Incremental Approximate Computing

Dhanya R Krishnan and Do Le Quoc and Pramod Bhatotia and Christof Fetzer and Rodrigo Rodrigues

Abstract Approximate computing is increasingly used for speeding up computations and efficiently utilizing the computing resources. The idea behind approximate computing is to return an approximate answer instead of the exact answer for user queries. The trick is to choose a representative sample of the data for computing instead of using the entire data. As a result, it allows users to trade-off query accuracy for response time, enabling interactive queries over massive data by running queries on data samples and presenting results annotated with meaningful error bars. At the same time, another technique called incremental computing tries to achieve the same goals as approximate computing, i.e., speeding up job execution and utilizing resource efficiently. Incremental computing relies on the *memoization* of intermediate results of sub-computations, and reusing these memoized results across jobs. This work makes the observation that these two computing paradigms are complementary, and can be married together! The idea is quite simple: *design a sampling algorithm that biases the sample selection to the memoized data items from previous runs*. To realize this idea, an online stratified sampling algorithm is designed. The algorithm uses self-adjusting computation to produce an incrementally updated approximate output with bounded error. The algorithm is implemented in a data analytics system called INCAPPROX.

Dhanya R Krishnan
TU Dresden, e-mail: dhanya.krishnan@icloud.com

Do Le Quoc
TU Dresden, e-mail: do.le_quoc@tu-dresden.de

Pramod Bhatotia
University of Edinburgh and Alan Turing Institute, e-mail: pramod.bhatotia@ed.ac.uk

Christof Fetzer
TU Dresden, e-mail: christof.fetzer@tu-dresden.de

Rodrigo Rodrigues
IST (Univ. Lisbon) & INESC-ID, e-mail: rodrigo.rodrigues@inesc-id.pt

1 Introduction

Stream analytics systems are an integral part of modern online services. These systems are extensively used for transforming raw data into useful information. Much of this raw data arrives as a continuous data stream and in huge volumes, requiring real-time stream processing based on parallel and distributed computing frameworks [1, 2, 13, 48, 61].

Near real-time processing of data streams has two desirable, but contradictory design requirements [48, 61]: *(i)* achieving low latency; and *(ii)* efficient resource utilization. For instance, the low-latency requirement can be met by employing more computing resources and parallelizing the application logic over the distributed infrastructure. Since most data analytics frameworks are based on the data-parallel programming model [24], almost linear scalability can be achieved with increased computing resources. However, low-latency comes at the cost of lower throughput and ineffective utilization of the computing resources. Moreover, in some cases, processing all data items of the input stream would require more than the available computing resources to meet the desired SLAs or the latency guarantees.

To strike a balance between these two contradictory goals, there is a surge of new computing paradigms that prefer to compute over a subset of data items instead of the entire data stream. Since computing over a subset of the input requires less time and resources, these computing paradigms can achieve bounded latency and efficient resource utilization. In particular, two such paradigms are incremental and approximate computing.

Incremental computing. Incremental computation is based on the observation that many data analytics jobs operate incrementally by repeatedly invoking the same application logic or algorithm over an input data that differs slightly from that of the previous invocation [15, 35, 36]. In such a workflow, small, localized changes to the input often require only small updates to the output, creating an opportunity to update the output incrementally instead of recomputing everything from scratch [4, 12]. Since the work done is often proportional to the change size rather than the total input size, incremental computation can achieve significant performance gains (low latency) and efficient utilization of computing resources [16, 18, 51].

The most common way for incremental computation is to rely on programmers to design and implement an application-specific incremental update mechanism (or a *dynamic algorithm*) for updating the output as the input changes [19, 21, 25, 26, 30, 34]. While dynamic algorithms can be asymptotically more efficient than recomputing everything from scratch, research in the algorithms community shows that these algorithms can be difficult to design, implement and maintain even for simple problems. Furthermore, these algorithms are studied mostly in the context of the uniprocessor computing model, making them ill-suited for parallel and distributed settings which is commonly used for large-scale data analytics.

Recent advancements in self-adjusting computation [4, 5, 6, 40] overcome the limitations of dynamic algorithms. Self-adjusting computation transparently benefits existing applications, without requiring the design and implementation of dynamic algorithms. At a high level, self-adjusting computation enables incremental updates by creating a dynamic dependence graph of the underlying computation, which records control and data dependencies between the sub-computations. Given a set of input changes, self-adjusting computation performs change propagation, where it reuses the *memoized* intermediate results for all sub-computations that are unaffected by the input changes, and re-computes only those parts of the computation that are transitively affected by the input change. As a result, self-adjusting computation computes only on a subset (*"delta"*) of the computation instead of re-computing everything from scratch.

Approximate computing. Approximate computation is based on the observation that many data analytics jobs are amenable to an approximate, rather than the exact output [27, 47, 49, 56]. For such an approximate workflow, it is possible to trade accuracy by computing over a partial subset instead of the entire input data to achieve low latency and efficient utilization of resources.

Over the last two decades, researchers in the database community proposed many techniques for approximate computing including sampling [9, 32], sketches [23], and online aggregation [37]. These techniques make different trade-offs with respect to the output quality, supported query interface, and workload. However, the early work in approximate computing was mainly targeted towards the centralized database architecture, and it was unclear whether these techniques could be extended in the context of big data analytics.

Recently, sampling based approaches have been successfully adopted for distributed data analytics [8, 33]. These systems show that it is possible to have a trade-off between the output accuracy and performance gains (also efficient resource utilization) by employing sampling-based approaches for computing over a *subset* of data items. However, these "big data" systems target batch processing workflow and cannot provide required low-latency guarantees for stream analytics.

Combining incremental and approximate computing. This paper makes the observation that the two computing paradigms, incremental and approximate computing, are complementary. Both computing paradigms rely on computing over a subset of data items instead of the entire dataset to achieve low latency and efficient cluster utilization. Therefore, this paper proposes to combine these paradigms together in order to leverage the benefits of both. Furthermore, incremental updates can be achieved without requiring the design and implementation of application-specific dynamic algorithms, and support approximate computing for stream analytics.

The high-level idea is to design a sampling algorithm that biases the sample selection to the memoized data items from previous runs. This idea is realized by designing an online sampling algorithm that selects a representative subset of data items from the input data stream. Thereafter, the sample is biased to include data items for which already there are memoized results from previous runs, while preserving the proportional allocation of data items of different (strata) distributions. Next, the user-specified streaming query is performed on this biased sample by making use of self-adjusting computation and the user is provided with an incrementally updated approximate output with error bounds.

2 IncApprox

2.1 System Overview

IncApprox is designed for real-time data analytics on online data streams. At the high level, the online data stream consists of data items from diverse sources of events or sub-streams. IncApprox uses a stream aggregator (e.g., Apache Kafka [3]) that integrates data from these sub-streams, and thereafter, the system reads this integrated data stream as the input.

IncApprox facilitates user querying on this data stream and supports a user interface that consists of a streaming query and a query budget. The user submits the streaming query to the system as well as specifies a query budget. The query budget can either be in the form of latency guarantees/SLAs for data processing, desired result accuracy, or computing resources available for query processing. IncApprox makes sure that the computation done over the data remains within the specified budget. To achieve this, the system makes use of a mixture of incremental and approximating computing for real-time processing over the input data stream, and emits the query result along with the confidence interval or error bounds.

2.2 System Model

In this section, the system model assumed in this work is presented.

Programming model. IncApprox supports a *batched stream processing* programming model. In batched stream processing, the online data stream is divided into small batches or sets of records; and for each batch a distributed data-parallel job is launched to produce the output.

Computation model. The computation model of IncApprox for stream processing is *sliding window computations*. In this model the computation window slides over the input data stream, where new arriving input data items are added to the window and the old data items are dropped from the window as they become less relevant to the analysis.

In sliding window computations, there is a substantial overlap of data items between the two successive computation windows, especially, when the size of the window is large relative to the slide interval. This overlap of unchanged data-items provides an opportunity to update the output incrementally.

Assumptions. IncApprox makes the following assumptions. Possible methods to enforce them are discussed in §3 and in the paper [39].

1. This work assumes that the input stream is stratified based on the source of event, i.e., the data items within each stratum follow the same distribution, and are mutually independent. Here a *stratum* refers to one sub-stream. If multiple sub-streams have the same distribution, they are combined to form a stratum.

2. This work assumes the existence of a virtual function that takes the user specified budget as the input and outputs the sample size for each window based on the budget.

2.3 Building Blocks

Techniques and the motivation behind INCAPPROX's design choices are briefly described.

Stratified sampling. In a streaming environment, since the window size might be very large, for a realistic rate of execution, INCAPPROX performs approximation using samples taken within the window. But the data stream might consist of data from disparate events. As such, INCAPPROX needs to make sure that every sub-stream is considered fairly to have a representative sample from each sub-stream. For this the system uses stratified sampling [9]. Stratified sampling ensures that data from every stratum is selected and none of the minorities are excluded. For statistical precision, INCAPPROX uses proportional allocation of each sub-stream to the sample [10]. It ensures that the sample size of each sub-stream is in proportion to the size of sub-stream in the whole window.

Self-adjusting computation. For incremental sliding window computations, INCAPPROX uses self-adjusting computation [4, 5, 6] to re-use the intermediate results of sub-computations across successive runs of jobs. In this technique the system maintains a dependence graph between sub-computations of a job, and reuse memoized results for sub-computations that are unaffected by the changed input in the computation window.

2.4 Algorithm

This section presents an overview algorithm of INCAPPROX. The algorithm computes a user-specified streaming *query* as a sliding window computation over the input data stream. The user also specifies a query *budget* for executing the query, which is used to derive the sample size (*sampleSize*) for the window using a cost function (see §2.2 and §3). The cost function ensures that processing remains within the query budget.

For each window, INCAPPROX first adjusts the computation window to the current start time t by removing all old data items from the *window* ($timestamp < t$). Similarly, the system also drops all old data items from the list of memoized items (*memo*), and the respective memoized results of all sub-computations that are dependent on those old data items.

Next, the system reads the new incoming data items in the *window*. Thereafter, it performs proportional stratified sampling (detailed in [39]) on the *window* to select a sample of size provided by the cost function. The stratified sampling algorithm ensures that samples from all strata are proportional, and no stratum is neglected.

Next, INCAPPROX biases the stratified sample to include items from the memoized sample, in order to enable the reuse of memoized results from previous sub-computations. The biased sampling algorithm biases samples *specific to each stratum*, to ensure reuse, and at the same time, retain proportional allocation [39].

Thereafter, on this biased sample, the system runs the user specified *query* as a data-parallel job *incrementally*, i.e., it reuses the memoized results for all data items that are unchanged in the window, and update the output based on the changed (or new) data items. After the job finishes, the system memoize all the items in the sample and their respective sub-computation results for reuse for the subsequent windows [39].

The job provides an estimated output which is bound to a range of error due to approximation. INCAPPROX performs error estimation to estimate this error bound and define a confidence interval for the result as: $output \pm error\ bound$ [39].

The entire process repeats for the next window, with updated windowing parameters and the sample size. (Note that the query budget can be updated across windows during the course of stream processing to adapt to the user's requirements.)

2.5 Estimation of Error Bounds

In order to provide a confidence interval for the approximate output, the error bounds are computed due to approximation.

Approximation for aggregate functions. Aggregate functions require results based on all the data items or groups of data items in the population. But since the system computes only over a small sample from the population, an *estimated* result based on the weightage of the sample can be obtained.

Consider an input stream S, within a window, consisting of n disjoint strata S_1, S_2 ..., S_n, i.e., $S = \sum_{i=1}^{n} S_i$. Suppose the i^{th} stratum S_i has B_i items and each item j has an associated value v_{ij}. Consider an example of taking sum of these values across the whole window, represented as $\sum_{i=1}^{n}(\sum_{j=1}^{B_i} v_{ij})$. To find an approximate sum, INCAPPROX first selects a sample from the window based on stratified and biased sampling, i.e., from each i^{th} stratum S_i in the window, the system samples b_i items. Then it estimates the sum from this sample as: $\hat{\tau} = \sum_{i=1}^{n}(\frac{B_i}{b_i}\sum_{j=1}^{b_i} v_{ij}) \pm \varepsilon$ where the error bound ε is defined as:

$$\varepsilon = t_{f,1-\frac{\alpha}{2}} \sqrt{\widehat{Var}(\hat{\tau})} \tag{1}$$

Here, $t_{f,1-\frac{\alpha}{2}}$ is the value of the t-distribution (i.e., *t-score*) with f degrees of freedom and $\alpha = 1 -$ confidence level. The degree of freedom f is expressed as:

$$f = \sum_{i=1}^{n} b_i - n \tag{2}$$

The estimated variance for sum, $\widehat{Var}(\hat{\tau})$ is represented as:

$$\widehat{Var}(\hat{\tau}) = \sum_{i=1}^{n} B_i * (B_i - b_i) \frac{s_i^2}{b_i} \tag{3}$$

where s_i^2 is the population variance in the i^{th} stratum. Since the bias sampling is such that the statistics of stratified sampling is preserved, the statistical theories [57] for stratified sampling has been used to compute the error bound.

Currently, IncApprox supports error estimation only for aggregate queries. For supporting queries that compute extreme values, such as minimum and maximum, the system can make use of extreme value theory [22, 41] to compute the error bounds.

Error bound estimation. For error bound estimation, first the sample statistic used to estimate a population parameter is identified, e.g., *sum*, and a desired confidence level is selected, e.g., 95%. In order to compute the margin of error ε using t-score as given in Equation 1, the sampling distribution must be nearly normal. The Central Limit Theorem (CLT) states that when the size of sample is sufficiently large ($>= 30$), then the sampling distribution of a statistic approximates to *normal distribution*, regardless of the underlying distribution of values in the data [57]. Hence, IncApprox computes t-score using a t-distribution calculator [46], with the given degree of freedom f (see Equation 2), and cumulative probability as $1 - \alpha/2$ where $\alpha = 1 -$ confidence level [43]. Thereafter, the system estimates the variance using the corresponding equation for the sample statistic considered (for *sum*, the Equation is 3). Finally, the system uses this t-score and estimated variance of the sample statistic and computes the margin of error using Equation 1.

3 Discussion

The design of IncApprox is based on the assumptions in §2.2. Solving these assumptions is beyond the scope of this paper; however, this section discusses some of the approaches that could be used to meet the assumptions.

I: Stratification of sub-streams. Currently this work assumes that sub-streams are stratified, i.e., the data items of individual sub-streams have the same distribution. However, it may not be the case. Next, two alternative approaches are discussed, namely bootstrap [28] and a semi-supervised learning algorithm [45] to classify evolving data streams. Bootstrap [28, 29] is a non parametric re-sampling technique used to estimate parameters of a population. It works by randomly selecting a large number of bootstrap samples with replacement and with the same size as in the original sample. Unknown parameters of a population can be estimated by averaging these bootstrap samples. Such a bootstrap-based classifier could be created from the initial reservoir of data, and the classifier could be used to classify sub-streams. Alternatively, a semi-supervised algorithm [45] can be employed to stratify a data stream. This algorithm manipulates both unlabeled and labeled data items to train a classification model.

II: Virtual cost function. Secondly, this work assume that there exists a virtual function that computes the sample-size based on the user-specified query budget. The query budget could be specified as either available computing resources or latency requirements. Two existing approaches—Pulsar [11] and resource prediction model [58, 59, 60, 31, 44] can be used to design such a virtual function for given computing resources and latency requirements, respectively (see details in [39]).

4 Related Work

IncApprox builds on two computing paradigms, namely, incremental and approximate computing. This section provides a survey of techniques proposed in these two paradigms.

Incremental computation. To apply incremental computing, the earlier big data systems (Google's Percolator [50], and Yahoo's CBP [42]) proposed an alternative programming model where the programmer is asked to implement an efficient incremental-update mechanism. However, these systems depart from the existing programming model, and also require implementation of dynamic algorithms on per-application basis, which could be difficult to design and implement.

To overcome the limitations, researchers proposed transparent approaches for incremental computation. Examples of transparent systems include Incoop [18, 17], DryadInc [51], Slider [13, 14], and NOVA [20]. These systems leverage the underlying data-parallel programming model such as MapReduce [24] or Dryad [38] for supporting incremental computation. IncApprox was built on transparent big data systems for incremental computation. In particular, the system leverages the advancements in self-adjusting computation [4, 12] to improve the efficiency of incremental computation. In contrast to the existing approaches, IncApprox extends incremental computation with the idea of approximation, thus further improving the performance.

Approximate computation. Approximation techniques such as sampling [9], sketches [23], and online aggregation [37] have been well-studied over the decades in the context of traditional (centralized) database systems. In the last five years, sampling-based techniques have been successfully employed in many distributed data analytics systems [8, 33, 53, 52]. The systems such as ApproxHadoop [33] and BlinkDB [7, 8] showed that it is possible to achieve the benefits of approximate computing also in the context of distributed big data analytics. However, these systems target batch processing and are not able to support low-latency stream analytics.

Recently, StreamApprox [55, 54] proposed an online sampling algorithm to "on-the-fly" take samples of input data streams in a distributed manner. Interestingly, the sampling algorithm is generalizable to two prominent types of stream processing models: batched and pipelined stream processing models. Lastly, PrivApprox [53, 52] employed a combination of randomized response and approximate computation to support privacy-preserving stream analytics.

INCAPPROX builds on the advancements in approximate computing for big data analytics. However, the system is different from the existing approximate computing systems in two crucial aspects. First, unlike the existing systems, ApproxHadoop and BlinkDB, that are designed for batch processing—INCAPPROX is targeted at stream processing. Second, the system benefits from both approximate and incremental computing.

5 Conclusion

This paper presents the combination of incremental and approximate computations. The proposed approach transparently benefits unmodified applications, i.e., programmers do not have to design and implement application-specific dynamic algorithms or sampling techniques. The approach is based on the observation that both computing paradigms rely on computing over a subset of data items instead of computing over the entire dataset. These two paradigms are combined by designing a sampling algorithm that biases the sample selection to the memoized data items from previous runs.

References

1. Apache Flink. https://flink.apache.org/. Accessed: Nov, 2017.
2. Apache Storm. http://storm-project.net/. Accessed: Nov, 2017.
3. Kafka - A high-throughput distributed messaging system. http://kafka.apache.org. Accessed: Nov, 2017.
4. U. A. Acar. *Self-Adjusting Computation*. PhD thesis, Carnegie Mellon University, 2005.
5. U. A. Acar, G. E. Blelloch, M. Blume, R. Harper, and K. Tangwongsan. An experimental analysis of self-adjusting computation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2009.
6. U. A. Acar, A. Cotter, B. Hudson, and D. Türkoğlu. Dynamic well-spaced point sets. In *Proceedings of the 26th Annual Symposium on Computational Geometry (SoCG)*, 2010.
7. S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: Building fast and reliable approximate query processing systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2014.
8. S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2013.
9. M. Al-Kateb and B. S. Lee. Stratified reservoir sampling over heterogeneous data streams. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM)*, 2010.
10. M. Al-Kateb, B. S. Lee, and X. S. Wang. Adaptive-size reservoir sampling over data streams. In *Proceedings of the 19th International Conference on Scientific and Statistical Database Management (SSBDM)*, 2007.

11. S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska. End-to-end performance isolation through virtual datacenters. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2014.
12. P. Bhatotia. *Incremental Parallel and Distributed Systems*. PhD thesis, Max Planck Institute for Software Systems (MPI-SWS), 2015.
13. P. Bhatotia, U. A. Acar, F. P. Junqueira, and R. Rodrigues. Slider: Incremental Sliding Window Analytics. In *Proceedings of the 15th International Middleware Conference (Middleware)*, 2014.
14. P. Bhatotia, M. Dischinger, R. Rodrigues, and U. A. Acar. Slider: Incremental Sliding-Window Computations for Large-Scale Data Analysis. In *Technical Report: MPI-SWS-2012-004*, 2012.
15. P. Bhatotia, P. Fonseca, U. A. Acar, B. Brandenburg, and R. Rodrigues. iThreads: A Threading Library for Parallel Incremental Computation. In *proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.
16. P. Bhatotia, R. Rodrigues, and A. Verma. Shredder: GPU-Accelerated Incremental Storage and Computation. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2012.
17. P. Bhatotia, A. Wieder, I. E. Akkus, R. Rodrigues, and U. A. Acar. Large-scale incremental data processing with change propagation. In *Proceedings of the Conference on Hot Topics in Cloud Computing (HotCloud)*, 2011.
18. P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquini. Incoop: MapReduce for Incremental Computations. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2011.
19. G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.
20. C. Olston et al. Nova: Continuous Pig/Hadoop Workflows. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2011.
21. Y.-J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 1992.
22. S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer, 2001.
23. G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends databases*, 2012.
24. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2004.
25. C. Demetrescu, I. Finocchi, and G. Italiano. *Handbook on Data Structures and Applications*. Chapman & Hall/CRC, 2004.
26. C. Demetrescu, I. Finocchi, and G. Italiano. *Handbook on Data Structures and Applications, Chapter 36: Dynamic Graphs*. 2005.
27. A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 2000.
28. D. M. Dziuda. *Data mining for genomics and proteomics: analysis of gene and protein expression data*. John Wiley & Sons, 2010.
29. B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 1986.
30. D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In *Algorithms and Theory of Computation Handbook*. CRC Press, 1999.
31. A. S. Ganapathi. Predicting and optimizing system utilization and performance via statistical machine learning. In *Technical Report No. UCB/EECS-2009-181*, 2009.
32. M. N. Garofalakis and P. B. Gibbon. Approximate Query Processing: Taming the TeraBytes. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2001.
33. I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. ApproxHadoop: Bringing Approximations to MapReduce Frameworks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

34. L. J. Guibas. Kinetic data structures: a state of the art report. In *Proceedings of the third Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 1998.
35. P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang. Nectar: Automatic Management of Data and Computation in Datacenters. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010.
36. B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou. Comet: Batched Stream Processing for Data Intensive Distributed Computing. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2010.
37. J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1997.
38. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2007.
39. D. R. Krishnan, D. L. Quoc, P. Bhatotia, C. Fetzer, and R. Rodrigues. IncApprox: A Data Analytics System for Incremental Approximate Computing. In *Proceedings of the 25th International Conference on World Wide Web (WWW)*, 2016.
40. R. Ley-Wild, U. A. Acar, and M. Fluet. A cost semantics for self-adjusting computation. In *Proceedings of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2009.
41. S. Liu and W. Q. Meeker. Statistical methods for estimating the minimum thickness along a pipeline. *Technometrics*, 2014.
42. D. Logothetis, C. Olston, B. Reed, K. Web, and K. Yocum. Stateful bulk processing for incremental analytics. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2010.
43. S. Lohr. *Sampling: Design and Analysis, 2nd Edition*. Cengage Learning, 2009.
44. S. Mallick, G. Hains, and C. S. Deme. A resource prediction model for virtualization servers. In *Proceedings of International Conference on High Performance Computing and Simulation (HPCS)*, 2012.
45. M. M. Masud, C. Woolam, J. Gao, L. Khan, J. Han, K. W. Hamlen, and N. C. Oza. Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowledge and information systems*, 2012.
46. C. Math. The Apache Commons Mathematics Library. http://commons.apache.org/proper/commons-math. Accessed: Nov, 2017.
47. S. Misailovic, D. M. Roy, and M. C. Rinard. Probabilistically accurate program transformations. In *Proceedings of the 18th International Conference on Static Analysis (SAS)*, 2011.
48. D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: A Timely Dataflow System. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP)*, 2013.
49. S. Natarajan. *Imprecise and Approximate Computation*. Kluwer Academic Publishers, 1995.
50. D. Peng and F. Dabek. Large-scale Incremental Processing Using Distributed Transactions and Notifications. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010.
51. L. Popa, M. Budiu, Y. Yu, and M. Isard. DryadInc: Reusing work in large-scale computations. In *Proceedings of the Conference on Hot Topics in Cloud Computing (HotCloud)*, 2009.
52. D. L. Quoc, M. Beck, P. Bhatotia, R. Chen, C. Fetzer, and T. Strufe. Privacy preserving stream analytics: The marriage of randomized response and approximate computing. https://arxiv.org/abs/1701.05403, 2017.
53. D. L. Quoc, M. Beck, P. Bhatotia, R. Chen, C. Fetzer, and T. Strufe. PrivApprox: Privacy-Preserving Stream Analytics. In *Proceedings of the 2017 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC)*, 2017.
54. D. L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe. Approximate Stream Analytics in Apache Flink and Apache Spark Streaming. *CoRR*, abs/1709.02946, 2017.
55. D. L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe. StreamApprox: Approximate Computing for Stream Analytics. In *Proceedings of the International Middleware Conference (Middleware)*, 2017.

56. S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE)*, 2011.
57. S. K. Thompson. *Sampling*. Wiley Series in Probability and Statistics, 2012.
58. A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Brief Announcement: Modelling MapReduce for Optimal Execution in the Cloud. In *proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of Distributed Computing (PODC)*, 2010.
59. A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Conductor: Orchestrating the Clouds. In *proceedings of the 4th international workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2010.
60. A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the Deployment of Computations in the Cloud with Conductor. In *proceedings of the 9th USENIX symposium on Networked Systems Design and Implementation (NSDI)*, 2012.
61. M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP)*, 2013.