Impacts of the Stream-Processing Model on Multi Geo-spatial Environments

Paulo Souza Junior, Alexandre Veith, Kassiano Matteussi, Claudio R. Geyer
Federal University of Rio Grande do Sul - Porto Alegre, Brazil
Institute of Informatics - PPGC
prrsjunior@inf.ufrgs.br

Abstract

Stream-Processing Systems (SPS) has become a crucial model in scientific and industrial domains in recent years. In this context, Multi Geo-spatial Environments are communally to host real time applications. It incurs in substantial communication costs, becoming a dominant operational expenditure factor. This work aims to evaluate the application slow-down impact in geographically distributed environments over different architecture models. The experiments were performed using a real scenario deployed on Microsoft Azure, where models are introduced and compared. Some of those models present up to 33% performance enhancement using batch-sized strategies.

1. Introduction

The most different technologies are converging to exchange data, as a consequence Internet of Things (IoT) is taking a vital place on the Big Data Analytics. Googles MapReduce (MR) has introduced a simple and scalable data processing technique that deals with the massive amount of data generated by all the things. Since the time of its introduction, Big Data Analytics has attracted a good amount of attention from both industry and academia. According to IDG, in the period 2014-2015 there was an increase of 125% in the number of organizations that implemented or deployed data-driven projects. Also, IDG estimates that companies will spend approximately 13.8 million dollars per annum on them in the next few years. Moreover, 58% of business investments are related to Big Data Analytics. Most of the initial use cases were in batch analytics (i.e., Batch-Processing), which meant that data was stored on a disk and then periodically processed by walking through it. However, Batch-Processing was not structured to have low latencies in the processing of the batches. Thus, a new model was introduced - Stream-Processing.

In recent years, this subclass of Big Data called fast data (i.e., high-speed real-time and near-real-time data streams) has also witnessed a vast increase in volume and availability. These specific data, (often denoted as events), are usually characterized by a small unit size (in the order of kilobytes) but have overwhelming collection rates. As a result of this continuous flow of data a Stream-Processing approach has emerged (Apache Storm [12], Apache Spark [15], Apache Flink [1], S4 [10], Apache Samza [16], and so on).

According to [13], Stream-Processing have been applied on multi-geospatial (i.e. multi-cloud) environments and it has pulled off for scientific discovery. Stream-processing allows to perform most of the processing independently on the data partitions across different sites and then to aggregate the results in a final phase. In some of the largest scenarios, the data sets are already partitioned for storage across multiple sites, which simplifies the task of preparing and launching a geographical-distributed processing. These scenarios include exponentially expanding data, and according to IDC by 2020 there will be around 44 Zettabytes of data that will require processing. Multi-Cloud (MC) has characteristics as heterogeneity and variety of network bandwidth. The use of MC is inevitable since a single site will not be enough to handle the high volume of data. As pointed out by Smartinsights, since 2013 the growth of data generated per minute in the main companies has been exponential.

Therefore, this article intends to overcome environmental limitations, proposing a model that is followed by a technique. From this assumption we assume that problems will arise with this kind of environment as network latency and data placement causing slow down in applications. A technique is mandatory to improve the throughput of event streams between distributed environments.

To address this challenge the data clustering and a small batches strategy will be used to obtain low latency for the stream-based processing ([3] and [14]). Finally, the evaluation is conducted by a comparative study of batch sizes, looking for particular approaches to batch the streams (i.e., time-based, event-based or hybrid form). In contemplation of determine which is the more adequate technique for the Stream-Processing in this scenario.

2. Background

This work will seek to reach a geographically distributed environment that has its decoupled functions as shown in Figure 1. The components mentioned here are related to the role of computing resources. Figure 1 presents a Geographically Distributed decoupled Environment composed by a Data Source which represents the data producer, Message Broker that handles message queues, and also by a Stream-processing engine as the core for data analysis.

Still, by the adoption of this architecture model is possible to raise the resource processing pool (e.g., stream processing clusters) at any time. The structure will guarantee the fittable in and out of the resources in the network.

The adopted kind of architecture is due to the possibility to get over the elasticity problem. In this scenario, they have created different execution clusters and therefore allowing the insertion of computational resources at any time. The structure will fit the in and out of the resources in the network.

Moreover, the distributed infrastructure for such processing model should not have restrictions on some online resources (i.e., environments should automatically adapt to

the requests, scalability). The growth of the Internet of Things (IoT) supports that point, where more and more sensors and "things" that are data sources are present in our applications.

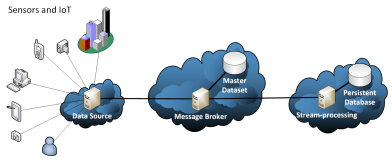

Figure 1. Geographically Distributed decoupled Environment

2.1. Multi Geo-Spatial Environments

The volume and velocity of data generation have been evolved in these last years. Consequently, it leads to the physical capacity constraint problems (i.e., single data center - Figure 2) [6].

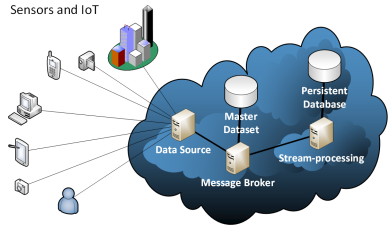

Figure 2. Single Data Center

Such restrictions related with Single Data Center can be seen as follows:

- **Memory**: Applications in Stream-processing usually make use of memory, and the data is growing exponentially. Through situations where there are enormous amount of data, the applications will have to decrease the size of the evaluation windows, and it will be able to affect negatively at the final application result;
- **Storage**: Data storage is one of the most important challenges of Big Data. It is the point that has been getting the attraction of researchers in recent years, especially in the batch-processing area [5, 2]. It involves the data placement and movement, especially problems with security. Although, both have impacts on the latency of execution;
- **CPU**: Clouds usually are environments that share the computational resource and also have limitations between the users (e.g., 300 cores in Microsoft Azure) [13]. These conditions will influence in the creation of the flows on the Stream-processing;

The structuring of a geographically distributed environment will assist in decreasing the restrictions given in a single site. Although, problems will arise related to data placement and data movement. The main one is the use of the Internet for the communication. In this scenario, it is not possible to define the time for the data transfer since the bandwidth and latency have wide fluctuations.

Therefore, methods and techniques should be evaluated to address the barriers of infrastructure. In the Stream-processing environments, there is a strong constraint related to the processing time. The result should be obtained in seconds or milliseconds (i.e., real-time or near real-time).

2.2. Stream-Processing Models

An important point that will influence the running time is the way the application will handle the data. Furthermore, as presented studies ([3] and [14]), an opportunity to overcome it is through the accumulation of the events and transfer them in data blocks (i.e., considering the time constraints given by the application). So, the latency cost of the transfer will be lower and the throughput of the events by time will be higher.

So within the options of Stream-Processing models, there are the following strategies of event aggregations: **One-at-a-time**, most of it is used to the real-time constraint. The events are transmitted directly to the Message Broker as are generated; **Batch-sized** used to the near real-time constraint. Events are retained in the source and sent to the Broker in batches.

Based on the mentioned techniques, it is important to select one that may improve the performance in proposed environment. So, we evaluate those approaches in a cloud infrastructure.

The accumulation of events will be viable in the process since it is intended to work with near real-time environments. By using batched-size model is expected to identify the best amount to accumulate or even check the interference the Internet in the environment. For the One-at-a-time approach each event will be added the network latency since it will be added only once in a Batch-sized approach.

In this study, the "mini-batch" size will vary primarily to verify the accumulation time and the impact caused by the application. To assess these impacts will use a fully distributed environment as shown in Figure 1 and seek to be identify the interference in the application run-time.

3. Decoupled Scenarios Performance Analysis

This section describes the evaluation and methodology to demonstrate the impacts of Stream-Processing Model on Multi Geo-spatial Environment. Moreover, we present the utilized tools, collected metrics and a performance analysis.

3.1. Methodology

We aims to measure all points within the decoupled scenario to be able to identify possibles bottleneck. The measured metrics were the completion time and the total number of events to verify each approach and, also we assessed the following points: events per seconds; Flink's events throughput; duration of each event;

The experiments comprise empirical evaluation performed on the Microsoft Azure cloud, at the PaaS level, using a VM on West of US as the Data Source, a VM on North of EU as the Message Broker and three VMs on East of Japan as the Stream-processing set. The system runs on A10 instances (processor Intel Xeon E5-2670 @ 2.6 GHz, 8 cores, 56GB DDR3-1600 MHz RAM and 120 GB storage) with Ubuntu Server. Each VM instance synchronizes

its time with the NTP Server. Experiments were designed based on the methodology proposed by [8] with a replication factor equal to 20 and 7 batch sizes. The experiments were performed as the following tools: **Data Source**: A Java implementation to produce parallel events; **Message Broker**: Apache Kafka 0.10.0; **Stream-processing Engine**: Apache Flink.

The stream-processing models evaluated were **One-at-a-time**: one event at a time and the **Batch-based**: batches with 2, 10, 150, 250, 500 and 1000 events. In order to evaluate the proposed environments the following tools were used: Iperf to measure the throughput and the bandwidth among the evaluated regions; Dstat-monitor was used for measuring resource consumption, CPU utilization, memory usage, disk and network I/O; Hping was used to obtain information regarding the network latency between the test environments; Ganglia has been used to monitor the reception and sending messages time through Apache Kafka and Flink.

For our experiments, we used a sentimental analysis application, as our Stream Processing Motor, where tweets were classified. The application derivates from a Natural Processing Language algorithm, based on a dictionary. Tests were performed in 140 runs, where the size of the tweet was $\approx 2KB$. In each run were sent 1.000,000 tweets where the experiment time took $\approx 12\ hours$.

3.2. Results

Figure 3 shows the entire execution over the decoupled environment, over the complete execution over all proposed scenarios, i.e., batch sizes. We noticed a significant variation in sending smaller batches. Most of it, caused by the substantial number of messages sent over the network, being more susceptible to failures and high latency that is caused mainly by the cloud services or the network links. Although, bigger batches performed less variation and

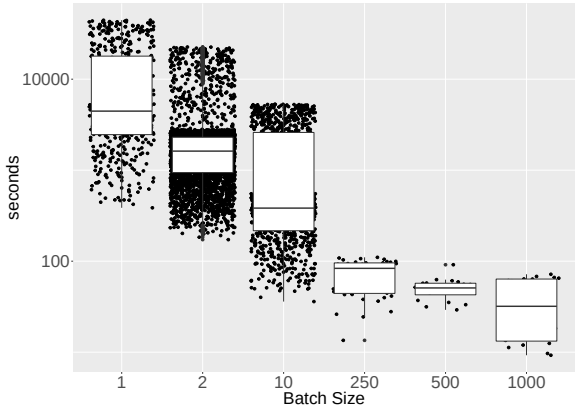

Figure 3. Execution time over the entire model

lower execution time. We can observe that in bigger batches there is fewer event consumption but with smaller processing time. Showing that for that type of environment is viable to send bigger messages than send more amounts of smaller messages. Batch sizes of 250 to 1000 shown promising results, providing stability and a shorter run time.

Since we adopt the use of a synthetic uniform data set, it has expected that the execution time may increase proportionally. However, it does not occur. The warm-up and cool-down execution of Apache Flink explain that behavior, which is the time of deployment of a new job and refresh of the framework.

Obviously, we also need to consider as well the characteristics of the evaluated application, that may benefit the approach. For this reason, the chosen application is I/O bound. It is because most of the Big Data Stream applications are considered I/O bound [7] the strategy may benefit most of the applications.

Moreover, it is possible to conclude that the aggregation is feasible since there is no further increase in execution time inside Flink. It also may prove, that there is an improvement of exploitation from the available resources since the throughput increased, but it needs further and deeply evaluations.

As seen in Figure 3 the batch sizes of 1 to 10 are significantly inferior in execution time, since the involved logistics to send plenty of messages besides using a bigger batch with multiples streams. At this point, Figure 4 reinforces that idea, considering that it presents the entire throughput time to prepare, send and process a batch-sized.

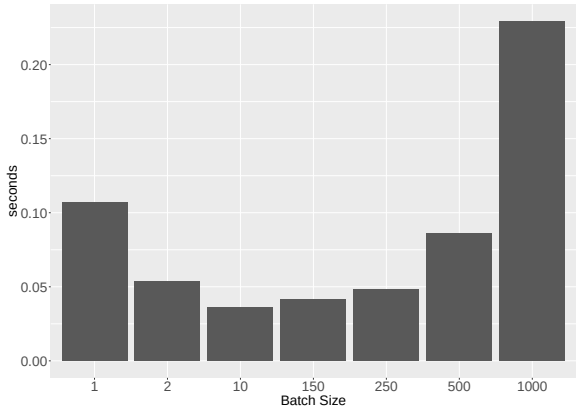

Figure 4. Event time

The Figure 4 shows the mean time of an event per each batch size. From the batch of size one to ten there's a time decrease of 66% that is a quite promising result. From the batch size of 10 to 500 it is possible to notice an increase in time. From this point we can see that the strategy is no longer efficient. Whatsoever, this is not a considerable growth since the amount messages almost doubles, comparing directly with the batch size of one message at a time. Conclusively, it is also possible to notice the degradation caused by the strategy of 1000 events in a batch. It is caused mostly by the overhead to aggregate that amount of events in memory.

4. Related Work

JetStream [14] is a set of strategies for efficient transfers of events between cloud data-centers. JetStream can self-adapt to the streams conditions by modeling and to monitor a set of context parameters. It further aggregates the available bandwidth by enabling multi-route streaming across the cloud sites. The research focus on events transfers between inter and intra-nodes. They propose an adaptive Cloud batching; the algorithm aggregates the streams

in batches in latency reduction. The main idea of JetStream is interesting. However, it just considers the latency and not the volatility. The proposal target environment where computational resources can leave unexpectedly and the scheduling policies must adapt to it.

In [9] are introduced solutions to enable elasticity over shared clusters. Mainly focused on the demand where the input rate of streams may vary drastically from each location. A monitor is proposed that discover bottlenecks in the stream data flow. Rescales are done in the capacity of processing seeking to solve the found bottlenecks. It concentrates principally on the capacity of processing, not the data itself, as proposed in this work.

SMART [4] is a framework to offer an efficient development of Big Data analysis for small and medium-sized organizations. SMART consider heterogeneous data sources and the data analysis focus on Geo-Distributed data, consider cost, failure probability, network overhead, I/O throughput and minimize the transfers between the computational resources. These parameters are not enough to work with Stream-processing and heterogeneity. To overcome the environment limitations, it is necessary to input information from the devices, such as memory, CPU speed, and storage.

Similarly, [11] purpose a generic, extensible, scalable, fine-grained re-configurable deployment and multi-cloud framework. It is based on a lightweight kernel and provides a hierarchical DSL (Domain Specific Language). DSL allows it to achieve a fine-grained level of administration. Counterpart the solution does not control the workload at the nodes and the possible failures, the Deployment Manager is just an interface to set the devices and the Virtual Machines.

5. Conclusions

Our findings demonstrate that it is necessary to evaluate the environment and the techniques used for sending batches, since it may vary according to network latency and environmental characteristics. Results proved that with small batches there is a wide variation in the time of the messages and possibly a high time within the duration of the event. In contrast to larger batches the number of messages is smaller, with less variability and preventing possible failures, thus may have a slight variation in time of the messages with relatively smaller durations.

Therefore, methods and techniques should be evaluated to address the barriers of infrastructure. In the Stream-processing environments, there is a strong constraint related to the processing time. The result should be in seconds or milliseconds (i.e., real-time or near real-time). As future work, a proposal of self-adaptative strategy may be made, to dynamic define size to the batching approach based on the constraints seen in those environments and following the necessity of each application.

References

[1] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke. The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964, Dec. 2014.

[2] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik. Scalable distributed stream processing. In *CIDR*, volume 3, pages 257–268, 2003.

[3] T. Das, Y. Zhong, I. Stoica, and S. Shenker. Adaptive stream processing using dynamic batch sizing. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 16:1–16:13, New York, NY, USA, 2014. ACM.

[4] J. C. S. dos Anjos, M. D. Assuno, J. Bez, A. Carissimi, J. P. C. L. Costa, F. Freitag, V. Markl, P. Fergus, R. Pereira, E. P. de Freitas, G. Fedak, and C. F. R. Geyer. Smart: An application framework for real time big data analysis on heterogeneous cloud environments. In *In proceedings of 15th IEEE International Conference on Computer and Information Technology (CIT-2015), Liverpool, England, UK, October 2015*. IEEE Computer Society, 2015.

[5] M. Garofalakis, J. Gehrke, and R. Rastogi. *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2016.

[6] L. Gu, D. Zeng, S. Guo, Y. Xiang, and J. Hu. A general communication cost optimization framework for big data stream processing in geo-distributed data centers. *IEEE Transactions on Computers*, 65(1):19–29, Jan 2016.

[7] B. Huang, S. Huang, J. Dai, J. Huang, and T. Xie. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, 00:41–51, 2010.

[8] R. Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[9] J. Li, C. Pu, Y. Chen, D. Gmach, and D. Milojicic. Enabling elastic stream processing in shared clusters. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 108–115, June 2016.

[10] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 170–177, Dec 2010.

[11] L. M. Pham, A. Tchana, D. Donsez, V. Zurczak, P. Y. Gibello, and N. de Palma. An adaptable framework to deploy complex applications onto multi-cloud platforms. In *Computing Communication Technologies - Research, Innovation, and Vision for the Future (RIVF), 2015 IEEE RIVF International Conference on*, pages 169–174, Jan 2015.

[12] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 147–156, New York, NY, USA, 2014. ACM.

[13] R. Tudoran, A. Costan, and G. Antoniu. Overflow: Multi-site aware big data management for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 4(1):76–89, Jan 2016.

[14] R. Tudoran, O. Nano, I. Santos, A. Costan, H. Soncu, L. Bougé, and G. Antoniu. Jetstream: Enabling high performance event streaming across cloud data-centers. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 23–34, New York, NY, USA, 2014. ACM.

[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[16] Z. Zhuang, T. Feng, Y. Pan, H. Ramachandra, and B. Sridharan. Effective multi-stream joining in apache samza framework. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 267–274, June 2016.