Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, Manisha Luthra. Zero-Shot Cost Models for Parallel Stream Processing. In Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM 2023), 5 pages, ACM press

Zero-Shot Cost Models for Parallel Stream Processing

Pratyush Agnihotri
Technische Universität Darmstadt

Boris Koldehofe
Technische Universität Ilmenau

Carsten Binnig
Technische Universität Darmstadt and DFKI

Manisha Luthra
Technische Universität Darmstadt and DFKI

ABSTRACT

This paper addresses the challenge of predicting the level of parallelism in distributed stream processing (DSP) systems, which are essential to deal with *different* high workload requirements of various industries such as e-commerce, online gaming, etc., where nowadays DSP systems are extensively used. Existing DSP systems rely on either manual tuning of parallelism degree or workload-driven learned models for tuning parallelism, which is inefficient and can lead to costly operator migrations and downtime when workload drifts occur. Thus, we argue for a learned model that can autonomously decide on the *right* parallelism degree while generalizing across workloads and meeting the current demands of DSP applications. We propose a novel approach that leverages zero-shot cost models to predict parallelism degree while generalizing across unseen streaming workloads *out-of-the-box*. We propose a rule-based strategy for the selection of parallelism degree and meaningful transferable features related to query workload and hardware that influence the parallelism decisions. We demonstrate the effectiveness of our system by reducing the required training dataset for parallelism prediction and achieving lower costs of parallel continuous query processing.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**.

KEYWORDS

Distributed stream processing, Zero-shot cost models, Parallelism prediction

ACM Reference Format:
Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, and Manisha Luthra. 2023. Zero-Shot Cost Models for Parallel Stream Processing. In *Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM '23), June 18, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3593078.3593934

1 INTRODUCTION

Why Parallel Stream Processing is Important? Distributed Stream Processing (DSP) enables the processing of large volumes of data in real-time while continuously delivering the result to the end-user. Nowadays, various industries such as online gaming (King) [2], e-commerce (Alibaba) [12], financial trading companies (Infront) [7], and satellite domain (ASRC Federal) [21] rely on a Distributed Stream Processing (DSP) systems for their core operations [6]. These industries need to continuously adapt the performance of the DSP system to meet the demands of workloads that are extremely *different* and very *high in scale*. For instance, *Alibaba* has to process more than 6 million transactions per second. Similarly, other industries like *King* (30 billion events per day), and *Infront* (24 billion events per day) need to keep accelerating their performance with increasing workload and to meet the Quality of Service (QoS) demands.

Another important challenge is that the query and data stream workload that these DSP systems deal with are very *different*. For instance, in Alibaba, customers' purchase history to provide personalized recommendations is different for each customer and thus would lead to different parallelism decisions of the recommendation query for each customer. Typically, DSP systems offer parallel processing by adding resources to process either a whole data stream or splitting it into sub-parts. For instance, Alibaba uses Blink [12], which adds resources to achieve parallelism with horizontal scaling. Still, adding more resources is costly, both monetary and leads to several operator migrations and downtime.

Research Question: This raises an important research question that we address: *how to predict additional resources for parallelism that can generalize across workloads?* Existing DSP systems like Apache Flink [3] and Heron [14] integrate parallelism decisions, commonly known as *parallelism degree*, as a manual knob to tune in the streaming query. Instead, we argue for a learned model that autonomously decides on the *right* parallelism degree with an ability to *generalize* across workloads and meet current demands of DSP applications. However, designing a DSP system for autonomous parallelism decisions is non-trivial because it depends on several dynamic and heterogeneous factors as follows: (1) Workload (data stream and query) characteristics such as event rate and selectivity of different kinds of operators and (2) Hardware characteristics of the resource type where an operator is placed, such as the number of cores. Thus, accurate modelling for parallel stream processing is a *key* challenge for DSP applications where workload changes (e.g., peaks in the data stream) are very common [13].

Existing Solutions and Their Pitfalls. Several works aim at addressing autonomous decisions for parallel stream processing by machine learning, e.g., using resource estimation [15, 18], parallelism [20], or stream partitioning prediction [17, 23]. Another predominant approach is by introducing elasticity, where resources are added or removed to deal with higher workloads [4, 9]. While these initial works have shown considerable performance using machine learning, they lack on either generalizing across workloads or are very costly as they do not efficiently scale both *horizontally* and *vertically*, leading to wasted resources or inferior performance [19].

An exciting direction that has shown promising initial results for unseen workloads are zero-shot models for databases [10, 11, 16, 22] and stream processing [8]. The main idea of zero-shot models is to (once) train on various representative workloads with so-called *transferable* features that enables the model to generalize. Further, the shared graph representation with the encoded transferable features allows training the zero-shot models for arbitrary queries. However, they are currently limited to the task of cost estimation for streaming queries, particularly latency and throughput prediction [8]. Thus, it is unclear how these zero-shot cost models can

Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, Manisha Luthra. Zero-Shot Cost Models for Parallel Stream Processing. In Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM 2023), 5 pages, ACM press

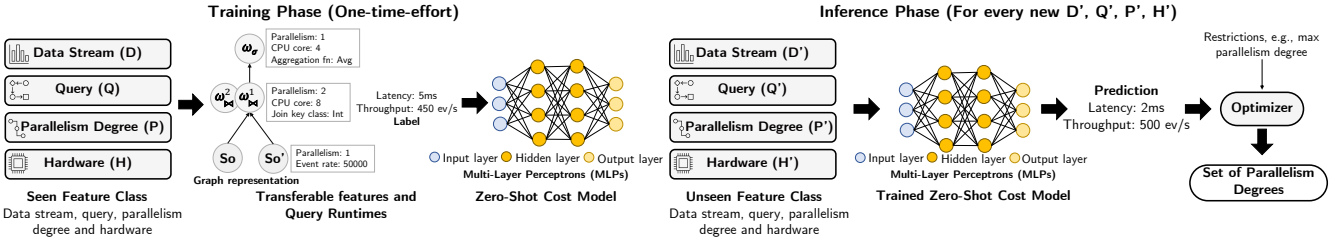

Figure 1: Overview of parallelism prediction in our system leveraging zero-shot cost model. For training (left figure), the zero-shot cost model is trained on the selected parallel query plans based on our *rule-based* strategy by executing and placing them on heterogeneous hardware. It is trained on selected *transferable* features encoded in a graph representation that enables the model to predict costs for unseen parallel streaming data configurations *out-of-the-box*. Thus, during inference (right figure), the trained model can provide accurate cost estimation for unseen parallelism degrees and different configurations of stream workloads and hardware without retraining or fine-tuning of the model. Finally, an optimizer is used to select a set of parallelism degrees with minimum cost estimates.

solve other optimization tasks of DSP systems, like predicting the parallelism degree of resources.

Parallelism Prediction by PANDA. In this short paper, we present an overview of PANDA that fulfils the aforementioned research gap by proposing a learned approach for parallelism prediction by leveraging zero-shot cost models [1, 8] while generalizing across unseen streaming workloads *out-of-the-box*. For this, we first need a large set of representative training data that covers a wide range of parallelism degrees for query plans on heterogeneous hardware. To find this representative dataset, we propose a *rule-based* strategy for parallelism degree selection that considers *simple* yet *effective* rules like estimated selectivity and event rates of operators to tune parallelism degrees during the training. This leads to a significant reduction of the training dataset, *half the training set* is sufficient using *rule-based* approach in contrast to random data collection for training zero-shot models for the task of parallelism prediction.

Second, we selected meaningful transferable features related to query workload and hardware that influences the parallelism decisions, such as the number of cores of hardware, which has a direct correlation to the parallelism degree that can be achieved on a given resource. Third, we encoded hardware into the graph representation that aids the model in joint parallelism and placement decisions so that the zero-shot cost models can learn about the parallelised performance on heterogeneous hardware. Finally, we trained the zero-shot cost model with the additional transferable features for parallelism decisions with two datasets collected using random and our rule-based strategy. Using the cost estimates in combination with an optimizer, we are able to find a set of parallelism degrees that leads to lower (estimated) costs of continuous query processing.

In the remaining sections, we first present an overview of our approach to tackle these challenges in Section 2 and present preliminary experiments in Section 3 and lastly conclude in Section 4.

2 SYSTEM OVERVIEW

In this section, we provide a high-level overview of PANDA system. We first explain the training and inference phase of zero-shot cost models for parallelism prediction and then explain the proposed rule-based strategy used to collect training data for the parallelism prediction task in Section 2.1 followed by an overview on the transferable feature and graph representation in Section 2.2.

Training phase. We leverage the zero-shot cost model proposed for DSP system [8] and train it for our task of parallelism prediction in a supervised way. For training, we propose a *rule-based* strategy where we deduce parallelism degrees based on intuitive rules such as setting a higher parallelism degree for higher event rates, as those lead to a reduced processing rate of the resources and hence they can benefit from higher parallelism. The specific rules devised to collect the training data are detailed in the section below (cf. Section 2.1) . Still, we collect a broad spectrum of queries with different parallelism degrees selected based on our devised rules and their respective costs (we consider the end-to-end latency of a query plan).

In this work, we restrict to *data-parallelism*[1] that means multiple instances of the same operator processes parts of datastream in a parallel way [18]. Additionally, we train the queries with different partitioning schemes depending on standard streaming operators such as hashing, rebalance and forward. Here, (1) the forward scheme simply forwards the datastream to the downstream operator when both upstream and downstream operators have an equal parallelism degree, (2) Rebalance distributes the tuples of the datastream in a round-robin fashion to the downstream operator instances, and (3) Hashing distributes the tuples based on a hash key when the parallelism degree of the downstream operator is greater than that of the upstream operator. Thus the model learns from a variety of distribution schemes of the tuples from the datastream for different parallelism degrees selected based on the rules. While this would seem like a huge effort, this is a one-time training where we limit it by systematic rules for collecting data for the parallelism prediction task. In contrast to workload-driven approaches [20] where a separate model needs to be trained with newly collected data this is way less – (only with 2500 queries the model performs very accurately, see Section 3).

For training data collection, the operators are deployed on heterogeneous resources using a fairness policy and parallelism degrees

[1]Another type is task-parallelism where different operators can be parallelised in sequence or in a pipeline fashion, but this is out of scope for this work.

Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, Manisha Luthra. Zero-Shot Cost Models for Parallel Stream Processing. In Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM 2023), 5 pages, ACM press

are collected with their corresponding costs (latency and throughput). Finally, the model is trained with a set of *transferable features* (cf. Table 1) encoded in a graph representation that signifies the streaming directed acyclic graph (DAG) with a mapping to hardware nodes. This transferable feature and graph representation together enables the zero-shot cost models to generalize across streaming workloads, hardware and parallelism degrees.

Inference and optimizer. The trained zero-shot cost model is used to derive costs for an unseen parallel query plan on a new hardware platform for a different parallelism degree *out-of-the-box*. A *key* enabler is the transferable feature and graph representation, where *transferable* means a feature that is independent of a specific streaming workload (data stream and query), e.g., tuple width of the data stream that can be specified independent of the underlying workload. An example of a non-transferable feature is a specific tuple attribute of a data stream such as "temperature values" that cannot be generalized across different streaming workloads. Furthermore, the ability to specify common query plans using the graph representation and the message passing across the graph and hardware nodes (multi-layer perceptrons (MLPs)) provides a means to transfer to unknown query plans. The cost prediction from the final layer of MLPs is used by the optimizer to select a set of parallelism degrees that leads to minimum costs (latency).

In the following, we detail on the rule-based strategy followed by the transferable feature and graph representation.

2.1 Rule-based Enumeration Strategy

Naively generating training data with different sets of parallelism degrees would result only for a single query type, millions of combinations, such that the model only sees different parallel query plans of a single query type. For instance, a parallel query plan with 9 operators (ω) that can be deployed on a single node with 8 cores (n_{core}) would lead to around 134 million parallel query plan combinations ($|n^{\omega}_{core}|$) to be enumerated (yet we do not deploy them on a cluster with 8 cores of node each). One way of enumeration that has previously shown promising results for cost estimates in zero-shot is a random enumeration [8]. However, the random selection of parallel query plans would result in noisy plans, e.g., randomly selecting higher parallelism degrees for downstream operators is less meaningful (except for joins), as we show later in the evaluation.

Therefore, we use predefined rules to systematically select and enumerate the *parallelism degree* (p) in a parallel streaming query plan based on the parameters that can be directly influenced by the parallelism ability of the hardware and thereby affect the performance. To be precise, we use estimated *selectivity* (sel), *incoming* (e_{in}) and *outgoing* (e_{out}) event rates of each operator in the dataflow graph to set the parallelism degrees. With these predetermined rules, we set meaningful parallelism degrees and enumerate parallel query plans that span a good range of resource utilization on a cluster of nodes so that the model learns meaningful costs. To better balance the trade-off between exploration and exploitation in the training data, we include estimated values of the above parameters, e.g., selectivity, such that the model also learns about "bad" parallel query plans (when the estimations are off).

Example: Assume we have two operators, upstream (ω_i) (predecessor) and downstream (ω_j) (successor in dataflow graph). To decide the parallelism degree of the downstream operator such as $\omega_j \in \{\omega_\sigma, \omega_\bowtie, \omega_\xi\}$, the selectivity and output rate of the upstream operator ω_i is computed. For instance, when a downstream operator is a window-aggregate ω_ξ followed by an upstream filter operator ω_σ, then a higher parallelism degree of ω_ξ does not make sense because it is followed by a filter operator that typically has a low selectivity. Therefore, we decrease the parallelism degree assignment of the window-aggregate ω_ξ operator by a factor depending on the selectivity of the upstream filter operator. To account for filter operators with high selectivity, the factor correspondingly leads to an increase in parallelism degree so the rules include those scenarios. Later, sel (ω_σ) is used to estimate outgoing event rate (e_{out}) for the next downstream operator (ω_j) in the parallel query plan and therefore set parallelism degrees of the next operators, respectively. By doing this selective assignment of parallelism degrees, we make the model aware of the different costs of parallel query plans by running only a few queries (as this is a very time-consuming process). Our hypothesis is that this rule-based strategy will enable the zero-shot cost model to predict costs for parallel query plans with a small number of training queries.

Node	Category	Feature	Description
Physical	hardware	CPU cores	Number of processing cores
	hardware	Node identifier	Unique identifier of every instance
	hardware	Total memory	Available memory of the node
	hardware	Network speed	Network link speed between nodes
	hardware	CPU frequency	CPU frequency on this instance
Logical	operator	Operator type	Type of operator
	operator	Parallelism degree	Parallel instances of the operator
	operator	Partitioning strategy	Strategy for data distribution (forward, rebalance, hashing)
	operator	Grouping identifier	Operators grouped together by DSP
	operator	Tuple width in	Incoming tuple width
	operator	Tuple width out	Outgoing tuple width
	operator	Selectivity	Ratio of output and input tuples
	operator	Input event rate	Incoming events on this instance
	operator	Output event rate	Outgoing events on this instance

Table 1: Selected transferable features for parallel query plans and hardware nodes for the parallelism prediction problem using zero-shot cost models.

2.2 Transferable Representation

There are three main ideas we extend the transferable representation of zero-shot cost models [8] for the task of parallelism prediction. (1) We include parallelism-specific (static) hardware features such as number of cores, total memory available, processing speed, etc. A selected list of transferable features for parallel query plans can be seen in Table 1. This is because the hardware properties related to the CPU directly correlate with the parallelisation capabilities of the node, this information is encoded as a transferable feature of a physical node. With this additional information, the model learns about the hardware capabilities of the node with higher parallelism degrees in relation to the runtime costs. (2) To encode parallelism degrees of an operator we consider two options *(i)* to have it as a transferable feature or *(ii)* to have a separate graph node for each new parallel operator instance of the parallel query plan. We select option *(i)* due to its simplicity and limited unique features of parallel graph nodes of an operator instance. We observe that for each new graph node of an operator instance, the transferable features that the model learns about the node are almost the same (if not entirely), which leads to duplicate nodes in the graph with redundant information, not to mention the added

Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, Manisha Luthra. Zero-Shot Cost Models for Parallel Stream Processing. In Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM 2023), 5 pages, ACM press

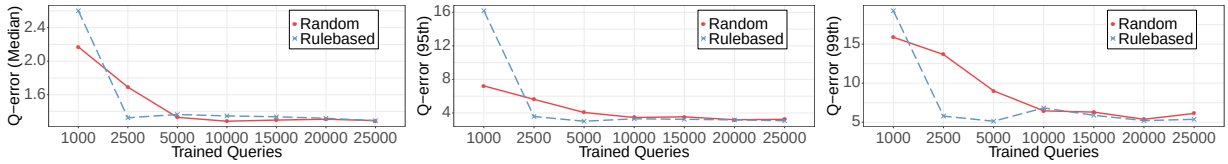

Figure 2: Trained queries required to achieve efficiency on the zero-shot model for different enumeration strategies

complexity of the model (with parallel nodes and edges). (3) We represent the hardware nodes separately in the graph representation with the mapping of the operators that denotes their deployment on the given hardware nodes. The physical graph deployment of the parallel query plans influences the overall cost that is observed, which is the reason this mapping needs to be learned by the model.

3 INITIAL EVALUATION

In this section, we show the efficacy of our *rule-based* strategy of training data collection and the zero-shot cost estimates for parallel query plans of DSP system. Before that, we explain our experiment setup and training data generation.

Experiment Setup and Metrics. We use a research testbed called *CloudLab* [5] to set up our Apache Flink cluster on varying hardware configurations with homogeneous and heterogeneous worker nodes as presented in Table 2 for data collection, training and inference. We used *Apache Flink* to deploy and manage the *Kubernetes cluster* for task placement using fairness policy and job distribution. We report *Q-error metric q(c, c')* that is defined as $max\left(\frac{c}{c'}, \frac{c'}{c}\right)$, with $q \geq 1$, which measures the relative deviation of the *true cost metric value c* (latency) and *its prediction c'* [8].

Clusters	Nodes	CPU	RAM (GB)	Disk (GB)	Intel Processor	CPU Speed (Ghz)
m510	270	8	64	64	Xeon D1548	2
rs620	48	16-20	128	100	Xeon E52667	2.2

Table 2: Combinations of hardware configuration used for data generation, training and inference

Training Data Generation. We implemented our parallel query plan generator on Apache Flink, a widely used DSP system. The plan generator enumerates parallel query plans using the *rule-based* strategy explained in Section 2.1 with different query characteristics such as event rates, operator types, etc. We execute the enumerated query plans on different cluster nodes shown in Table 2 with distinct hardware properties, e.g., cores from $8-20$, etc., with 8000 queries each for the query structure with in total 24000 queries used for train (80%), test (10%), and validation (10%) phases. This allows our zero-shot cost model to learn from different deployments of parallel query plans. In addition, we used different enumeration strategies *random* and *rule-based* strategies to enumerate parallelism degrees for comparison. In the following, we show how accurate are the zero-shot cost estimates using the transferable features selected for parallel query plans with random and rule-based strategies. Next, we compare the two enumeration strategies against different amounts of training queries.

Experiment 1: Accuracy of zero-shot cost estimates. In Table 3, we present the efficacy of our zero-shot model for three query

Enumeration Strategy	Query Structure	Latency	
		median	95th
Random	Linear	1.2302	2.8055
	2-way-join	1.3270	3.5349
	3-way-join	1.2630	2.7992
	Overall	1.2788	3.0608
Rule-based	Linear	1.2171	2.4263
	2-way-join	1.3601	3.4281
	3-way-join	1.3580	3.3730
	Overall	1.3042	3.2566

Table 3: Q-errors (median, and 95th) for unseen combinations of parallelism degree and workload properties.

structures: linear query comprising a filter and window-aggregate, 2-way-join comprising a window-join, window-aggregate and a filter, 3-way-join with 2 window-joins, window-aggregate and a filter as well as accuracy of the overall model (all query structures). Moreover, we compare the accuracy results for the two enumeration strategies, random and rule-based. A key observation from the results is that the cost estimates are very accurate for latency and overall, the model is able to provide good estimates for parallel query plans. The second observation is that the rule-based approach provides slightly better results in comparison to random, however, at this level the difference is not very prominent. To further investigate the effect, we provide results in the next experiment, where we observe the performance of the two strategies as we increase the number of trained queries.

Experiment 2: Efficiency of enumeration strategies with trained queries. In Figure 2, we analyze the accuracy of the model with the increasing amount of training queries to examine our hypothesis from Section 2.1. We observe with median, $95th$ and $99th$ percentile of q-errors for latency estimates that rule-based strategy converges soon with less amount of queries (2500) to better accuracy in comparison to the random strategy that needs at least more than 5000 queries to achieve better accuracies. Thus, we see that our rule-based strategy is able to show promising results even with less training data, which is typically the case in zero-shot cases. In our follow-up work, we want to investigate further the performance of our rule-based strategy for zero-shot cases where the model has not seen anything about the parallel query plan.

4 CONCLUSION

In this paper, we propose a zero-shot approach for parallelism prediction of streaming queries that can provide accurate cost estimates of unseen parallel query plans and in turn select optimal parallelism degrees. To train the zero-shot cost model, we propose a rule-based strategy that systematically enumerates parallel query plans. Moreover, we present *novel* transferable features that have an influence on the performance of parallel query plans. Our results show that the proposed zero-shot cost models lead to very accurate cost estimates for parallel query plans and the proposed rule-based

strategy is able to achieve very accurate results with a small amount of training data (2500 queries), which is quite promising for zero-shot data and tasks. In future, we want to further dive into these aspects and analyse the performance of the proposed model on more zero-shot data and tasks.

ACKNOWLEDGEMENT

This work is co-funded by the German Research Foundation (DFG) in the Collaborative Research Center (CRC) 1053 MAKI as well as DFKI Darmstadt.

REFERENCES

[1] Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, and Manisha Luthra. 2022. PANDA: Performance Prediction for Parallel and Dynamic Stream Processing. In *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*. Association for Computing Machinery, 180–181.

[2] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. 2017. State Management in Apache Flink®: Consistent Stateful Distributed Stream Processing. *Proc. VLDB Endow.* 10, 12 (2017), 1718–1729.

[3] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering* 38, 4 (2015).

[4] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. 2018. Decentralized self-adaptation for elastic data stream processing. *in FGCS* 87 (2018), 171–185.

[5] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 1–14.

[6] Data Flair. 2016. *Apache Flink Use Cases – Real life case studies of Apache Flink*. [Online; accessed 20-05-2022].

[7] Sebastian Frischbier, Mario Paic, Alexander Echler, and Christian Roth. 2019. Managing the Complexity of Processing Financial Data at Scale - An Experience Report. In *Complex Systems Design and Management*. Springer International Publishing, 14–26.

[8] Roman Heinrich, Manisha Luthra, Harald Kornmayer, and Carsten Binnig. 2022. Zero-Shot Cost Models for Distributed Stream Processing. In *DEBS*. Accepted for publication.

[9] Thomas Heinze, Zbigniew Jerzak, Gregor Hackenbroich, and Christof Fetzer. 2014. Latency-Aware Elastic Scaling for Distributed Data Stream Processing Systems. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*. Association for Computing Machinery, 13–22.

[10] Benjamin Hilprecht and Carsten Binnig. 2022. One Model to Rule them All: Towards Zero-Shot Learning for Databases. *In CIDR* (2022).

[11] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-Shot Cost Models for out-of-the-Box Learned Cost Prediction. *Proc. VLDB Endow.* 15, 11 (2022), 2361–2374.

[12] Xiaowei Jiang. 2021. Blink: How Alibaba Uses Apache Flink. https://www.ververica.com/blog/blink-flink-alibaba-search. [Online; accessed 27-05-2022].

[13] Alexandros Koliousis, Matthias Weidlich, Raul Castro Fernandez, Alexander L Wolf, Paolo Costa, and Peter Pietzuch. 2016. Saber: Window-based hybrid stream processing for heterogeneous architectures. In *SIGMOD*. 555–569.

[14] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. 2015. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of data*. 239–250.

[15] Xunyun Liu and Rajkumar Buyya. 2020. Resource management and scheduling in distributed stream processing systems: a taxonomy, review, and future directions. *in CSUR* 53 (2020), 1–41.

[16] Yao Lu, Srikanth Kandula, Arnd Christian König, and Surajit Chaudhuri. 2021. Pre-training Summarization Models of Structured Datasets for Cardinality Estimation. In *VLDB*. https://www.microsoft.com/en-us/research/publication/pre-training-summarization-models-of-structured-datasets-for-cardinality-estimation/

[17] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. 2015. Predictable low-latency event detection with parallel complex event processing. *in IEEE IoTJ* 2 (2015), 274–286.

[18] Henriette Röger and Ruben Mayer. 2019. A comprehensive survey on parallelization and elasticity in stream processing. *in CSUR* 52 (2019), 1–37.

[19] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. 2016. Big data analytics on Apache Spark. *International Journal of Data Science and Analytics* 1 (2016), 145–164.

[20] Jungeun Shin, Diana Arroyo, Asser Tantawi, Chen Wang, Alaa Youssef, and Rakesh Nagi. 2022. Cloud-Native Workflow Scheduling Using a Hybrid Priority Rule and Dynamic Task Parallelism. In *Proceedings of the 13th Symposium on Cloud Computing*. Association for Computing Machinery, 72–77.

[21] Eric Velte. 2021. Kafka Migration for Satellite Event Streaming Data. https://www.confluent.io/events/kafka-summit-americas-2021/kafka-migration-for-satellite-event-streaming-data/. [Online; accessed 10-03-2023].

[22] Ziniu Wu, Peilun Yang, Pei Yu, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2022. A Unified Transferable Model for ML-Enhanced DBMS. *Conference on Innovative Data Systems Research* (2022).

[23] Eleni Zapridou, Ioannis Mytilinis, and Anastasia Ailamaki. 2022. Dalton: Learned Partitioning for Distributed Data Streams. *Proc. VLDB Endowment* 16, 3 (2022), 491–504.