Online Data Processing on S4 engine: A Study Case on Natural Disasters

Pablo González Cantergiani
Estudiante Magíster en Ingeniería Informática
Universidad de Santiago de Chile
Email: pablo.gonzalezca@usach.cl

Daniel Wladdimiro Cottet
Estudiante Magíster en Ingeniería Informática
Universidad de Santiago de Chile
Email: daniel.wladdimiro@usach.cl

***Abstract*—Streaming data processing has gained a lot of attention the last years due to their ability to process data in an online manner, allowing to support time-critical tasks such as authorities decision making on emergencies, fraud detection, among others. In this work we analyze a study case of real-time stream processing on the Yahoo! S4 engine. We propose a model to detect people's needs in case of a natural disaster using social network data. We evaluate its performance focusing the experimentation on how to define replication level for the processing operators. Our results show that the complexity of the tasks performed by the engine operators are critical for the whole system performance. We demonstrate that replicating these operators the system can improve up to a 20% the load balancing among the processing units and reduce the total processing time up to 19%.**

***Keywords*—Performance, load balancing, data streaming proccesing, parallel programming, distributed computing, social networks, natural disasters.**

I. Introduction

In the last years Internet users and the information generated by them have grown exponentially. With the arrival of Web 2.0 paradigm, users have change the way they interact creating a huge increment of the data available. A clear example are social networks such as Facebook [1] and Twitter [2]. These applications deal every day with millions of users that generate a stream of data as a result of their online interactions. Traditionally, data is stored to be analyzed in a off-line manner, however, many applications such as banks fraud detection, Twitter trending topics calculation and Web search engines click counting require of on-line processing to give near real-time responses.

Stream processing engines (SPEs) are specially designed to cope with real time data processing. SPEs use a graph oriented paradigm, representing operators as vertices, also called processing elements (PEs) and edges represents the flows of I/O data (Fig. 1). This organization allows SPEs to cope with thousands of events per second. Solutions such as Apache S4 [3], Storm of Twitter [4], TimeStream [5], StreamCloud [6], SEEP [7], D-Stream [8], Samza [9], WheelMill [10] and Kinesis of Amazon [11] operate under this common basis where operators (filters, aggregators, counters among others) are organized on a graph and they are distributed over the nodes to be processed.

Events are described as a tuple (key, value) which are distributed based on their keys generated using uniform hashing techniques. Key based data distribution has been widely used for sharing load among participants in many scenarios: P2P environments, Web Search engines, among others. However, under highly skewed data distributions only few keys will occurs frequently overloading those nodes in charge of their processing (hot spots).

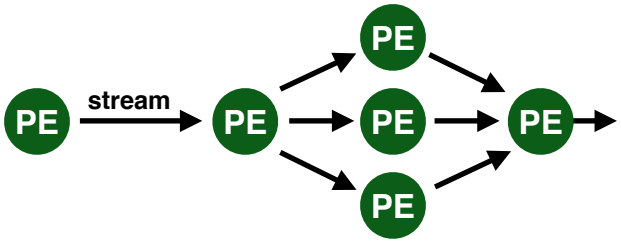

Fig. 1. Stream processing paradigm.

Nowadays real-time processing is required to assist many processes such as authorities decision making in case of emergencies or disasters. In this work we are particularly interested on the former scenarios since Chile is frequently hit by natural disaster such as earthquakes. Since 1990, Chile has lived more than 50 emergencies, among them, some of the strongest earthquakes recorded. In 2010 earthquake, social networks such as Twitter had a strong influence in people finding and information dissemination about the disaster.

Motivated by this we have built a Twitter classification tool that detects people's needs in post-disaster scenarios (see Section III). In order to cope with real-time processing we have created an implementation of this tool over a well-known stream processing engine developed by Yahoo! called S4. Even though our tool reaches good results detecting and classifying needs, in this work we are interesting in analyze the load balancing issue that arise in this type of processing engines.

S4 employs a push-based method to distribute events to the processing nodes. We observe that S4 push-based method suffers of load balancing problems, overloading those processing units in charge of the most frequent keys or the more complex operators. In order to deal with this problem we study the performance impact of replicating the operators in order to share load among multiple replicas. SPEs do not provide the tools to define the level of replication of the system or which component must be replicated.

Using a Twitter dataset from the Chilean Earthquake we

recreate the stream of tweets generated between February 27 - March 2, 2010 and classify them in order to recognize people's needs such as: information, water, electricity and food. This classification tool was developed in the context of a PMI-InEs project at University of Santiago. Our results shows that operators replication allows to reduce processing time in more that a 19% for 1M tweets processed and load balancing is improved up to 20%. These results are promising since a greater impact can be reached on massive tweets processing.

In this work we analyze a study case of real-time stream processing, where we propose a model to detect people's needs in case of a natural disaster using social network data. We evaluate its performance focusing the experimentation on how to define replication level for the processing operators.

The remainder of this article is organized as follows: In Section II we present the related works. Section III presents our processing model specifying the techniques employed to reach a scalable processing and the tasks deployed on the operators. Section IV details the processing steps and finally in Section V we present our concluding remarks and future work.

II. Related work

A. Stream Processing Engines

Currently there is a variety of tools to process big data, generally based on a programming framework called MapReduce proposed by Google [12]. As new requirements arise from the diversity of information, emerging platforms have been developed for real-time processing which are supported for example by Apache, Amazon, Twitter, Google among others.

These tools, called Stream Processing Engines, work with the definition of a topology, usually as a directed graph where each node represents operators or task and the edges stream flow, although certain tools provide the flexibility to allow the generation of cycles or that events re-enter a previous task, since it works on data stream processing is suggested it to be always a downstream imitating a group of pipes following forking paths of the graph.

Performance on SPE has been analyzed by Zhang et al. [13] optimizing stream application to use minimal computing resource to sustain maximal event throughput. Authors state that characterizing the resource usage of PEs is critical to performance optimization. Zhang et al. uses workload characterization to predict PE complexity.

Towards the end of achieving auto-scalability some works like [14] and [15] recently applied replication as our work. However, they are focused on migrations and operators merging. Heinze et al. [16] has applied its elasticity techniques to a real-world use case based on different workloads from the Frankfurt stock exchange. Other real-world applications implemented by SPEs are for example social network such as Twitter [3] and cloud monitoring [17].

B. Apache S4

Apache S4 is a multipurpose platform that enables distributed and scalable processing data streams from different data sources [3], and is strongly influenced under the MapReduce model.

The goals of the system are:

- Providing a simple interface to the application development for processing information in streaming.
- An architecture that allows to work on both cluster and home computer hardware.
- Using a decentralized and symmetric architecture, all nodes share the same responsibility and functionality.
- Presenting the programmer a flexible design and as standard as possible.

S4 basic units are the processing elements (PEs) and the messages exchanged between them. Each PEs is instantiated and distributed uniformly among the nodes used by S4.

The messages exchanged between the processing units are called events and are described as a pair (key, value). Each time a processing unit sends a message a hash function is applied to the key that maps the event to the corresponding node and therefore to the PE responsible for the key, if this did not exist, the node creates a new PE and the value is returned.

On S4, operators are called PE which are associated with a key for the events that will consume and a key for the events to be sent. When we work with a tuple of key-value messages, for each key there may be more than one associated value, so changing its value allows to replicate processing units in the system, as seen in Fig 2.

III. Processing Model

In this work we were interested on processing tweets generated after a natural disaster such as the Earthquake of Chile occurred during the morning of February 27, 2010. To process this data, we develop a tool on S4 oriented to classify the tweets in order to recognize post-disaster people's needs. The classification process consist on 5 steps. Each operator or PE has to perform one particular task of the 5 step process. The PEs are mapped to their processing units based on their id-keys. This distribution method can easily overload one particular node, generating hot-spots in the system degrading it performance. In order to deal with this issue we exploit replication of the overloaded PEs.

The proposed classification process consist of 5 steps: *Recollecting*, *Scheduling*, *Filtering*, *Relevance* and *Ranking*, which are detailed in the following. A final subsection discusses the replication process.

A. Recollecting

This operator focuses on collecting data from the source. In our study case we retrieve data from the Twitter API. We access the streaming API which deliver access to Twitter's global stream of tweets. We have used a collection tweets from the earthquake of Chile. Using this data we recreate the stream of tweets generated during the 4 days after the disaster (February 27 to March 2) and process it using our tool. A special PE is in charge of this task called *adapter*.

B. Scheduling

The scheduling operator focuses on recognizing each type of tweet which might be: information, water, electricity or food. Once the tweet is recognize as a particular need, the operator redirect the tweet to the next PE in the graph based on it type. Type recognition is a process based on a bag of words generated by developers and validated by undergraduate students. They associate a set of common words that refers to one particular type of need. Despite the process is straightforward, we have to deal with typos and shortening which increases the complexity of the recognition process. In order to deal with this issue, the recognizing process exploits the Hamming distance. Specifically, we have used a threshold of two, for example, the Hamming distance between the word *necesitamos* and the misspelled word *nesecitamos* is two, so we consider that the first word match with the second word.

C. Filtering

The filter operator exploits a Naive Bayesian model to identify if tweets are objective or subjective. This information is important in the following step. To create these models we perform an automatic classification through bags of positive and negative words. The bag of words were manually created by developers and validated by undergraduate students based on the information of other disaster tweets datasets.

We give a higher value to objective tweets which have more information, for example, tweets created by news sources. We consider they are more reliable than the subjectives ones. If the tweet is subjective it is checked whether it is positive or negative in order to benefit the tweets based on the identified characteristics by applying weights constants in the ranking process.

D. Relevance

On post-disaster scenarios it is crucial to identify whether the information is coming from a trustworthy source. In an effort to identify those trustable tweets we create an operator in charge of the tweet's evaluation to give more relevance to the trustworthy ones. Trustworthiness is calculated in two dimensions: author information and tweet information. From the author side, information such as the number of tweets generated, the number of followers/followees and an account verification state are considered to calculate the reputation of the author. From the tweet side, the number of retweets, favorite marks, and the associated timestamp are exploited to calculate its reputation.

In order to calculate the ranking, we performed a normalization process of the obtained values, where the scheduling operator had to perform a barrier tweets. This was computed every certain number of tweets, to get statistics such as the maximum number of followers, the number of favorites, etc. This data is used to normalize each tweet.

These two values are used in conjunction with the ranking result by the next operator called ranking to create a final ranking.

E. Ranking

Ranking operator is responsible of creating the final ranking using the trust values generated on the previous steps. In this operator we can define which component of the ranking is given more relevance. A ranking function based on weight allow us to control the impact on the ranking of one particular component. It is important to mention that a synchronization barrier must be perform at this level in order to calculate the ranking for a given number of tweets.

F. Replication

The proposed classification tool is deployed on the S4 stream processing engine which has a push-based method to distribute events to the processing nodes. In our experience using S4 as stream processing engine, we observe that push-based method creates processing hot-spots, overloading those processing units in charge of the most frequent keys. In order to deal with this problem we study the performance impact of replicating the operators in order to share load among multiple replicas.

Creating multiple replicas require of a special type of operator that we call *scheduler*. The scheduler main function is to distribute tweets towards the replicas in a uniform manner to avoid overload a particular one. In our implementation the scheduler distribute the tweets in a round-robin manner. In Fig. 2 we present the scheduler and a replicated operator. It is important to notice that despite it seems that PE_3 is a new bottleneck or hot-spot, this PE have to deal with pre-processed information coming from the replicated PE (e.g. an integer counter or word-filtered information).

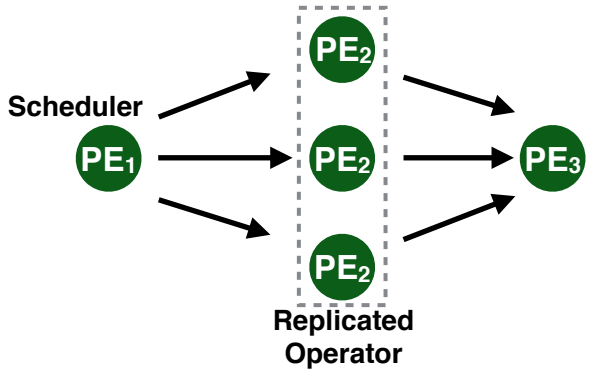

Fig. 2. Replication tasks in the system.

Despite new PE's are added to the processing graph performance is improved. SPE's were designed to give support to real-time processing, thus replicating operators allows us to avoid queues and keep tweets processing flowing at low cost in terms of overhead.

IV. Experiments & Analisys

For our experiments we use our Tweets Classification Tool deployed on S4 to classify people's needs in a post-disaster scenario. The classification process is composed of 5 processing steps previously detailed in Section III. Our dataset consists of 1M spanish tweets generated after the Chilean

earthquake of February 27, 2010. Tweets belong to a window time of 4 days tweets from February 27 to March 2, 2010. The dataset contains more than 4.5M of tweets in english, portuguese and spanish, but only 1M tweets are in the former language. We are interested on spanish tweets to identify Chilean people needs on a post-disaster scenario.

Our classification tool was developed to retrieve data directly from the Twitter API, therefore we modified its implementation in order to support the use of our earthquake dataset. Using this previously collected data, we simulate the stream generated at the disaster period based on the timestamp of the tweets.

Our goal is to analyze the application performance and show that replication can be an easy-way to overcome load balancing issues on SPEs. Load balancing problems among processing nodes can degrade the overall system performance compromising real time processing.

All our tests were executed on a machine Intel(R) Xeon(R) CPU E5-2650 v2 @2.60 GHz with 4 processors, 16 physical cores and 32 GB of memory.

We have evaluated 5 configurations, were configuration 1 (a.k.a Conf.1) corresponds to our baseline case were no replication is performed. In other words, the S4 traditional sequential processing. The other 4 configurations include the operator's replication strategy. The level of replication of each configuration is presented in Table I. Replication level was set based on the complexity of the operators.

TABLE I. PROCESSING GRAPHS CONFIGURATIONS

	PE_1	PE_2	PE_3	PE_4
Conf. 1	1	1	1	1
Conf. 2	2	5	1	1
Conf. 3	5	20	2	1
Conf. 4	10	50	5	1
Conf. 5	25	100	10	1

In all the configurations PE_4 remains without replication because this operator computes the ranking of the tweets. In order to perform a complete ranking of tweets it is necessary to process all data at the same place. We adopt this approach because of its simplicity. However, it is also possible to replicate this operator and perform a local ranking on the replicas in order to parallelize its processing. There are several solutions in literature that solve this problem with results close to a full ranking [18].

It is important to mention that operators replication is a low cost operation. However, there is a maximum number of replicas where we can exploit in an optimum manner of their parallelism. In an scenario with too many replicas we waste resources due to the its usage without improving performance.

In Fig. 3 we evaluate the average time that takes to the classification tool to finish the processing of 250K, 500K, 750K and 1M tweets. This time comprises the instantiation time of all the PEs and the processing time. We measure the average time for the 5 configurations. We notice that Conf.1

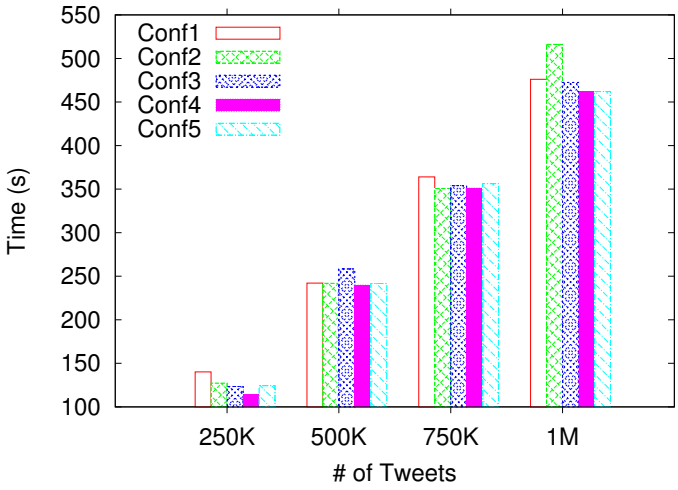

Fig. 3. Average application running time.

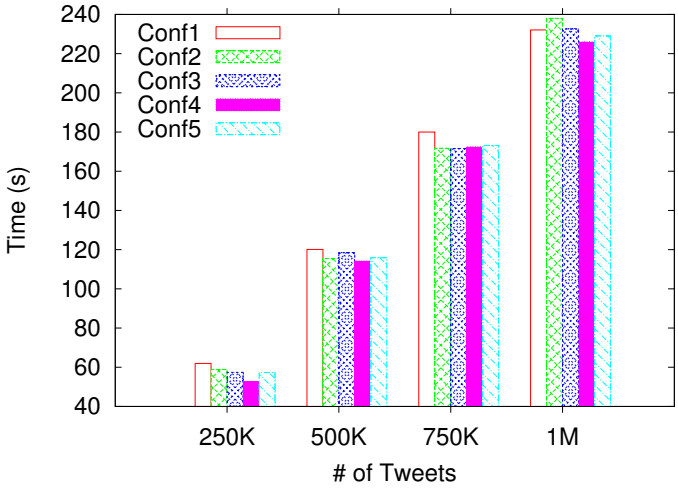

Fig. 4. Average delay time for each configuration.

has the higher average time, in most of the configurations. For the case of 1M tweet we observe that Conf. 2 presents a higher time. This is the result of bigger processing queues. If we observe the other cases, Conf.2 outperforms Conf.1, however with larger datasets queues become longer due to the underestimated level of replication. On the other hand, Conf.4 shows to be the most efficient level of replication for all the datasets size evaluated. Conf.5 presents similar results to Conf.4, however for a small number of tweets this level of replication is overestimated and replication overhead becomes relevant increasing the end time. The good results of Conf. 4 and 5 were expected because PE_2 performs the most complex task and both configurations present the higher replication level for that PE.

We have also evaluated the average time to process the different tweets datasets. This time does not include the instantiation time. In Fig. 4 we present our results. We can observe a similar behavior compared to the previous figure. Conf.0 and Conf.4 present a similar behavior improving processing time up to a 19%. If we compared both figures, we can see that a

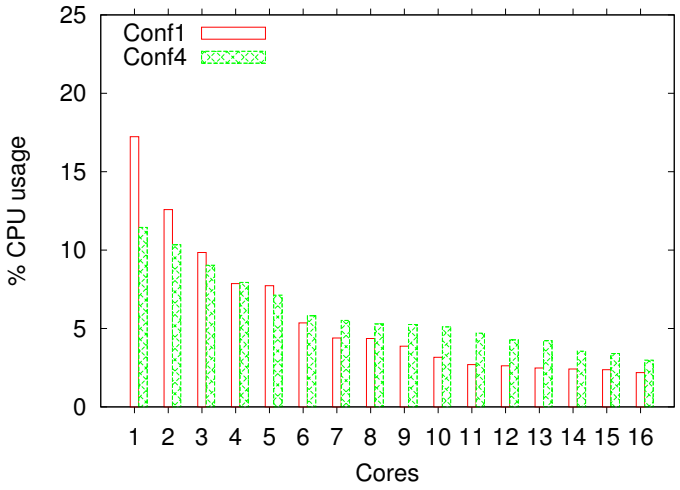

Fig. 5. Average CPU usage.

considerable time is spent on PEs instantiation. This time is has a higher impact for the smallest dataset.

In order to evaluate load balancing we measure the average CPU usage. We evaluate this metric at each physical core. In Fig. 5 we present the CPU usage at each core for Configuration 1 and 4 (Conf.1 and Conf.4 respectively). As can be seen, Conf.4 presents a better distribution of load compared to the Conf.1 which does not uses replication. This result is confirmed by the standard deviation of both configurations. Conf.0 presents a standard deviation of 4.3 among the 16 cores while in Conf.4 is 2.5. We have also evaluate load balancing conditions among cores using the well-known Gini Coefficient (which is being used in other disciplines, such as economics and ecology) [19] [20]. Gini ranges from a minimum value of 0, when all cores have equal load, to a maximum of 1, when every individual, except one has a load of zero. Therefore, as Gini comes closer to 0, load imbalances are reduced, whereas, as Gini comes closer to 1, load imbalances are increased. We calculate Gini coefficient for the results presented in Fig. 5. Our results confirm an important improvement of 20% in load balancing among cores. Configuration 1 presents a Gini coefficient of 0.64 while configuration 4 reaches a Gini of 0.51.

Our results show that the complexity of the tasks performed by the engine operators are critical for the whole system performance. We have demonstrated that by replicating the most complex operators the system can improve up to a 20% the load balancing among the processing units and reduce the total processing time up to 19%.

It is important to mention that SPEs do not provide the tools to define the level of replication of the system or which component must be replicated. In this work we have presented some directions about metrics to define replication level for the processing operators over SPEs.

V. Conclusion

In this work we study the load balancing issues that can arise on stream processing engines due to the distribution of processing elements to nodes. We have evaluated the performance of S4 stream processing engine using a tweet's classification tool deployed on this engine.

In order to deal with overloaded nodes, we exploit the replication of operators in order to distribute the load among several processing units. We state that replication allows to reduce the nodes overloading, and then improve the performance of the engine. However, there is no directions about which operator must be replicated. We show that the complexity of the task performed by the operator and the frequency of the keys are both important factors to evaluate when replication is used.

We evaluate different graph configurations including the replication method and compared with a baseline case of S4. Despite our solution is straightforward, it clearly states that a previous analysis of the processing model to deploy over S4 will enhance the performance of the SPE. Our results show an improvement of more than 19% in the processing time for 1M of tweets. On bigger data streams results can be even more significant. Moreover, we show that load balancing among processing units is improved up to a 20% compared to the case without replication.

The best results we obtained from those configuration that have a higher level of replication on the most complex operators (Conf.4 and 5). This is dependent of the complexity of the operator, which has a higher probability of creating a bottleneck and thus compromise the processing performance. In consequence, not only key-distribution plays an important role on the processing performance but also the complexity level of the operator. Both cases must be analyzed in order to deploy an ad-hoc replication to overcome with the overload issues. However, it is still an open problem how to define the best number of replicas for a given complexity. Our future work is oriented on that direction.

Acknowledgment

The authors would like to thank CITIAPS. This article was partially funded by project PMI USA1024.

References

[1] "Facebook." [Online]. Available: https://www.facebook.com

[2] "Twitter." [Online]. Available: https://twitter.com

[3] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ser. ICDMW '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 170–177. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2010.172

[4] "Storm." [Online]. Available: https://github.com/nathanmarz/storm/wiki

[5] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, "Timestream: reliable stream computation in the cloud," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 1–14.

[6] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez, "Streamcloud: An elastic and scalable data streaming system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2351–2365, 2012.

[7] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch, "Integrating scale out and fault tolerance in stream processing using operator state management," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 725–736.

[8] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: fault-tolerant streaming computation at scale," in *SOSP*, M. Kaminsky and M. Dahlin, Eds. ACM, 2013, pp. 423–438.

[9] "Samza." [Online]. Available: http://samza.incubator.apache.org/

[10] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: Fault-tolerant stream processing at internet scale," in *Very Large Data Bases*, 2013, pp. 734–746.

[11] "Amazon kinesis." [Online]. Available: http://aws.amazon.com/kinesis/

[12] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI*. USENIX Association, 2004, pp. 137–150.

[13] X. J. Zhang, S. Parekh, B. Gedik, H. Andrade, and K.-L. Wu, "Workload characterization for operator-based distributed stream processing applications," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '10. New York, NY, USA: ACM, 2010, pp. 235–247. [Online]. Available: http://doi.acm.org/10.1145/1827418.1827466

[14] B. Gedik, S. Schneider, M. Hirzel, and K.-L. Wu, "Elastic scaling for data stream processing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 6, pp. 1447–1463, June 2014.

[15] K. G. S. Madsen, P. Thyssen, and Y. Zhou, "Integrating fault-tolerance and elasticity in a distributed data stream processing system," in *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '14. New York, NY, USA: ACM, 2014, pp. 48:1–48:4. [Online]. Available: http://doi.acm.org/10.1145/2618243.2618288

[16] T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer, "Auto-scaling techniques for elastic data stream processing," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '14. New York, NY, USA: ACM, 2014, pp. 318–321. [Online]. Available: http://doi.acm.org/10.1145/2611286.2611314

[17] M. Smit, B. Simmons, and M. Litoiu, "Distributed, application-level monitoring for heterogeneous clouds using stream processing," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2103 – 2114, 2013.

[18] N. Ailon, "Aggregation of partial rankings, p-ratings and top-m lists," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 415–424. [Online]. Available: http://dl.acm.org/citation.cfm?id=1283383.1283427

[19] L. Ceriani and P. Verme, "The origins of the gini index: extracts from variabilit e mutabilit (1912) by corrado gini," *The Journal of Economic Inequality*, vol. 10, no. 3, pp. 421–443, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10888-011-9188-x

[20] T. Pitoura, N. Ntarmos, and P. Triantafillou, "Replication, load balancing and efficient range query processing in dhts," in *Advances in Database Technology - EDBT 2006*, ser. Lecture Notes in Computer Science, Y. Ioannidis, M. Scholl, J. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, Eds. Springer Berlin Heidelberg, 2006, vol. 3896, pp. 131–148. [Online]. Available: http://dx.doi.org/10.1007/11687238_11