A Pre-Aggregation Strategy Towards Reduction on Job Completion Time in Stream Process Systems

Breno Fanchiotti Zanchetta
Federal University of Rio Grande do Sul
Institute of Informatics
Room 205 – ZIP 15.064 – 91.501-970
Porto Alegre – RS – Brazil
bfzanchetta@hotmail.com
bfzanchetta@gmail.com

Paulo R. Souza Jr, Kassiano J. Matteussi
Vincius P. Perego, Julio C. S. Anjos
Claudio F. R. Geyer, Edison P. de Freitas
Federal University of Rio Grande do Sul
Institute of Informatics
Room 205 – ZIP 15.064 – 91.501-970
Porto Alegre – RS – Brazil
{prrsjunior, kjmatteussi, vpperego}@inf.ufrgs.br
{jcsanjos, geyer, edison.pignaton}@inf.ufrgs.br

Abstract

A very essential part of science of Big Data is called streaming, and treats a very small-sized data called Stream, and possesses real real time processing requirements. There are several economic real world applications that make use of this tool in order to retrieve high trending topics and make economic decisions. Regarding the most known techniques, this aggregation technique can be labeled as batching and implies in real gains on various applications fidelity with small latency trade-offs. This work aims to test many solutions of pre-aggregation in an initial controlled environment and further test it on a network-stressed scenario, towards the mission of finding the best topology configuration that reduces network congestion. Every test was performed in a private cluster and the results point towards a set of values that generate real time gains on Flink's processing motor for given intervals of configuration variables.

Index Terms: Big Data, Stream Processing, Data Aggregation.

1. Introduction

Big Data is a term in evidence in communication media such as Facebook and Twitter, which make use of this topic in their daily activities. Current statistics found in [7] presents data numbers estimatives: Facebook gets 4.75 billion shares, 4.5 billion likes, 420 million status updates and 300 million photos every day. This scenario generates tremendous amounts of data that have different types, sizes and importances. Current Big Data solutions are divided mainly into Stream processing and Batch processing. The first is used for smaller pieces of information with high income rates and, the second uses bigger sized data with slight smaller input flow.

The basic unit in Stream Processing is called stream, and may contain different purposes depending on the system that produced it. For instance, [6] focuses a new approach and language on airplane error calculation using streams generated by the airplane error logs. This approach used Streams in order to send error logs from airplane coupled sensors to its main computer for calculations that involve planes angle correction, altitude, temperature and others.

Even though it demands less complexity than a real time airplane software that prevents error calculations, Twitter still is very important in order to retrieve trending topics and other economic tendencies, as mentioned by [1]. Also, tweets consist of easily accessible data with very small size, therefore it is used in this work as dataset.

There are two main strategies on streaming: sending one tuple at a time or aggregating multiple streams into batches and sending them for processing. Roughly, aggregating streams technique show execution gain because this strategy reduces context switch on the big data processing engine. Also, it increases throughput and latency a priori with increases on the applications fidelity.

Its important to explain the feature called Window, which is an engine feature that allows its running application to perform the operators calculations together in time intervals like a physical sliding window, reducing the overhead on the application. Allied to the window, this work proposes a prototype that tests if there is an optimal aggregation variable size that generates Flinks time reduction, also aiming reduction of bandwidth latency and achieving an increase in throughput.

This work is structured in background and related work, followed by third section which specifies the methods used in this work and the proposed solution. Section four demonstrates results, followed by section five with discussions of these results. Finally, there is section six with future work trends.

2. Background and Related Work

2.1. State-of-the-art

The work contained on [2] serves as a guideline for stream application development. It approaches 8 basic rules that every stream processing environment must attend in order to respect real time requisites. This work brings more attention towards the fourth rule: The requirement of real time stream processing is that a developed engine must guarantee predictable and repeatable outcomes.

In order to attend this importance, a comparison metric was chosen to compare final output logs from each and every execution to maintain a certain level of repeatable outcomes. Logically, in real world applications there is no true way to predict stream input order, however, as [6] states, during tests a few guidelines must be respected in order to maintain this real time feature.

Also, as [3] points out, there is no standardized course of action for designing a topology for streaming applications, but a priori. One of the works contribution is justified by this argument, since it provided experiments based on empirical choices that were reassured by [8], [4] and [5]. These works also developed stream aggregation topologies without specifying the strategy behind them. All these facts reassure this paper work intents to run tests based on empirical evaluation.

The work found at [2] reduces computational cost by using a single persistent random variable that marks the lifetime of each key on the cache memory. This work does not intend to include either cached memory or time variables in stake, since time management demands clock synchronization and making usage of cached memory is not trivial. Meanwhile, the cluster random access memory is a brand resource, it must be dealt accordingly.

The work described in [10] shows how event-based stream processing may differ from aggregation technique, expliciting that the second technique is able to provide gain in executions whenever there are events out-of-order from more than one source. This was used as basis for this work, mainly because singled source datasets proved to be less afected by aggregation strategy in most of the test case scenarios and thus, the validation of this prototype might be accessed by the use of different dataset delivery strategies.

2.2. Related Work

The work contained in [9] proposes a streaming platform in real time. It tries to determine the optimal batching values in order to increase network transfer rates between multiple routes. It makes good usage of the fact that some big data providers dont charge fees to provide intranodal data transfers but charge internodal transfers.

In order to validate their hopes of reducing transfer times, their prototype perform intranodal replications and recalculate latency between the new nodes and distributed applications. Its possible that some of the new replications contain smaller latencies than the previous original data. Therefore, tests that oscillate batching sizes are performed for each of these new replications in order to detect the quickest path and best batching size.

All these calculations are performed autonomously and the system self-adapts to changes in the network, choosing the new best batching size and best route based on latency exhausting calculations. Our prototype differs from this work because it seeks to find optimal values overall for a given configuration in order to perform future tests for generic environments. Also, the previously mentioned work performs a series of calculations, batching reordering, replications and other techniques that dont convince their gains.

3. Materials

3.1. Methods and Model

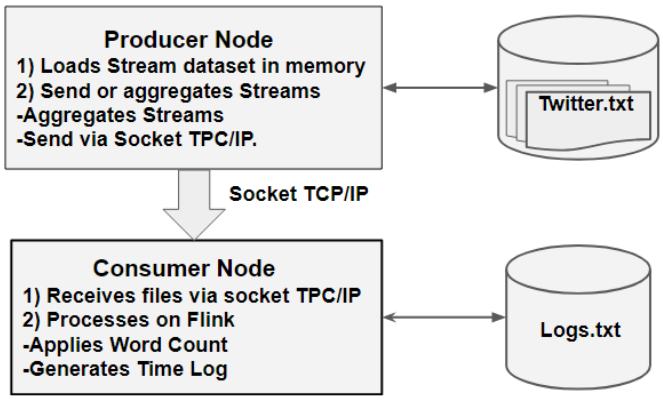

Figure 1. Model

This works used real world dataset, pre-collected by Twitters API in java, containing multiple tweets. Also, the dataset received a few documents from USA criminalistic department and AIDS statistics from health department.

3.2. Implementation

The TCP/IP protocol was used in the Python code in order to stablish communication with the processing engine and feed the dataset to it. First node of the private cluster performs the data aggregation and send it for validation job on second node. This work chose Word Count application in order to answer a question: Is there a value for pre-aggregation that generates execution time gain on [1]Flink?

In order to perform experiments on streaming, this work chose Apache Flink software to be used as stream processing engine because of it's easy to be configured and used.The earlier implementation was performed in Java language. Since procedures that involve high I/O operations with stream aggregation resulted in poor outcomes, this work moved towards a Python approach and started to show promising results.

The pre-aggregation class was developed in Python, configured to receive aggregation sizes per stream that comes in. As a single node of the cluster was configured to be the stream producer and the second was configured to hold the Flink as the consumer node.

It was set a fixed number of lines as a aggregation parameter to prove that pre-aggregation influences Flink-window processing time positively. For every stream that the Python nodes receives, it aggregates a fixed number of lines until the stream is all read. If the stream is over, the pre-aggregation fetches other streams in order to fill the entire aggregation variable.

Result: Aggregation time on Python,
Stream processing time on Flink
Initializes n aggregation size;
Initializes concatenation string;
Starts time initialization;
while *has stream file* **do**
  while *file has next line* **do**
    Concatenates line to String;
    if *String greater than n* **then**
      Send String to Flink ;
      Restarts String ;
  end
end

Algorithm 1: Aggregation Strategy

Finally, when this aggregation variable fills entirely, the concatenation is sent. Concerning configurations, 33 repetitions of each test were set for each possibility of variable permutation, including different aggregation sizes, 5 second window or not, with different sizes of parallelism, on the two given datasets. The experiment has 6 strategies of aggregation, 2 datasets, 2 types of window and 4 sizes of parallelism. Permuting these factors, the entire experiment had 6x2x2x4=96 possibilities, which the 33 experiments ran singularly, in order to achieve a close behaviour of the desired pattern (so 96x33=3168 test case total).

1 https://flink.apache.org/

3.3. Equipment

The tests were executed on a private Cluster, with 5x Power Edge 1950 8 Cores (4 Real and 4 Virtual) containing 16GB of RAM. LAN Network bandwidth is known to be 100 Mbps and it was used Ubuntu Server version 16.04 LTS.

4. Results

Results showed that small consistent datasets with no Flink window had the execution time varied negatively for all values in the training group, with one exception of the largest aggregation size (1000 concatenation size). The larger 1,6GB dataset showed that there is a slight advantage when performing aggregation with 500 units of concatenation in average.

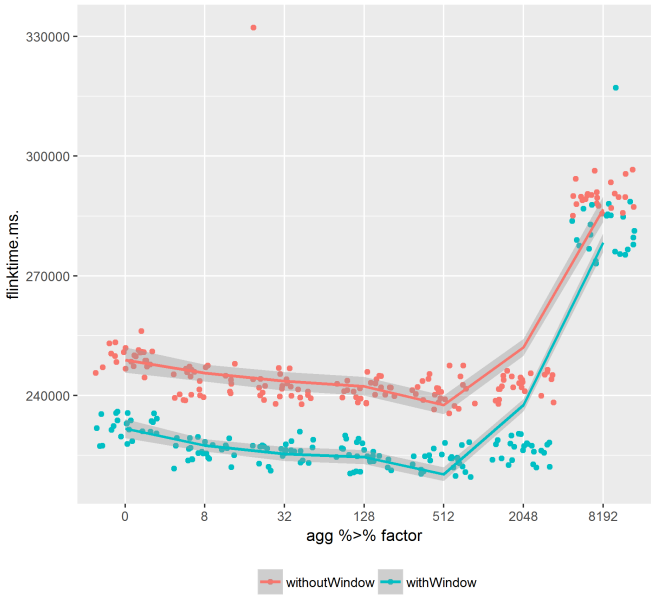

Figure 2. Big Dataset Jitter

On the other hand, Flink 5-seconded window experiments showed different results. For bigger 1,6GB dataset any aggregation showed a slight gain in execution up to 750ms. About the last test, using a 121,7MB with the same 5 second Flink window, there was an increase in processing time.

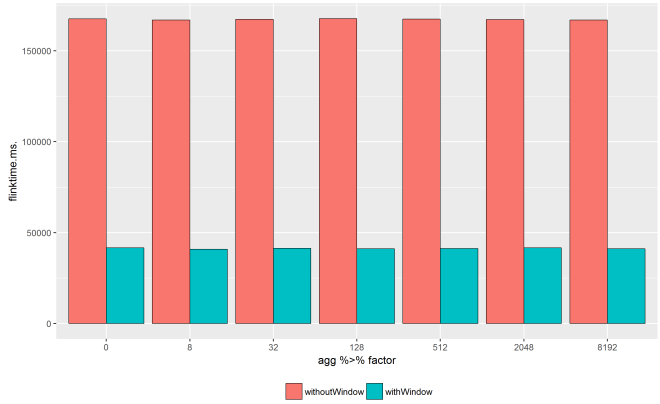

Figure 3. Tiny Dataset Jitter

Figure 1 points that for bigger datasets there is an optimal value of 512. On the other hand there is a strong effort to avoid high values of batching and no aggregation, since these strategies obtained worse results. This was visible for any value higher than 512. Figure 2 points that for smaller datasets there is also a high tendency to increase time in the engine when using a window regardless of the agreggation strategy. However, for multiple aggregation sizes on very small datasets the results were very insignificant and didn't pose to show any relative gains.

5. Discussion

At the beginning, this work tried to detect a relation between stream pre-aggregation and time gain on Flink word count application given two real world datasets. In order to test this work model, multiple runs were executed on a couple of nodes of a private cluster for the given dataset and distinctive variables of configuration. It was shown that for greatter datasets, the engine benefited from aggregation strategies and window, with best value of 512 for the given problem in the given topology. For smaller datasets, there was a gain in window use, but no significant gain in aggregation strategy.

6. Conclusion and Future Work

Future work will focus on developing dynamic strategies of updating variables settings which might be accomplished by dynamic tables. Whilst this approach focuses on tabled methods, there is also room for insertion of probabilistic models or machine learning.

Also, some changes might be done towards measurement of throughput and latency as secondary metrics. The implementation of a simulator might accomplish more results in this case.

References

[1] P. Basanta-Val, N. Fernandez-Garcia, L. Sanchez-Fernandez, and J. A. Fisteus. Patterns for distributed real-time stream processing. *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[2] N. Duffield, Y. Xu, L. Xia, N. Ahmed, and M. Yu. Stream aggregation through order sampling. *arXiv preprint arXiv:1703.02693*, 2017.

[3] R. Guerraoui, E. Le Merrer, R. Patra, and B.-D. Tran. Frugal topology construction for stream aggregation in the cloud. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. Ieee, 2016.

[4] Q. Huang and P. P. Lee. Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams. In *INFOCOM, 2014 Proceedings IEEE*, pages 1420–1428. IEEE, 2014.

[5] R. Huebsch, M. Garofalakis, J. M. Hellerstein, and I. Stoica. Sharing aggregate computation for distributed queries. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 485–496. ACM, 2007.

[6] S. Imai, E. Blasch, A. Galli, W. Zhu, F. Lee, and C. A. Varela. Airplane flight safety using error-tolerant data stream processing. *IEEE Aerospace and Electronic Systems Magazine*, 32(4):4–17, 2017.

[7] D. Noyes. The top 20 valuable facebook statistics–updated february 2015. *Retrieved from Zephoria: https://zephoria.com/social-media/top-15-valuable-facebookstatistics*, 2015.

[8] O. Papapetrou, M. Garofalakis, and A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *Proceedings of the VLDB Endowment*, 5(10):992–1003, 2012.

[9] R. Tudoran, A. Costan, O. Nano, I. Santos, H. Soncu, and G. Antoniu. Jetstream: Enabling high throughput live event streaming on multi-site clouds. *Future Generation Computer Systems*, 54:274–291, 2016.

[10] K. Wahner. Wahner, kai. "real-time stream processing as game changer in a big data world with hadoop and data warehouse." internet. 2014. [Online]. Acessado em: 30-06-2017.