Data Streaming and its Application to Stream Processing: Tutorial

Conference Paper · June 2017

DOI: 10.1145/3093742.3095108

2 authors:

Leonardo Querzoni
Sapienza University of Rome
118 PUBLICATIONS **1,345** CITATIONS
SEE PROFILE

Nicolo Rivetti
Technion - Israel Institute of Technology
18 PUBLICATIONS **128** CITATIONS
SEE PROFILE

Tutorial: Data Streaming and its Application to Stream Processing

Leornado Querzoni
Sapienza University of Rome
querzoni@dis.uniroma1.it

Nicolo Rivetti
Technion - Israel Institute of Technology
nrivetti@technion.ac.il

ABSTRACT

In this tutorial paper we present the results of recent research findings in the area of data streaming applied to stream processing systems. In particular, we introduce the data streaming model, detailing the main algorithmic results in this research field. We then move to detail how such algorithms can be applied to modern distributed stream processing systems to improve their efficiency. Finally we outline several open research directions in this field.

CCS CONCEPTS

•Software and its engineering →Distributed systems organizing principles; •Theory of computation →*Online learning algorithms; Sketching and sampling;*

KEYWORDS

Stream Processing, Data Streaming, Load Shedding, Load Balancing

1 INTRODUCTION

Nowadays stream analysis is used in many context where the amount of data and/or the rate at which it is generated rules out other approaches (*e.g.,* batch processing). Industrial control systems, sensor networks, business intelligence are all examples of industrial sectors that today strongly rely on stream analysis to extract actionable information from intense data streams in quasi-real time.

Application scenarios in these contexts required in the last few years the development of new techniques and frameworks to support the desired requirements. The data streaming model provides randomized and/or approximated solutions to compute specific functions over (distributed) stream(s) of data-items in worst case scenarios, while striving for small resources usage. Stream processing, which is somehow complementary and more practical, provides efficient and highly scalable frameworks to perform soft real-time generic computation on streams, relying on cloud computing platforms. While data streaming algorithms and stream processing systems were born separately and have so far seen parallel evolutions, their duality opens the opportunity for making stream processing systems more efficient and performant through data streaming solutions. This tutorial paper provides a brief overview of the data streaming model, and describes how current limitations of stream processing systems may be surpassed or at least mitigated through data streaming solutions. This new path in the evolution of stream processing systems offers promising results that have immediate applicability in real-world problems; however, it still presents complex open problems that may possibly appeal the research community.

The rest of this paper is structured as follows. Section 2 introduces the common system model considered for data streaming algorithms tailored on the stream processing model. Section 3 is dedicated to an introduction to data streaming [23]. In particular it provides an overview of the main algorithmic results from this research field, most of which have shown their applicability to stream processing. Then Section 4 shows how the previous algorithms can be used to build more efficient stream processing systems (*i.e.,* Apache Storm [34], Apache Spark [33], Flink [32], *etc.*) and applications. Finally, section 5 explores Future Research Directions in the area of data streaming applied to stream processing systems.

2 SYSTEM MODEL

Data streaming focuses on estimating metrics over streams, which is an important task in data intensive applications where it is necessary to quickly and precisely process a huge amount of data. This can be applied in particular to stream processing, both at system and at application level. Here we formalize a system model bridging between the classical data streaming distributed model (*i.e.,* Distributed Functional Monitoring [12]) and the Stream Processing model.

We consider a distributed stream processing system (SPS) deployed on a cluster where several computing nodes exchange data through messages sent over a network. The SPS executes a stream processing application represented by a *topology*: a directed acyclic graph interconnecting operators, represented by nodes, with data-streams, represented by edges. Each topology contains at least a *source*, *i.e.,* an operator connected only through outbound data-streams, and a *sink*, *i.e.,* an operator connected only through inbound data-streams.

Data injected by the source is encapsulated in units called tuples and each stream is a sequence of tuples. Without loss of generality, here we assume that each tuple is a finite set of key/value pairs that can be customized to represent complex data structures. To simplify the discussion, in the rest of this work we deal with streams of unary tuples with a single non negative integer value. Thus tuples in a data-stream can be represented as items in a stream $\sigma = \langle t_1, \ldots, t_j, \ldots t_m \rangle$, where each item t is drawn from a universe of size n. More in details, t denotes the value of the item in the set $[n] = \{1, \ldots, n\}$, while the subscript j denotes the position in the

DEBS'17, Barcelona, Spain
© 2017 ACM. 978-1-4503-5065-5…$15.00
DOI: http://

stream in the index sequence $[m] = \{1, \ldots, m\}$. Then m is the size (or number of items) of the stream σ.

Let consider two operators, S and O, connected through a directed edge from S to O, *i.e.,* a stream $\sigma_{S \to O}$. Both operators can be parallelized, creating s independent instances $S_1, \ldots, S_s \in \mathcal{I}$ of S and k independent instances $O_1, \ldots, O_k \in \mathcal{K}$ of O, as well as by partitioning the stream $\sigma_{S \to O}$ into $s \times k$ sub-streams $\sigma_{S_1 \to O_1}, \ldots, \sigma_{S_s \to O_k}$. In other words, each operator S_i ($i \in [s]$) has k outbound sub-streams $\sigma_{S_i \to O_1}, \ldots, \sigma_{S_i \to O_k}$ and each operator O_p ($p \in [k]$) has s inbound sub-streams $\sigma_{S_1 \to O_p}, \ldots, \sigma_{S_s \to O_p}$. We denote the union of the outbound sub-stream of operator instance S_i with $\sigma_{S_i} = \cup_{p=1}^{k} \sigma_{S_i \to O_p}$. Similarly, we denote the union of the inbound sub-streams of operator instance O_p with $\sigma_{O_p} = \cup_{i=1}^{s} \sigma_{S_i \to O_p}$.

Each operator instance has a FIFO input queue where tuples are buffered while the instance is busy processing previous tuples.

Most works in the data streaming field do not take into account that the characteristics of the stream may vary over time or assume that the whole semi-infinite stream is of interest. However, this is seldom the case in a practical setting such as stream processing: in many applications, there is a greater interest in considering only the "recently observed" items and the characteristics of the stream may change, alternating between steady and transient phases. This gives rise to the *sliding window* model formalised by Datar, Gionis, Indyk and Motwani [13]: items arrive continuously and expire after exactly M steps. The relevant data is then the set of items that arrived in the last M steps, *i.e.,* the *active* window. A step can either be defined as a sample arrival, then the active window would contain exactly M items and the window is said to be count-based. Otherwise, the step is defined as a time tick, either logical or physical, and the window is said to be time-based. More formally, let consider the count-based sliding window and recall that by definition t_j is the item arriving at step j. In other words, the stream σ at step m is the sequence $\langle t_1, \ldots, t_{m-M}, t_{m-M+1}, \ldots, t_m \rangle$. Then, the desired function ϕ has to be evaluated over the stream $\sigma(M) = \langle t_{m-M}, t_{m-M+1}, \ldots, t_m \rangle$.

3 DATA STREAMING PROBLEMS AND SOLUTIONS

In a streaming setting, performing real time analysis of large streams with small capacities in terms of storage and processing, analyzing input data relying on full space algorithms may not be feasible. Two main approaches exist and are used in data streaming to monitor massive data streams in real time: sampling and summaries. The first one consists in regularly sampling the input streams so that only a limited amount of data items is locally kept. This allows to exactly compute functions on these samples, which are expected to be representative. However, the accuracy of this computation, with respect to the whole stream, fully depends on the volume of data that has been sampled and the location of the sampled data. The summary approach consists in scanning on the fly each piece of data of the input stream, and keep locally only compact synopses or sketches containing the most important information about data items. This approach enables to derive some data streams statistics with guaranteed error bounds. In general, the research done so far has focused on computing functions or statistical measures with ε or (ε, δ)-approximations in poly-logarithmic space over the size and/or the domain size of the stream. More formally, an algorithm A is said to be an (ε, δ)-approximation of the function ϕ if the output $\widehat{\phi}$ of A is such that $\mathbb{P}[\mid \widehat{\phi} - \phi \mid > \varepsilon\phi] \leq \delta$, where $\varepsilon, \delta > 0$ are given as precision and probability of failure parameters.

Estimating the frequency moments is one of the first problems that was solved through data streaming methods, namely in the seminal paper from Alon, Matias and Szegedy [1]. A stream σ implicitly defines a frequency vector $\overrightarrow{f} = \langle f_1, \ldots, f_t, \ldots, f_n \rangle$ where f_t is the frequency of item t, *i.e.,* the number of occurrences of t in σ. The stream frequency moments $F_\imath = \sum_{J=1}^{n} f_J^\imath = \| f \|_\imath$ are a useful measure to understand statistical properties of the data. In particular $F_\imath$ where $\imath \geq 2$ indicate the skew of the data, which is a relevant information in many distributed applications. The authors of [1] present the AMS algorithm estimating the frequency moments and show how it can be applied to estimate the join size using $O(1/\varepsilon^2 \log 1/\delta(\log n + \log m))$. More recently, Vengerov *et al.* [35] proposed the last of a long list of improvements of [1] to estimate the join size with filter conditions, always relying on the AMS basic estimator.

Flajolet and Martin [15] authored another seminal paper in the data streaming literature, where they present an elegant solution to the estimation of the number of distinct items in a stream, *i.e.,* F_0. More recently, Baryossef *et al.* presented an (ε, δ)-approximation, known as the BJKST algorithm, which is the current state of the art providing an estimation $\widehat{F_0}$ such that $\mathbb{P}[|\widehat{F_0} - F_0| > \varepsilon] < \delta$ in $O(1/\varepsilon^2 \log(1/\delta) \log n)$ bits.

The estimation of the stream frequency distribution, *i.e.,* return an estimation of the frequency of occurrence for each distinct item of the stream, provides a more detailed view with respect to a single aggregate value for the whole stream. Charikar, Chen and Farach-Colon [9] proposed the Count-Sketch algorithm in 2002. In 2005, Cormode and Muthukrishnam [11] proposed the Count-Min Sketch, which is quite similar to the Count-Sketch since both rely on 2-*universal hash-functions*[1] and a matrix of counters which size depends on the required accuracy. The space complexity of this algorithm is $O(\frac{1}{\varepsilon} \log \frac{1}{\delta}(\log m + \log n))$ bits. Therefore, the following quality bounds hold: $\mathbb{P}[\mid \widehat{f_t} - f_t \mid \geq \varepsilon \parallel \overrightarrow{f}_{-t} \parallel_1] \leq \delta$, while $f_t \leq \widehat{f_t}$ is always true (where $\overrightarrow{f}_{-t}$ represents the frequency vector $\overrightarrow{f}$ without the scalar f_t). The bound on the estimation accuracy of the Count-Min Sketch is is weaker than the one provided by the Count-Sketch for skewed distribution. On the other hand, the Count-Min Sketch has a smaller memory footprint by shaving a $1/\varepsilon$ factor.

Papapetrou *et al.* [26] present the ECM-Sketch, extending the Count-Min Sketch [11] to the sliding window model. The wave-based version of ECM-Sketch replaces each counter of the Count-Min Sketc matrix with a *wave* [17] data structure. Each wave is a set of lists, the number and the size of such lists is set by the parameter ε. Then the wave-based ECM-Sketch space complexity is $O(\frac{1}{\varepsilon} \log \frac{1}{\delta} \left(\frac{1}{\varepsilon} \log^2 \varepsilon N + \log n \right))$ bits. Rivetti *et al.* [30] replace the entries of the matrix with a data structure that uses additional

[1]A hash function is said to be 2-universal if it assigns truly random values in the co-domain to any value of its domain.

memory only when required. The author show a 4 fold memory improvement while being more accurate than [26].

Identifying (and estimated the frequencies) of heavy hitter boils down to find the items in the stream whose frequency f_t is above a given (user defined) threshold $\Theta \geq 0$. There is a tight relation between this and the frequency estimation problems since one can be reduced to the other, but with some limitations. Moreover, the heavy hitters problem can be reduced to the top-k (*i.e.,* finding the k most frequent items) problem, however the converse does not hold. Misra and Gries [22] provide an elegant deterministic sampling algorithm for this problem. Sticky Sampling and Lossy Counting are other two well known algorithms proposed by Manku and Motwani [20], however their space complexities have been superseded by another counter-based solution: the Space-Saving algorithm presented by Metwally *et al.* [21]. Similarly to the Misra-Greis [22] algorithm, it maintains an associative array of $\langle item, counter, error \rangle$ triples which maximum size is set to $\lceil 1/\varepsilon \rceil$. The heavy hitters are all items in the array where $counter - error \geq \Theta m$, and their frequency estimation is the value of $counter$. The over-estimation made by the algorithm on the estimated frequency $\widehat{f_t}$ of heavy hitter t verifies $f_t + \varepsilon m \geq \widehat{f_t} \geq f_t$ for any $0 < \varepsilon < \Theta \leq 1$.

Ben-Basat *et al.* [3] present the Compact Space-Saving algorithm, a complete overhaul of the Space Saving, and an extension in the sliding window model, the Windowed Compact Space-Saving algorithm. The authors show both asymptotic and practical (65% to 85%) improvement over the vanilla Space-Saving algorithm.

4 IMPROVING STREAM PROCESSING SYSTEMS

Stream processing systems [14] (SPS) are today gaining momentum as a tool to perform analytics on continuous data streams. These applications have become ubiquitous due to increased automation in telecommunications, health-care, transportation, retail, science, security, emergency response, and finance. As a result, various research communities have independently developed programming models for streaming. The communities that have focused the most on streaming optimizations are digital signal processing, operating systems and networks, complex event processing and databases. In their survey, Hirzel *et al.* [18] propose a catalogue encompassing 11 stream processing optimizations developed by these communities. In this section we will review some of the most recent research findings applying data streaming to stream processing.

Load balancing in distributed computing is a well known problem that has been extensively studied since the 80s [5, 36]. We can identify two ways to perform load balancing in stream processing systems [18]: either when placing the operators on the available machines [6, 7] or when assigning load to the parallelized instances $O_\jmath \in \mathcal{K}$ of operator O. If O is stateless, then tuples are randomly assigned to instances (*random* or *shuffle grouping*). Rivetti *et al.* [29] present an algorithm aimed at reducing the overall tuple completion time in shuffle-grouped stages. In particular they extend the Count-Min Sketch [11] to estimate at runtime the execution time of the tuples ingested by the parallel instances $O_\jmath \in \mathcal{K}$. This information is then back propagated to the previous operator S which is then able to schedule (online) the tuples on its outbound sub-streams σ_{S_i} improving the balance among the instances of O.

Key grouping is another grouping technique used by stream processing frameworks to simplify the development of parallel stateful operators. Through key grouping a stream of tuples is partitioned in several disjoint sub-streams depending on the values contained in the tuples themselves. Each operator instance target of one sub-stream is guaranteed to receive all the tuples containing a specific key value. A common solution to implement key grouping is through hash functions that, however, are known to cause load imbalances on the target operator instances when the input data stream is characterized by a skewed value distribution. Gedik [16] proposes a solution to this problem leveraging consistent hashing and the Lossy Counting [20] algorithm. Rivetti *et al.* [31], using the Space-Saving [21] algorithm, provide near-optimal load distribution by refining the work of Gedik [16]. The basic intuition is that heavy hitters induce most of the unbalance in the load distribution. Then, in both work the author leverage solutions to the heavy hitter problem to isolate/carefully handle the keys that have a frequency larger than/close to $1/k$. Rivetti *et al.* [31] outperform [16] through a better management of non heavy hitter keys, however do not handle changes in the distribution over time. On the other hand, Nasir *et al.* [25] applied the power of two choices approach to this problem, *i.e.,* relax the key-grouping constraints by spreading each key on 2 instances, thus restricting the applicability of their solution but also achieving better balancing. The authors proposed a further improvement [24] by spreading the keys on more than 2 instances.

Caneill *et al.* [4] present another work focusing on load balancing through a different angle. In topologies with sequences of stateful operators, and thus key grouped stages, the likelihood that a tuple is sent to different operator instances located on different machines is high. Since network is a bottleneck for many stream applications, this behavior significantly degrades their performance. Caneill *et al.* [4] use the Space-Saving [21] algorithm to identify the heavy hitters and analyze the correlation between keys of subsequent key-grouped stages. This information is than leveraged to build a key-to-instance mapping that improves stream locality for stateful stream processing applications .

As load balancing strives to fully use the available resources, an orthogonal problem is to provide the system with enough resources to run with good performances. Load shedding is a technique employed by stream processing systems to handle unpredictable spikes in the input load whenever available computing resources are not adequately provisioned. A load shedder drops tuples to keep the input load below a critical threshold and thus avoid unbounded queuing and system trashing. Similarly to [29], Rivetti *et al.* [29] estimate at runtime the execution time of the tuples ingested by the parallel instances $O_\jmath \in \mathcal{K}$ through an extension of the Count-Min Sketch [11]. The preceding operator S leverages this information to shed the load based on the (estimated) execution time instead of on a pre-defined or over-simplified cost model as most previous work do.

5 FUTURE RESEARCH DIRECTIONS

The works discussed in the previous section mainly focus on balancing the load assigned to the parallelized instances $O_\jmath \in \mathcal{K}$ of operator O. Clearly, there are several other ways in which stream processing can benefit by leveraging data streaming, opening several promising research directions. This includes studying the applicability of other data streaming algorithms, the application to other system level problems as well as application level issues.

Application level

Approximated query [19] are one of the means devised in the database community to provide fast (while not exact) answers to queries, which resembles the lambda architecture studied in the stream processing area. Previously we have introduced the notion of load-shedding. An interesting contribution from Reiss *et al.* [28] is to switch from an accurate to an approximate computation when the load becomes to intense. In both cases, data streaming may provide the means to introduce those concepts into stream processing in a rigorous way.

System level

Data streaming has shown its strength in the database community, for instance providing efficient estimation of the join size [1, 35]. There are many query plan optimizations performed in database systems based on top of such estimations. The stream processing community may want to port those into stream processing systems, leveraging data streaming to collect the relevant information to dynamically optimize the deployment phase / query plan. Similarly, data streaming may be applied to adaptive online scheduling and auto-parallelization, two topics with a lot of ongoing research which mostly relies on machine learning techniques. Focusing on specialized stream processing systems, we can see even further possible applications, *e.g.,* improve the partitioning in graph based computation [10]. In this field, recent contributions [27] have shown how partitioning quality may depend on topological characteristics of the input graph (e.g. vertex degree distribution). Such characteristics may be tracked using data streaming solutions.

Other metrics

There are many other, and more complex, metrics that have been studied in the data streaming community, such as computing the entropy of a stream [8], that may find innovative application in stream processing. For instance, one could use a estimation of a similarity distance between two streams [2] to detect redundancy in the stream processing application.

REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, STOC, 1996.

[2] E. Anceaume and Y. Busnel. A Distributed Information Divergence Estimation over Data Streams. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):478–487, 2014.

[3] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner. Heavy hitters in streams and sliding windows. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[4] M. Caneill, A. El Rheddane, V. Leroy, and N. De Palma. Locality-aware routing in stateful streaming applications. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, 2016.

[5] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The State of the Art in Locally Distributed Web-server Systems. *ACM Computing Surveys*, 34(2):263–311, 2002.

[6] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli. Optimal Operator Placement for Distributed Stream Processing Applications. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, DEBS, 2016.

[7] D. Carney, U. Çetintemel, A. Rasin, S. Zdonik, M. Cherniack, and M. Stonebraker. Operator Scheduling in a Data Stream Manager. In *Proceedings of the 29th International Conference on Very Large Data Bases*, VLDB, 2003.

[8] A. Chakrabarti, G. Cormode, and A. McGregor. A Near-optimal Algorithm for Computing the Entropy of a Stream. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, SODA, 2007.

[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding Frequent Items in Data Streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP, 2002.

[10] R. Chen, J. Shi, Y. Chen, and H. Chen. PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. In *Proceedings of the 10th European Conference on Computer Systems*, EuroSys, 2015.

[11] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[12] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for Distributed Functional Monitoring. *ACM Transactions on Algorithms*, 7(2):21:1–21:20, 2011.

[13] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[14] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Co., 2010.

[15] P. Flajolet and G. N. Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[16] B. Gedik. Partitioning Functions for Stateful Data Parallelism in Stream Processing. *The VLDB Journal*, 23(4):517–539, 2014.

[17] P. B. Gibbons and S. Tirthapura. Distributed Streams Algorithms for Sliding Windows. *Theory of Computing Systems*, 37(3):457–478, 2004.

[18] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm. A Catalog of Stream Processing Optimizations. *ACM Computing Surveys*, 46(4):41–34, 2014.

[19] Q. Liu. *Approximate Query Processing*, pages 113–119. Springer US, Boston, MA, 2009.

[20] G. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB, 2002.

[21] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Proceedings of the 10th International Conference on Database Theory*, ICDT, 2005.

[22] J. Misra and D. Gries. Finding Repeated Elements. *Science of Computer Programming*, 2:143–152, 1982.

[23] S. Muthukrishnan. *Data streams: algorithms and applications*. Now Publishers Inc, 2005.

[24] M. A. U. Nasir, G. D. F. Morales, N. Kourtellis, and M. Serafini. When two choices are not enough: Balancing at scale in distributed stream processing. In *Proceedings of the IEEE 32nd International Conference on Data Engineering*, ICDE, 2016.

[25] M. A. U. Nasir, G. D. F. Morales, D. G. Soriano, N. Kourtellis, and M. Serafini. The Power of Both Choices: Practical Load Balancing for Distributed Stream Processing Engines. In *Proceedings of the 31st IEEE International Conference on Data Engineering*, ICDE, 2015.

[26] O. Papapetrou, M. N. Garofalakis, and A. Deligiannakis. Sketch-based Querying of Distributed Sliding-Window Data Streams. *Proceedings of the VLDB Endowment*, 5(10):992–1003, 2012.

[27] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, and G. Iacoboni. Hdrf: Stream-based partitioning for power-law graphs. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 243–252. ACM, 2015.

[28] F. Reiss and J. M. Hellerstein. Data Triage: An Adaptive Architecture for Load Dhedding in TelegraphCQ. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE, 2005.

[29] N. Rivetti, E. Anceaume, Y. Busnel, L. Querzoni, and B. Sericola. Proactive Online Scheduling for Shuffle Grouping in Distributed Stream Processing Systems. In *Proceedings of the 17th ACM/IFIP/USENIX International Middleware Conference*, Middleware, 2016.

[30] N. Rivetti, Y. Busnel, and A. Mostefaoui. Efficiently Summarizing Distributed Data Streams over Sliding Windows. In *Proceedings of the 14th IEEE International Symposium on Network Computing and Applications*, NCA, 2015.

[31] N. Rivetti, L. Querzoni, E. Anceaume, Y. Busnel, and B. Sericola. Efficient Key Grouping for Near-Optimal Load Balancing in Stream Processing Systems. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, DEBS, 2015.

[32] The Apache Software Foundation. Apache Flink. https://flink.apache.org/.

[33] The Apache Software Foundation. Apache Spark. http://spark.apache.org/.

[34] The Apache Software Foundation. Apache Storm. http://storm.apache.org.

[35] D. Vengerov, A. C. Menck, M. Zait, and S. P. Chakkappen. Join size estimation subject to filter conditions. *Proceedings VLDB Endowment*, 8(12):1530–1541, Aug. 2015.

[36] S. Zhou. *Performance Studies of Dynamic Load Balancing in Distributed Systems*. PhD thesis, UC Berkeley, 1987.