Aten: A Dispatcher for Big Data Applications in Heterogeneous Systems

Paulo de Souza Junior, Kassiano J Matteussi, Julio C S dos Anjos, Jobe D D Dos Santos, Alexandre da Silva Veith, Claudio R. Geyer

Aten: A Dispatcher for Big Data Applications in Heterogeneous Systems

Paulo R. R. de Souza Junior, Kassiano J. Matteussi
Julio C. S. dos Anjos, Jobe D. D. dos Santos
Claudio Fernando Resin Geyer
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil 91509–900
{prrsjunior, kjmatteussi, jcsanjos
jobe.dylbas, geyer}@inf.ufrgs.br

Alexandre da Silva Veith
INRIA, LIP
ENS Lyon, France
alexandre.veith@ens-lyon.fr

***Abstract*—Stream Processing Engines (SPEs) have to support high data ingestion to ensure the quality and efficiency for the end-user or a system administrator. The data flow processed by SPE fluctuates over time, and requires real-time or near real-time resource pool adjustments (network, memory, CPU and other). This scenario leads to the problem known as skewed data production caused by the non-uniform incoming flow at specific points on the environment, resulting in slow down of applications caused by network bottlenecks and inefficient load balance. This work proposes Aten as a solution to overcome unbalanced data flows processed by Big Data Stream applications in heterogeneous systems. Aten manages data aggregation and data streams within message queues, assuming different algorithms as strategies to partition data flow over all the available computational resources. The paper presents preliminary results indicating that is possible to maximize the throughput and also provide low latency levels for SPEs.**

***Keywords*—Stream Processing, Big Data, Heterogeneous Systems, Skewed Stream.**

I. Introduction

Organizations and companies can extract information from huge volumes of data from several devices, and this is possible since Big Data enabled the analysis of massive amounts of data. Although, there is still loss of data, without extracting potential knowledge for lack of means of mining the information or not having the necessary resources to process it. In order to enable the extraction of that knowledge, vast capacity of computational resources are necessary. Cloud computing has emerged as a scalable and volatile infrastructure, which provides an extensive number of geographically distributed data centers, which is ideal for Big Data Analytics.

The Internet of Things (IoT) has a role in that process, because of the possibility of collecting data from the most varied devices. All of this has been possible with the adoption by Big Data of the MapReduce model introduced in 2004 by [1] which enabled the capacity of analyzing huge amounts of data. It has evolved constantly and now it is being divided into two main subclasses that are not only based on MapReduce: Stream Processing and Batch Processing. Batch Processing is the manipulation of intrinsically large data (usually already available for analysis), thus not requiring real-time data. Since Batch-processing was not structured to have low latencies, a new model was introduced, the Stream-Processing model, which deals with quite small batches known as events. These specific tiny data (often denoted as events) are usually characterized by a small unit size (in the order of kilobytes) that has overwhelming collection rates.

Another Big Data application subclass called *fast data* [2] (i.e., high-speed real-time and near-real-time data streams, Stream-Processing) has witnessed an increase in volume and availability. Stream-Processing events are easily ingested by the system and are picked up by multiple and different devices, thus generating large income ratio over input streams that require little response time. Due to these characteristics, the event processing occurs in memory (like in the frameworks Apache Storm [3], Apache Spark [4], S4 [5], Apache Samza [6], Apache Flink [7], and so on). Stream Processing Engines (SPEs) like Apache Flink require shock absorbers that are responsible for the storage of events, simultaneously or definitively, and guarantee the consuming of each received event. Message queue frameworks such Apache Kafka, currently the most popular, provides scalability, replication, and repartitions that enable parallelism in consuming events [8]. Frameworks in the SPEs, typically require environments that are capable of supporting large-scale data processing, structured and unstructured data, high data ingestion rates, and resource availability variations. Nonetheless, recent studies [9]–[11] met unexpected network contention problems (e.g., low Throughput and high end-to-end latency) and poor distribution of data load as the main issues while processing Big Data Streaming applications in Cloud.

Aten aims at the implementation of solutions for Stream processing constraints - such as imbalanced load distribution and resource management caused by skewed queues and,

consequently, low throughput of events - by proposing and evaluating multiple methods of stream partitioning and communication optimization, in order to increase the throughput of events and exploitation of resource usage (e.g., CPU, Memory). Aten is built using popular Big Data tools, promoting a high-level model that can be adapted to individual needs. In addition, Aten was first designed as a module to improve and extend the SMART architecture, conceived by Anjos [12].

This paper is structured as follows: Section II presents the background necessary to understand the concepts of Big Data models, architectures, and engines. Section III discusses the related work, separated in subsections based on the focus of each one: Heterogeneous systems, Communication Optimization, and Stream Partition in which are indicated the limitations and advantages of those proposals. Section IV presents an overview of the SMART Architecture and introduces Aten as a module to integrate it. Section V presents the evaluation of the proposed model detailing the experimental setup and preliminary results, analysis of the throughput of events and resource usage to each evaluated scenario. Finally, Section VI provides conclusions and outcomes of the overall work besides discuss some future work.

II. Background

Cloud computing has become a powerful platform to perform large-scale and complex computing. Cloud scenarios provide advantages such as security, efficiency, flexibility, pay-per-use methods and scalable data storage [13] [14]. It is ideal for Big Data processing that resembles large datasets - composed of structured and unstructured data. Big data enables users to process and analyze distributed queries across multiple sites, returning resultant sets promptly through the use of Batch and Stream Processing models.

Big Data processing models can be separated in two subclasses, the batch and the stream models. There is a recent increase in stream popularity due to the requirement of real-time responses. However, it does not exclude entirely the usage of batch.

Batch processing represents the execution of a series of jobs without manual intervention (non-interactive) and leads to the execution of a series of batches as inputs, rather than as a single input [15]. Not too distant, Stream Processing (SP) scenarios are usually represented by Direct Acyclic Graphs (DAG) – in which the vertices are operators and edges are streams. It can result in geographically distributed data generation using thousands of sources, continuously and simultaneously with unpredictable flows. Moreover, SP systems work sequentially or incrementally within events or by window times (a time slice). It produces a broad range of data analysis as correlations, aggregations, filters, and sampling [16]–[18].

Architectures such as Lambda were proposed for batch and real-time processing [19] and attempts to balance latency, throughput, and fault tolerance while using batch and RT processing to provide comprehensive, accurate, real-time views of the data. In contrast, we also can also cite Kappa architecture. Kappa simplifies Lambda architecture avoiding data replication and providing event processing (device data streams) and works only on service and real-time layers [20].

Engines have been proposed for carrying out data analysis tasks in a scalable and efficient manner over most varied types of architectures [21]. Indeed, frameworks for batch processing as Hadoop MapReduce (HMR) [22] are widely used. The HMR is the most used implementation of the MapReduce model and represents an open-source version that provides resilient, high-throughput access to application data. On the other hand, stream-processing frameworks such as Apache Storm [23], Flink [24] and Spark [25] are widely used for real-time processing and data analysis. These frameworks discretize incoming data streams into temporary short time windows and then perform micro-batch processing. This process aims to improve the scalability and fault-tolerance of distributed stream processing tools by avoiding straggler tasks.

III. Related Work

The related work of this work can be separated into the following topics: Heterogeneous Systems, Communication Optimization, and Stream Partitioning. All these topics converge in SPEs, providing solutions to pertinent constraints in every related proposal.

A. Heterogeneous Systems

Cirus framework [26] proposes a generic, elastic, scalable, re-configurable deployment and multi-cloud framework, focusing on Ubilytics (Ubiquitous Big Data Analytics). The paper evaluates the genericity and elasticity of Cirus through a use case using a set of real datasets. Cyrus collects and analyzes IoT data for M2M services (machine-to-machine), which are indicated by the author as producers of vast amounts of data. In addition, scalability and orchestration of the platform are provided through the use of the Roboconf platform [27]. Three abstract components describe the architecture; each specialized component is transformed into a concrete component when the application is instantiated in the Cloud environment. These components are represented as layers in the Cirus model, which are IoT edges, message brokers and big data analytics (BDA) platforms. Also, the BDA platform of Cirus is divided into three main layers (serving, batch and speed) established by the Lambda architecture. Validation is conducted through the deployment of a real Smart-Grid use case, making future predictions of consumption based on real-time data, collected from households, and intends to evaluate the elasticity of the proposal. The prototype consists of OpenHAB as the IoT edge that produces income, Mosquitto as message broker and Apache Spark as the BDA.

In [28] proposals are introduced to enable elasticity over shared heterogeneous clusters. This work focuses on unbal-

anced input rates from streams processing system over different areas. The proposed solutions do not address constraints related to the distribution of workload in the system but aim to give resource power to bypass bottlenecks. The model proposes a monitor that discovers bottlenecks in the stream data flow. In order to solve these bottlenecks, another module, which communicates with the monitor, is responsible for re-scaling the capacity of processing of each point, by deploying additional resources. It seeks to solve the found bottlenecks, and reestablish the data flow over the network. It concentrates principally on the capacity of processing to solve network congestion problems found in stream processing systems.

The SMART Architecture (SA) proposed by Anjos [29] considers heterogeneous and geo-distributed data sources, data analysis, consider costs, failure probability, network overhead, I/O throughput and minimize the transfers between the computational resources.

The SA proposed by Anjos [29] offers an efficient architecture for Big Data analysis for small and medium-sized organizations. The implementation considers heterogeneous systems, varied data sources and geo-distributed environments. Also, are considered the cost, fault tolerance, network overhead, I/O throughput, as well as the minimization of data transfers between computational resources [30]. However, these parameters are not enough to work with Stream-processing and heterogeneity. To overcome the environment limitations, it is necessary to input the characteristics from the devices, such as memory, CPU speed and storage. Those information's are important for the load balancing what could be used to maximize the throughput of the Big Data applications.

B. Communication Optimization

JetStream [31] proposes a set of strategies for efficient transfers of events between cloud data-centers. JetStream can self-adapt to the conditions of streams by modeling and monitoring a set of context parameters. It further aggregates the available bandwidth by enabling multi-route streaming across the cloud sites. The research focuses on events transfers between inter and intra-nodes. An adaptive Cloud batching where an algorithm aggregates the streams in batches to reduce the latency. The main idea of JetStream is to overcome limitations in the transfers across different topologies of a network. However, it just considers the latency and not the volatility. The proposal focuses on scheduling policies must adapt to the environment.

A study of approaches to determine batch sizes for stream applications is present in [32] and in NEPTUNE [33]. These approaches commonly loaded the workload (batch-based) before being delivered, so a trade-off is discussed. Also, algorithms were proposed to find the smallest batching interval that ensures system stability for the longest possible period, and that adapts to unstable scenarios. The evaluations were conducted assuming metrics like latency, network throughput and bandwidth, to validate the appropriated size for different scenarios of network conditions, parallelism, operators, and resource contention. Moreover, in NEPTUNE, the solution proposed represents a dispatcher that handles workflow and reshapes data in order to optimize communication, resource utilization and seeks to guarantee real-time or near real-time responses.

C. Stream Partitioning

Distributed Stream Processing Engines (DSPE) have grown considerably since the ascension of Stream Processing in Big Data scenarios and its necessity of real-time processing. Notwithstanding, the popularity of batch processing is not losing strength, as it can be placed side by side with DSPE (e.g., Lambda Architecture). Lots of data are created day by day, and it comes with a flow, regularly within streams. There are peaks overflows of data produced in different places over several conditions that may lead to the imbalance.

In [34] is presented DSPE as a solution to handle vast volumes of data that come at high rates, and require low latency answers. This related work intends to solve load inbalance caused by skewed streams in heterogeneous clusters. The following main three solutions to partitioning data streams are discussed: **Key grouping** in which all messages with the same key are processed by the same worker, which is affirmed to be the perfect choice for *stateful* operators; **Partial key grouping** that is adapted from the traditional power of two choices [35] for load balance in map-reduce operators; **Shuffle grouping** that forwards messages blindly, usually in round-robin, it is indicated for *stateless* operators. Finally, a consistent grouping scheme also is proposed, that intends to achieve fair assignment of messages to operators considering skewed streams and heterogeneity.

Furthermore, it is possible to find in the literature different, from simple to complex, approaches to partition data streams [34], [36]–[39]. Most of the approaches intend to reduce communication costs and maximize resource usage in a certain level of parallelism. Some of the common approaches are: *Random* partitioning, in which the incoming flow is distributed randomly to each worker following a uniform distribution; *Broadcast* partitioning, which provides the same message to each worker; *Rebalance* partitioning, which partitions elements in an old-fashioned round-robin as it attempts to create equal load per partition. It is indicated in Apache Flink [37] as an ideal solution for performance optimization in the presence of data skew; *Rescale* partitioning, which also partitions elements in a round-robin way as in *Rebalance*. However, it does not completely rebalance the data stream. *Rescale* considers the level of parallelism of each worker and produces a subset of downstream operations. The subgroup of operations to which the upstream operation sends elements depends on the degree of parallelism of both the upstream and downstream operation. That being told, the load is split fairly

considering the parallelism of the upstream and downstream operations.

There are several approaches for splitting data streams and network optimization, but no pattern is set since there is not a definitive solution to every scenario. It is necessary to evaluate and experiment empirically in order to find a suitable solution. In the next section we present Aten as a dispatcher module integrated in SMART Architecture.

IV. Big Data Dispatcher

Big data applications need to handle and process huge amounts of data. In fact, the data analysis requires as fast as possible real-time solutions to craft data insights. The SMART presents an ideal approach to support big data processing and analysis. In this section we present SMART and Aten which is placed inside the SMART architecture and has concerns related to the flow, size and repartition of data.

A. SMART Architecture Overview

SMART Architecture (SA) presents a framework that offers an efficient deployment of Big Data analysis for small and medium-sized organizations [29]. The complete overview of SA is presented in Figure 1.

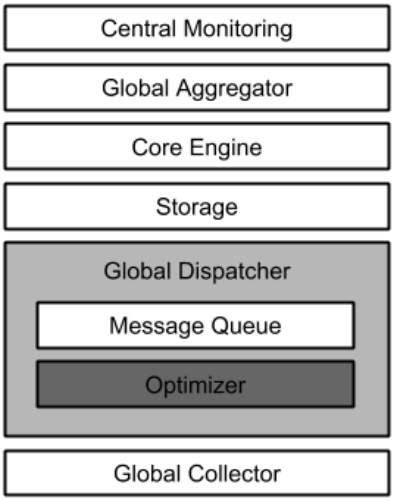

Fig. 1. SMART Architecture

Based on Figure 1 and following the top-down approach, we describe the Central Monitoring of SA as a module to monitor distributed systems for both homogeneous and heterogeneous environments, which enables monitoring clouds, grids, and IoT devices. Also, it offers a user-friendly Interface that shows information and insights in real time; The **Global Aggregator** orchestrates data aggregation results and keeps it safe for end-users; **Core Engine** supports hybrid processing such as the provisioning of streaming and batch applications at the same time over Cloud/Multi-Cloud and Multi-Grid environments.

The intermediate results, processed in the Core Engine, are serialized for the Global Aggregator that carries out the data consolidation; The **Storage** layer handles data in protected and unprotected mode. The data are stored in relation or non-relational databases; The **Global Dispatcher** has as objective to decouple data from the lower layers in the message queue mechanism.

In the following section, we present in details Aten a as dispatcher module of SA; **The Global Collector** coordinates the management and obtainment of data from several sources and control the data integrity mechanisms. The data is collected and serialized under a standard TCP/IP, which forms the communication stack for the Global Dispatcher.

B. Model Implementation

Aten is placed at the Global Dispatcher module of SA. In SA the data is decoupled from the lower layers in the message queue mechanism. It is put in a FIFO queue so that it can be distributed to servers in accordance to the availability of their resources in both Cloud or Multi-Cloud, and Grid or Multi-Grid environments. Furthermore, a simulation process can be used, which implements an execution time prediction that will be used by the Decision Engine to improve the accuracy of the scheduling mechanism [40].

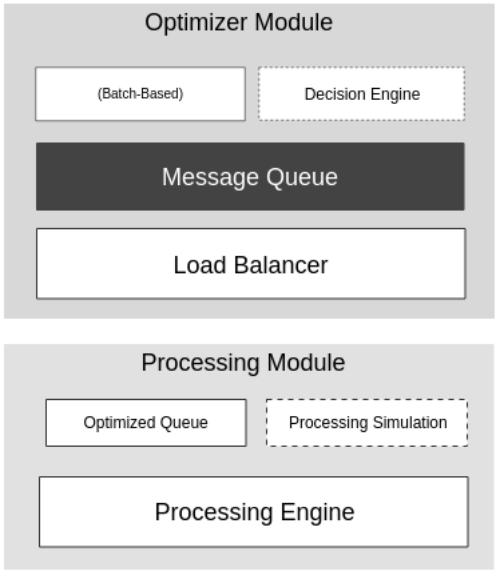

Fig. 2. Aten Modules

The inner view of Aten structure is shown in Figure 2, which is separated into two layers: the optimizer layer analyses the volume of input data and employs the Decision Engine to make decisions about message size (batch-based, data-resize, buffering events) and data through distinct environments in the load balancer (data stream partition); the processing layer is responsible for consuming data from the queues and providing a result. Furthermore, SA provides a simulation to predict the behavior of hybrid infrastructure distribution, which is ideal

for lambda architecture environments, but this is not the focus of this proposal.

The batch-based module sets the ideal message size in order to optimize communication, as a static number value *n* that defines the amount of messages to be buffered, this value is determined empirically (since there is not an ideal value, but one that is found empirically [41]). This module reduces the number of messages being transmitted over the network, consequently reducing the number of failures, message packaging time, and requests. This approach promotes a significant increase in the throughput in SP engines.

The decision engine (DE) concerns the data stream partition algorithm; it assumes the characteristics of the environment and defines the algorithm that will handle the message queues. The load balancer (LB) is responsible for running that algorithm, setting the approach and defining the flow partition to each data consumer. There are several algorithms to split data stream, such as *Random*, shuffle, *Broadcast*, and so on. Thus, we evaluated these algorithms in an isolated scenario as well attempted to show the effect it will cause in Big Data streaming applications alongside different solutions.

Once the algorithm of data stream partition is defined, a new optimized queue is generated, which contains the partitions that are consumed by the Processing Engine (PE). The PE is implemented using Apache Flink and consumes data from the Apache Kafka topics. Aten LB is implemented with Apache Kafka, which is responsible for managing those queues and providing the necessary replication and partitions in order to achieve ideal parallelism. Alongside LB, we also explored the Apache Flink partition API, which allows implementing customized algorithms of the data partition such as *Rebalance* and *Rescale* which fit the changes and behaviors of the framework. It attempts to update data stream partitions assuming the available resources, but does not update it dynamically.

The following section discusses the evaluation of different setups with and without the Aten's model, the experimental platform, and provides results related to the performance of Aten in a Cloud Computing Platform.

V. Evaluation

This section presents the experimental environment used to evaluate Aten, followed by its performance analysis. The evaluation is conducted in order to demonstrate the functionality and effectiveness of Aten dispatching events in Big Data applications.

A. Experimental Environment

The experiments were performed with Microsoft Azure Cloud Computing Platform & Services in distributed datacenters. The created setup intended to analyze the viability of the solution on multi-cloud environments (i.e., multi-tenancy, resource consumption, application behavior, latency). All experiments were performed selecting Virtual Machines (VM) provided by Microsoft Azure.

In order to validate the proposal, the following VMs characteristics were selected: D11 instances, 2 virtual cores, 2GB RAM and 20GB SSD storage; A3 Basic instances with 4 virtual cores, 7GB and 120GB HDD storage; A0 Basic instances with 1 virtual cores, 1GB RAM and 120GB HDD storage; A10 Basic instances with 8 virtual cores, 56GB RAM and 1TB HDD storage. Every VM instance has the Intel Xeon E52670 2.6GHz processor, DDR31600MHz RAM and the Operational system is Ubuntu Server 16.04.

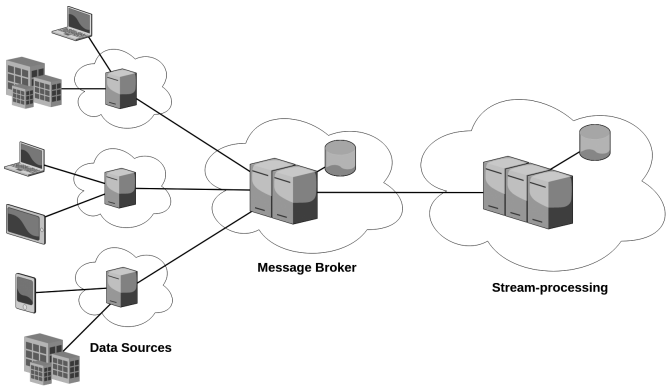

Fig. 3. Experimental Platform

Different types of VM were selected to provide a level of heterogeneity. However, the producers are equal in resource type, but the data generation rates are set. This is necessary to provoke the skewed flow of messages in the environment. The dataset is composed of *tweets* previously collected using Twitter API. All *tweets* were filtered using the last US election as subject. The dataset was composed by 10 GB where each producer receives a portion of the data "2GB slice". If necessary, this data distribution is repeated again.

The distribution of the experimental platform is illustrated in Figure 3, which is made in three levels: the data producer using 10 D11 instances in which each instance produces data as 100 clients (threads) in different ratios, the producer is implemented in Python and it is located in a Microsoft Azure Datacenter in Brazil; The broker concerns the temporary storage of the messages to be later consumed by the processing engine. It is distributed in 3 A3 Basic VM instances that are managed by Zookeeper[1] and runs Apache Kafka as message queue framework, located in the east US datacenter. Finally, the processing engine is composed by 4 VMs instances with disparate resource capabilities: A10 (Smart-f-Master), D11 (Smart-f-Slave1), A2 (Smart-f-Slave2) and A3 (Smart-f-Slave3), located in the west US datacenter. This unusual setup is necessary to achieve heterogeneity of resources. In the software layer, the framework solution for Big Data Streaming used is Apache Flink, where an application of word count is evaluated.

[1]https://zookeeper.apache.org/

Experiments were designed based on the methodology proposed by [42] with a replication factor equal to 30 and the variables discussed below. The evaluated scenarios were: *Default*, which is Flinks default partition of streams; *Broadcast*, which distributed all stream flows to every machine; *Random*, which selects the destination randomly; *Rebalance*, which adapts the stream flow change; and *Rescale* by parallelism level. In order to verify each approach, the measured metrics were completion time, Flinks throughput per elements (events), memory and CPU consumption.

B. Performance Analysis

The throughput of elements (events) can be seen in Figure 4, in which the right title of each graphic names its algorithm, but not the *Default*, which is the regular execution with no algorithm at all. Columns 1 and 128 represent the communication optimization, in which the value 1 represents the one-at-a-time approach and value 128 represents the batch-sized approach with static value equals to 128. The filled area represents the machines behavior in the PE for each allocated machine (Smart-f-master, Smart-f-slave1, Smart-f-slave2, Smart-f-slave3). Furthermore, the entire execution is relative to the consumption of all the dataset described in the previous section.

The Table I comprises the overall performance of both approaches: default (1) x batch-based (128). This is presented to support the numerical analysis of the behavior provided by Figure 4.

TABLE I
OVERALL PERFORMANCE

Algorithm	Gain (%)
Broadcast	-1,2
Default	26,6
Random	20,2
Rebalance	4,1
Rescale	44

The execution of each approach is quite different, since *Broadcast*, *Rescale* and *Default* present less execution time to process the received messages. It is evident that the distribution of data is not ideal for scenarios using the Random algorithm since it presents the most prolonged execution time to process the entire dataset. The *Random* algorithm represents the worst result since it has a low throughput of events and the distribution is nearly equal in every machine. The *Rebalance* algorithm presents better results, although it is not ideal, since there still is inadequate distribution and low throughput.

The *Random* and *Rebalance* results are closer, which is indicated by the fact that *Rebalance* defines the data stream flow in round-robin which is pretty much what the *Random* algorithm does but in a random order. Since it does not update over the execution, this will result in less effective performance if the defined partition is already inefficient.

The *Default* execution presents well-distributed data with high throughput, at least in the beginning which deteriorates over the execution, principally because of the introduction of the skewed data generation. The initial peak in *Default* would represent an ideal distribution with high throughput, which thoroughly explores resource capability. It is also visible the difference of the batch-based that slightly increases the throughput and becomes clear of its effectiveness after a while, finalizing first than the one-at-a-time.

The *Broadcast* also presents high throughput and can process the whole data faster than the *Default* execution. This is clear since the data is available to every machine and does not have any dependency that may delay the execution. Of course the less capable machine, Smart-f-slave3 does not process much, since it turns out to be overlapped by the other machines. This solution does not seem ideal for this particular case since all data is being repeatedly processed in the entire capacity of every machine and possibly being discarded if it was already processed. Thus, it is a fact that it improves resource usage and throughput, but perhaps not in the cleverest and cheapest manner.

Finally, *Rescale* results indicate it as the best suitable solution for this particular case, in which the data is being adapted considering the parallelism level and capability of the machines. It presents higher throughput than the other algorithms for every machine, which indicates optimal resource usage. Furthermore, the batch-based approach culminates in even higher throughput as can be seen the gain in Table II, reducing the event duration time and providing faster answers.

TABLE II
BATCH-BASED GAIN

Algorithm	Gain (%)
Default	0
Broadcast	16
Random	-32,7
Rebalance	-69
Rescale	33,9

The results presented in Table II indicates the gain of each algorithm in comparison with obtained time of *Default* algorithm used by Flink.

VI. CONCLUSION AND FUTURE DIRECTIONS

Aten is a high-level model that is located between Apache Kafka and Flink, seeking to optimize the communication and data flow to achieve improved data distribution within heterogeneous systems and culminate the usage of network, CPU and Memory.

Preliminary results show the effect of different algorithms in Aten, allowing throughput and resource exploitation to be

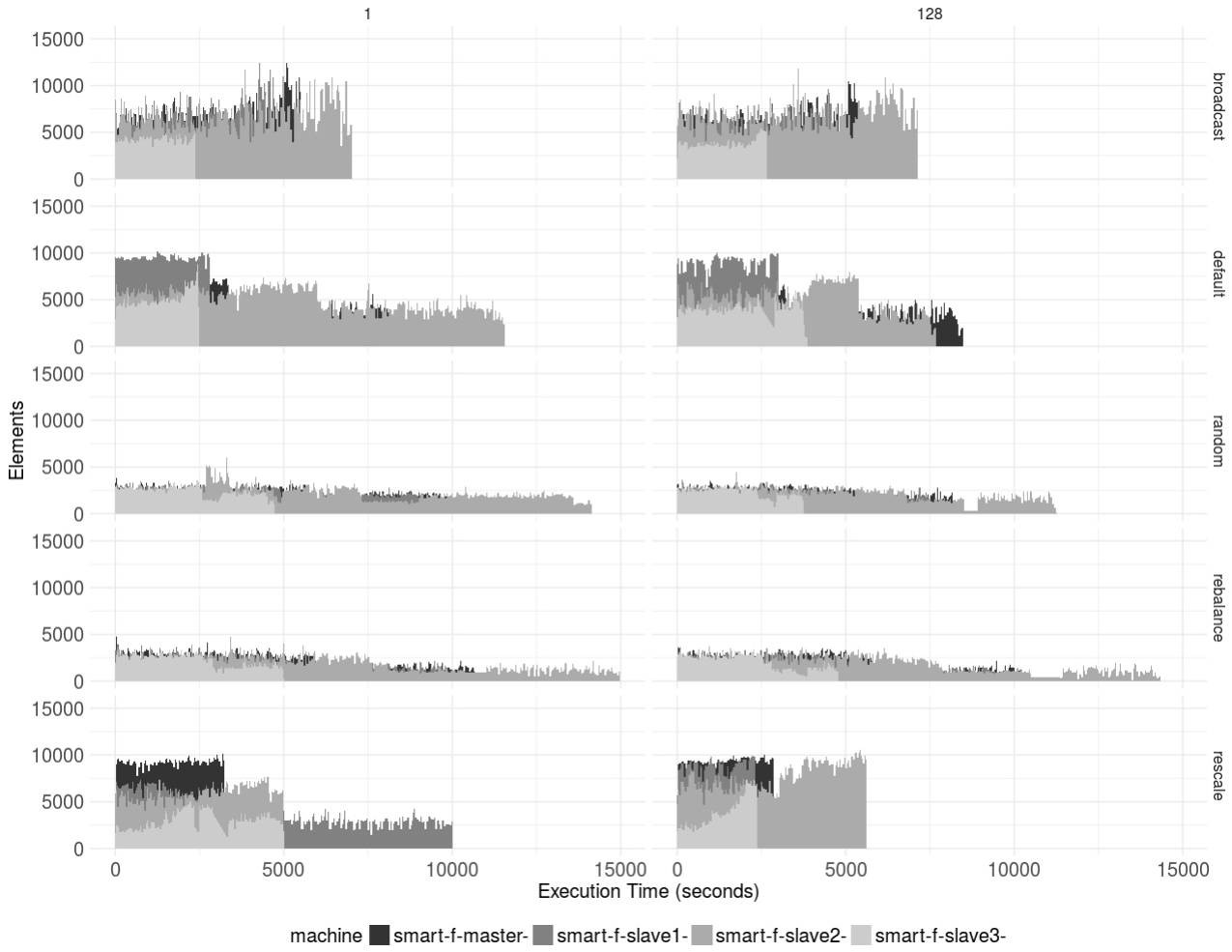

Fig. 4. Throughput of elements

increased. *Rescale* algorithm turns out to be the most effective for this scenario, but it may not be the same for different applications. It is also noted that even for algorithms that shown poor performance like *Random* and *Rebalance* was possible to increase its throughput by changing the default batch size. However, in order to maximize, even more, the throughput of the streaming systems a more detailed evaluation is required, since the performance is workload and application sensitive.

In future works, we intend to evaluate dynamic approaches that self-adapt to changes such as data skew and volatile environments in the multi-cloud environment. These approaches concern the size of the batch-based assuming application constraints (i.e., window size), and provide dynamic and adaptive algorithms to partition data stream over environmental changes with a more detailed evaluation.

ACKNOWLEDGMENT

This work has been partially supported by Microsoft Corporation, the project ”GREEN-CLOUD: Computacao em Cloud com Computacao Sustentavel” (16/2551-0000 488-9) and ”SmartSent” (#17/2551-0001 195-3) from FAPERGS and CNPq Brazil, program PRONEX 12/2014 and PROPESQ-UFRGS-Brazil for supporting this work.

REFERENCES

[1] J. D. Sanjay Ghemawat, “Mapreduce: Simplified data processing on large clusters,” 2004, accessed on: 07/08/2017. [Online]. Available: Retrievedfrom:http://labs.google.com/papers/mapreduce.html

[2] N. Miloslavskaya and A. Tolstoy, “Application of big data, fast data, and data lake concepts to information security issues,” in *Proceedings of the 4th International Conference on Future Internet of Things and Cloud Workshops. FiCloudW’16*. IEEE, 2016, pp. 148-153.

[3] 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 International Conference on Management of Data. SIGMOD ’14*. ACM, 2014, pp. 147-156.

[4] A. Inc, “Apache spark: Lightning-fast cluster computing,” 2016, accessed on: 07/08/2017. [Online]. Available: Retrievedfrom:http://spark.apache.org

[5] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, “S4: Distributed stream computing platform,” in *Proceedings of the International Conference on Data Mining Workshops. ICDMW’10*. IEEE, 2010, pp. 170-177.

[6] Z. Zhuang, T. Feng, Y. Pan, H. Ramachandra, and B. Sridharan, "Effective multi-stream joining in apache samza framework," in *Proceedings of the International Congress on Big Data. BigData Congress'16*. IEEE, 2016, pp. 267-274.

[7] 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 International Journal on Very Large Data Bases*, vol. 23, no. 6, pp. 939-964, 2014.

[8] P. L. Noac'h, A. Costan, and L. Boug, "A performance evaluation of apache kafka in support of big data streaming applications," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec 2017, pp. 4803–4806.

[9] J. Abawajy, "Comprehensive analysis of big data variety landscape," *Journal of Parallel, Emergent and Distributed Systems. Taylor & Francis*, vol. 30, no. 1, pp. 5-14, 2015.

[10] M. G. Xavier, I. C. De Oliveira, F. D. Rossi, R. D. Dos Passos, K. J. Matteussi, and C. A. De Rose, "A performance isolation analysis of disk-intensive workloads on container-based clouds," in *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP'15*. IEEE, 2015, pp. 253-260.

[11] M. G. Xavier, K. J. Matteussi, F. Lorenzo, and C. A. De Rose, "Understanding performance interference in multi-tenant cloud databases and web applications," in *Proceedings of the International Conference on Big Data, Big Data'16*. IEEE, 2016, pp. 2847–2852.

[12] J. C. S. dos Anjos, M. D. Assuno, J. Bez, C. Geyer, E. P. de Freitas, A. Carissimi, J. P. C. L. Costa, G. Fedak, F. Freitag, V. Markl, P. Fergus, and R. Pereira, "Smart: An application framework for real time big data analysis on heterogeneous cloud environments," in *Proceedings of the International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, CIT/IUCC/DASC/PICOM'15*. IEEE, 2015, pp. 199–206.

[13] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of big data on cloud computing: Review and open research issues," *Journal of Information Systems. Elsevier*, vol. 47, no. Supplement C, pp. 98-115, 2015.

[14] C. Rista, D. Griebler, C. A. F. Maron, and L. G. Fernandes, "Improving the network performance of a container-based cloud environment for hadoop systems," in *Proceedings of the International Conference on High Performance Computing Simulation. HPCS'17*. IEEE, 2017, pp. 619–626.

[15] Y. Ikura and M. Gimple, "Efficient scheduling algorithms for a single batch processing machine," *Oper. Res. Lett.*, vol. 5, no. 2, 1986.

[16] Amazon, "O que são dados em streaming?" 2016, accessed on: 04/05/2017. [Online]. Available: https://aws.amazon.com/pt/streaming-data/

[17] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, "A catalog of stream processing optimizations," *Journal of Computing Surveys. ACM*, vol. 46, no. 4, pp. 46:1-46:34, 2014.

[18] T. Buddhika, R. Stern, K. Lindburg, K. Ericson, and S. Pallickara, "Online scheduling and interference alleviation for low-latency, high-throughput processing of data streams," *Transactions on Parallel and Distributed Systems. IEEE*, vol. PP, no. 99, pp. 1-1, 2017.

[19] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Manning Publications Co., 2015.

[20] J. Meehan, S. Zdonik, S. Tian, Y. Tian, N. Tatbul, A. Dziedzic, and A. Elmore, "Integrating real-time and batch processing in a polystore," in *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*. IEEE, 2016, pp. 1-7.

[21] M. D. de Assunção, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," pp. 1- 17, 2018.

[22] A. Hadoop, "Hadoop," 2009, accessed on: 04/05/2017. [Online]. Available: http://hadoop.apache.org

[23] A. Storm, "Storm documentation," 2014, accessed on: 04/05/2017. [Online]. Available: http://storm.apache.org/documentation/Home.html

[24] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. IEEE*, vol. 36, no. 4, pp. 17-29, 2015.

[25] Apache Spark, "Spark streaming programming guide," 2014, accessed on: 04/05/2017. [Online]. Available: http://spark.apache.org/docs/latest/streaming-programming-guide.html

[26] L. M. Pham, A. El-Rheddane, D. Donsez, and N. de Palma, "Cirus: an elastic cloud-based framework for ubilytics," *Annals of Telecommunications*, vol. 71, no. 3, pp. 133-140, 2016.

[27] L. M. Pham, A. Tchana, D. Donsez, N. de Palma, V. Zurczak, and P. Y. Gibello, "Roboconf: A hybrid cloud orchestrator to deploy complex applications," in *2015 IEEE 8th International Conference on Cloud Computing*, 2015, pp. 365–372.

[28] 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)*, 2016, pp. 108–115.

[29] J. C. S. dos Anjos, M. D. A. ao, 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.

[30] V. A. da Silva *et al.*, "Strategies for big data analytics through lambda architectures in volatile environments," in *Proceedings of Preprint arXiv. TA'2017*. Elsevier, 2016, pp. 114-119.

[31] R. Tudoran, A. Costan, O. Nano, I. Santos, H. Soncu, and G. Antoniu, "Jetstream: Enabling high throughput live event streaming on multi-site clouds," *Journal of Future Generation Computer Systems. Elsevier*, vol. 54, no. Supplement C, pp. 274-291, 2016.

[32] T. Das, Y. Zhong, I. Stoica, and S. Shenker, "Adaptive stream processing using dynamic batch sizing," in *Proceedings of the International Symposium on Cloud Computing. SOCC'14*. ACM, 2014, pp. 16:1-16:13.

[33] T. Buddhika and S. Pallickara, "Neptune: Real time stream processing for internet of things and sensing environments," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 1143–1152.

[34] M. A. U. Nasir, H. Horii, M. Serafini, N. Kourtellis, R. Raymond, S. Girdzijauskas, and T. Osogami, "Load balancing for skewed streams on heterogeneous cluster," *CoRR*, vol. abs/1705.09073, 2017.

[35] M. A. U. Nasir, G. D. F. Morales, D. García-Soriano, N. Kourtellis, and M. Serafini, "Partial key grouping: Load-balanced partitioning of distributed streams," *CoRR*, vol. abs/1510.07623, 2015.

[36] B. Gedik, "Partitioning functions for stateful data parallelism in stream processing," *The VLDB Journal*, vol. 23, no. 4, 2014.

[37] A. Katsifodimos and S. Schelter, "Apache flink: Stream analytics at scale," in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, April 2016, pp. 193–193.

[38] A. H. Atashkar, N. Ghadiri, and M. Joodaki, "Linked data partitioning for rdf processing on apache spark," in *2017 3th International Conference on Web Research (ICWR)*, April 2017, pp. 73–77.

[39] V. Cardellini, M. Nardelli, and D. Luzi, "Elastic stateful stream processing in storm," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, July 2016, pp. 583–590.

[40] J. C. Anjos, I. Carrera, W. Kolberg, A. L. Tibola, L. B. Arantes, and C. R. Geyer, "Mra++: Scheduling and data placement on mapreduce for heterogeneous environments," *Future Generation Computer Systems*, vol. 42, pp. 22 – 35, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X14001642

[41] Q. Huang and P. P. Lee, "Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams," in *Proceedings of the 33th Annual International Conference on Computer Communications, INFOCOM'14*. IEEE, 2014, pp. 1420-1428.

[42] R. Jain, *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*, 2nd ed. Wiley, 1991.