may arrive for processing [3], hence a data stream S may be considered as an ordered sequence of items $\langle s, \tau \rangle$, where s is a relational tuple with a defined schema and τ its timestamp value. Given that processing should be carried out in main memory so as to meet real-time requirements, *windows* have been introduced to restrict the amount of stream tuples being processed each time.

In effect, windowing is a *Stream-to-Relation* operator [3] that repetitively provides a temporary relation (*"state"*) consisting of a countable portion of items from stream S. At any instant $\tau \in \mathbb{T}$, the transient window state $W(S(\tau))$ is derived by setting specific constraints on a designated windowing attribute [13]. Usually, such constraints involve timestamps, hence the most widely used windows are *time-based* (e.g., sliding, tumbling or landmark)[1]. As we discussed in [15], [16], these conditions can be abstracted with a *scope* function, suitably parameterized according to query specifics; the scope determines window's structure, i.e., its time-varying *bounds*, the actual *extent*, as well as its progression with time. More details about the taxonomy and properties of windowing constructs can be found in [15].

Time-based sliding windows are almost ubiquitous in stream processing [1], [5], [8], since the focus is primarily on recent data. Their specification parameters [15] include:

- A temporal extent (or "range") spanning ω units backwards from current time τ_c. Only items with timestamp $t \in (\tau_c - \omega, \tau_c]$ qualify for actual window state.
- A sliding step of β time units controls transition to successive state, i.e., how frequently to check for qualifying items. Upon sliding, fresh tuples are included in the new state at the expense of expiring items discarded from the rear (i.e., remotest) bound of the window.
- Initiation time $\tau_0 \in \mathbb{T}$ specifies when that window is initially applied against the stream.

This processing paradigm is performed along a single timeline, where all time instants are of similar detail. However, time dimension is intuitively liaised to multiple levels of resolution with respect to $\mathbb{T}$, termed *time granules*. Each granule γ_k at level k consists of a fixed number of discrete instants $\tau \in \mathbb{T}$, while a set of consecutive granules at level k can be merged into a greater granule at $k+1$, thus iteratively defining several levels of *granularity* [6], like seconds, minutes, hours, days etc. We assume that each granule is composed of a contiguous set of successive timestamps with no "holes" inside or "gaps" between granules at the same level [7]. In practice, since varying-size intervals like months or years are seldom used in stream processing, our focus is on time granules of uniform size (mainly seconds, minutes, hours, and days). Overall, this scheme implies a hierarchical composition of granules from primitive time units (in our case, timestamps drawn from $\mathbb{T}$).

In the sequel, we introduce an extension to the sliding window paradigm, by allowing several time horizons of varying extent to be concurrently specified and maintained against a stream S. As we explain, each such frame may be seen as a separate level of granularity that completely subsumes all shorter ones, provided that every horizon is anchored at current time τ_c.

III. Specifying Multi-granular Windows

A. Window Semantics and Properties

Suppose that at time instant $\tau_0 \in \mathbb{T}$ a window W with $n > 0$ levels of granularity is initially applied over a data stream S. At level $k = 0, \ldots n-1$, window W specifies a pair $\langle \omega_k, \beta_k \rangle$, consisting of a temporal extent ω_k that stems backwards from current instant τ_c, as well as a forward sliding step β_k. In practice, extents and slides can be expressed in various time units (e.g., seconds, minutes, hours, etc.). Referring to the example in Section 1, a frame of $\omega_0 = 60$ sec refreshed every $\beta_0 = 1$ sec is specified for level 0, whereas at next levels the respective pairs are $\langle \omega_1, \beta_1 \rangle = \langle 10 \text{ min}, 1 \text{ min} \rangle$ and $\langle \omega_2, \beta_2 \rangle = \langle 1 \text{ h}, 5 \text{ min} \rangle$. Without loss of generality, we assume that at any level k, extents ω_k and slides β_k are all expressed in the same primitive units from $\mathbb{T}$. In essence, a multi-granular window W employs a set of scopes with user-specified finite intervals anchored at current time τ_c. At each level k, a *subwindow* W_k is defined with

[1] *Tuple-based* (i.e., count or partitioning) windows that depend on the succession of stream items are beyond the scope of this paper.

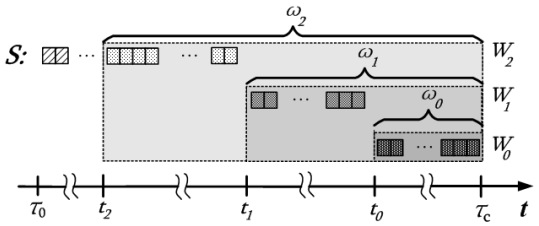

Figure 1. State of a 3-level sliding window.

its own time horizon and refresh frequency, yet all of them nested under the widest W_{n-1}.

By construction, for any level $k = 1, \ldots, n-1$ it holds that $\beta_{k-1} \leq \beta_k$ and $\omega_{k-1} < \omega_k$, so a granular hierarchy of subwindows is created (Fig. 1). For any subwindow W_k, we also assume that $\omega_k = \mu_k \cdot \beta_k$ where $\mu_k \in \mathbb{N}^*$, so each temporal extent is actually composed of a fixed number of primary blocks, i.e., *granules* of a fixed size per level. Thus, for level k, each granule γ_k has a size of β_k units and subwindow W_k spans μ_k such granules. Optionally, by stipulating that $\beta_k = \lambda_k \cdot \beta_{k-1}, \lambda_k \in \mathbb{N}^*$, a symmetric shift of time frames takes place, so when W_k slides forward, so do all other subwindows W_i at lower levels $i < k$.

Every W_k always contains a finite number of items, termed *"subwindow state"* or *substate*, for short. Naturally, smallest subwindow W_0 covers the most recent data. Subsequent extents at greater granules $k > 0$ span wider intervals (i.e., more distant in the past). Therefore, overlaps exist among substates, so a tuple s that qualifies for W_k, should also qualify for $W_i, \forall i > k$. Overall window state W is the union of stream items qualifying for each substate, i.e., $W = \cup W_k$. A tuple s expires entirely from W as soon as s gets withdrawn from widest subwindow W_{n-1}.

Nonetheless, since each level prescribes its own sliding step β_k, nested subwindows may not be always aligned with current time τ_c, hence not concurrently refreshed. Consequently, substates are not necessarily synchronized, because some tuples may have just been inserted into W_{k-1}, but not yet admitted into W_k. The finer the granule, the more frequent the change (i.e., insertion and expiration of stream items) at the corresponding substate. At time instant τ_c, the actual bounds of subwindow W_k can be determined through its *scope*, i.e., a time interval

$$scope_k(\tau_c) = [\max(\tau_0, t_k), \quad \tau_c - \mod(\tau_c - \tau_0, \beta_k)]$$

where $t_k = \tau_c - \mod(\tau_c - \tau_0, \beta_k) - \omega_k + 1$ denotes the rear bound of W_k. Both bounds have a time-varying lag of $\mod(\tau_c - \tau_0, \beta_k)$ units behind current timestamp τ_c and proceed in tandem, but only periodically and at a diverse pace for each W_k when $\mod(\tau_c - \tau_0, \beta_k) = 0$. Figure 1 illustrates the case when all subwindows just got simultaneously refreshed, so their front bounds coincide with τ_c, while rear bounds t_k are accordingly adjusted. Next, we explain how all substates can be smoothly maintained without loss of tuples in transit between successive levels.

B. Maintenance of Subwindow States

Although sliding by default, a multi-granular window cannot be implemented as a single queue, due to its inherent distinction in hierarchical granules with overlapping substates. Neither is it efficient to maintain each subwindow as an autonomous sliding frame, because of the prohibitive overhead for replicating a given tuple in as many substates as it currently qualifies to. This fact dictates for a scheme specifically tailored to multi-granular state maintenance.

An important observation is that, as soon as a given subwindow W_k slides forward, its extent ω_k subsumes all ω_i at any lower level $i < k$. Normally, the substate of W_k includes tuples already qualifying for any $W_i, \forall i < k$. In other words, state maintenance for a subwindow should facilitate maintenance of all its upper substates as well, because a subset of their qualifying tuples is readily available. For a given W_k, it would then suffice to handle only stream items with timestamps ranging in $[t_k, t_{k-1})$, i.e., tuples not covered by subordinating window frames (shown with a different shading in Fig. 1). So, an adequate structure would be a chain of interconnected nodes g_k, each one implementing the queue of items assigned solely to interval $[t_k, t_{k-1})$. Note that each such interval covers $\lceil \frac{\omega_k - \omega_{k-1}}{\beta_k} \rceil$ granules, because each granule at level k is β_k time units long.

The only problem is that subwindows generally do not slide concurrently, but each has its own pace. By default, $\beta_{k-1} \leq \beta_k$, so subwindow W_{k-1} gets refreshed more frequently compared to W_k. Items expiring from temporal extent ω_{k-1} are not assigned into next node g_k immediately, but as soon as $\mod(\tau_c - \tau_0, \beta_k) = 0$. Thus, each node in the chain must be accompanied by an auxiliary node, which simply acts as a buffer of items expiring from the preceding subwindow. Overall, this scheme is a chain of alternating "buffer" δ_i and "core" nodes g_i, like an assembly line that seamlessly produces the overall window state W. A given δ_k maintains items referring to a time interval of size up to the sliding step β_k of its successive core node g_k. To better convey the hierarchical structure of a multi-granular window, Fig. 2 depicts this chain as a flight of *"stairs"* with as many levels as the prescribed subwindows. Note that no tuples are ever duplicated among nodes, since each tuple s is assigned to a single core or buffer node, i.e., the exact one corresponding to the finest granule that covers the timestamp of s. Besides, no qualifying item is ever missing from overall window W, because the chain of nodes covers consecutive intervals with no temporal gaps.

Although the capacity (in stream items) of each node queue cannot be known in advance, their maximum size (in time units) is guided by window specification, namely sliding step β_k for buffer nodes and the difference $\omega_k - \omega_{k-1}$ for core ones. Once W_k slides forward, its corresponding

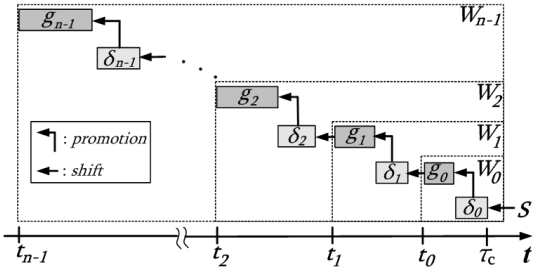

Figure 2. Stairwise chain framework for multi-granular window maintenance.

core node g_k accepts from its preceding buffer node δ_k any fresh items that span an interval of β_k units (*"promotion"* of tuples). In exchange, expiring items referring to the remotest interval of β_k units are discarded from g_k and get buffered into the next δ_{k+1} node (*"shift"*). At the lowest level, node δ_0 simply accepts incoming raw tuples, whereas there is no need to buffer items expiring from the widest subwindow W_{n-1}, as they get discarded from overall window state.

Observe that two successive instantiations of a given subwindow state may share stream items (if $\beta_k < \omega_k$), but newer substate W_k (produced after a slide) does not subsume previous one W'_k in its entirety. Such window update pattern is not monotonic [12], meaning that states cannot be maintained considering insertions of fresh items only, but expirations as well. Still, time is monotonic, so for any qualifying item $s \in W_k$ its expiration time t_{exp} is known beforehand, i.e., when s must be evicted from g_k. Hence, each subwindow can be characterized as *weakest non-monotonic*, exactly like a typical sliding window [12], [16]. For any tuple s, its expiration t_{exp} needs updating when s migrates to higher substate all along its lifetime through the multi-granular window. Such expiration timestamps help invalidating query results (e.g. aggregates) computed over previous window instantiations [16].

C. Subwindow State Reporting

Since a window is principally meant for repetitively providing a finite stream portion, timely state reporting for any granularity level(s) of interest is crucial. Semantics of a multi-grained window and its maintenance through a stairwise scheme allow two alternative reporting modes:

(i) *Continuous state reporting.* At any current instant τ_c, the state at a requested horizon $[\tau_c - h, \tau_c]$ is composed from the union of partial substates. This requires visiting the series of core and buffer nodes starting from the bottommost δ_0 in the stairwise chain and ending up to the first node g_k that fully covers the given horizon, i.e., $\omega_k \geq h$.

(ii) *Periodic state reporting.* Upon sliding by β_k units, corresponding W_k has to report its qualifying tuples. At this stage, immediately after promotion and before any shifting operations take place, node δ_k is empty, while g_k maintains tuples up to its rear bound t_k. Provided that $\beta_k = \lambda_k \cdot \beta_{k-1}, \lambda_k \in \mathbb{N}^*, \ k < n$, it turns out that buffer nodes $\delta_i, \ i < k$ are empty too. Hence, as soon as core node g_k reports, all other g_i lower in the stairwise hierarchy ($i < k$) have to report as well. Starting from g_0 and ascending stairs up to level k, the state of any intermediate subwindow W_k can be issued incrementally, taking advantage of already refreshed states for all its nested W_i (Fig. 2).

In realistic situations, only periodic reporting is meaningful and also feasible, as the processing mechanism might have to cope with an increased amount of incoming data. To avoid query evaluation for non-reporting subwindows, each substate report is signalled with current timestamp τ_c. This instructs the query processor (e.g. for computing aggregates) to deal with newly refreshed substates only. Note that "freshness" of data in any subwindow W_k is strictly controlled by its sliding step β_k and does not depend on the potentially fluctuating arrival rate ρ of stream items.

IV. Operations over Multi-granular Windows

Windows primarily serve as a means of unblocking execution of continuous queries, so that relational operators (join, aggregation, etc.) can emit incremental results when evaluated against finite window state(s) instead of the unbounded stream. With regard to multi-granular windowing, any typical operator can be applied against an isolated subwindow with no implications, thanks to its inherent sliding behavior. Yet, the main purpose of multi-level constructs is to concurrently focus on diverse time horizons. Next, we discuss advanced operations exemplifying the powerfulness of multi-granular window semantics, also taking advantage of the stairwise processing scheme. Our objective is to avoid duplication in computations and reduce update cost at coarser granules, benefiting from intermediate calculations available at finer ones.

A. Multi-grained Aggregation

Concurrently aggregating data streams over various time periods, e.g., average temperature during the last hour, past 24 hours and past week, may offer a succinct perception of the flowing measurements. Beyond approximate answering [11], [17], fast and reliable aggregates can be attained with the proposed multi-granular window, eliminating the need to maintain separate sliding windows.

In that case, instead of storing original stream items, nodes in the hierarchical "stairwise" framework (Fig. 2) can be adjusted to retain aggregate values, each spanning a distinct granule of β_k units. Buffer nodes δ_k simply accumulate intermediate sub-aggregates at the granularity of β_{k-1} time units expiring from the previous core node g_{k-1}. Then, these items can incrementally produce another aggregate at granularity β_k. When g_k slides forward, it accepts a single

aggregated value from node δ_k, which in turn becomes empty because its sub-aggregates are no longer needed.

At level k, the respective core node g_k maintains a partial aggregate value α_k as well as the sequence of contributing sub-aggregates (each of granularity β_k). That subwindow slides every β_k time units and a new sub-aggregate is inserted in the sequence, at the expense of the oldest one referring to the remotest granule of same size β_k. This latter item is evicted from g_k and buffered at node δ_{k+1} waiting for its promotion to core node g_{k+1}. Depending on the kind of aggregation, value α_k is updated accordingly, considering sub-aggregates derived from modified items, i.e., one inserted and one evicted from substate W_k.

At each level, this process requires constant time $O(1)$ per item for typical *progressive* aggregates like `SUM`, `COUNT` or `AVG`, since there is no need to scan the entire substate in case of expirations [16]. With regard to *retrospective* aggregates (`MIN`, `MAX`), as soon as the actual aggregate value expires from g_k, a search must be carried out across the sequence of items currently in g_k to find a replacement value. Apparently, the cost is $O(\lceil \frac{\omega_k}{\beta_k} \rceil)$ whenever a min/max value expires, but only $O(1)$ otherwise. Overall, the amortized cost is expectedly small, except for the unusual case of streams with ascending (for `MIN`) or descending values (for `MAX`), which trigger a substate probe at every slide [16].

To compute an aggregate for the time horizon of subwindow W_k, only nodes lower in the hierarchy should be accessed in order to obtain their partial $\alpha_i, i \leq k$. It is easy to see that every raw stream item is examined only once, namely when it gets admitted into node δ_0. Afterwards, a partial aggregate must be checked only when it enters into or exits from a core node. Of course, the number of partial aggregates is expected to shrink towards higher levels, as greater sliding steps translate to coarser granules.

Almost without modifications, a variant for *aging* multi-grained aggregation is also applicable, by simply associating a fixed weight per level. Time-decaying weights [10] gradually diminish for coarser granules so as to weaken their contribution to overall aggregate, which is biased towards more recent stream items pertaining to finer granules.

B. Online Multi-granular Regression

Linear regression over multi-granular windows may offer a versatile monitoring function that could assist in discovering trends or statistical forecasting for streaming time series [2], [9]. Suppose a data stream S with a value $s(t)$ at every successive instant $t \in \mathbb{T}$. Without loss of generality, we assume that multiple concurrent items could be aggregated into a single value, while missing values at any instant can be filled in with the latest item received.

A *linear fit* for stream items $s(t)$ qualifying for a given subwindow $W_k = \langle \omega_k, \beta_k \rangle$ is an estimation function:

$$\hat{s}(t) = \sigma \cdot t + \xi \tag{1}$$

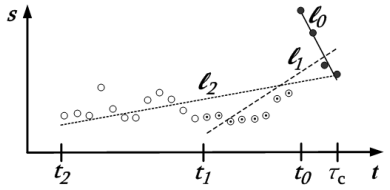

Figure 3. Trendlines at diverse time horizons.

where parameter σ signifies the slope and ξ the intercept of estimated value $\hat{s}(t)$ with $t \in [\tau_c - \omega_k + 1, \tau_c]$ and $\tau_c \in \mathbb{T}$. Typically [14], the *least square error* (*LSE*) linear fit ℓ can be obtained when choosing coefficients as follows:

$$\sigma = \frac{\sum (t - \bar{t})(s(t) - \bar{s}(t))}{\sum (t - \bar{t})^2} = \frac{\sum (t - \bar{t}) \cdot s(t)}{\sum (t - \bar{t}) \cdot t} \tag{2}$$

$$\xi = \bar{s}(t) - \sigma \cdot \bar{t} \tag{3}$$

$$\text{where} \quad \bar{t} = \frac{\sum t}{\omega_k} \quad \text{and} \quad \bar{s}(t) = \frac{\sum s(t)}{\omega_k} \tag{4}$$

Note that $\bar{t}$ is the average over all instants t pertaining to current extent $[\tau_c - \omega_k + 1, \tau_c]$, whereas $\bar{s}(t)$ represents the mean against the respective stream values[2]. Figure 3 illustrates trendlines calculated over three nested subwindows with extents of $\omega_0 = 4$, $\omega_1 = 12$, and $\omega_2 = 24$ time units from current timestamp τ_c. Stream items received during each time horizon ω_k contribute to estimates of respective linear fit ℓ_k. Apparently, multiple estimates about the actual trend of streaming time series are possible, depending on the granular resolution that is taken into account.

Our main concern is whether regression coefficients can be maintained in an online fashion for all subwindows specified. For a given extent ω_k, it can be verified that:

$$\bar{t} = \frac{(\tau_c - \omega_k + 1) + \tau_c}{2} = \tau_c - \frac{\omega_k - 1}{2} \tag{5}$$

so the running average $\bar{t}$ among its time instants has a closed formula and is monotonically ascending in pace with the timestamp order, but at a constant lag of $\frac{\omega_k - 1}{2}$ units behind current instant τ_c. When subwindow W_k slides forward with the advancement of time, apparently $\bar{t}$ increases; meanwhile, for a fixed instant t considered in both substates of W_k (i.e., before and after the slide), the difference $t - \bar{t}$ is not constant but decreasing. Although it appears that σ requires recomputation after each slide, Eq. (2) can be written as follows after simple algebraic manipulations:

$$\sigma = \frac{\sum t \cdot s(t) - \sum \bar{t} \cdot s(t)}{\sum t^2 - \sum \bar{t} t} = \frac{\sum t \cdot s(t) - \bar{t} \cdot \sum s(t)}{\sum t^2 - \bar{t} \cdot \sum t}$$

[2]Each $\sum$ ranges over all successive time instants $t \in [\tau_c - \omega_k + 1, \tau_c]$ within a given subwindow W_k, i.e., $\sum_{t=\tau_c-\omega_k+1}^{\tau_c}$.

Finally, by taking Eq. (4) into account, regression coefficients in Eq. (2) and (3) can be expressed as:

$$\sigma = \frac{\sum t \cdot s(t) - \bar{t} \cdot \sum s(t)}{\sum t^2 - \omega_k \cdot \bar{t}^2} \tag{6}$$

$$\xi = \frac{1}{\omega_k} \cdot \sum s(t) - \sigma \cdot \bar{t} \tag{7}$$

Luckily, the above expressions can be computed incrementally for an individual subwindow W_k, since all involved terms are distributive. Adjusting any of these terms incurs simple additions (or subtractions) to existing summations over stream values and time instants according to fresh insertions into (or respectively, expirations from) W_k. What's more, we can take advantage of already computed summations when it comes for greater subwindows. Indeed, every summation affects a compact stream chunk or a range of successive timestamp values, so it remains valid all along the hierarchy and its result can be readily utilized for subsequent summations over coarser granules. To this goal, a variant of the "stairwise" chain can be utilized, where each buffer or core node retains the summation terms of Eq. (6) and (7) spanning its period of interest. To calculate regression coefficients for a given W_k it suffices to visit nodes bottom-up to level k and sum the respective terms, before evaluating Eq. (6) and (7) for the respective time horizon ω_k. Overall, linear fit coefficients σ and ξ can be computed in a single pass with $O(1)$ cost per stream item, but only reported every β_k units for a given subwindow W_k.

C. Identifying Recurring Stream Items

Online checking for particular stream values that repeatedly keep arriving during diverse time horizons can be useful for detecting persistent attacks in networks, frequent customers in web auctions etc. Identifying such recurring items fits well with the notion of multi-granular windows, as time periods of interest can be conveniently expressed with sliding temporal extents. Of course, since subwindows are nested, appearance of an item in the bottommost frame would suffice to signal its presence in all horizons. However, the objective here is different: we need to identify values that emerge again and again, albeit not necessarily in a periodic fashion. Thus, we consider an item as recurring if, ascending the granular hierarchy, it is found *at least once more* in every successive time frame.

More concretely, the problem can be stated as follows. Let a window W over a data stream S specify n nested subwindows $W_k = \langle \omega_k, \beta_k \rangle$, $k = 0, \ldots, n-1$. If instant $t_k \in \mathbb{T}$ signifies the current rear bound of subwindow W_k, we wish to identify stream items s that are present at least once in every interval $[t_k, t_{k-1})$ in which W can be seamlessly subdivided (for level $k = 0$, the respective interval is $[t_0, \tau_c]$). At level k, such an interval consists of granules at β_k units each, so it suffices to mark each granule with a single bit (i.e., 1 for presence and 0 for absence of s).

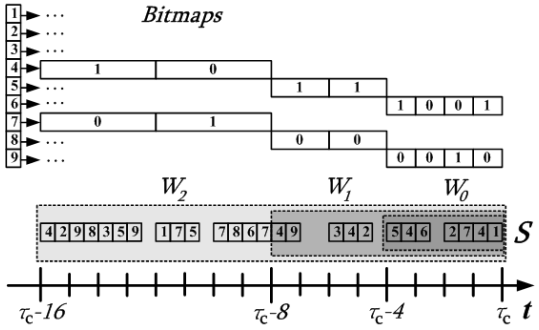

Figure 4. Bitmaps for detecting recurring items in a 3-granular window over stream S.

So, at any level, a bitmap of varying size indicates the pattern of appearances for a given s. Due to iterative construction of granules, this series of bitmaps can be shifted as time evolves, appending a new least-significant bit (LSB) at the bottommost level signifying fresh occurrence of s, while suitably coalescing β_k most-significant bits from level $k-1$ into a single LSB for a coarser granule at level k.

Interestingly, our stairwise framework can be adjusted to detect recurring items online, by accommodating such bitmaps in core and buffer nodes. Instead of storing original stream values, a core node g_k maintains $\lceil \frac{\omega_k - \omega_{k-1}}{\beta_k} \rceil$ bits for a given item s, whereas its adjoined buffer δ_k temporarily holds up to β_k bits for granules expiring from g_{k-1}. Hence, we may verify presence of s in level k by simply applying a logical OR over the respective bitmap. Then, taking the AND of resulting bits over all levels, we can safely deduce whether s is recurring. The final result shall confirm presence of s if at least one bit is set in every bitmap, whereas no recurrence is inferred if at least one bitmap has no bit set. A separate stairwise scheme is needed for each distinct item s of interest, hence a vector of bitmap chains (each with a small memory footprint) is actually maintained.

Figure 4 illustrates detection of recurring items over a 3-granular window applied against a stream S, specifying pairs $W_0 = \langle 4, 1 \rangle$, $W_1 = \langle 8, 2 \rangle$, $W_2 = \langle 16, 4 \rangle$ of temporal extents and slides $\langle \omega_k, \beta_k \rangle$. Assuming that at time τ_c all subwindows have just slid forward, the contents of two bitmap chains are depicted. For convenience, buffer nodes are omitted since they have just been emptied, whereas sizes of bit cells reveal the range of their respective granule. From the bitmaps, it is easy to conclude that value 4 is recurring, as it has a bit set in every level, whereas 7 is not.

Updating a bitmap chain costs $O(n)$ per distinct item, since at most one LSB per level has to be adjusted when all subwindows slide forward. As the number n of levels is a small constant, maintenance cost is constant as well. Memory cost is $O(nr)$ as it depends on –presumably moderate– count r of distinct items, each requiring a set of n bitmaps. Last but not least, by maintaining frequency counts instead

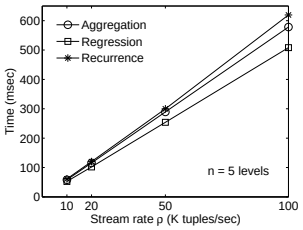

Figure 5.

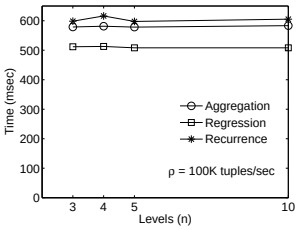

Figure 6.

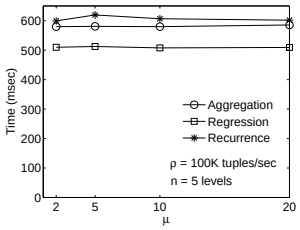

Figure 7.

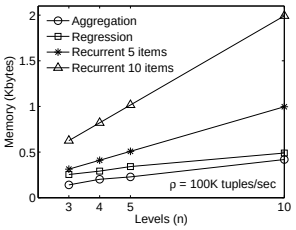

Figure 8.

of bits for each distinct item per level, accurate statistics can be calculated (e.g., for ranking persistent network threats).

V. Experimental Evaluation

In this section, we report comprehensive results from an empirical validation of the proposed multi-granular window framework against a real dataset[3], which contains network traffic traces for wide-area TCP connections with the Lawrence Berkeley Laboratory (LBL). All algorithms were implemented in C++ and simulated on an Intel Core 2 Duo 3GHz CPU running GNU/Linux with 2GB of main memory. Results are averages of actual measurements per timestamp over complete (i.e., not "half-filled") windows. Due to space limitations, we omit the rather straightforward comparison with a baseline approach that utilizes separate sliding windows at diverse granularities. Apparently, our multi-level state maintenance requires space comparable to that consumed by the widest window at the coarsest granule, with significant time savings due to sharing computation across hierarchically organized substates. Next, we show diagrams from most representative simulations for windowed operations with diverse parameter settings.

First, for checking scalability with diverse stream arrival rates ρ, Fig. 5 plots maintenance cost for a 5-level window with maximum extent $\omega_5 = 40$ timestamps. Apparently, the number of window updates increases linearly with the stream volume per timestamp, yet incurring a constant processing cost per item. Observe that identifying all recurrent items (based on attribute for transmission protocol) has a slightly greater cost, as it has to maintain more values (i.e., bitmaps) compared to scalar aggregation (`SUM` on connection duration) and maintenance of regression coefficients (on connection duration, too). To verify robustness of the proposed techniques, all subsequent experiments were conducted with $\rho = 100$K tuples/sec.

Smooth maintenance of multiple substates is an important benefit from the proposed paradigm. This is reflected in Fig. 6, where each operation requires almost constant processing time for a varying number n of levels, fixing $\omega_{n-1} = 40$ timestamps. Indeed, by exploiting already computed results from subordinate levels, each successive subwindow practically considers a smaller stream portion as opposed to a separate sliding window with identical specifics. Hence, each extra level incurs little additional maintenance cost, while focusing farther in the past.

The next experiment considers 5-level windows with diverse maximum extent ω_{n-1}, yet we assume that each level contains the same number μ of granules (remember that $\mu_k = \frac{\omega_k}{\beta_k}$ controls the rate of refresh for k-th subwindow). As illustrated in Fig. 7, even wider scopes incur no significant overhead, because only values in subwindow bounds are actually affected (shifted or promoted) at every slide. The amount of these "delta" changes does not actually depend on the size of temporal extents or the degree of overlaps among substates, but on the number n of levels only. Similar conclusions were drawn for sliding steps at diverse λ_k parameters; results are omitted due to lack of space.

With respect to memory consumption, a n-granular window W has to retain as much tuples as an autonomous sliding window comparable to its widest frame W_{n-1}. Thanks to inherent nesting of substates, this constitutes a clear advantage over isolated maintenance of separate windows. But chiefly, this policy yields space savings when evaluating operations. As shown in Fig. 8, the memory footprint is almost negligible for aggregation and regression, increasing sublinearly with the number of levels, because only intermediate results are kept for disjoint intervals and no actual stream tuples. However, space cost escalates linearly with distinct count r of recurring items (plotted for $r = 5, 10$), since a separate bitmap is required per distinct item.

Overall, experiments indicate that multi-granular window semantics can support scalable execution of composite operations in near real-time and at reduced space overhead.

VI. Related Work

Several –predominantly sliding– window variants are integrated in stream processing engines such as AURORA [1], STREAM [5], TelegraphCQ [8] and have recently made their way into commercial systems like StreamBase, Oracle CEP or Coral8 [16]. For their specification, time is usually considered as an infinite succession of timestamps, with no distinction of granules whatsoever [6], [7]. Our proposal for multi-level windows has no resemblance to partitioned

[3]Also used for window simulations in [12], [16]. Publicly available from `http://ita.ee.lbl.gov/html/contrib/LBL-CONN-7.html`.

windows either [3], [15]; time horizons are user-defined intervals, whereas partitions are data-driven and their contents depend on non-windowing attributes (i.e., not timestamps).

Time granularities have been successfully employed in stream mining. The pyramidal time frame proposed in [2] for clustering massive data streams, essentially trades storage requirements against approximation quality by keeping denser stream snapshots closer to current time. A similar tilt time frame [9] is applied for regression analysis and can substantially reduce the amount of items stored or aggregated in time-series data cubes. In both models, the notion of maintaining evolving window states is missing, as items are clustered or summarized over fixed time periods.

Sliding window aggregation has given rise to interesting algorithms; even so, the notion of diverse time horizons remains mostly overlooked. Time-decaying aggregation [10] suggests a family of functions that weight importance of a stream item by its recency. Indexing partial aggregates over intervals at multiple resolutions [4] enables the requested aggregate to be suitably composed from a union of varying-size intervals. In [13], window extents are subdivided into disjoint *"panes"* so as to compute sub-aggregates over each such slice and finally "roll-up" intermediate results.

For aggregates at multiple granularities, the fixed-time window model in [18] employs a specialized disk-based index on tuple validation intervals. Granules refer to disjoint intervals for gradually coarser aggregation, offering better approximation for most recent data. But this approach is mostly geared towards data warehousing rather than massive in-memory stream processing. An inverted histogram [17] organizes multiple windows into a monotonic search space for approximating multi-granularity aggregates with error guarantees. The hierarchical sliding window model in [11] is closer in spirit to our own. It also works over various time horizons, but their focus is on approximating stream chunks at varying digests. Using a multi-granularity tree, aggregation affects diverse sets of items per level; neither are such "snapshots" nested nor can partial aggregates from lower levels be reutilized upwards. In contrast, our framework offers clear advantages to progressively computing subwindow states, as well as exact and timely response to complex operations over multiple time horizons.

VII. Conclusions

In this paper, we provided a foundation for multi-level sliding windows extending to diverse time frames over a data stream. We introduced a simple, yet effective processing scheme for reliable maintenance of stream items qualifying to successive instantiations of all nested subwindows. In addition, we investigated and experimentally validated techniques for online answering to advanced continuous queries against multiple time horizons.

In the future, we plan to apply this framework to shared processing of multiple windowed queries, potentially subsuming their arbitrary ranges and sliding steps into a minimal set of optimized multi-granular schemes. Generalizing multi-level semantics to other window types (tuple-based ones, in particular) is also a challenging research topic.

References

[1] D.J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a New Model and Architecture for Data Stream Management. *VLDB Journal*, 12(2):120-139, August 2003.

[2] C. Aggarwal, J. Han, J. Wang, and P.S. Yu. A Framework for Clustering Evolving Data Streams. In *VLDB*, pp. 81-92, September 2003.

[3] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *VLDB Journal*, 15(2):121-142, June 2006.

[4] A. Arasu and J. Widom. Resource Sharing in Continuous Sliding-Window Aggregates. In *VLDB*, pp. 336-347, September 2004.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *ACM PODS*, pp. 1-16, 2002.

[6] C. Bettini, C.E. Dyreson, W.S. Evans, R.T. Snodgrass, and X. Sean Wang. A Glossary of Time Granularity Concepts. *Temporal Databases: Research and Practice*, LNCS 1399, pp. 406-413, 1998.

[7] C. Bettini, X. Sean Wang, and S. Jajodia. A General Framework for Time Granularity and Its Application to Temporal Reasoning. *Annals of Mathematics and Artificial Intelligence*, 22(1-2): 29-58, 1998.

[8] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, V. Raman, F. Reiss, and M.A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, January 2003.

[9] Y. Chen, G. Dong, J. Han, B.W. Wah, and J. Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *VLDB*, pp. 323-334, August 2002.

[10] E. Cohen and M. Strauss. Maintaining Time-Decaying Stream Aggregates. In *ACM PODS*, pp. 223-233, June 2003.

[11] J. Feng, Y. Wang, J. Yao, and T. Watanabe. Multi-Granularity Aggregation Index for Data Stream. In *CW*, pp. 767-771, 2008.

[12] L. Golab and M. Tamer Özsu. Update-Pattern-Aware Modeling and Processing of Continuous Queries. In *SIGMOD*, pp. 658-669, 2005.

[13] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. Tucker. No Pane, No Gain: Efficient Evaluation of Sliding-Window Aggregates over Data Streams. *SIGMOD Record*, 34(1):39-44, March 2005.

[14] D. Montgomery, E. Peck, and G. Vining. *Introduction to Linear Regression Analysis*. Wiley, 4th edition, 2006.

[15] K. Patroumpas and T. Sellis. Window Specification over Data Streams. In *ICSNW*, LNCS 4254, pp. 445-464, March 2006.

[16] K. Patroumpas and T. Sellis. Maintaining Consistent Results of Continuous Queries under Diverse Window Specifications. *Information Systems*, doi:10.1016/j.is.2010.02.001.

[17] S. Qin, W. Qian, and A. Zhou. Approximately Processing Multi-granularity Aggregate Queries over Data Streams. In *ICDE*, 2006.

[18] D. Zhang, D. Gunopulos, V.J. Tsotras, and B. Seeger. Temporal and Spatio-temporal Aggregation over Data Streams using Multiple Time Granularities. *Information Systems*, 28(1-2):61-84, 2003.