S

Sliding-Window Aggregation Algorithms

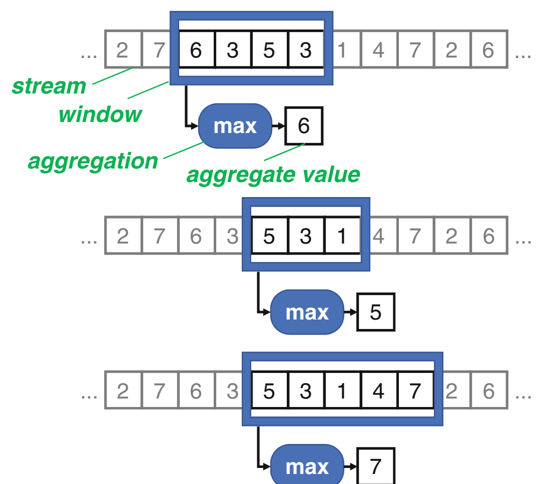
Kanat Tangwongsan[1], Martin Hirzel[2], and Scott Schneider[2]
[1]Mahidol University International College, Salaya, Thailand
[2]IBM Research AI, Yorktown Heights, NY, USA

Synonyms

Sliding-window fold; Stream reduce; SWAG

Definition

An *aggregation* is a function from a collection of data items to an aggregate value. In *sliding-window aggregation*, the input collection consists of a window over the most recent data items in a stream. Here, a *stream* is a potentially infinite sequence of data items, and the decision on which data items are *most recent* at any point in time is given by a window policy. A *sliding-window aggregation algorithm* updates the aggregate value, often using incremental-computation techniques, as the window contents change over time, as illustrated in Fig. 1.



Sliding-Window Aggregation Algorithms, Fig. 1 Sliding-window aggregation definitions

Overview

Sliding-window aggregation summarizes a collection of recent streaming data, capturing the most recent happenings as well as some history. Including some history provides context for decisions, which would be missing if only the current data item were used. Using the most recent data helps identify and react to present trends, which would be diluted if all data from the beginning of time were included.

Aggregation is one of the most fundamental data processing operations. This is true in general, not just in stream processing. Aggregation is versatile: it can compute counts,

© Springer International Publishing AG 2018
S. Sakr, A. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-63962-8_157-1

averages, or maxima, index data structures, sketches such as Bloom filters, and many more. In databases, it shows up as a basic relational algebra operator called group-by-aggregate and denoted γ (Garcia-Molina et al. 2008). In spreadsheets, it shows up as a function from a range of cells to a summary statistic (Sajaniemi and Pekkanen 1988). In programming languages, it shows up as a popular higher-order function called `fold` (Hutton 1999). In MapReduce, it shows up as reduce (Dean and Ghemawat 2004), which has been leveraged in many tasks, including computations that do not diminish the volume of data.

In stream processing, aggregation plays a similarly central role. But unlike the abovementioned cases, which focus on data at rest, streaming aggregation must handle data in motion. In particular, sliding-window aggregation must handle inserting new data items into the window as they arrive and evicting old data items from the window as they expire. Supporting this efficiently poses algorithmic challenges, especially for non-invertible aggregation functions such as `max`, for which there is no way to "subtract off" expiring items. From an algorithmic perspective, handling sliding windows with both insertion and eviction is more challenging than handling just insertion. Yet, there are two cases where eviction does not matter: unbounded and tumbling windows. Unbounded windows appear, for instance, in CQL (Arasu et al. 2006). Because they grow indefinitely, it is sufficient to update aggregations upon `insert` and not keep the data item itself around; they never need to call `evict`. Tumbling windows are more common; because they clear the entire contents of the window at the same time, there is no need to call `evict` on individual elements of the window.

Sliding windows are most commonly first-in, first-out (FIFO), resembling the behavior of a queue. What to keep in a sliding window and how often the aggregation is computed are controlled by policies, cataloged elsewhere (Gedik 2013); they may be count-based (e.g., the past 128 elements) or time-based (e.g., the past 12 min), among others. Regardless of policies, FIFO sliding-window aggregation (SWAG) can be formulated as an abstract data type with the following operations:

- `insert`(v) appends the value v to the window.
- `evict`() removes the "oldest" value in the window.
- `query`() returns the aggregation of the values in the window.

Metrics of interest in SWAG implementations are throughput, latency, and memory footprint. SWAG implementations also differ in generality: to enhance efficiency, aggregation operations, when feasible, are applied incrementally – that is, modifying a running sum of sort in response to data items arriving or leaving the window. To what extent this can be exploited depends on the nature of the aggregation operation.

Past work (Gray et al. 1996; Tangwongsan et al. 2015) cast most aggregation operations as binary operators, written $\oplus$, and has categorized them based on algebraic properties. Table 1 lists common aggregation operations with their properties and groups them into categories. An aggregation operator is *invertible* if there exists some function $\ominus$ such that $(x \oplus y) \ominus y = x$ for all x and y. Using $\ominus$, SWAGs can implement eviction as an undo. A function is *associative* if $x \oplus (y \oplus z) = (x \oplus y) \oplus z$ for all x, y, and z. SWAGs can take advantage of associativity by applying $\oplus$ at arbitrary places inside the window. Without associativity, SWAGs are restricted to applying $\oplus$ only at the end, upon insertion. A function is *commutative* if $x \oplus y = y \oplus x$ for all x and y. SWAGs are able to ignore the insertion order of data items for commutative aggregation operators. An aggregation operator is *rank-based* if it relies upon an ordering by some attribute of each data item, for instance, to find the ith-smallest.

Table 2 presents an overview of the SWAG algorithms presented in this article, with their asymptotic complexity, space usage, and restrictions. The most straightforward SWAG algorithm is called **Recalc**, since it always recalculates all values. Upon any `insert` or `evict`, Recalc walks the entire window and recomputes the

Sliding-Window Aggregation Algorithms, Table 1 Aggregation operations. Check marks (✓), crosses (×), and question marks (?) indicate that a property is true for all, false for all, or false for some of the given group, respectively

	Invertible	Associative	Commutative	Rank-based
• **Sum-like**: sum, count, average, standard deviation, …	✓	✓	✓	×
• **Collect-like**: collect list, concatenate strings, ith-youngest, …	✓	✓	×	?
• **Median-like**: median, percentile, ith-smallest, …	✓	✓	✓	✓
• **Max-like**: max, min, argMax, argMin, maxCount, …	×	✓	?	×
• **Sketch-like**: Bloom filter (Bloom 1970), CountMin (Cormode and Muthukrishnan 2005), HyperLogLog (Flajolet et al. 2007)	×	✓	✓	×

Sliding-Window Aggregation Algorithms, Table 2 Summary of aggregation algorithms and their properties, where n is the window size and n_{max} is the size of the smallest contiguous range that contains all the shared windows

	Algorithmic complexity		Restrictions
	Time	Space	
Recalc	Worst-case $O(n)$	$O(n)$	None
Subtract-on-Evict (SOE)	Worst-case $O(1)$	$O(n)$	Sum-like or collect-like
Order Statistics Tree (OST) (Hirzel et al. 2016)	Worst-case $O(\log n)$	$O(n)$	Median-like
Reactive Aggregator (RA) (Tangwongsan et al. 2015)	Average $O(\log n)$	$O(n)$	Associative
DABA (Tangwongsan et al. 2017)	Worst-case $O(1)$	$O(n)$	Associative, FIFO
B-Int (Arasu and Widom 2004)	Shared $O(\log n_{max})$	$O(n_{max})$	Associative, FIFO
FlatFIT (Shein et al. 2017)	Average $O(1)$	$O(n)$	Associative, FIFO

aggregation value by using all available elements. Its performance is obviously $O(n)$, where n is the current number of elements in the window. Recalc serves as the baseline comparison for all other SWAG algorithms. **Subtract-on-evict** (SOE) is a $O(1)$ algorithm, but it is not general: it can only be used when the aggregation is invertible. Upon every `insert`, SOE updates the current aggregation value using $\oplus$, and upon every `evict`, SOE updates that value using $\ominus$. The **order statistics tree** (OST) adds subtree statistics to the inner nodes of a balanced search tree (Hirzel et al. 2016). Values are put in both a queue and the tree, making both `insert` and `evict` $O(\log n)$. But `query` calls for aggregations such as the median or pth percentile become $O(\log n)$ because such information can be derived by traversing a path that is no longer than the height of the tree.

This section gave a brief overview with background and some simple aggregation algorithms. In general, research into SWAG algorithms tries to avoid $O(n)$ costs (unlike Recalc) but maintain generality (unlike SOE and OST). The next section will discuss the more sophisticated algorithms from Table 2 that offer improvements toward this goal.

Key Research Findings

The most successful techniques in speeding up sliding-window aggregation have been data structuring and algorithmic techniques that yield asymptotic improvements. They are the most effective when the aggregation function meets certain algebraic requirements. For instance, there are important aggregation operations that are associative, but not necessarily invertible nor commutative.

Pre-aggregation of data items that will be evicted at the same time is a technique that can be applied together with all SWAG algorithms discussed in this article. When data items are co-evicted, the window need not store them individually but can instead store partial aggregations, reducing the effective window size n in Table 2. Pre-aggregation algorithms include paned windows (Li et al. 2005), paired windows (Krishnamurthy et al. 2006), and Cutty windows (Carbone et al. 2016). Windows are sometimes coarsened to enable pre-aggregation, improving performance at the expense of some approximation.

B-Int (Arasu and Widom 2004), designed to facilitate sharing across windows, stores a "shared" window S that contains inside it all the windows being shared. To facilitate fast queries, B-Int maintains pre-aggregated values for all base intervals that lie within S. *Base intervals* (more commonly known now as dyadic intervals) are intervals of the form $[2^\ell k, 2^\ell (k+1) - 1]$ with $\ell, k \geq 0$. The parameter ℓ defines the level of a base interval. This allows a query between the i-th data item and j-th data item within S to be answered by combining at most $O(\log |i - j|)$ pre-aggregated values, resulting in logarithmic running time.

The **Reactive Aggregator** (RA) (Tangwongsan et al. 2015) is implemented via a balanced tree ordered by time, where internal nodes hold the partial aggregations of their subtrees, and offers $O(\log n)$ amortized time. Instead of the conventional approach to implementing balanced trees by frequent rebalancing, RA projects the tree over a complete perfect binary tree, which it stores in a flat array. This leads to higher performance than other tree-based SWAG implementations in practice, since it saves the time of rebalancing as well as the overheads of pointers and fine-grained memory allocation.

For latency-sensitive applications, the aggregation algorithm cannot afford a long pause. **DABA** (Tangwongsan et al. 2017) ensures that every SWAG operation takes $O(1)$ time in the worst-case, not just on average. This is accomplished by extending Okasaki's functional queue (Okasaki 1995) and removing dependencies on lazy evaluation and automatic garbage collection. **FlatFIT** (Shein et al. 2017) is another algorithm that achieves $O(1)$ time although in the amortized sense. This is accomplished by storing pre-aggregated values in a tree-like index structure that promotes reuse, reminiscent of path compression in the union-find data structure.

There are a number of other generic techniques that tend to apply broadly to sliding-window aggregation. Window partitioning is sometimes used as a means to maintain group-by aggregation and obtain data parallelism through fission (Schneider et al. 2015). When stream data items arrive out of order, a holding buffer can be used to reorder them before they enter the window (Srivastava and Widom 2004). Alternatively, in the case that the stream is formed by merging multiple sub-streams, out-of-order streams may be solved by pre-aggregating each data source separately and consolidating partial aggregation results as late as possible when doing an actual query (Krishnamurthy et al. 2010).

Examples of Application

Many applications of stream processing depend heavily upon sliding-window aggregation. This section describes concrete examples of applying sliding-window aggregation to real-world use cases. Understanding these examples helps appreciate the problems and guide the design of solutions.

Medical service providers want to save lives by getting early warnings when there is a high likelihood that a patient's health is about to deteriorate. For instance, the Artemis system analyzes data from real-time sensors on patients in a neonatal intensive care unit (Blount et al. 2010). Among other things, it counts how often the blood oxygen saturation and the mean arterial blood pressure fall below a threshold in a 20-s sliding window. If the counts exceed another threshold, Artemis raises an alert.

Financial agents engaged in algorithmic trading want to make money by buying and selling stocks or other financial instruments. Treleaven et al. review the current practice for how that works technologically (Treleaven et al. 2013). Streaming systems for algorithmic trading make their decisions based on predicted future prices. One of the inputs for these predictions is a moving average of the recent history of a price, for example, over a 1-hour sliding window.

Road traffic can be regulated using variable tolling to implement congestion-pricing policies. One of the most popular benchmarks for streaming systems, linear road, is based on variable tolling (Arasu et al. 2004). The idea is to regulate demand by charging higher tolls for driving on congested roads. To do this, the streaming system must determine whether a road is congested. This works by using sliding-window aggregation to compute the number and average speed of vehicles in a given road segment and time window.

The above list of use cases is by no means exhaustive; there are many more applications of sliding-window aggregation, for instance, in phone providers, security, and social media.

Future Directions for Research

Research on sliding-window aggregation has focused mainly on aggregation functions that are associative and on FIFO windows. Much less is known for other nontrivial scenarios. Is it possible to efficiently support associative aggregation functions on windows that are non-FIFO? Besides associativity and invertibility, what other properties can be exploited to develop general-purpose algorithms for fast sliding-window aggregation? How can SWAG algorithms take better advantage of multicore parallelism?

Cross-References

▶ Adaptive Windowing
▶ Incremental Sliding Window Analytics
▶ Stream Query Optimization
▶ Stream Window Aggregation Semantics and Optimisation

References

Arasu A, Widom J (2004) Resource sharing in continuous sliding window aggregates. In: Conference on very large data bases (VLDB), pp 336–347

Arasu A, Cherniack M, Galvez E, Maier D, Maskey AS, Ryvkina E, Stonebraker M, Tibbetts R (2004) Linear road: a stream data management benchmark. In: Conference on very large data bases (VLDB), pp 480–491

Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. J Very Large Data Bases 15(2):121–142

Bloom BH (1970) Space/time trade-offs in hash coding with allowable errors. Commun ACM 13(7):422–426

Blount M, Ebling MR, Eklund JM, James AG, McGregor C, Percival N, Smith K, Sow D (2010) Real-time analysis for intensive care: development and deployment of the Artemis analytic system. IEEE Eng Med Biol Mag 29:110–118

Carbone P, Traub J, Katsifodimos A, Haridi S, Markl V (2016) Cutty: aggregate sharing for user-defined windows. In: Conference on information and knowledge management (CIKM), pp 1201–1210

Cormode G, Muthukrishnan S (2005) An improved data stream summary: the count-min sketch and its applications. J Algorithms 55(1):58–75

Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Symposium on operating systems design and implementation (OSDI), pp 137–150

Flajolet P, Fusy E, Gandouet O, Meunier F (2007) HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In: Conference on analysis of algorithms (AofA), pp 127–146

Garcia-Molina H, Ullman JD, Widom J (2008) Database systems: the complete book, 2nd edn. Pearson/Prentice Hall, New Dehli

Gedik B (2013) Generic windowing support for extensible stream processing systems. Softw Pract Exp 44(9): 1105–1128

Gray J, Bosworth A, Layman A, Pirahesh H (1996) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: International conference on data engineering (ICDE), pp 152–159

Hirzel M, Rabbah R, Suter P, Tardieu O, Vaziri M (2016) Spreadsheets for stream processing with unbounded windows and partitions. In: Conference on distributed event-based systems (DEBS), pp 49–60

Hutton G (1999) A tutorial on the universality and expressiveness of fold. J Funct Program 9(1):355–372

Krishnamurthy S, Wu C, Franklin M (2006) On-the-fly sharing for streamed aggregation. In: International conference on management of data (SIGMOD), pp 623–634

Krishnamurthy S, Franklin MJ, Davis J, Farina D, Golovko P, Li A, Thombre N (2010) Continuous analytics over discontinuous streams. In: International conference on management of data (SIGMOD), pp 1081–1092

Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005) No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. ACM SIGMOD Rec 34(1):39–44

Okasaki C (1995) Simple and efficient purely functional queues and deques. J Funct Program 5(4): 583–592

Sajaniemi J, Pekkanen J (1988) An empirical analysis of spreadsheet calculation. Softw Pract Exp 18(6):583–596

Schneider S, Hirzel M, Gedik B, Wu KL (2015) Safe data parallelism for general streaming. IEEE Trans Comput 64(2):504–517

Shein AU, Chrysanthis PK, Labrinidis A (2017) FlatFIT: accelerated incremental sliding-window aggregation for real-time analytics. In: Conference on scientific and statistical database management (SSDBM), pp 5:1–5:12

Srivastava U, Widom J (2004) Flexible time management in data stream systems. In: Principles of database systems (PODS), pp 263–274

Tangwongsan K, Hirzel M, Schneider S, Wu KL (2015) General incremental sliding-window aggregation. In: Conference on very large data bases (VLDB), pp 702–713

Tangwongsan K, Hirzel M, Schneider S (2017) Low-latency sliding-window aggregation in worst-case constant time. In: Conference on distributed event-based systems (DEBS), pp 66–77

Treleaven P, Galas M, Lalchand V (2013) Algorithmic trading review. Commun ACM 56(11):76–85