SheerMP: Optimized Streaming Analytics-as-a-Service over Multi-site and Multi-platform Settings

George Stamatakis
gstamatakis@athenarc.gr
Athena Research Center

Antonios Kontaxakis*
antonios.kontaxakis@ulb.be
Universite Libre de Bruxelles

Alkis Simitsis
alkis@athenarc.gr
Athena Research Center

Nikos Giatrakos
ngiatrakos@athenarc.gr
Athena Research Center
Technical University of Crete

Antonios Deligiannakis
adeli@athenarc.gr
Athena Research Center
Technical University of Crete

ABSTRACT

Analytics are in the core of many emerging applications and can greatly benefit from the abundance of data and the progress in the processing capabilities of modern hardware. Still, new challenges arise with the extreme complexity of deciding how to execute analytics workflows given the plethora of choices of various cloud providers, the fragmented nature of diverse Big Data technologies, and the difficult task of resource provisioning to dynamically satisfy the demands of running streaming analytics over time. In this paper, we demonstrate a prototype system that optimizes streaming analytics workflows across Big Data platforms and computer clusters. Our system is the first that (i) considers a multi-user setup, (ii) examines the availability of multiple (potentially, geo-dispersed) compute choices, and (iii) provides a holistic framework covering a wide variety of practical optimization and adaptive resource allocation scenarios over a variety of streaming Big Data platforms.

1 INTRODUCTION

Analytics-as-a-Service (AaaS) providers offer subscription-based, domain-specific Business Intelligence (BI) solutions. End-to-end BI solutions collect relevant data, analyze them and present results in a manner largely tailored to business analysts. Still, under the hood, IT professionals have to deal with the complexity of the hardware and software infrastructure. Administering such an infrastructure designed to serve multiple, concurrently running analytics workflows of various end-users involves complex decisions. The choice of compute clusters (e.g., corporate data center, one or more cloud providers), Big Data platforms and hardware resources for multiple users with dozens of analytics requests are dimensions of a hard to manage decision problem.

Consider the following real-world, motivating scenario from the financial domain. In the AaaS model, a financial advisory company provides BI services to their customers via client applications enabling them to design analytics workflows and submit them to a financial AaaS provider. The provider executes the requested analytics tasks over an infrastructure composed of one or more Big Data platforms deployed on multiple compute clusters, henceforth termed sites. For instance, Microsoft Azure HDInsight supports Apache Spark and Kafka, while Google Cloud also offers Apache Flink. These sites may be located near regional market data centers. Even for a single on-premises compute cluster that accumulates all relevant market data, the financial AaaS provider often exports their workload peaks to public clouds operating over a hybrid cloud setting.

Such analytics workflows are streaming in nature. For instance, customers want to maintain a diversified portfolio. Thus, they require continuous analysis of stock trades and bids to extract knowledge about highly correlated pairs of stocks and counts of bids per stock. In that, financial advisors can automatically recommend bids on uncorrelated stocks or (using the aforementioned count) indicate the leaders in pinpointed correlations.

For efficiently exploiting the available infrastructure, we need to make informed decisions about the sites and the resources required for the workflows running on a multitude of Big Data platforms. Given the complexity of the underlying multi-site, multi-platform setting and the often unpredictably bursty amount of client requests, this becomes a non-trivial, multi-objective, multi-dimensional optimization problem. Performance metrics for guiding such decisions combine throughput, communication cost, processing and network latency as well as various Service Level Agreement (SLA) constraints. In addition, these are not one-shot decisions. In streaming workflows, data rates, data distributions, workflow concurrency degree, etc, which affect the said metrics, are highly volatile variables. Running workflows require continuous monitoring through lightweight metrics collection, which may often lead to reevaluation of workflow execution strategy (i.e., workflow execution plans) and elastic (adaptive) resource allocation at runtime.

Contributions. We present SheerMP, a system that provides end-to-end support for optimized AaaS infrastructure administration under the presence of large-scale streaming workflows. SheerMP is the first that (i) considers a multi-user setup, (ii) examines the availability of multiple (potentially, geo-dispersed) compute choices, and (iii) provides a holistic framework covering a wide variety of practical optimization and adaptive resource allocation scenarios. SheerMP automates optimization decisions, submits and migrates streaming analytics workflows, and monitors their execution over a variety of streaming Big Data platforms. Our prototype supports popular streaming Big Data platforms, such as Apache Flink, Spark Structured Streaming, and Kafka.

Related Work. Cross-platform systems [1, 6, 9, 10, 13, 15, 19, 20] focus on batch, instead of streaming settings [14] and do not consider runtime adaptation. Systems supporting streams focus on a single streaming platform [1, 17]. Stream processing systems [5, 7, 11, 12, 18, 21, 23] have touched upon aspects of optimization in the context of adaptive re-scaling of streaming analytics workflows. Nonetheless, these works focus on a single engine as well, e.g., Apache Storm, Heron, and Spark Streaming.

[*] Work done while the author was with Athena Research Center.

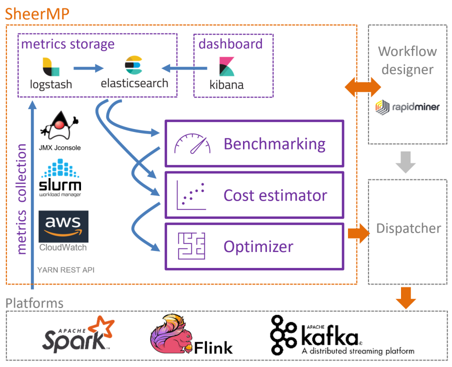


Figure 1: System architecture

2 DEMONSTRABLE FEATURES OF SheerMP

Figure 1 shows the architecture of SheerMP. The workflow lifecycle starts with the workflow designer. Next, the flow is sent to the optimizer, which generates a workflow execution plan making cost model-driven decisions. Finally, the workflow is sent to the dispatcher (not explained here) and gets scheduled for execution on one or more platforms and sites.

Workflow creation. SheerMP integrates RapidMiner (RM) Studio[1] to enable graphical workflow design. Workflows are imported to RM via our REST API connector using JSON encoding. We support all stream transformation operators provided by Spark Structured Streaming, Flink, and Kafka, as well as online machine learning [8] and data synopses operators [16] provided by the streaming extension of RM. SheerMP uses a dictionary to map platform-agnostic operators of logical workflows designed in RM to physical operators for the supported platforms. RM connection objects are used to derive the resource capacity at available sites.

Statistics collection. SheerMP collects a variety of performance metrics for running workflows following common practices met in APM systems [3] and periodically probes metrics via JMX or taps onto endpoints such as Yarn REST APIs and CloudWatch. The metrics collected at runtime are stored in an ELK stack[2] deployment.

Benchmarking. We use automated micro-benchmarks to profile operators and feed our cost models avoiding a cold start. The optimizer uses the cost models at runtime to generate plans for new workflows or reevaluate plans of long-running workflows. The Benchmarking module submits sample jobs in a principled fashion and uses our Statistics Collector to collect metrics at both, (a) the *operator level*, such as job duration, input/output type (i.e. Kafka, or custom source/sink), records/bytes sent/received, key ranges and distribution of incoming tuples, provisioned resources (workers, CPU, memory), choices of compute cluster/cloud and Big Data platform, etc., as well as (b) the *workflow level*, such as throughput, latency, communication and state size (accounting for migration cost), CPU/memory/queue utilization, containers allocated, and so on.

Cost estimation. Currently, we support two cost estimators, one employing traditional, System-R like techniques and one using Bayesian Optimization (BO) inspired by [2]. For the latter, we use the statistics collected from micro-benchmarks and/or workflow execution to train a Gaussian Process Regression (GPR) model for the various operators and (parts of) workflows executed in different (Big Data platform, site) pairs.

The BO-based cost models are built following best practices based on our experience. One method is as follows. (a) We choose an acquisition function to prescribe which micro-benchmarks to run. A good choice is the Expected Improvement (EI) [2] that reduces fast the prediction uncertainty in the GPRs based on a small number of micro-benchmarks. (b) Then, we choose a uniform sample (e.g., 5%) of all possible micro-benchmarks to fit an initial GPR. (c) Next, we improve the GPR by executing 10% of all possible micro-benchmarks, each determined by the EI function. At workflow runtime, the GPRs of our cost model are improved incrementally with new statistics throughout the execution.

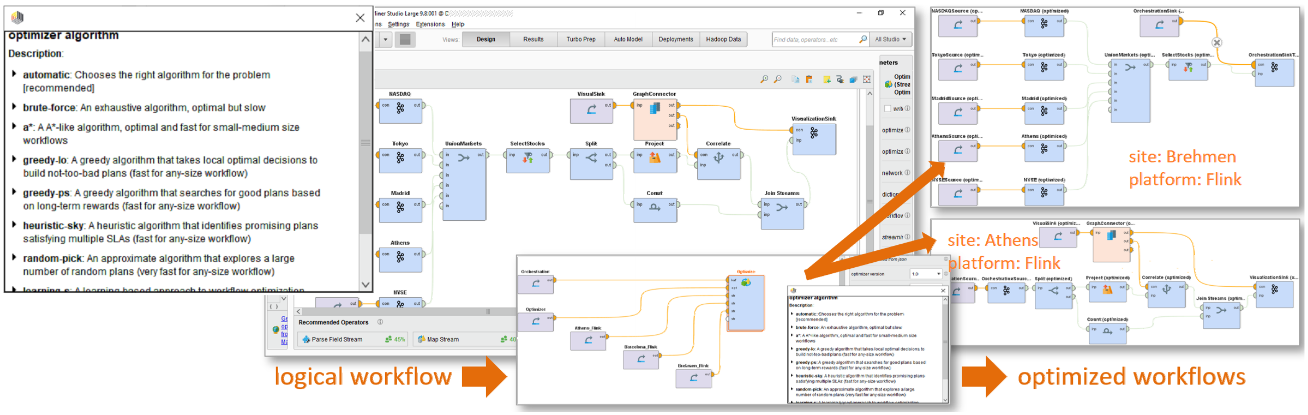
Optimization. The Optimizer solves a multi-criteria optimization problem with objectives involving throughput maximization, communication cost, processing and network latency, CPU, memory usage, and migration cost reduction. We maximize a weighted combination of these objectives under resource capacity and SLA constraints. We provide a suite of optimization algorithms (Figure 2) with operator-based and plan-based algorithmic families.

Operator-based. These algorithms start from upstream, operators of a topologically sorted workflow, compute a candidate physical design (e.g., site and platform assignment), and progressively build an execution plan by considering downstream operators: (a) Exhaustive search (ES): enumerates all (operator, platform, site) combinations for each operator. (b) A*-like search: our novel A* variation [8] employs heuristics to prune the search space using a priority queue and returns a graph (instead of a path) with physical operators. (c) Greedy search (GS): it keeps only the best physical instantiation for each operator before examining (operator, platform, site) configurations for its downstream ones. At runtime adaptation (which we discuss next), these algorithms execute from scratch.

Plan-based. These algorithms start with a good-enough execution plan placing operators close to data sources (e.g. Kafka topics). At runtime adaptation, they improve the current plan by exploring one (operator, platform, site) migration action at a time. Plan-based algorithms are inherently parallelizable, as migration actions can be examined in tandem, and work well with large workflows. (a) Hybrid BFS (hBFS): exhaustive BFS strategy similar to hybrid top-down, bottom-up BFS [4]. (b) Random-pick (RP): picks a random sampling of possible migration actions, inspired by QuickPick [22]. (c) Heuristic-sky (HS): examines one (operator, platform, site) migration action and if the resulted plan improves at least one performance dimension, it attempts more changes on that plan; otherwise, it skips the search towards this direction. (d) Greedy-lo & Greedy-ps (GLO & GPS): greedy ensembles that perform fast plan enumeration using counting and number base switch. GLO keeps the best plan per operator and attempts more (operator, platform, site) migration actions on each such plan, pruning the rest of the search space. GPS keeps only the best overall plan and attempts more migration actions on it.

The plan-based greedy ensembles work as follows. For a workflow with O operators that can be placed into P platforms and S sites (other dimensions can be added easily), the possible plans are $(P * S)^O$. To speed up the algorithm, we consider each operator as a vector that can take P*S values and construct an index into the O vectors as a O-digit, base-(P*S) number. Each digit of this number is an index into one of the vectors. Then, with a simple counting we can produce the respective plans. For example, for 4 operators and 4 possible values (e.g., 2 platforms and 2

[1] https://rapidminer.com/
[2] https://www.elastic.co/what-is/elk-stack


optimizer algorithm

Description:

- **automatic**: Chooses the right algorithm for the problem [recommended]
- **brute-force**: An exhaustive algorithm, optimal but slow
- **a***: A A*-like algorithm, optimal and fast for small-medium size workflows
- **greedy-lo**: A greedy algorithm that takes local optimal decisions to build not-too-bad plans (fast for any-size workflow)
- **greedy-ps**: A greedy algorithm that searches for good plans based on long-term rewards (fast for any-size workflow)
- **heuristic-sky**: A heuristic algorithm that identifies promising plans satisfying multiple SLAs (fast for any-size workflow)
- **random-pick**: An approximate algorithm that explores a large number of random plans (very fast for any-size workflow)

Figure 2: Optimizer algorithms (left), logical workflow (middle), connections to sites (bottom), optimized workflows (right)

sites) we count from 0 to $4^4 = 256$, and convert each number to 4 base-4 digits, and use these digits as indices into the operator vectors to get a plan. In this example, the possible actions are [P1|S1, P1|S2, P2|S1, P2|S2]. Hence, the number 164 maps to 2210, which represents the plan: O1(P2S1) -> O2(P2S1) -> O3(P1S2) -> O4(P1S1). This technique lends itself nicely to parallelism and enables efficient pruning of large chunks of the plan space.

The 'automatic' option in Figure 2 picks an algorithmic family and an algorithm based on the workflow size (#operators) and an application-defined cutoff execution time for the optimization process. Operator-based algorithms are preferred for small workflows (20-25 operators) and plan-based for larger ones. The GUI presents the algorithms from computationally expensive (more accurate) to faster ones (less accurate). Within each algorithmic family, an algorithm is faster than its preceding one by an order of magnitude due to more aggressive pruning.

Adaptation. Long-running streaming workflows may become suboptimal as their inputs change or their execution environment changes as other workflows are added or terminated. To maintain the offered service within the given SLAs, periodically we reevaluate problem workflows, i.e., workflows whose performance (e.g., latency, throughput, resource utilization) deteriorates beyond an acceptable margin. Then, we perform an adaptation event using either of the algorithmic families as described above. Preserving the state of the running workflows, we migrate running workflows on-the-fly to other sites or platforms with one of either two goals: improve the performance of the affected workflow or improve the performance of the entire workload. To do so, the optimizer may suggest a new plan to satisfy the new runtime conditions, balancing out the migration cost with the expected performance improvement, i.e., $cost\_new\_plan - cost\_current\_plan - migration\_cost$. These costs are weighted combinations of all performance dimensions. The migration cost combines communication cost and network latency due to state transfers to other site(s), dropped throughput and increased processing latency during job migration.

Our techniques are specifically designed for cross-platform stream processing because they (a) support runtime adaptation, (b) provide instantly plans for arbitrarily complex workflows and networks, with parallel optimization algorithms, and (c) employ BO-based cost models with GPRs for computationally inexpensive, incremental cost model updates. Such optimization features are crucial for AaaS with highly volatile statistical properties of streaming workloads.

3 OUR PRESENTATION

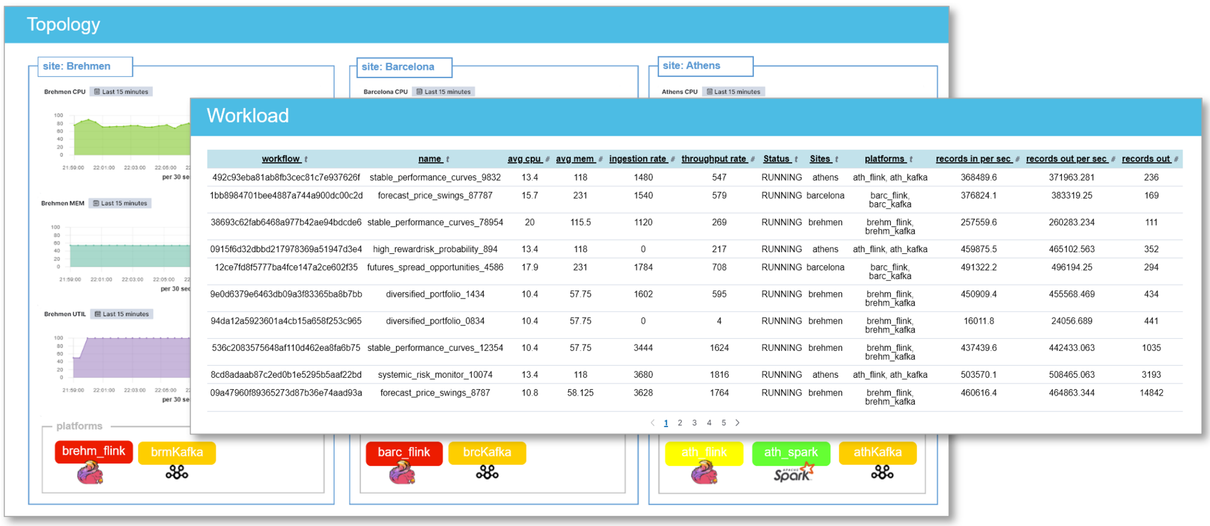
Our presentation script will be using the diversified portfolio maintenance scenario discussed in the introduction and further described below. We will provide a multitude of apriori designed streaming analytics workflows running concurrently comprising a demanding workload with varying characteristics.

SheerMP users will be able to graphically create and submit a streaming workflow. After submission, but before deploying the corresponding jobs, the users can iterate over this process and either examine the workflow as is or they can pick an optimization algorithms from SheerMP's suite and visually inspect the generated physical workflow. Finally, they will deploy the workflow for execution.

The users can then lively explore the initial resource allocation, workflow performance, as well as subsequent adaptation decisions for the workflow via the admin dashboard. Figure 3 illustrates example snapshots of SheerMP's admin dashboard. SheerMP practical benefits will be deduced via intuitive graphical representations of resource consumption, performance, and workflow adaptation in the admin dashboard. To evaluate the effectiveness of SheerMP, a separate dashboard will provide median, average L_1 and R^2 scores computed over the predicted values of the optimization objectives versus the actual ones, as workflows are deployed or adapted.

Workflow optimization. In the scenario of financial AaaS presented earlier, we use real Level 1 (stock trades), Level 2 (bids on stocks) data (~5000 stocks/~10 TB from various regional market data centers/clusters) provided by a European financial company. Figure 2(middle) presents an application of the diversified portfolio maintenance workflow. The workflow ingests Level 1 and Level 2 stock data from markets via a number of Kafka sources, unions market data and aims at discovering cross-correlations among pairs of stocks as well as identify the leaders (with more bids) in these pairs.

This workflow is particularly resource demanding as the straightforward way to detect cross-correlations results in a complexity that is quadratic to the number of ingested streams. Ingested market data are filtered to include any potential subset of stocks. Data streams are split with Level 2 data being directed to the bottom branch of the workflow where counts of bids per stock within a time window are computed. Simultaneously, at the top branch of the workflow, Level 1 stock data streams are directed to a Project operator which keeps only the timestamp and price attributes of each trade of a stock. Then, correlated stock pairs



workflow	name	avg cpu	avg mem	ingestion rate	throughput rate	Status	Sites	platforms	records in per sec	records out per sec	records out
492c93eba81ab8fb3cec81c7e937626f	stable_performance_curves_9832	13.4	118	1480	547	RUNNING	athens	ath_flink, ath_kafka	368489.6	371963.281	236
1bb8984701bee4887a744a900dc00c2d	forecast_price_swings_87787	15.7	231	1540	579	RUNNING	barcelona	barc_flink, barc_kafka	376824.1	383319.25	169
38693c62fab6468a977b42ae94bdcde6	stable_performance_curves_78954	20	115.5	1120	269	RUNNING	brehmen	brehm_flink, brehm_kafka	257559.6	260283.234	111
0915f6d32dbbd217978369a51947d3e4	high_rewardrisk_probability_894	13.4	118	0	217	RUNNING	athens	ath_flink, ath_kafka	459875.5	465102.563	352
12ce7fd8f5777ba4fce147a2ce602f35	futures_spread_opportunities_4586	17.9	231	1784	708	RUNNING	barcelona	barc_flink, barc_kafka	491322.2	496194.25	294
9e0d6379e6463db09a3f83365ba8b7bb	diversified_portfolio_1434	10.4	57.75	1602	595	RUNNING	brehmen	brehm_flink, brehm_kafka	450909.4	455568.469	434
94da12a5923601a4cb15a658f253c965	diversified_portfolio_0834	10.4	57.75	0	4	RUNNING	brehmen	brehm_flink, brehm_kafka	16011.8	24056.689	441
536c2083575648af110d462ea8fa6b75	stable_performance_curves_12354	10.4	57.75	3444	1624	RUNNING	brehmen	brehm_flink, brehm_kafka	437439.6	442433.063	1035
8cd8adaab87c2ed0b1e5295b5aaf22bd	systemic_risk_monitor_10074	13.4	118	3680	1816	RUNNING	athens	ath_flink, ath_kafka	503570.1	508465.063	3193
09a47960f89365273d87b36e74aad93a	forecast_price_swings_8787	10.8	58.125	3628	1764	RUNNING	brehmen	brehm_flink, brehm_kafka	460616.4	464863.344	14842

‹ 1 2 3 4 5 ›

Figure 3: Example snapshots showing the topology of a multi-site, multi-platform setting and a list of running workflows

(surpassing a given threshold) are computed and get joined with the respective count of bids per stock from the bottom branch of the workflow. Cross-correlations and leader stocks are directed to proper Kafka topics for visualization purposes.

This workflow presents several optimization opportunities. Figure 2 illustrates options of optimization algorithms (at the left) and available sites (middle-bottom) to execute the workflow. On the right, we see example results of flow optimization with the A*-like algorithm. In this case, the initial flow is split into two parts: the part handling incoming data traffic is sent to 'Brehmen Flink' site, and the computationally demanding correlation and join operations are sent to the 'Athens Flink' site.

Monitoring & Runtime Adaptation. Next, we will demonstrate the monitoring and runtime adaptation capabilities of SheerMP. We will present the system at normal operation where multiple streaming analytics workflows run concurrently. For ease of presentation, we will present a small number of 10-12 streaming workflows running, and we will monitor workload operations in the presence of events, such as (a) varying data rates, (b) long-running workflows hogging system resources, (c) addition/removal of workflows, and so on. We will then observe how SheerMP performs runtime adaptation to remediate ad hoc problems. Figure 3 shows example snapshots of one of our dashboards depicting site metrics (e.g., CPU, memory and queue utilization), platforms load, and workflow execution related metrics (e.g., resource allocation metrics, site and platform placement). In online mode, the dashboard shows workflows added or removed dynamically, or placed at different locations and platforms.

Experimental Highlights. The workflow of Figure 2 optimized by SheerMP across 3 clusters (each running Apache Flink with 32 task slots) achieves 2x to 10x higher throughput for our financial AaaS scenario's workload, compared to running it as a single job on one cluster. In the same scenario, our cost models, trained according to the best practices described in Section 2, can accurately capture trends (>80% R^2 score) across performance dimensions throughout the workflow execution.

Acknowledgments. This work has received funding from the EU Horizon 2020 research and innovation program INFORE under grant agreement No 825070.

REFERENCES

[1] D. Agrawal, S. Chawla, B.C. Rojas, and et al. 2018. RHEEM: Enabling Cross-Platform Data Processing - May The Big Data Be With You! -. *PVLDB* (2018).
[2] O. Alipourfard, H. Harry Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *NSDI*.
[3] A. Arvanitis, S. Babu, E. Chu, A. Popescu, A. Simitsis, and K. Wilkinson. 2019. Automated Performance Management for the Big Data Stack. In *CIDR*.
[4] S. Beamer, A. Buluç, K. Asanovic, and D. A. Patterson. 2013. Distributed Memory Breadth-First Search Revisited: Enabling Bottom-Up Search. In *IPDPSW*.
[5] V. Cardellini, F. L. Presti, M. Nardelli, and G. R. Russo. 2018. Decentralized self-adaptation for elastic Data Stream Processing. *Future Gener. Comput. Syst.* 87 (2018), 171–185.
[6] S. Chatterjee, M. Jagadeesan, W. Qin, and S. Idreos. 2021. Cosine: A Cloud-Cost Optimized Self-Designing Key-Value Storage Engine. *PVLDB* 15, 1 (2021).
[7] T. Das, Y. Zhong, I. Stoica, and S. Shenker. 2014. Adaptive Stream Processing using Dynamic Batch Sizing. In *SoCC*.
[8] A. Deligiannakis, N. Giatrakos, Y. Kotidis, V. Samoladas, and A. Simitsis. 2021. Extreme-Scale Interactive Cross-Platform Streaming Analytics - The INFORE Approach. In *SEA-Data@VLDB 2021*, Vol. 2929. 7–13.
[9] K. Doka, N. Papailiou, D. Tsoumakos, C. Mantas, and N. Koziris. 2015. IReS: Intelligent, Multi-Engine Resource Scheduler for Big Data Analytics Workflows. In *SIGMOD*.
[10] A. J. Elmore, J. Duggan, M. Stonebraker, and et al. 2015. A Demonstration of the BigDAWG Polystore System. *PVLDB* 8, 12 (2015).
[11] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy. 2017. Dhalion: Self-Regulating Stream Processing in Heron. *PVLDB* (2017), 1825–1836.
[12] T. Z. J. Fu et al. 2015. DRS: Dynamic Resource Scheduling for Real-Time Analytics over Fast Streams. In *ICDCS*. 411–420.
[13] I. Gog, M. Schwarzkopf, N. Crooks, M. P. Grosvenor, A. Clement, and S. Hand. 2015. Musketeer: all for one, one for all in data processing systems. In *EuroSys*.
[14] H. Herodotou, Y. Chen, and J. Lu. 2020. A Survey on Automatic Parameter Tuning for Big Data Processing Systems. *ACM Comput. Surv.* 53, 2 (2020).
[15] I. Konstantinou, E. Angelou, D. Tsoumakos, C. Boumpouka, N. Koziris, and S. Sioutas. 2012. TIRAMOLA: elastic nosql provisioning through a cloud management platform. In *SIGMOD*.
[16] A. Kontaxakis, N. Giatrakos, and A. Deligiannakis. 2020. A Synopses Data Engine for Interactive Extreme-Scale Analytics. In *CIKM*. 2085–2088.
[17] J. Meehan, S. Zdonik, S. Tian, Y. Tian, N. Tatbul, A. Dziedzic, and A. J. Elmore. 2016. Integrating real-time and batch processing in a polystore. In *HPEC*. 1–7.
[18] M. Petrova, N. Butakova, D. Nasonova, and M. Melnikа. 2018. Adaptive performance model for dynamic scaling Apache Spark Streaming. *Procedia Computer Science* 136 (2018), 109–117.
[19] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. 2012. Optimizing analytic data flows for multiple execution engines. In *SIGMOD*.
[20] A. Simitsis, K. Wilkinson, U. Dayal, and M. Hsu. 2013. HFMS: Managing the lifecycle and complexity of hybrid analytic data flows. In *ICDE*.
[21] S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M. J. Franklin, B. Recht, and I. Stoica. 2017. Drizzle: Fast and Adaptable Stream Processing at Scale. In *SoSP*.
[22] F. Waas and A. Pellenkoft. 2000. Join Order Selection - Good Enough Is Easy. In *BNCOD*, Vol. 1832. 51–67.
[23] J. Xu, Z. Chen, J. Tang, and S. Su. 2014. T-Storm: Traffic-Aware Online Scheduling in Storm. In *ICDCS*.