Rethinking Adaptability in Wide-Area Stream Processing Systems

Albert Jonathan, Abhishek Chandra, and Jon Weissman
University of Minnesota Twin Cities

Abstract

Adaptability is an important property of stream processing systems since the systems need to maintain low latency and high throughput execution of long-running queries. In a wide-area environment, dynamics are common not only because of the workload variability but also the nature of wide-area network (WAN) bandwidth that frequently changes. In this work, we study the adaptability property of stream processing systems designed for a wide-area environment. Specifically, we (1) discuss the challenges of reconfiguring query executions in a wide-area environment, (2) propose ideas how to adapt existing reconfiguration techniques used in centralized Cloud to a wide-area environment, and (3) discuss the tradeoffs between them. A key finding is that the best adaptation technique to use depends on the network conditions, types of query, and optimization metrics.

1 Introduction

Streaming analytics has been increasingly popular due to the real-time need of many big data analytics applications. This can be seen in a recent development of various distributed stream processing systems to process continuous data streams with low latency and high throughput [26, 44, 24, 37, 6, 19]. Most of the work in streaming analytics has focused on a centralized Cloud environment. Unfortunately, using these systems for processing geo-distributed data is very inefficient and it may result in a wasteful resource utilization [41, 29, 40]. Examples of such analysis include real-time global trend detection on social network, and real-time log monitoring of geo-distributed CDN servers [28, 31].

Adopting existing centralized Cloud-based systems into wide-area settings requires rethinking some of their designs due to the fundamental differences of the runtime environment. For example, centrally aggregating geo-distributed data without WAN awareness has been shown to be very inefficient due to the highly heterogeneous and limited WAN bandwidth. This may result in 19× higher latency [29] or 250× higher bandwidth utilization [41] compared to the WAN-aware deployment.

In this paper, we rethink the adaptability property of stream processing systems in a wide-area environment. Adaptability is a key property of stream processing systems due to the long-running nature of stream queries. Yet, runtime dynamics such as varying workload pattern [21, 22], changes in network topology [18], occurrence of stragglers [3], node additions, and failures [12] are common and inevitable in distributed systems. Thus, the systems need to gracefully adapt to these dynamics to maintain low latency and high throughput execution.

Existing work has addressed the adaptability property of stream processing systems, but focuses on a centralized Cloud environment [7, 30, 11, 42, 39]. In such an environment, dynamics are typically caused by workload variability, stragglers, or node failures [3]. In a wide-area environment, the scarce and heterogeneous WAN bandwidth further adds extra challenges since it is highly dynamic in practice (topology changes every ∼5-10 minutes [15, 18]). Previous work [31] has addressed the importance of adaptability in wide-area streaming analytics, but relies on aggregation and approximation that may introduce errors, which may not be applicable for queries that require exact computations.

In this paper, we discuss different reconfiguration approaches that can be used to handle WAN dynamics. Specifically, we discuss the challenges of *scale out* in wide-area settings and propose a reconfiguration technique that dynamically *re-plans* query executions based on the WAN condition. We show that adopting these reconfiguration approaches into wide-area settings requires rethinking of their designs due to the different assumptions of the environment. Finally, we discuss the tradeoffs between these approaches and how they can be used in practice. A key finding is that the best adaptation technique to use depends on dynamic network conditions.

2 Background & Related Work

2.1 Distributed Stream Processing Systems

Stream Processing Models: A streaming analytics query is typically written using a SQL-like declarative language [36, 4]. It is translated by a *query optimizer* into a query plan represented as a directed acyclic graph (DAG), where the vertices are stream operators and the edges are data flow. A *job scheduler* then takes the DAG and deploys each of the operators based on its scheduling policy (e.g., locality-aware). Once a query has been deployed, it typically runs continuously [2, 19, 39].

Today's distributed stream processing systems can be generally categorized based on their processing models: *Bulk Synchronous Parallel* (BSP) model, and *Continuous Operator* (CO) model. The BSP model partitions continuous data streams into a set of micro batches and processes each batch similar to the batch processing systems [44, 8]. On the other hand, the CO model processes each data/event independently as it arrives (except for explicit grouping such as windowing) [37, 6, 24]. In this work, we consider the CO model since in general it provides lower latency and higher throughput execution [10] and more importantly, it incurs less communication overhead [39] that is critical in wide-area settings.

Adaptability in Stream Processing Systems: Adaptability is an important property of stream processing systems because most stream workloads are long-running and runtime dynamics are inevitable in distributed systems. These dynamics include varying workload pattern [21, 22], stragglers [3], and even failures [12]. Thus, distributed systems designed for long-running workloads need to gracefully adapt to runtime dynamics without sacrificing their normal execution performance.

Stream processing systems that use the BSP model typically adapt to dynamics during the synchronization stage between two coordination intervals. At this stage, a job scheduler may quickly reconfigure an already-running job by rescheduling or scaling out/in each operator. Recent work on this model has proposed a technique to handle runtime changes by dynamically adapting the processing interval [11] and disjoining the processing and synchronization intervals [39]. On the other hand, stream processing systems that use the CO model typically integrate scale out and fault tolerance using a *checkpoint-then-restart* mechanism [7, 30, 2, 20]. In this mechanism, the systems will (1) synchronize operators to checkpoint their states using distributed checkpointing algorithm [9, 27, 5], (2) suspend the execution for reconfiguration, then (3) restart the execution from the last checkpointed state. The checkpoint stage is particularly useful for ensuring *exactly-once* processing semantic.

2.2 Wide-Area Data Analytics

Recent work in wide-area data analytics [29, 41] has addressed the importance of WAN-aware job scheduling that is critical to the overall query execution performance. Clarinet [40] further addresses the importance of bringing the WAN awareness into the query optimizer in selecting the optimal execution plan based on the WAN bandwidth availability, while Gaia [16] proposes optimization techniques for geo-distributed machine learning. Although their work are related to ours, they assume that WAN bandwidth between sites are static and the workloads remain stable over the runtime of the queries, which is reasonable for short-running queries. However, these assumptions are not reasonable for streaming analytics queries that are long-running.

Other work [28, 17, 14, 31] has also looked at the problem of streaming analytics in a wide-area environment. Pietzuch et al. [28] focuses on an operator placement problem in an ad-hoc wide-area topology, while Hwang et al. [17] proposes a replication technique to ensure reliable stream processing over the WAN. Heintz et al. [14] proposes an algorithm that trades-off timeliness and accuracy in the context of windowed-group aggregation. JetStream [31] models data streams using the data cube abstraction and proposes aggregation/degradation techniques to handle WAN bandwidth constraints. Although aggregating data streams as proposed in JetStream can handle WAN dynamics, it may introduce a certain degree of errors which may not be applicable for applications that require exact processing (e.g., billing query). Instead, it would be desirable to adapt to runtime dynamics without sacrificing the quality of the queries.

2.3 Dynamics in Wide-Area Environment

Most of the work in adaptive stream processing systems has focused on a centralized Cloud environment where the main sources of dynamics are workload variability and stragglers that lead to computational resource bottlenecks. In this case, a system can reconfigure a query execution by provisioning more computing resources and scaling out some of its operators across multiple worker nodes [7, 34, 43]. In a wide-area environment, the limited WAN bandwidth between sites is typically the main cause of bottleneck [29, 41, 40]. Furthermore, this WAN constraint is very dynamic in practice [18, 15]. In practice, WAN topology that connects multiple geo-distributed sites may change every 5 to 10 minutes [18]. Thus, how stream processing systems adapt to WAN dynamics will define their performance. In this paper, we discuss how to adapt existing centralized Cloud-based reconfiguration techniques to wide-area stream processing systems to handle WAN dynamics.

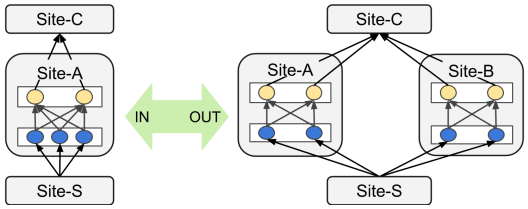
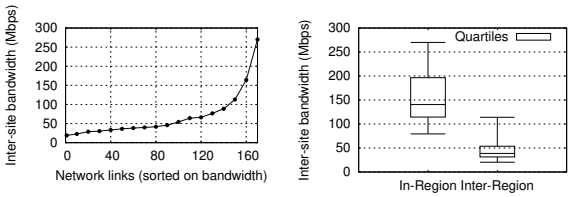


Figure 1: Scale out/in operators across sites



(a) Bandwidth heterogeneity (b) Bandwidth distribution

Figure 2: Inter-site EC2 bandwidth distribution

3 Dynamic Reconfiguration

In this section, we discuss different reconfiguration approaches in wide-area stream processing systems. Specifically, we discuss how to *scale out* stream operators (§3.1) and propose a query *re-planning* technique (§3.2) to adapt to wide-area dynamics.

3.1 Scaling Out Resources Across Sites

A common approach to handle runtime dynamics in a centralized Cloud environment is by scaling out/in operators across multiple worker nodes *within* a data center/site [30, 42, 7, 43, 34, 25]. Existing research has proposed techniques to ensure the semantic correctness [30] and how to re-distribute workload across multiple worker nodes [43, 34, 25]. This reconfiguration approach only changes the deployment and the parallelism of each operator without changing the query execution plan itself.

Scaling out operators *within* a site can be considered as a *scale up* case in wide-area settings, which can be used to solve computational resource bottleneck. However, this does not handle potential bottleneck in data transmission over the WAN that may be caused by an increasing data rate and/or a decreasing bandwidth capacity. We consider the *load* of a link from Site-S to Site-D as a ratio between the data transmission rate over its bandwidth capacity ($L_S^D = \frac{streamRate}{bandwidth_{S,D}}$). A ratio > 1 indicates that the network link is contended which may result in stale results and performance degradation.

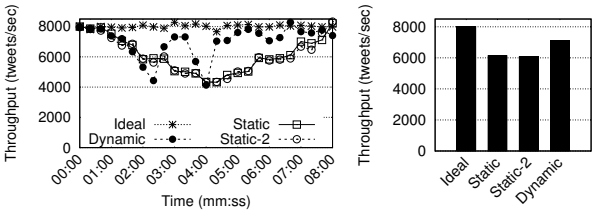
Figure 1 presents an example showing how scaling out operators across sites may reduce the load of overloaded links. Suppose that the stream rate from Site-S to Site-A is higher than its bandwidth capacity ($L_S^A > 1$) which results in a bandwidth contention. In this case, scaling out operators from Site-A to Site-B and partitioning the data streams across both sites can reduce L_S^A. However, this may impose an additional overhead of aggregating the results of the scaled operators. Thus, the system may *scale in* the operators back to a single-site deployment when the load decreases ($L_S^A < 1$).

Scaling out operators can also handle overloaded links between sites that are used for transmitting data streams to downstream operators. In streaming analtyics, the output data rate typically depends on its input rate as well as the type of the operators, and most operators (e.g., selection/projection/filter/reduce) result in a reduced output rate [1]. In this case, scaling out operators across sites may also reduce the egress links' load of a particular site. For example, if the link from Site-A to Site-C is overloaded ($L_A^C > 1$), scaling out operators to Site-B may reduce L_A^C by distributing its load with L_B^C.

Operator Migration and Fault Tolerance: Alternatively, an operator can be migrated to another site without changing its parallelism, p. This can be viewed as a special case of a scale out where the scheduler essentially scales the operator out with a parallelism of 0 at the original site and p at the new site. The question on *how much parallelism an operator should have at each site* itself depends on the scaling policy, which will be discussed later. The scale out mechanism can also be used to handle operator failures since restarting a failed operator can be viewed as scaling out the operator from 0 parallelism to p as shown in a previous work [7].

Where to scale: One question that needs to be addressed in scaling out operators across multiple sites is *where an operator should be scaled out to*. In a centralized environment, this question may not be as critical as in a wide-area environment since network links within a data center typically have a very high bandwidth ($15\times$ to $60\times$ higher than inter-data center bandwidth [16]) and they are less heterogeneous, as presented in Figure 2(a) which shows the measured bandwidth links between 14 Amazon EC2 data centers. We can see that the bandwidth of a link that connects two sites can be as high as $20\times$ compared to the other links. This bandwidth heterogeneity will affect the overhead of scaling operators with large states [42] since it requires distributing the states over the WAN. Thus, unlike in a centralized Cloud environment, scaling out resources in a wide-area environment needs to consider the heterogeneity of WAN bandwidth.

We propose a solution to mitigate the scale out overhead by scaling out operators only to sites that are located within the same region/continent. The key insight behind this policy is based on the observation that WAN bandwidth between sites that are located within the same region is typically much higher than those that connect sites across regions (Figure 2(b)). Hence, scaling out operators within a region may mitigate both the state migration and the aggregation overhead.



(a) Throughput over time (b) Average throughput

Figure 3: Benefit of scale out during network dynamic

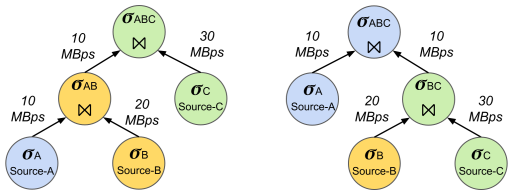


Figure 4: Different query plans for the same query

Scale out policy: There has been work that looks at load balancing algorithms for stream processing systems within a cluster environment [35, 43, 7, 13]. They typically adopt a scale out policy based on CPU utilization [7], the load distribution among worker nodes [25, 32] or rely on application writers to determine the proper policies specifically for their applications [42, 30, 33].

The problem of load sharing in wide-area streaming analytics is challenging due to the highly heterogeneous and dynamic WAN bandwidth. In the case of scaling out stateless operators, the system can adapt a shuffle-based load balancing algorithm by incorporating a weight factor to each link that defines its available bandwidth. Thus, the workload will be distributed proportionally based on the available bandwidth of each link. The weight factor of each link needs to be updated as the bandwidth changes. In the case of stateful operators, a shuffle-based load balancing may not be applicable since it imposes an additional aggregation overhead. Recent work [25] has proposed an alternative key-based load balancing algorithm based on the "power of two choices" [23]. Although this algorithm can be used in practice, we are still investigating how to adapt this policy into a wide-area environment.

Preliminary Experiment: We conducted a preliminary experiment to show the benefit of scale out to handle WAN dynamics. We implemented our reconfiguration technique as a prototype module on Apache Flink [1] and deployed our system on a localized CloudLab [2] cluster that emulated a real wide-area environment. The bandwidth between sites were controlled using `iperf` and we introduced dynamics by periodically changing the bandwidth between sites by 10% to 30% with a maximum deviation of 50% from the original bandwidth.

The workload was based on a real Twitter trace whose playback rate had been scaled to approximately 8000 tweets/sec to reflect the actual rate of Twitter [38]. We deployed an application that periodically outputs the top-k most popular topics and their sentiment scores for each country. The tweets were distributed based on their geolocation information across 4 different sites.

[1] http://flink.apache.org/

[2] https://www.cloudlab.us

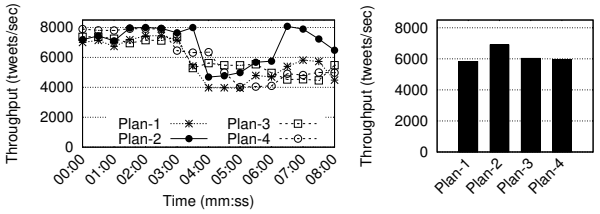
Figure 3 shows the benefit of scale out. All approaches executed the same execution plan. Both `Static` and `Static-2` did not react to runtime dynamics and the only difference between them was that the latter used $2\times$ computational resources. We see that a higher number of computational resources did not affect the performance since the bottleneck was in the network. On the other hand, the `Dynamic` reacted to load changes by scaling out operators to nearby sites whenever the throughput dropped below 5000 tweets/sec. We also plotted `Ideal` showing the ideal case where no bandwidth was contended. We see that although the `Dynamic` had a couple throughput drops caused by the transient suspension of execution, it resulted in a higher overall throughput compared to the static cases (Figure 3(b)). We believe this overhead can be mitigated by scaling out operators and redistributing the streams concurrently.

3.2 Changing Query Execution Plans

In this section we propose a query re-planning technique to adapt to WAN dynamics. Our motivation is based on the observation that the optimal execution plan of a wide-area data analytics query may change over time depending on the WAN bandwidth availability [40].

Consider an example in Figure 4 which shows two different execution plans for the same query. It consumes input streams from 3 sources: A, B, and C that are located at different sites, and joins them using a *full hash join*, which is commutative. The query optimizer may prefer the first plan if the bandwidth is sufficient since it consumes lower bandwidth ($20MBps$) compared to the second plan ($30MBps$). However, if the link from Site-A to Site-B is overloaded, while the link from Site-B to Site-C has sufficient bandwidth, the second plan will result in a better performance. Thus, the query optimizer may re-plan the query based on the WAN condition.

Re-planning queries with stateful operators: The main challenge in re-planning a query is in migrating the states of stateful operators. Although the query optimizer guarantees the correctness of executing different plans, they may have different intermediate operators with different state semantics. For example, the state of $\sigma(A \bowtie B)$ may not be the same as $\sigma(B \bowtie C)$. Thus, the state of $\sigma(A \bowtie B)$ cannot be migrated.



(a) Throughput over time (b) Average throughput

Figure 5: Different query plan performance

In order to ensure the exactly once processing semantic when changing a query plan, the query has to have finite states where reconfiguration can be done at the end of the state interval. This is similar to the coordination interval in the BSP model. For example, in the case of window operation that groups streams every T time units, the reconfiguration can be performed at the end of the interval T once the states have been processed or re-initiated. Thus, reordering operators is only applicable for applications that are either stateless or have finite states.

Preliminary Experiment: To show the benefit of re-planning a query execution, we conducted an experiment where we compared 4 different execution plans for the same query. The only difference between them is the order of the aggregation operators. The setup was the same as the setup in §3.1. Figure 5(a) shows that none of the plans performs the best at all time. Although Plan-2 resulted in the highest overall throughput (Figure 5(b)), Plan-3 and Plan-4 outperformed the other plans at a certain interval. This shows that changing a query's plan may improve its overall execution performance.

4 Discussion

Approximation vs. Reconfiguration: In general, reducing stream rate using approximation/aggregation is highly desirable to mitigate WAN bandwidth utilization. However, they may introduce a certain degree of error or inaccuracy that may not be applicable for applications that require exact computation. On the other hand, reconfiguration does not affect the quality of an execution but, it may incurs higher overhead. In fact, both aggregation and reconfiguration can be jointly used to handle runtime dynamics. For example, a system may first reduce stream rate and start reconfiguring when the approximation results in a significant loss of quality. Alternatively, it may scale out first to maintain high accuracy and use approximation if it is unable to satisfy the performance goal. Thus, both techniques can be used together to maintain a high quality result while maintaining a high performance execution in the face of network constraint.

Scale Out vs. Query Re-planning: We conducted an initial experiment that explores different cases where scale out can outperform query re-planning and vice versa (see Figure 6). The workload that was used in this experiment is the same as in the two previous experiments. In a certain case, both techniques can handle runtime dynamics (Case-2). However, we observed that scale out typically consumes more resources and incurs extra aggregation overhead, which can be mitigated if the query optimizer can identify an alternative execution plan that better fits the condition (Case-3). However, re-planning also has a drawback of having higher reconfiguration overhead and limited applicability as discussed in §3.2. Scale out may also result in better performance by distributing the workload across multiple sites in the case where the query optimizer is unable to find an alternative plan that avoids overloaded links (Case-4). We believe that the decision as to which approach should be used highly depends on many factors such as the type of dynamics and queries, and this requires further research.

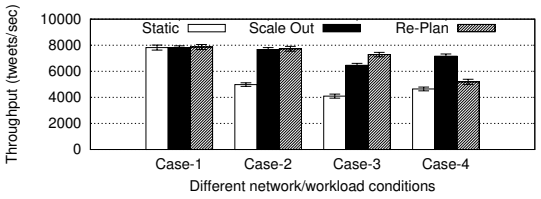


Figure 6: Benefit of Scale Out vs. Re-planning

Reconfiguration for concurrent queries: In addition to handling runtime dynamics, reconfiguration can also be used to optimize the deployment of multiple concurrent queries. Since, streaming analytics queries may not be batch-scheduled in general, rescheduling existing executions or changing their query plans may result in a global optimal deployment. We are currently looking at the opportunity of reconfiguration for optimizing multiple concurrent query executions.

5 Conclusion

In this paper, we study the problem of adaptability in wide-area stream processing systems. We discuss different reconfiguration approaches: *scale out* and *query re-planning* to adapt to WAN dynamics. Specifically, we address the challenges of adapting existing reconfiguration techniques into a wide-area environment, propose initial ideas on how to efficiently apply these techniques, and discuss the trade-offs between them. We believe that further research needs to be done in this area which includes minimizing the reconfiguration overhead, the reconfiguration policy, and state management across sites.

6 Acknowledgement

The authors would like to acknowledge grant NSF CNS-1619254 and CNS-1717834 that supported this research. We also thank the anonymous HotCloud reviewers for their insightful feedback.

References

[1] AGARWAL, S., KANDULA, S., BRUNO, N., WU, M.-C., STOICA, I., AND ZHOU, J. Re-optimizing data-parallel computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (2012), USENIX Association, pp. 21–21.

[2] AKIDAU, T., BALIKOV, A., BEKIROĞLU, K., CHERNYAK, S., HABERMAN, J., LAX, R., MCVEETY, S., MILLS, D., NORDSTROM, P., AND WHITTLE, S. Millwheel: fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment 6*, 11 (2013), 1033–1044.

[3] ANANTHANARAYANAN, G., GHODSI, A., SHENKER, S., AND STOICA, I. Effective straggler mitigation: Attack of the clones. In *NSDI* (2013), vol. 13, pp. 185–198.

[4] ARMBRUST, M., XIN, R. S., LIAN, C., HUAI, Y., LIU, D., BRADLEY, J. K., MENG, X., KAFTAN, T., FRANKLIN, M. J., GHODSI, A., ET AL. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015), ACM, pp. 1383–1394.

[5] CARBONE, P., EWEN, S., FÓRA, G., HARIDI, S., RICHTER, S., AND TZOUMAS, K. State management in apache flink®: consistent stateful distributed stream processing. *Proceedings of the VLDB Endowment 10*, 12 (2017), 1718–1729.

[6] CARBONE, P., KATSIFODIMOS, A., EWEN, S., MARKL, V., HARIDI, S., AND TZOUMAS, K. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 36*, 4 (2015).

[7] CASTRO FERNANDEZ, R., MIGLIAVACCA, M., KALYVIANAKI, E., AND PIETZUCH, P. 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* (2013), ACM, pp. 725–736.

[8] CHAMBERS, C., RANIWALA, A., PERRY, F., ADAMS, S., HENRY, R. R., BRADSHAW, R., AND WEIZENBAUM, N. Flumejava: easy, efficient data-parallel pipelines. In *ACM Sigplan Notices* (2010), vol. 45, ACM, pp. 363–375.

[9] CHANDY, K. M., AND LAMPORT, L. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS) 3*, 1 (1985), 63–75.

[10] CHINTAPALLI, S., DAGIT, D., EVANS, B., FARIVAR, R., GRAVES, T., HOLDERBAUGH, M., LIU, Z., NUSBAUM, K., PATIL, K., PENG, B. J., ET AL. Benchmarking streaming computation engines: storm, flink and spark streaming. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International* (2016), IEEE, pp. 1789–1792.

[11] DAS, T., ZHONG, Y., STOICA, I., AND SHENKER, S. Adaptive stream processing using dynamic batch sizing. In *Proceedings of the ACM Symposium on Cloud Computing* (2014), ACM, pp. 1–13.

[12] FORD, D., LABELLE, F., POPOVICI, F. I., STOKELY, M., TRUONG, V.-A., BARROSO, L., GRIMES, C., AND QUINLAN, S. Availability in globally distributed storage systems. In *OSDI* (2010), vol. 10, pp. 1–7.

[13] GEDIK, B. Partitioning functions for stateful data parallelism in stream processing. *The VLDB Journal 23*, 4 (2014), 517–539.

[14] HEINTZ, B., CHANDRA, A., AND SITARAMAN, R. K. Trading timeliness and accuracy in geo-distributed streaming analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (2016), ACM, pp. 361–373.

[15] HONG, C.-Y., KANDULA, S., MAHAJAN, R., ZHANG, M., GILL, V., NANDURI, M., AND WATTENHOFER, R. Achieving high utilization with software-driven wan. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 15–26.

[16] HSIEH, K., HARLAP, A., VIJAYKUMAR, N., KONOMIS, D., GANGER, G. R., GIBBONS, P. B., AND MUTLU, O. Gaia: Geo-distributed machine learning approaching lan speeds. In *NSDI* (2017), pp. 629–647.

[17] HWANG, J.-H., CETINTEMEL, U., AND ZDONIK, S. Fast and reliable stream processing over wide area networks. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on* (2007), IEEE, pp. 604–613.

[18] JAIN, S., KUMAR, A., MANDAL, S., ONG, J., POUTIEVSKI, L., SINGH, A., VENKATA, S., WANDERER, J., ZHOU, J., ZHU, M., ET AL. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 3–14.

[19] KULKARNI, S., BHAGAT, N., FU, M., KEDIGEHALLI, V., KELLOGG, C., MITTAL, S., PATEL, J. M., RAMASAMY, K., AND TANEJA, S. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015), ACM, pp. 239–250.

[20] LIN, W., FAN, H., QIAN, Z., XU, J., YANG, S., ZHOU, J., AND ZHOU, L. Streamscope: Continuous reliable distributed processing of big data streams. In *NSDI* (2016), vol. 16, pp. 439–453.

[21] Stream-processing with mantis. `https://medium.com/netflix-techblog/stream-processing-with-mantis-78af913f51a6`, year=2016.

[22] MEISNER, D., SADLER, C. M., BARROSO, L. A., WEBER, W.-D., AND WENISCH, T. F. Power management of online data-intensive services. In *ACM SIGARCH Computer Architecture News* (2011), vol. 39, ACM, pp. 319–330.

[23] MITZENMACHER, M. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems 12*, 10 (2001), 1094–1104.

[24] MURRAY, D. G., MCSHERRY, F., ISAACS, R., ISARD, M., BARHAM, P., AND ABADI, M. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 439–455.

[25] NASIR, M. A. U., MORALES, G. D. F., KOURTELLIS, N., AND SERAFINI, M. When two choices are not enough: Balancing at scale in distributed stream processing. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on* (2016), IEEE, pp. 589–600.

[26] NEUMEYER, L., ROBBINS, B., NAIR, A., AND KESARI, A. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on* (2010), IEEE, pp. 170–177.

[27] NOGHABI, S. A., PARAMASIVAM, K., PAN, Y., RAMESH, N., BRINGHURST, J., GUPTA, I., AND CAMPBELL, R. H. Samza: stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment 10*, 12 (2017), 1634–1645.

[28] PIETZUCH, P., LEDLIE, J., SHNEIDMAN, J., ROUSSOPOULOS, M., WELSH, M., AND SELTZER, M. Network-aware operator placement for stream-processing systems. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on* (2006), IEEE, pp. 49–49.

[29] PU, Q., ANANTHANARAYANAN, G., BODIK, P., KANDULA, S., AKELLA, A., BAHL, P., AND STOICA, I. Low latency geo-distributed data analytics. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 421–434.

[30] QIAN, Z., HE, Y., SU, C., WU, Z., ZHU, H., ZHANG, T., ZHOU, L., YU, Y., AND ZHANG, Z. Timestream: Reliable stream computation in the cloud. In *Proceedings of the 8th ACM European Conference on Computer Systems* (2013), ACM, pp. 1–14.

[31] RABKIN, A., ARYE, M., SEN, S., PAI, V. S., AND FREEDMAN, M. J. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In *NSDI* (2014), vol. 14, pp. 275–288.

[32] RIVETTI, N., QUERZONI, L., ANCEAUME, E., BUSNEL, Y., AND SERICOLA, B. 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* (2015), ACM, pp. 80–91.

[33] SATZGER, B., HUMMER, W., LEITNER, P., AND DUSTDAR, S. Esc: Towards an elastic stream computing platform for the cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on* (2011), IEEE, pp. 348–355.

[34] SCHNEIDER, S., ANDRADE, H., GEDIK, B., BIEM, A., AND WU, K.-L. Elastic scaling of data parallel operators in stream processing. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* (2009), IEEE, pp. 1–12.

[35] SHAH, M. A., HELLERSTEIN, J. M., CHANDRASEKARAN, S., AND FRANKLIN, M. J. Flux: An adaptive partitioning operator for continuous query systems. In *Data Engineering, 2003. Proceedings. 19th International Conference on* (2003), IEEE, pp. 25–36.

[36] THUSOO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ZHANG, N., ANTONY, S., LIU, H., AND MURTHY, R. Hive-a petabyte scale data warehouse using hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on* (2010), IEEE, pp. 996–1005.

[37] TOSHNIWAL, A., TANEJA, S., SHUKLA, A., RAMASAMY, K., PATEL, J. M., KULKARNI, S., JACKSON, J., GADE, K., FU, M., DONHAM, J., ET AL. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), ACM, pp. 147–156.

[38] Twitter statistics. `http://www.internetlivestats.com/twitter-statistics/`, year=2017.

[39] VENKATARAMAN, S., PANDA, A., OUSTERHOUT, K., ARMBRUST, M., GHODSI, A., FRANKLIN, M. J., RECHT, B., AND STOICA, I. Drizzle: Fast and adaptable stream processing at scale. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), ACM, pp. 374–389.

[40] VISWANATHAN, R., ANANTHANARAYANAN, G., AND AKELLA, A. Clarinet: Wan-aware optimization for analytics queries. In *OSDI* (2016), vol. 16, pp. 435–450.

[41] VULIMIRI, A., CURINO, C., GODFREY, P. B., JUNGBLUT, T., PADHYE, J., AND VARGHESE, G. Global analytics in the face of bandwidth and regulatory constraints. In *NSDI* (2015), vol. 7, pp. 7–8.

[42] WU, Y., AND TAN, K.-L. Chronostream: Elastic stateful stream computation in the cloud. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on* (2015), IEEE, pp. 723–734.

[43] XING, Y., ZDONIK, S., AND HWANG, J.-H. Dynamic load distribution in the borealis stream processor. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on* (2005), IEEE, pp. 791–802.

[44] ZAHARIA, M., DAS, T., LI, H., SHENKER, S., AND STOICA, I. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. *HotCloud 12* (2012), 10–10.