# A Survey of Real-Time Big Data Processing Algorithms


**Devesh Kumar Lal and Ugrasen Suman**

**Abstract** Data collection and processing in real time is one of the most challenging domains for big data. The sustainable proliferation of unbounded streaming data has become arduous for data collection, data pre-process, data optimization, etc. Real-time streaming for data collection can effectively be performed by windowing mechanism. In this communication, we have discussed various windowing mechanisms such as sliding window, tumbling window, landmark window, index-based window, adaptive size tumbling window, and partitioned-based window. The reliability measure, which depends upon selection of appropriate windowing mechanism, has also been discussed. These window-based algorithms have been compared on the basis of CPU utilization, memory consumption, time efficiency, and operation compatibility. In this paper, we have surveyed various aggregation algorithms such as reactive aggregator, flatFAT, flatFIT, B-Int, DABA, and two stacks aggregator and compared them based on time complexity. Remarkably, a hybrid window mechanism has been introduced in this study which can handle the most recent data stream and variable rate of data stream by sliding window and tumbling window, respectively.

**Keywords** Aggregation algorithms · Big data · Real-time data processing · Streaming data · Window algorithms

## 1 Introduction

Big data is used for enormous data sets, which become arduous to process by traditional methods like distributive mechanisms. Big data processing challenge gets intensified as data sets acquire velocity (also known as streaming data). Streams of data may be generated from IoT sensor network, internetwork traffic data, stock market, etc. The performance of streaming data analytics in real time with minimum

D. K. Lal (✉) · U. Suman
School of Computer Science & IT, Devi Ahilya University, Indore, India
e-mail: devesh2222@gmail.com

U. Suman
e-mail: ugrasen123@yahoo.com



V. K. Gupta et al. (eds.), *Reliability and Risk Assessment in Engineering*,
Lecture Notes in Mechanical Engineering,
https://doi.org/10.1007/978-981-15-3746-2_1

latency becomes intricate. Unbounded streams of data from various sources continuously proliferate over time. This sustainable proliferation of data stream needs to process in a real-time mode, to minimize the accumulation of data sets in a particular node. Processing of data stream cannot be performed at once; it requires some finite data elements, from unbounded data set.

Sliding window is the most common mechanism for data selection. It gives a finite window particularly based on time or space [1–6]. Time-based window accumulates data sets depend upon number of clock cycle, while space-based window depends upon its size. Increase in coarse window is directly proportional to its latency so that it is adjusted between coarse and fine size of a window. Window accumulations are primarily of two types; namely count-based window and time-based window [7, 8]. Stream of data resides in window for a fixed number of counts, which is termed as count-based window. A window is specified by a fixed-size temporal extent, which usually, most recent time interval and comes under time-based window [9, 10]. Figure 1 depicts a typical windowing mechanism for streaming data. In this mechanism, fixed windows slide over unbounded data streams.

Infeasibility to process entire data stream at once procreates the concept of windowing. Here, the most recent data streams reside into a window. Windowing is widely accepted mainly because of two benefits. Firstly, window makes it possible to implement streaming version of aggregation, and secondly, the most recent data stream can be availed, which is predominant in real-time analytics as compared to historical data. Centralized and distributive window are the two types of processing mechanisms of windowing system. Centralized window creates bottleneck, where data streams are accumulated at a single node, which may discard the new stream data gathered at a node. The rate of incoming data stream might be equal or less than the processing time of the outgoing stream, else creates the accumulation of data streams [11, 12].

Distributive window partitions the window into smaller multiple sub-windows to be deployed into different nodes, for parallel processing. There exist various window techniques such as sliding window, tumbling window, hopping window, partitioned window. [5, 13–17]. The existing window techniques deal with the smaller chunks

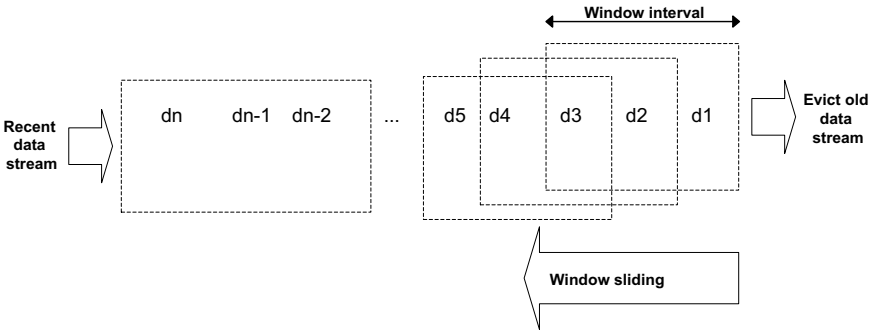


**Fig. 1** Windowing system in real-time big data

of data, formed from unbound sets of data stream, which considers the entire data stream in a sequential order, resulting in the enhancement of reliability.

Sliding window is the most widely used technique in data streaming. It uses aggregation mechanisms, which are applied over a window [18]. In this paper, we have applied window algorithms and aggregate algorithms for data streaming. Also, these algorithms are compared with their time and space complexity. Various challenges exist in windowing and aggregators algorithms to design a holistic aggregation for real-time data streams. Many aggregator algorithms are based on operators applied over single data stream node. The challenges get intensified for scalable nature of data stream. Therefore, we have introduced a scalable approach of aggregation sliding-window algorithm able to process into multiple nodes at a time.

## 2 Windowing Algorithms

Real-time unbound data stream can be evaluated with a fixed size of deterministic window where all types of aggregation and filter operations are performed. A finite number of data chunks can reside in the main memory at a particular time. This data chunk is termed as window, which contains multiple tuples. This window can be processed in centralized or distributed manner. Balkesen et al. [19] propose common dependent tuples require serial manner processing, whereas independent tuples process in parallel manner. Input streams are partitioned into independent chunks known as panes. These collective panes belong to similar window are processed and managed by same process instance.

Balkesen et al. [19] proposed sliding-window partitioning mechanism, where window is broken down into smaller multiple chunks. These multiple chunks are processed in parallel. In tumbling window, elements are assigned to a fixed length, non-overlapping windows of a specified window size such that only, one particular window evicts and inserts at a time [20].

Chen et al. [21] proposed the two classifications of sliding-window algorithm. First classification is based on data independent mode, which is extricated by equivalence partition with round-robin technique. Second approach, classifies data-dependent mode of data streams using a sliding-window index technique. In this approach, data-dependent stream uses a sliding-window index that maintains the window size. It maintains the window by inserting the new data stream into the window and evicting the old data stream resulting in low resource utilization and high system latency.

Tangwongsan et al. [22] and Hirzel et al. [23] proposed the outlining of a chunked-array queue implementation for FIFO windows. The main operations based on queues, such that insert from the back of the queue and evict the front element of the queue. Additionally, the queue provides a bidirectional iterator, which serves as pointers into the queue. The complexity is managed by maintaining a doubly-linked list of chunks, each a fixed-size array of elements.

Patroumpas et al. [18] proposed landmark window where windows have their lower or upper bound fixed at a specific time known as a landmark letting others bound

**Table 1** Comparison of windowing mechanisms

| Parameter | Window algorithm | | | |
|---|---|---|---|---|
| | CPU utilization | Memory consumption | Time efficiency | Operation compatibility |
| Sliding window | Low | Moderate | High | High |
| Tumbling window | Moderate | Moderate | Low | Low |
| Landmark window | Low | Moderate | Low | Low |
| Index-based window | Low | Moderate | High | High |
| Adaptive-sized tumbling window | High | Moderate | High | Moderate |
| Partitioned-based window | High | Low | High | High |

to follow the evolution of time. The existing windowing techniques are compared on the basis of CPU utilization, memory consumption, time efficiency, and operation compatibility. Comparison of various windowing algorithms is depicted in Table 1.

Smaller finite sets from unbound data stream procreate window, which become significant by applying operations such as aggregation, joins. Among them, aggregation is a common important feature in streaming applications. Such applications often need an aggregated summary of the most recent data in a stream, which is deemed the most relevant. A poorly chosen algorithm can cause high latencies and high memory consumption, leading to losses, missed opportunities, and quality-of-service violations. Aggregation operation depends on its window, which aims the most efficient algorithm. The most commonly used aggregation algorithms are discussed in the following sections.

## 3 Aggregation Algorithm

In traditional approach, aggregation is performed from scratch on a chunk window of streaming data. These windows are aggregated as per user-defined query. Increase in window size may reflect enhancement of overall complexity because entire window gets aggregated at a time. Tangwongsan et al. [24] proposed reactive aggregator; it manages values irreversibility, handles non-commutative of processed stream, and provides solution from out-of-order window semantics. In this aggregator, a constant stride maintains which applies over entire window. The length of stride is smaller so that instance aggregated value is stored in a flat array. A user-defined aggregated value is fetched from a query. Reactive aggregator uses FlatFAT aggregator operator mechanism, which stores aggregated results in a pointer-less and tree-based data

structure. These mechanisms reduce the overhead cost such as size of pointer, inclusion of tree-based aggregation. In FlatFAT, aggregated operator allows processing of multiple queries at time into a particular window. These aggregations are performed by storing partial aggregated values in tree leaf node. Final aggregated values are retrieved through applying multiple queries over partial aggregated results. The leaf node of a tree contains partial aggregated values while root node consists of maximum range allowed for a result.

Shein et al. [25] proposed an incremental sliding-window aggregator for real-time analytics. Here, they have introduced a flat and fast index traverse such as FlatFIT approach. FlatFIT maintains indexing structures using a two circular array and a stack. Here, stack maintains and stores indices, whereas circular array interconnected with the help of indices.

The partial aggregated results are stored in an index structure, which reduces recalculation at the time of performing final aggregation. Tangwongsan et al. [22] proposed a De-Amortized Banker's Aggregator (DABA) sliding-window aggregator. DABA uses chunked-array queue data structure for performing sliding-window aggregator operation. Here, every chunk represents a linear size array, which is interlinked with different reference pointers. In this algorithm, various reference pointers help to calculate partial aggregated results. These partial results recombine to form a complete desired aggregated result.

Base Intervals (B-Int) proposed by Arasuand Widom [26] is a final aggregation technique. It uses a multi-level data structure that consists of dynamic intervals of different lengths. The number of partial depends upon the level, such as first level consists of one partial, the second level has two partial, and so on, until we reach to maximum supported range length. The whole data structure is organized in a circular fashion so that the rightmost interval on any level is followed by the leftmost interval from the same level. When producing the final aggregate, B-Int also determines the minimum number of intervals needed to represent the desired range and aggregates the entire range. During insertions, B-Int only updates the intervals that end with the inserted value, instead of updating the entire structure bottom up until reaching the top layer. However, this slows down look-ups, since more intervals are needed to be aggregated to produce result.

Two-stack algorithm is proposed and implemented by Tangwongsan et al. [24] using FIFO window mechanism. In this approach, aggregation is performed over stack, by storing aggregated value in stack-based manner. Two-stack approach spends linear time in evict and storage. A comparative chart of aggregation algorithms is presented in Table 2. The comparison is based on time complexity and usage of its window mechanism.

The comparison of different aggregators on the basis of their time complexity indicates the adaptation of suitable windowing mechanisms. In real-time big data processing, aggregation operation applies on a fraction of window. Therefore, it becomes essential to determine appropriate window and aggregator algorithms. Table 2 shows FlatFIT aggregation approach is best suitable for sliding window and tumbling window. These aggregators may be compared with space complexity as well.

**Table 2** Comparison of aggregation algorithms

| Algorithm | Time complexity | Windowing |
|---|---|---|
| Traditional aggregators | O($n$) | Smaller window |
| Reactive aggregators | Average (log $n$) | TW, SW, LW |
| FlatFIT | $n - 1$ | SW, TW |
| B-Int | $n$ . log($n$) | Shared window |
| DABA | Worst O(1) | SW, TW, LW |
| Two stacks | Average O(1) | IBW, SW |

TW—Tumbling window, SW—Sliding window, LW—Landmark window, IBW—Index-based window

## 4 Challenges and Proposed Work

Windowing mechanisms bind with various challenges such as selection of appropriate window size. If selection of proper window with relative aggregation is not specific, then approach will become costly. A coarse size window may perceive higher latency as compared to fine-sized window. It has some limitation based on eviction and insertion of data streams in window. Aggregations approaches may be distributive, non-distributive, commutative, etc. Generalized aggregator requires specifically designed algorithms, which can contribute number of use cases. Aforementioned aggregator algorithm is based on single data stream node [27].

Here, we introduce the hybrid window model with the combination of sliding window and tumbling window. Sliding window accumulates recent data streams, while tumbling window keeps a fixed interval of data stream that is responsible for the rate of change. Tumbling window slices short interval from sliding window. Tumbling window is executed in distributed manner. Unbound data stream arrives in multiple nodes at a time, which may be extricated by distributive mode and it may also handle generalized holistic aggregation operations. Window can be distributed over multiple nodes. Partitioned window is computed separately in different node. Results are reassembled to get the aggregated value.

## 5 Conclusions

In real-time data processing, latency can be minimized by using a suitable windowing and aggregation mechanism. The selection of appropriate window and aggregation algorithm is based on various factors such as aggregation operation, type of window, requirements for latency, type of processing, and size of data streams. A generalized solution for all types of use cases in real-time stream processing is rare. Individual algorithm for window and aggregator mechanism cannot satisfy all types of requirements. This paper presents a brief study and comparison of various data stream algorithms, also identified research gaps in aforesaid algorithms.

**Acknowledgements** I offer most sincere gratitude to the Council of Scientific and Industrial Research (CSIR), Government of India, for financial support in the form of Junior Research Fellowships.

## References

1. Gibbonsand BP, Tirthapura S (2002) Distributed streams algorithms for sliding windows. In: Proceedings of the fourteenth annual ACM symposium on parallel algorithms and architectures. ACM
2. Rivetti N, Busnel Y, Mostefaoui A (2015) Efficiently summarizing data streams over sliding windows. In: 2015 IEEE 14th international symposium on network computing and applications (NCA). IEEE
3. Mousavi H, Zaniolo C (2013) Fast computation of approximate biased histograms on sliding windows over data streams. In: Proceedings of the 25th international conference on scientific and statistical database management. ACM
4. Badiozamany S, Orsborn K, Risch T (2016) Framework for real-time clustering over sliding windows. In: Proceedings of the 28th international conference on scientific and statistical database management. ACM
5. Wei Z, Liu X, Li F, Shang S, Du X, Wen JR (2016) Matrix sketching over sliding windows. In: Proceedings of the 2016 international conference on management of data. ACM
6. Wu F, Wu Q, Zhong Y, Jin X (2009) Mining frequent patterns in data stream over sliding windows. In: 2009 international conference on computational intelligence and software engineering, 2009, CiSE. IEEE, New York
7. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. ACM
8. Epasto A, Lattanzi S, Vassilvitskii S, Zadimoghaddam M (2017) Submodular optimization over sliding windows. In: Proceedings of the 26th international conference on world wide web international world wide web conferences steering committee
9. Zhang L, Zhanhuai L, Yiqiang Z, Min Y, Yang Z (2007) A priority random sampling algorithm for time-based sliding windows over weighted streaming data. In: Proceedings of the 2007 ACM symposium on applied computing. ACM
10. Braverman V, Ostrovsky R, Zaniolo C (2009) Optimal sampling from sliding windows. In: Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems ACM
11. Balazinska M, Hwang JH, Shah MA (2009) Fault-tolerance and high availability in data stream management systems. In: Encyclopedia of database systems. Springer US, 1109–1115
12. Liberty E (2013) Simple and deterministic matrix sketching. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. ACM
13. Patroumpas K, Sellis T (2009) Window update patterns in stream operators. In: East European conference on advances in databases and information systems. Springer, Berlin
14. Bhatotia P, Acar UA, Junqueira FP, Rodrigues R (2014) Slider: incremental sliding window analytics. In: Proceedings of the 15th international middleware conference. ACM
15. Badiozamany S (2016) Real-time data stream clustering over sliding windows. Diss. Acta Univ Ups
16. Zhang L, Lin J, Karim R (2017) Sliding window-based fault detection from high-dimensional data streams. IEEE Trans Syst Man Cybernet Syst 47(2):289–303
17. Golab L (2004) Querying sliding windows over online data streams. In: International conference on extending database technology. Springer, Berlin

18. Patroumpas K, Sellis T (2006) Window specification over data streams. In: Current trends in database technology–EDBT, pp 445–464
19. Balkesen C, Tatbul N (2011) Scalable data partitioning techniques for parallel sliding window processing over data streams. In: International workshop on data management for sensor networks (DMSN)
20. Marcu OC, Tudoran R, Nicolae B, Costan A, Antoniu G, Hernandez MSP (2017) Exploring shared state in key-value store for window-based multi-pattern streaming analytics. In: Proceedings of the 17th IEEE/ACM international symposium on cluster, cloud and grid computing. IEEE Press
21. Chen H, Wang Y, Wang Y, Ma X (2016) GDSW: a general framework for distributed sliding window over data streams. In: IEEE 22nd international conference on parallel and distributed systems (ICPADS). IEEE
22. Tangwongsan K, Hirzel M, Schneider S (2017) Low-latency sliding-window aggregation in worst-case constant time. In: Proceedings of the 11th ACM international conference on distributed and event-based systems. ACM
23. Hirzel M, Schneider S, Tangwongsan K (2017) Sliding-window aggregation algorithms: tutorial. In: Proceedings of the 11th ACM international conference on distributed and event-based systems. ACM
24. Tangwongsan K et al (2015) General incremental sliding-window aggregation. In: Proceedings of the VLDB endowment vol 8(7), pp 702–713
25. Shein AU, Chrysanthis PK, Labrinidis A (2017) FlatFIT: accelerated incremental sliding-window aggregation for real-time analytics. In: Proceedings of the 29th international conference on scientific and statistical database management. ACM
26. Arasu A, Widom J (2004) Resource sharing in continuous sliding-window aggregates. In: Proceedings of the thirtieth international conference on very large data bases, vol 30. VLDB Endowment
27. Cormode G, Yi K (2011) Brief announcement: tracking distributed aggregates over time-based sliding windows. PODC 11