Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing

Pratyush Agnihotri[1], Boris Koldehofe[2], Roman Heinrich[1,3], Carsten Binnig[1,3], and Manisha Luthra[1,3]

[1] Technische Universität Darmstadt, Germany
[2] Technische Universität Ilmenau, Germany
[3] DFKI Darmstadt, Germany

Abstract. The paper introduces PDSP-Bench, a novel benchmarking system designed for a systematic understanding of performance of parallel stream processing in a distributed environment. Such an understanding is essential for determining how Stream Processing Systems (SPS) use operator parallelism and the available resources to process massive workloads of modern applications. Existing benchmarking systems focus on analyzing SPS using queries with sequential operator pipelines within a homogeneous centralized environment. Quite differently, PDSP-Bench emphasizes the aspects of parallel stream processing in a distributed heterogeneous environment and simultaneously allows the integration of machine learning models for SPS workloads. In our results, we benchmark a well-known SPS, Apache Flink, using parallel query structures derived from real-world applications and synthetic queries to show the capabilities of PDSP-Bench towards parallel stream processing. Moreover, we compare different learned cost models using generated SPS workloads on PDSP-Bench by showcasing their evaluations on model and training efficiency. We present key observations from our experiments using PDSP-Bench that highlight interesting trends given different query workloads, such as *non-linearity* and *paradoxical* effects of parallelism on the performance.

Keywords: Parallel and distributed stream processing · Benchmark

1 Introduction

Benchmarking parallel dataflows is important. Recent advancements in Stream Processing (SP) have introduced a variety of systems, such as Apache Flink and Storm for analyzing data streams in real-time. These systems are essential for many applications that handle immense data volumes. For instance, Netflix uses Flink to process over $1.3TB$ of data in its daily tasks, necessitating the use of many parallel operator instances to keep up with the high arrival rates and processing of data tuples [7]. For this, SP systems offer a *data flow* abstraction to specify operator parallelism in the query and provide data partitioning strategies to manage data stream partitions. While such *parallel data flows* have become an intrinsic part of every SP system, there is, however, very limited understanding of the performance of SPS under massively parallel dataflows.

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

Benchmark System	C1: P/S	C2: He/Ho	D/C	Infrastructure	C3: Learned SPS Models	Application Suite		Scalability
						Real-world	Synthetic	
Linear Road [4]	S	Ho	C	Single machine	No	1	-	No
YSB [18]	S	Ho	C	Single machine	No	1	-	No
StreamBench [43]	S	Ho	D	VMs	No	-	7	Partially
RIoTBench [52]	S	Ho	D	VMs	No	4	-	No
OSPBench [58]	S	Ho	D	Cloud AWS EC2	No	-	1	No
HiBench [31]	S	Ho	D	Local Cluster	No	-	4	No
BigDataBench [60]	S	Ho	D	Local Cluster	No	-	1	Partially
ESPBench [29]	S	Ho	D	VMs	No	5	-	No
SPBench [24]	P	Ho	C	VMs	No	4	-	Partially
DSPBench [13]	P	Ho	D	Azure Cloud Cluster	No	13	2	Partially
PDSP-Bench	**S/P**	**He/Ho**	**C/D**	**CloudLab, Geni Cluster, On-premise**	**Yes**	**14**	**9**	**Fully**

Table 1: Comparison of the existing benchmarking system for SP with PDSP-Bench emphasizing the research challenges. Our work can effectively benchmark both parallel data flow graphs and heterogeneous hardware as well as can be used as a benchmarking system for training ML models on SP workloads. Abbreviations used are S: Sequential plans, P: Parallel plans, He: Heterogeneous hardware, Ho: Homogeneous hardware, D: Distributed cluster, and C: Centralized or single machine.

Key challenges of existing work. Most of the existing benchmarking systems for SP are tailored towards the understanding of sequential dataflows [4,18,60,36]. Those benchmarking parallel dataflows [58,24,13,65] are restricted to a homogeneous environment for resource placement and offer limited capabilities in terms of scaling workloads, e.g., event rate and query parameters like window length. We believe a thorough analysis of parallel data flow graph placement on heterogeneous resources will reveal interesting insights into the behavior of distinct operators on various hardware resources and vice versa. Another unique aspect of our work is the ability to scale workload generation – both data streams and queries – by offering a benchmarking platform, which in fact can also be used for machine learning of SPS workloads, becoming increasingly important nowadays [27,64]. In summary, we identify three primary challenges for PDSP-Bench by analyzing key existing benchmarking systems for SP workloads presented in Table 1.

C1: Lack of expressiveness. Most existing benchmarks [18,4,60,31] often overlook the importance of benchmarking parallel dataflow applications, thus focusing only on sequential dataflows with a limited set of operators. For instance, StreamBench [43] overlooks essential operators, such as window functions, crucial for concurrent partitioning and efficient resource utilization.

C2: Shift to heterogeneity. The shift towards heterogeneous hardware requirement for benchmarking requires complex resource management, i.e., the underlying system must manage parallel resource mapping on varied hardware architectures, network links, and storage. Although benchmarking systems exist that assess parallel dataflows, like DSPBench [13] and SPBench [24], the benchmarks are restricted to homogeneous hardware reducing their relevance as real-world workloads often require heterogeneous environments [65]. For instance, Netflix runs on 1400+ nodes on 50+ distinct clusters with varied CPU cores [7] to deal with their demands of massively parallel dataflow applications.

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

C3: Integrating learned SPS models. Most importantly, the rapid advancement in SP mechanisms using machine learning (ML), necessitates a scalable and resource-friendly benchmarking system, ensuring its long-term relevance and utility in assessing future SPS with learned components. Recently, ML has been successfully applied for cost-based optimizations in SPS to support heterogeneous placements [27,26] and deciding parallelism strategies [64,2] and showed promising performance. This increasing surge of development in learned SPS models calls for a benchmarking platform that allows fair comparison between them by integrating the models and generating consistent training data for them. However, existing work do not provide a means to integrate ML models such that they can be compared in a "fair" way with consistent metrics (cf. Table 1, Learned SPS models).

Our proposal: We propose PDSP-Bench[4], a novel benchmarking system specifically designed to tackle the three primary challenges faced by existing benchmarks: lack of expressiveness in benchmarking parallel dataflows, the necessity for heterogeneous hardware support, and the integration of learned SPS models. Unlike existing benchmarks, PDSP-Bench enables the creation and evaluation of parallel query structures (PQP) across a diverse range of operators and input data streams, which we divide into synthetic and real-world workloads, thus offering an expressive and scalable solution. We also provide mechanisms to configure and manage heterogeneous hardware resources, by integrating resources from testbeds like CloudLab with different configurations, which are essential for accurately reflecting real-world deployment scenarios. Furthermore, PDSP-Bench facilitates the integration of learned SPS models, allowing for systematic training and evaluation of these models on diverse streaming workloads. This integration is increasingly important given the surge of use of ML for optimizing SPS performance [2,26,64]. The system's ability to generate large corpora of streaming datasets ensures that the ML models are trained on data representative of actual streaming workloads. (cf. Section 3)

The evaluation of PDSP-Bench involves extensive experiments that highlight its capabilities in benchmarking parallel stream processing. By using Apache Flink as the System Under Test (SUT), the evaluation demonstrates the impact of varying parallel query complexities, hardware configurations, and workload parameters on system performance. The results show the importance of considering both parallelism and heterogeneity to achieve optimal performance in real-time data processing applications. Moreover, we also integrate and train various learned cost models for streaming queries and showcase their performance in terms of model and training efficiency. (cf. Section 4)

2 PDSP-Bench: System Overview

The main goal of PDSP-Bench is to enable benchmarking of parallel and distributed stream processing (PDSP) systems considering heterogeneous environments for query deployment. As such, PDSP-Bench aims to enable the creation of large corpora of streaming datasets across three dimensions: *query*, *data* and

[4] Source code: `https://github.com/pratyushagnihotri/PDSPBench`

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

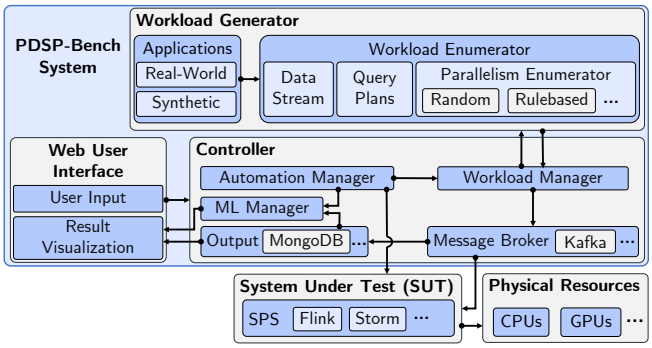


Fig. 1: PDSP-Bench system overview

resource diversity. Such large corpora of datasets can be used in training ML models for learning optimizations of SPS such as cost of executing streaming queries and their placement on heterogeneous hardware. We demonstrate this by training and evaluating learned cost models using the dataset generated by PDSP-Bench.

While PDSP-Bench supports both sequential and parallel query plans (PQPs)[5], we mainly focus on PQP to show our novel contributions to tackle challenges. PDSP-Bench has three main components: (1) workload generator, (2) controller and (3) web user interface (cf. Figure 1). System under Test (SUT) represents the underlying SPS like Apache Flink or Storm that are being evaluated by PDSP-Bench. We present an overview of our solution (**S#**) towards the goal and show how we address the aforementioned challenges (**C#**) of existing work using PDSP-Bench components as follows.

C1: Lack of expressiveness. S1: To specify PQP with a wide range of operators and input data streams, PDSP-Bench provides a core component known as workload generator as seen in Figure 1. The task of this component is to enumerate different factors of workload – both data and query – e.g., *parallelism degrees* to generate meaningful PQP to be executed on SUT, e.g., on Flink, hence enabling *query* and *data* diversity. These inputs on the enumeration can be given by the user via the web user interface that is managed by the controller as discussed later, but can be also configured directly into PDSP-Bench. A *key* issue we solve thereby is to generate PQP that are both *valid* and *representative* of current streaming applications. Thus PQP must represent both standard streaming and user-defined operators that we selected from open-source data stream processing datasets like DEBS Grand Challenges. We believe a combination of synthetic and real-world workloads is necessary to be able to properly assess SUT's performance and generate datasets that are representative for ML in streaming platforms. We discuss this component in Section 3.

C2: Shift to heterogeneity. S2: We provide interfaces to the users to configure hardware resources that are used in turn to execute the generated

[5] By PQP, we mean a given query structure with parallelism degrees that can generate multiple queries of this type of structure, e.g., linear PQP will generate a plan with parallel instances of filter operators with random filter literals.

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

PQP workload created by the former component. The controller and web user interface (WUI) components alleviate this complexity of configuring different hardware and hence enabling *resource* diversity for query execution and their deployment by automating it. We support the evaluation of heterogeneous CPU architectures such as Intel, AMD but also distinct network, memory and storage parameters by integrating CloudLab cluster, but other cloud providers can also be integrated easily. Thus, the complex mechanism of creation of machines and query deployment using hefty resource providers like Kubernetes and Yarn in SUT is hidden using these components.

C3: Integrating learned SPS models. S3: The entire benchmarking system design holistically guides the users to specify PQP and its properties as well as their execution on different hardware resources that in turn can used to generate data to train and evaluate ML models. For instance, such PQP execution data can be used as features together with the performance metrics as labels, such as end-to-end latency, on a given SUT to train a *cost model* that predicts those metrics. Moreover, controller component allows integration of different ML models to support training on different sizes of SP workloads. To evaluate models, we report metrics such as accuracy (q-error) and training overhead (queries and time) as well as investigate trade-offs between them.

As a solution, we present the PDSP-Bench workflow that shows how to use PDSP-Bench (cf. Figure 1) to generate streaming workloads that can be used to train ML models and in turn also be used to infer on a PQP from PDSP-Bench using the trained model. All the user inputs are collected using the WUI that are forwarded to the controller to orchestrate the benchmarking process. It allows users to select from existing applications in the suite (real-world or synthetic), but also provides a means to create novel applications in the form of PQP. Moreover, we provide other input parameters like parallelism enumeration strategies, workload and execution parameters such as event rate and query execution time (to limit the query as they are long-running) explained in the next section. We also allow to store the generated workload in a database, e.g., MongoDB that can be used for training ML models. Thus, ML Manager in the controller uses the "same" training data to train available ML models, e.g., a cost model can be trained to predict the costs of a PQP. This integrated approach allows "fair" comparison between ML models using our reported metrics such as training overhead. Thus, the reporting of benchmarks must also support training- and inference-related metrics and not just performance metrics. During the execution of PQP on the selected SUT, the performance as well as the training metrics can be visualized in real-time on WUI.

We will focus on the workload generator component that is core to the PDSP-Bench system in the following.

3 Workload Generator

An important research question we answer in this work is *"how to systematically generate workloads (data and query) for a comprehensive suite of dataflows to benchmark parallel and distributed streaming capabilities of SUT"*. The workload generator component plays a pivotal role in this question, as it generates

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

data streams and parallel query plans (PQP) derived using an enumerator (Section 3.1) for our integrated synthetic and real-world applications aiding to benchmark any given SUT (Section 3.2)[6]. While we generate workload by varying parameters related to data, query and resources given in Table 3, e.g., event rates of upto 4 million events per second and parallelism degrees upto 128, they are in practice limited by the amount of resources which are available (e.g., the CloudLab cluster nodes m510, c630 and c6525_25g). Thus the scale of workloads that PDSP-Bench can be much higher given the availability of the high amount of resources. In the following, we focus on how we diversify across these parameters.

3.1 Workload Enumerator

This component enables *data* and *query* diversity by orchestrating the generation and a variety of data streams and PQP as described in the following.

Data stream: For synthetic applications, a common strategy to generate synthetic data is to *randomly* select from a given valid data range to avoid exhaustive enumeration of the given parameters, which is extremely time consuming and practically impossible to do within a reasonable timeframe. In fact, *domain randomization* is a common technique used for synthetic data generation to train ML models like deep neural networks such that it learns from the features of interest. The rationale behind this method is to have variability in the data so significant that the models trained on this data could generalize to the real-world data with no additional training [56]. Thus, to address this PDSP-Bench includes a method for generating synthetic data streams by randomly varying over (1) tuple width (# data items in a single tuple of a data stream), (2) its data types (the data type per data item), and (3) event rates (# event tuples produced per time unit) crucial both for rigorously testing SUT's capabilities and collecting meaningful training data for ML models (cf. Table 3 for data ranges). While we generate data using the defined range, these values are highly configurable in the PDSP-Bench. To enable data streams from real-world applications, we use `kafka` as a data producer that is connected via PDSP-Bench to the SUT. We *repeat* the data stream read from the source to mimic infinite data streams.

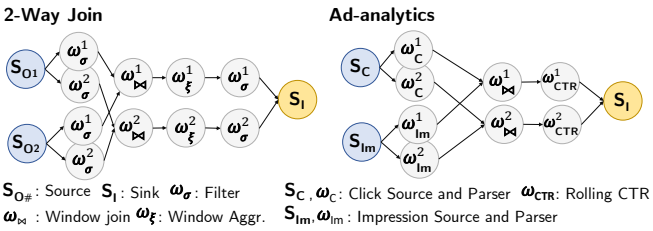


Fig. 2: Example of PQP: synthetic 2-way join and real-world ad analytics application.

[6] Selected data: `https://github.com/pratyushagnihotri/pdsp-bench_experiment_data`

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

Query: For synthetic query plans, we offer an extensive range of PQP from an array of query structures, including simple linear queries with one filter to complex configurations involving multi-way joins and multiple chained filters. To give an example representation of such a 2-way join see Figure 2 (left). Moreover, we randomly enumerate over multiple parameters of these query structures such as filter function (e.g., $<$, $\leq$), its data type (the filter value's data type), window type (e.g., sliding, tumbling), window policy (e.g., time, count), etc., to generate queries that again can be used to evaluate a given SUT and is representative of workloads required to train a ML model. While we generate these query parameters randomly, an important question arises how we balance query properties, e.g., selectivity. For instance, random selection of filter literals may result that data never passes the generated filter. To avoid this, we use selectivity estimation methods [2] to estimate selectivity of given filter operator such that queries with only valid literals are generated where sel_{ω_σ} is not 0. For real-world queries, we enable users to choose from the given range of applications available in our benchmark suite but also use them as a basis PQP to generate more queries. For instance, consider the ad analytics application in Figure 2 (right) where the users can choose to generate more queries by adding more filter operators, choosing a different window count for the join, etc. This way, we allow users to execute given applications but also generate queries to evaluate SUT capabilities for dynamics and uncertainty inherent to real-world applications. Moreover, this flexibility allows to generation of representative PQP aligned to the real world to train ML models. The full list of query applications is described in Table 2 and explained in the next section. Other data ranges to configure query plans available in PDSP-Bench can be seen in Table 3.

Parallelism enumerator: While random enumeration is meaningful as it represents real-world data ranges for the parameters discussed so far, e.g., tuple widths for data stream and window length for operators, we note that random enumeration of parallelism degrees for operators is not the same. The random selection of parallelism degrees in PQP will result in very noisy queries or even invalid queries, e.g., selecting higher parallelism degrees for downstream operators is less meaningful since there are anyways less tuples that have to be processed as tuples move down in the data flow graph (e.g., after filter operator). Moreover, random selection of parallelism degrees, e.g., $\omega_\sigma = 1$, $\omega_\bowtie = 10$ in the 2-way join example query in Figure 2 leads to a plan that is very bad in performance because it first limits processing capabilities by selecting only one instance of filter and hence there is limited use of 10 instances of join operators and highly wasteful of resources. While such bad plans still might be interesting for benchmarking SUT to cover corner cases, learning ML models with such bad plans is not meaningful as they are not encountered in real-world. Thus, we employ different strategies for parallelism degree enumeration in PDSP-Bench that can be selected by the user depending on the needs. For instance, we provide `Rule-based` strategy that selects *meaningful* parallelism degrees for PQP derived based on literature [35] but also random enumeration as defined below.

`Random` selects a parallelism degree randomly within the given range, usually upto maximum number of cores available on physical resources, introducing

variability for comprehensive performance assessment of SUT. `Rule-based` goes beyond randomness and selects parallelism based on workload characteristics and physical resources. It considers factors such as event rates, operator selectivity, and the number of cores, enabling a more targeted enumeration of parallelism for upstream and downstream operators. This approach seeks to optimize performance by aligning parallelism with the specific demands and capacities of the system [35]. `Exhaustive` aims to test every unique combination of parallelism degrees, ensuring that each combination is tested. `MinAvgMax` cycles through generating queries with minimum, average, and maximum numbers of parallelism degrees, systematically exploring the effects of varying parallelism degrees on system performance, from least to most intensive use of resources. `Increasing` evaluates the impact of incremental change in parallelism, starting at the minimum degree and increasing stepwise to the maximum for each operator up the dataflow graph. `Parameter-based` is designed for rapid testing, as it configures parallelism based on user input.

3.2 Applications

We include a selection of applications in the PDSP-Bench benchmarking suite by analyzing previous research works in databases [18,4] and stream processing [13,29,36,24]. The applications are chosen based on a set of criterion that capture the diversity of streaming workloads, including data sources' tuple width, the data items' type, as well as the different operators and their complexity, e.g., standard SPS and user-defined operators in data flow graphs. To thoroughly assess PDSP in heterogeneous environments, we classified these applications into real-world and synthetic categories. This approach ensures a detailed evaluation of the SUT capabilities, with a special emphasis on its performance (latency and throughput) but also readiness for future demands across various conditions from typical to peak usage scenarios for ML integration.

The real-world applications reflect genuine data streams, such as *social media feeds*, *financial transactions* and *IoT sensor data*, which are crucial for mimicking actual system loads and behaviors for the benchmarking process as presented in Table 2. For instance, DEBS 2014 Smart Grid data [52] serves as a real-world benchmark, reflecting energy usage patterns from smart plugs. On the other hand, synthetic applications also represent real-world scenarios by including standard SPS operators like filters, aggregates and joins, but the data streams are generated artificially, allowing to stress test SUT under hypothetical future scenarios with high data volumes. This dual approach ensures a balanced assessment of SUT's performance, scalability, and adaptability, preparing it for current and future data processing challenges. We enlist all the (both synthetic and real-world) applications included in PDSP-Bench in Table 3. While we provide the applications described above, PDSP-Bench can be easily extended by integrating new jobs from other benchmarks like YSB [18] and Nexmark [57].

4 Evaluation

In this section, we present an extensive evaluation using PDSP-Bench. Due to many workloads both synthetic and real-world and varying parameters in

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

Applications	Area	Description
Word Count (WC) [38]	Text Processing	Processes a text stream, tokenizes sentences into words, and counts the occurrences of each word in real-time using a key-based aggregation.
Machine Outlier (MO) [34]	Network Monitoring	Detects network anomalies in machine usage data streams using the *BFPRT algorithm* [63] to identify outliers based on statistical medians.
Linear Road (LR) [4]	Traffic Management	Processes vehicle-generated location data through four queries: *toll notification*, *accident notification*, *daily expenditure*, and *total travel time*, to calculate charges or detect incidents.
Logs Processing (LP) [54]	Web Analytics	Processes HTTP Web Server log data to extract insights using two queries: (i) counts visits within specified intervals, and (ii) tallies status codes.
Google Cloud Monitoring (GCM) [37]	Cloud Infrastructure	Analyzes cloud computing data by calculating average CPU usage over time, either grouped by job or category, with results processed through sliding windows and specific grouping operators.
TPC-H (TPCH) [10]	E-commerce	Processes a stream of order events to emit high-priority orders, utilizing operators to structure, filter, and calculate the occurrence sums of order priorities within specified time windows.
Bargain Index (BI) [6]	Finance	Analyzes stock quotes streams to identify bargains by calculating the price-to-volume ratio against a threshold using *VWAP* and *Bargain Index* Calculators, emitting qualifying quotes.
Sentiment Analysis (SA) [21]	Social Network	Determines the emotional tone of tweets by assessing sentiment using TwitterAnalyzer and SentimentClassifier operators, which apply Basic or LingPipe classifiers to score and label the tweets.
Smart Grid (SG) [20]	Sensor Network	Analyzes smart home energy usage through two queries that calculate global and local average loads using sliding window.
Click Analytics (CA) [43]	Web Analytics	Analyzes user interactions with online content through two queries: grouping click events by Client ID for repeat and total visits per URL, and identifying geographical origins using a Geo-IP database.
Spike Detection (SD) [53]	Sensor Network	Processes sensor data streams from a production plant to detect sudden temperature spikes by calculating average temperatures over sliding windows, and identify spikes exceeding 3% of the average.
Trending Topics (TT) [45]	Social Network	Processes stream of tweets using the TwitterParser and TopicExtractor operators to identify trending topics on Twitter based on aggregated popular topics based on predefined thresholds.
Traffic Monitoring (TM) [41]	Sensor Network	Processes streaming vehicle data using TrafficEventParser and RoadMatcher operators to match vehicle locations to road segments then calculates average speed per segment using the AverageSpeedCalculator.
Ad Analytics (AD) [47]	Advertising	Processes real-time data on user engagement with digital ads by parsing clicks and impressions, calculating their counts within time windows, and computing the click-through rate (CTR) with a rolling CTR operator.
Synthetic Queries	Standard DSP Queries	Assess standard streaming workloads by randomly generating diverse data streams and query structures with increasing complexity. It supports various data types and standard operators like filter, window aggregate, window join, and groupby to evaluate streaming operators through synthetic query structures, from simple linear to complex multi-join queries.

Table 2: Benchmarked parallel query structures based on synthetic and real-world applications (based on [32,13,29,24,65].

)

PDSP-Bench, we select interesting observations (**O#**) from our evaluation for the following evaluation questions.

Exp. 1: Impact of PQP complexity on performance. How increasing parallel query complexity such as number and type of operators as well as parallelism degree can influence performance?

Exp. 2: Impact of heterogeneous hardware on performance. How heterogeneous hardware can impact the execution of PQP?

Exp. 3: Integration of ML models in PDSP-Bench.(1) How different learned cost models perform on various PQP? (2) What is the influence of parallelism enumeration strategies on training efficiency?

Environment Setup and Implementation. We use the *Cloudlab research testbed* [22] to perform all our experiments as it provides the necessary distributed infrastructure (cf. Table 4) for configuring and deploying an SUT cluster, enabling us to benchmark using PDSP-Bench. For this initial evaluation, we select *Apache Flink v1.16.1* as SUT, however this can be exchanged by any SPS. In addition, PDSP-Bench uses an *Apache Kafka* on a separate machine from SUT to produce data at different event rates for the various applications. PQP from different query structures ($\approx 30k$ PQPs) are executed three times for 3 minutes each on clusters of 10 nodes (cf. Table 4) and correspond-

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

ing performance metrics are collected and stored locally as well as in *MongoDB* database. Furthermore, the *controller* is implemented in *Django* and *WUI* is developed with *Vue.js* to take users' input, such as cluster setup, SUT deployment and PQP as described in Section 2.

Metrics. For experiments 1 and 2, we focus on *end-to-end latency*, though PDSP-Bench can be used to measure other performance metrics depending upon SUT benchmarking requirements. *End-to-end latency* represents the time interval starting from the production of the first data tuple from the data source until when the output of the query result is delivered to the data sink. It is the sum of the processing latency of each operator (including window time) in the processing pipeline, the network latency of data transmission from a data source within operators to the data sink, and the input and output latency of reading and writing data to and from external systems like IoT data sources. Given that operators might be distributed across different locations based on the CloudLab resources, network latency is a significant factor. [2,44] We report the mean of three runs of measuring median latency (*50th percentile*).

In experiments 3 and 4, we report Q-error q(c, c') which is a well-known metric to measure the accuracy of ML models [39]. In the context of learned cost models it gives relative deviation of the true cost c (latency) with its prediction c'. In addition, we compare the proposed enumeration strategies to generate parallelism degrees (cf. Section 3) in terms of accuracy with Q-error and training time of the models using these strategies.

Diversity	Parameters	Parameter Data Range
Query	Real-world query structures	Refer Table 2 for full list
	Synthetic query structures	Linear, 2-chained filter, 3-chained filter, 4-chained filter, 2-way join, 3-way join, 4-way join, 5-way join, 6-way join
	Parallelism degree categories	$1 \leq$ XS < 8, $8 \leq$ S < 16, $16 \leq$ M < 32, $32 \leq$ L < 64, $64 \leq$ XL < 128, $128 \leq$ XXL
	Window duration (ms)	50, 100, 150, 200, 250, 325, 750, 1k, 1.5k, 2k, 2.5k, 3k, 4k, 5k, 6k, 7k, 8k, 9k, 10k
	Window length (tuples)	2, 3, 4, 5, 7, 10, 17, 25, 37, 50, 62, 75, 82, 100, 150, 200, 250, 300, 350, 400
	Sliding length (ratio)	[0.3, 0.4, 0.5, 0.6, 0.7] x Window length
	Window types and policy	type: sliding and tumbling, policy: count and time-based
	Window aggr. functions	min, max, avg, mean, sum
	Join and filter data types	string, integer, double
	Filter functions	$\leq, \geq, \neq, =, <, >$, startsWith, endsWith, endsNotWith, startsNotWith
Data	Tuple width and data type	[1 - 15] x [str., doubles, int]
	Event rate (events/sec)	10, 100, 1k, 5k, 10k, 50k, 100k, 200k, 500k, 1mn, 2mn, 4mn
	Partitioning strategy	Strategy for data distribution (forward, rebalance, hashing)
Resource	Cluster type	Homogeneous: m510, Heterogeneous: c6320, c6525_25g
ML models	Learned cost models	Linear regression (LR) [23], Multi-layer perceptron (MLP) [30], Random forest (RF) [16], Graph neural networks (GNN) [62,2,26]

Table 3: PDSP-Bench benchmark parameters for SUT (order by complexity).

Cluster Type	Clusters	Node	CPU	RAM (GB)	Disk (GB)	Processor	Speed (Ghz)	Network Link Speed
Ho	m510	10	8	64	256	Xeon D	2	10 Gbps
He	c6525_25g	10	16	128	480	AMD EPYC	2.2	
He	c6320	10	28	256	1024	Haswell	2.0	

Table 4: Cluster of resources utilized on CloudLab tested to perform benchmarking to SUT. Here, "Ho" and "He" are homogeneous and heterogeneous

Evaluation Parameters. Table 3 outlines evaluation parameter range evaluated by PDSP-Bench for data stream, PQP and resources (cf. Section 3.1). Table 4 presents the used hardware configuration from CloudLab testbed. Although we evaluate different event rates, we present results on the highest event

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

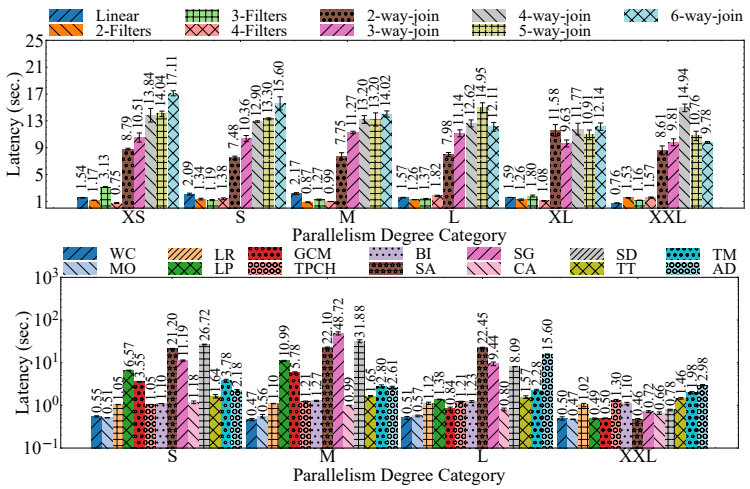


Fig. 3: Impact of parallelism degree on PQP performance from synthetic (top) and real-world applications (bottom). The analysis shows distinct performance behaviors, highlighting the benefits of parallelism in improving end-to-end latency and the need to consider query and operator characteristics in PDSP environments. Results indicate a complex interplay between standard operations and UDOs, with *paradoxical* effects and *non-linear* performance trends related to parallelism degree. (Note: Bottom figure omits XS and XL; their performance mirrors S and L)

rate of $4mn$ events/sec. unless otherwise specified as intuitively higher scale of events will benefit from parallelism. Thus, we consider the occurence of a number of events over a fixed interval of time, so data is modelled as poisson distributed since many real-world applications, e.g., network traffic, sensor networks, etc., are poisson distributed. However, in PDSP-Bench we can also model other common data distributions such as zipf.

4.1 Exp. 1: Impact of PQP complexity

We benchmark Flink using two PQP categories: a) synthetic, primarily with standard SPS operators (cf. Figure 3 *top*) and b) real-world, combining standard SPS and user-defined operators (UDOs) (cf. Figure 3 *bottom*). We select homogeneous resources of `m510` cluster (with 10 nodes) to analyze *only* parallelism degree diversity. Here, the complexity of a PQP correlates both the composition of various operators and the parallelism degree applied to execute them. For instance, *linear* parallel query structure, with a single source, multiple filters, and window aggregation without joins, has simpler data flow and lower computational demands, leading to reduced end-to-end latency. However, the presence of dual filters introduces computational requirements that can affect latency based on the data volume and filter complexity. In contrast, multi-way joins significantly increase complexity and computational overhead, necessitating effective parallel processing to manage latency implications. We present key findings from this analysis.

O1- *Increasing parallelism can speed-up multi-way join queries.* We observe an interesting trend in Figure 3 (top) when parallel query structures transitioned

from linear query to more chained filters and joins. Initially, adding filters keeps latency consistent across parallelism categories *XS to XXL*. However, introducing join operators leads to a tipping point where latency increases linearly due to the complexity of coordinating joins across distributed datasets. At the same time, the parallel instances help in handling workload and reducing latency with increasing complexity. A similar trend is observed in real-world applications Figure 3 (bottom). PQP with standard SP operators such as *WC*, *LR* show consistent performance, while those with data-intensive UDOs such as *SA, SG, SD* show significant performance improvement with increasing parallelism. This highlights that parallelism benefits PQP with data-intensive operators more than those with less data-intensive operators.

O2*- Increasing parallelism can speed-up queries, but not in all cases.* While increasing parallelism generally improves performance by distributing the workload, for some PQP, there is a paradoxical effect on end-to-end latency as PQP complexity increases. Beyond a certain threshold of parallelism ($32 \leq L < 64$), particularly with multiple joins, the overhead of managing parallel operations, such as data shuffling and synchronization, outweighs the benefits, leading to increased latency. This is similar to adding more lanes in city traffic, which can cause congestion at merge points. Thus, performance improvements in multi-way joins are small or negligible as parallelism increases from L to XL. In contrast, PQP from real-world applications show performance benefits at extremely high parallelism. For instance, $32 \leq L < 64$ parallelism significantly improves latency in `SG` and `SD`, while `AD` shows negligible performance gains even beyond *128*, reflecting the complexity of operators in the query.

O3*- Queries with UDOs shows unpredictable performance.* We also investigate the performance intricacies of standard SP operators compared to UDOs.Our findings reveal distinct scalability and computational overhead that underline the relationship between operator types and parallel processing efficiency. Standard SP operators exhibit predictable scalability due to their well-defined semantics. For instance, a *flatMap* operation in a *WC* application scales almost linearly with increased parallelism, requiring no complex state management. Conversely, UDOs, which embed custom logic, show variable scalability due to state handling and coordination needs. In the `AD` application, custom aggregation and joining logic on a sliding window result in non-linear scaling, where increased parallelism leads to higher overhead, sometimes degrading performance.

O4*- The non-linear effect of parallelism on latency:* PDSP-Bench provides a critical insight into the relationship between parallelism degree of PQP and performance is non-linear. As the parallelism of standard SP operators in a query increases, the performance does not linearly increase as not necessarily each PQP advantages from parallelism. This results from the previous observations O1 - O3 because of different factors like complexity of operators and paradox of parallelism. For instance, *SA, SG, SD* has high latency for parallelism categories S and M which starts improving at level L, with significant improvements when parallelism exceeds 128.

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

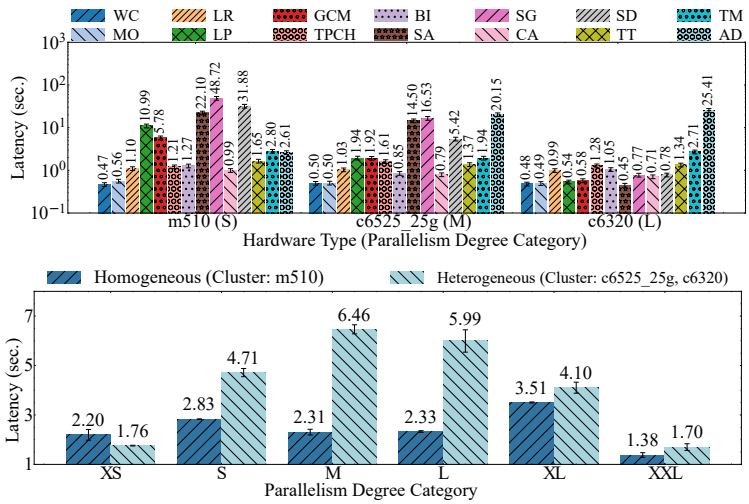


Fig. 4: Impact of heterogeneous hardware on performance for PDSP, with varying parallelism degree and resource processing capabilities on real-world (top) and synthetic (bottom) applications. Evaluation shows that parallel processing benefits from hardware diversity but requires understanding hardware characteristics and workload distribution to optimize performance and avoid of pitfalls such as diversity dilemma.

4.2 Exp. 2: Impact of heterogeneous hardware on performance

Next, we evaluate the impact of homogeneous (`m510`) and heterogeneous hardware (`c6525_25g,c6320`) clusters on performance for PDSP and executed PQP across clusters of 10 nodes each. Figure 4 (top) represents the mean *end-to-end latency* of PQP with parallelism degree category as per # cores on hardware of each cluster to analyze the performance of real-world applications. For instance, `m510` cluster has hardware with 8 cores, so selected PQP with S parallelism degree category. Similarly, PQP with parallelism degree categories M and L as clusters `c6525_25g` and `c6320` have hardwares with 16 and 28 cores, respectively. Figure 4 (bottom) shows the mean *end-to-end latency* across different parallelism categories of PQP for three types of clusters for synthetic applications.

O5*- Powerful heterogeneous environment does not necessarily accelerate queries.* By evaluating PDSP across diverse hardware configurations, we encounter the *diversity dilemma* where the theoretical advantages of heterogeneous environments sometimes clash with practical performance outcomes in Figure 4 (top). We anticipate that the diversity in computational processing capabilities would universally accelerate parallel processing and enhance performance. However, we notice that while applications *SA*, *CA*, *SD* significantly benefited, showing exponential decrease in latency. On the other hand, *AD* struggles to improve the performance in heterogeneous configuration due to the complexity of UDOs coupled with the communication overhead across different instances. This finding suggests that the theoretical benefits of hardware diversity require careful orchestration for workload distribution and resource management strategies to leverage heterogeneous environments effectively.

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

***O6**- Finding optimal parallelism for queries is non-trivial.* We also evaluate parallel processing capabilities of synthetic PQP on various hardware clusters as shown in Figure 4 (bottom). We notice no consistent balancing point of parallelism exists for all workloads. Until this point, increased parallelism might not yield further benefits and could even hinder performance due to higher communication and synchronization overhead. Finding this point in a heterogeneous environment might be even more challenging. In homogeneous clusters, latency generally increases with parallelism from XS to XL, then decreases at XXL. A similar trend is observed in heterogeneous clusters, with latency highest at M and decreasing for L, XL, and XXL. This suggests heterogeneous clusters benefit more from increased parallelism due to diverse computational capabilities. Initial higher latency at M may result from workload distribution imbalance, improving as parallelism grows and better utilizing diverse resources.

***O7**- Homogeneous or heterogeneous clusters: There is no clear choice for all cases.* PDSP-Bench provides insight about notable performance enhancement with increasing parallelism and diverse hardware configurations as presented in Figure 4 (top). Specifically, PQP from real-world applications benefit significantly from increasing parallelism and hardware processing capability, leading to improved performance. Conversely, PQP from synthetic applications demonstrate better performance on homogeneous clusters than heterogeneous ones, due to the efficient handling of standard SP operators. Conversely, in heterogeneous clusters, despite high computational potential, challenges arise from uneven workload distribution and increased communication overheads due to varying speeds across different hardware units.

4.3 Exp. 3: Integration of ML models in PDSP-Bench

(1) Performance of learned cost models. The `ML Manager` (cf. Section 2) of PDSP-Bench offers benchmarking of performance of various ML models tailored for parallel and distributed stream processing environments. It uses data collected during benchmarking as labeled datasets for training and inference. In this evaluation, we focus on assessing the effectiveness of different learned cost models in predicting performance of streaming queries such as end to end latency.

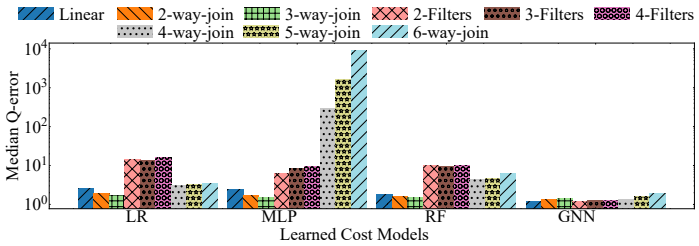
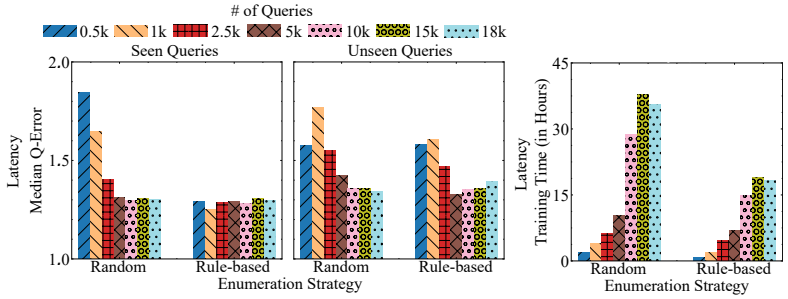


Fig. 5: Comparison of accuracy of various ML models: Linear regression (LR) [23], Multi-layer perceptron (MLP) [30], Random forest (RF) [16], Graph neural networks (GNN) [62,2,26] for various parallel query structures.



(a) Amount of queries **(b)** Training time

Fig. 6: Comparison of accuracy enumeration strategies. The rule-based strategy achieves better accuracy with (a) 2.5k queries compared to the random strategy. It converges in (b) 4.5 hours, three times faster than the random approach.

We integrate four distinct ML models architectures into the `ML Manager`: (1) Linear regression (LR) [23]: traditionally used for its simplicity and effectiveness in prediction tasks, (2) Multi-layer Perceptrons (MLP) [30]: known for capturing nonlinear relationships in data, (3) Random forest (RF) [16]: utilizes decision trees to improve prediction accuracy, and (4) Graph neural networks (GNN) [62,2,26]: applies graph structures to model complex relationships in data. It encodes PQP as a DAG [2] within GNN, allowing the model to treat different operators within PQP as nodes, and the relationships between them as edges. These models are selected based on their diverse approaches to model and handle the complexities of stream processing queries. The accuracy of these models is measured using Q-error $q(c, c')$ *where* $q \geq 1$ metric as mentioned before. Here, q-error being close to 1 represents better prediction accuracy. We also implement early stopping for each ML model to prevent overfitting and ensure efficient training times. Early stopping is based on monitoring the validation loss, halting training if it did not improve for 100 consecutive epochs. This method was uniformly applied across all models to maintain consistency. The training of these model performed on $m510$ clusters.

O8- *Graph representation assists in learning dynamic behavior of SPS.* The primary aim of benchmarking ML models is to evaluate how these diverse cost models perform across various streaming queries. In Figure 5 indicates that the GNN model consistently surpasses other models in predicting cost i.e., latency, even as query complexity increases. The high accuracy of GNN can be attributed to the fact that the graph-based representation enables it to capture and utilize the intricate dependencies within the query structures effectively and dynamic behaviors of stream processing systems, leading to more accurate and reliable performance predictions compared to other models. Similar observation was made for PQP related to real-world applications and thus we only present results on synthetic PQPs.

(2) Influence of ML training strategies. We investigate the influence of various enumeration strategies (cf. Section 3.1), designed to optimize data collection efforts and reduce training time in PDSP-Bench. For instance, Figure 6 represent our findings by comparing the efficacy of random and rule-based parallelism enumeration strategies for training GNN models for latency prediction.

We stick to GNN models for this evaluation because of the observations made in the previous evaluation (O8).

O9- *Data-efficient training for high accuracy with reduced training time.* Figure 6a illustrates that the GNN model trained using rule-based enumeration strategy requires only $2.5k$ queries for accurate latency predictions for both seen (linear, 2-way and 3-way join) and unseen (other remaining synthetic) queries. This efficiency is attributed to the strategy's systematic collection of data, which focuses on determining and exploring around selected parallelism degrees and generating more meaningful graph representations for GNN than the random strategy. Further analysis in Figure 6b reveals that the rule-based strategy requires significantly less training time—approximately three time less than the time of the random strategy, i.e., 4.6 hours. This benchmarking of enumeration strategies underscores a crucial insight: adopting data-efficient training methods using PDSP-Bench can significantly enhance model training efficacy.

5 Related Work

We divide existing benchmarking systems for DSP systems into - *(i)* DSPS- *(ii)* TPC and *(iii)* machine learning (ML) in benchmarking systems.

Benchmarking Systems for DSPS. Despite numerous benchmarks for database management systems, standardised and systematic benchmarks specifically designed for stream processing architectures are scarce. Our review of existing systems [32,51,42,33,28,37,8,9] reveals significant gaps, particularly in addressing the nuances of DSP (cf. Table 1). The LRB [4] and similar efforts like the YSB [18,19,25] and BigDataBench [60] often remain focused on batch processing rather than real-time streaming. Emerging micro-benchmarks such as HiBench [31], StreamBench [61], RIoTBench [52] and OSPBench [58] bring advancements in streaming benchmarks but still fall short in adequately testing scalability and handling real-time streaming requirements. These benchmarks typically do not address essential aspects of DSP systems like parallelism, hardware diversity, and variable workloads comprehensively. Recent benchmarking systems such as DSPBench [13] and SPBench [24] address some aspects of parallelism but do not provide a holistic view of parallel stream processing. They often neglect critical elements such as the degree of parallelism, data partitioning strategies, and typically rely on homogeneous hardware setups, which do not reflect real-world DSP environments [65].

TPC Benchmarks. TPC [48] has developed several benchmarks to evaluate various aspects of computing systems [17,11,46,12,55,15,49]. Among these TPC benchmarks, TPCx-IoT [49] is the most relevant to DSP as it addresses scenarios involving continuous data ingestion and real-time analytics.While the TPC benchmarks have contributed to standardizing the evaluation of transactional and analytical processing systems, there remains a distinct gap in benchmarking system designed specifically for parallel and distributed stream processing due to fundamental differences in system architecture and operational goals.

TPC benchmarks provide extensive coverage of database and batch processing scenarios, which are more oriented towards singular query execution on

static or slowly evolving data sets. These benchmarks fall short in addressing the unique requirements of stream processing systems such real-time data processing, continuous ingestion, and immediate response to dynamic changes in data streams. In addition, none of the existing TPC benchmarks are designed to assess performance under these conditions, focusing instead on batch or transactional processing where data latency and continuous data flow are less critical. They do not adequately test data streaming partitioning and required parallelism for dynamically scaling systems in distributed environments.

Machine Learning in DSP systems. In the context of benchmarking of learned component of SPS such as performance prediction, existing benchmarks like DeepBench [5], MLPerf [50], Fathom [1], CleanML [40] and TPCx-AI [14] have laid significant groundwork. These benchmarks are predominantly tailored to assess specific aspects of ML systems, from the underlying hardware's computational abilities to the effectiveness of algorithms on static dataset. For instance, MLPerf focuses on the computationally intensive aspects of ML such as model training and inference, TPCx-AI provides a more comprehensive evaluation by including the entire data processing pipeline. These benchmarks are valuable for organizations to optimize their end-to-end ML workflows, e.g., Dell Technologies show significant performance improvements using TPCx-AI [59] in providing high-performance ML solutions. Their use of the benchmark has highlighted improvements in hardware efficiency and cost-performance ratios, underscoring the benchmark's utility in real-world applications. While existing ML benchmarks [3,8] are valuable in evaluating the capabilities and performance of various systems, a significant gap exists in terms of incorporating these findings into predictive performance models. For instance, DeepBench and MLPerf primarily focus on predefined tasks for benchmarking the raw performance of hardware and machine learning frameworks but do not extend to predicting future system performance based on evolving workloads or system changes. Similarly, other ML benchmarks typically evaluate performance under static conditions. They do not provide insights into system performance under fluctuating workloads or infrastructure changes, nor do they incorporate real-time data streams crucial for continuous decision-making and predictive analytics in dynamic environments.

6 Conclusion

This paper introduces PDSP-Bench that addresses the critical need for an advanced benchmarking system that reflects the complexities of modern parallel SP environments. Its ability to support heterogeneous hardware, integrate ML models, and evaluate a wide range of parallel query structures makes it a valuable tool for researchers and practitioners in the field of stream processing. Our findings underline the constructive impact of parallelism and hardware diversity in improving performance, but also the necessity of a refined understanding of massive parallel dataflows. In future, we plan to expand on heterogeneous hardware architectures like GPUs, followed by a in-depth analysis and training of ML models based on data collected using PDSP-Bench.

Acknowledgements

This research is funded by the DFG as part of project C2 within the Collaborative Research Center (CRC) 1053– MAKI; BMBF and the state of Hesse as part of the NHR Program and HMWK cluster project 3AI (The Third Wave of AI). We also want to thank hessian.AI at TU Darmstadt and DFKI Darmstadt.

References

1. Adolf, R., Rama, S., Reagen, B., Wei, G.Y., Brooks, D.: Fathom: Reference workloads for modern deep learning methods. In: 2016 IEEE International Symposium on Workload Characterization (IISWC). pp. 1–10. IEEE (2016)
2. Agnihotri, P., Koldehofe, B., Stiegele, P., Heinrich, R., Binnig, C., Luthra, M.: Zerotune: Learned zero-shot cost model for parallelism tuning in stream processing. In: IEEE 40th International Conference on Data Engineering (ICDE). p. 1–16 (2024)
3. Ahmed, N., Barczak, A.L., Rashid, M.A., Susnjak, T.: Runtime prediction of big data jobs: performance comparison of machine learning algorithms and analytical models. Journal of Big Data **9**(1), 67 (2022)
4. Arasu, A., Cherniack, M., Galvez, E., Maier, D., Maskey, A.S., Ryvkina, E., Stonebraker, M., Tibbetts, R.: Linear road: a stream data management benchmark. In: VLDB. pp. 480–491 (2004)
5. Belloni, S., Ritter, D., Schröder, M., Rörup, N.: Deepbench: Benchmarking json document stores. In: Proceedings of the 2022 workshop on 9th International Workshop of Testing Database Systems. pp. 1–9 (2022)
6. Biem, A., Bouillet, E., Feng, H., Ranganathan, A., Riabov, A., Verscheure, O., Koutsopoulos, H., Moran, C.: Ibm infosphere streams for scalable, real-time, intelligent transportation services. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. pp. 1093–1104 (2010)
7. Blog, N.T.: Keystone Real-time Stream Processing Platform. `https://t.ly/61XZJ` (2018), [Online; accessed 03-03-2024]
8. Boden, C., Rabl, T., Schelter, S., Markl, V.: Benchmarking distributed data processing systems for machine learning workloads. In: Performance Evaluation and Benchmarking for the Era of Artificial Intelligence: 10th TPC Technology Conference, TPCTC 2018. pp. 42–57. Springer (2019)
9. Boden, C., Spina, A., Rabl, T., Markl, V.: Benchmarking data flow systems for scalable machine learning. In: Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond. pp. 1–10 (2017)
10. Boncz, P., Neumann, T., Erling, O.: Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark. In: Technology Conference on Performance Evaluation and Benchmarking. pp. 61–76. Springer (2013)
11. Boncz, P., Neumann, T., Erling, O.: Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark. In: Technology Conference on Performance Evaluation and Benchmarking. pp. 61–76. Springer (2013)
12. Bond, A., Johnson, D., Kopczynski, G., Taheri, H.R.: Profiling the performance of virtualized databases with the tpcx-v benchmark. In: Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things: 7th TPC Technology Conference, TPCTC 2015, Kohala Coast, HI, USA, August 31–September 4, 2015. Revised Selected Papers 7. pp. 156–172. Springer (2016)

13. Bordin, M.V., Griebler, D., Mencagli, G., Geyer, C.F., Fernandes, L.G.L.: Dspbench: A suite of benchmark applications for distributed data stream processing systems. IEEE Access **8**, 222900–222917 (2020)
14. Brücke, C., Härtling, P., Palacios, R.D.E., Patel, H., Rabl, T.: Tpcx-ai - an industry standard benchmark for artificial intelligence and machine learning systems. Proceedings of the VLDB Endowment **16**(12), 3649–3661 (2023)
15. Cao, P., Gowda, B., Lakshmi, S., Narasimhadevara, C., Nguyen, P., Poelman, J., Poess, M., Rabl, T.: From bigbench to tpcx-bb: Standardization of a big data benchmark. In: Performance Evaluation and Benchmarking. Traditional-Big Data-Internet of Things: 8th TPC Technology Conference, TPCTC 2016. pp. 24–44. Springer (2017)
16. Chen, J., Li, K., Tang, Z., Bilal, K., Yu, S., Weng, C., Li, K.: A parallel random forest algorithm for big data in a spark cloud computing environment. IEEE Transactions on Parallel and Distributed Systems **28**(4), 919–933 (2016)
17. Chen, S., Ailamaki, A., Athanassoulis, M., Gibbons, P.B., Johnson, R., Pandis, I., Stoica, R.: Tpc-e vs. tpc-c: Characterizing the new tpc-e benchmark via an i/o comparison study. ACM Sigmod Record **39**(3), 5–10 (2011)
18. 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: IEEE IPDPS. pp. 1789–1792 (2016)
19. Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Liu, Z., Nusbaum, K., Patil, K., Peng, B., et al.: Benchmarking streaming computation engines at yahoo. Technical Report (2015)
20. DEBS, A.: DEBS 2014 Grand Challenge: Smart homes. `https://debs.org/grand-challenges/2014/` (2016), [Online; accessed 25-04-2024]
21. Dubey, S.: Real Time Sentiment Analysis. `https://github.com/voltas/real-time-sentiment-analytic` (2015), [Online; accessed 25-04-2024]
22. Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., Akella, A., Wang, K., Ricart, G., Landweber, L., Elliott, C., Zink, M., Cecchet, E., Kar, S., Mishra, P.: The design and operation of CloudLab. In: In USENIX Conference. p. 1–14 (2019)
23. Ganapathi, A., Kuno, H., Dayal, U., Wiener, J.L., Fox, A., Jordan, M., Patterson, D.: Predicting multiple metrics for queries: Better decisions enabled by machine learning. In: 2009 IEEE 25th International Conference on Data Engineering. pp. 592–603 (2009)
24. Garcia, A.M., Griebler, D., Schepke, C., Fernandes, L.G.: Spbench: a framework for creating benchmarks of stream processing applications. Springer Computing **105**(5), 1077–1099 (2023)
25. Grier, J.: Extending the yahoo! streaming benchmark. URL http://data-artisans.com/extending-the-yahoo-streamingbenchmark (2016)
26. Heinrich, R., Binnig, C., Kornmayer, H., Luthra, M.: Costream: Learned cost models for operator placement in edge-cloud environments. In: IEEE 40th International Conference on Data Engineering (ICDE). p. 1–16 (2024)
27. Heinrich, R., Luthra, M., Kornmayer, H., Binnig, C.: Zero-shot cost models for distributed stream processing. In: ACM DEBS. p. 85–90 (2022)
28. Hesse, G., Lorenz, M.: Conceptual survey on data stream processing systems. In: Proceedings of the IEEE 21st International Conference on Parallel and Distributed Systems. p. 797–802. IEEE Computer Society (2015)

29. Hesse, G., Matthies, C., Perscheid, M., Uflacker, M., Plattner, H.: Espbench: the enterprise stream processing benchmark. In: ACM/SPEC ICPE. pp. 201–212 (2021)
30. Hosseinzadeh Talaee, P.: Multilayer perceptron with different training algorithms for streamflow forecasting. Neural Computing and Applications **24**, 695–703 (2014)
31. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In: IEEE ICDEW. pp. 41–51 (2010)
32. Ihde, N., Marten, P., Eleliemy, A., Poerwawinata, G., Silva, P., Tolovski, I., Ciorba, F.M., Rabl, T.: A survey of big data, high performance computing, and machine learning benchmarks. In: Performance Evaluation and Benchmarking: 13th TPC Technology Conference, TPCTC 2021, Copenhagen, Denmark, August 20, 2021, Revised Selected Papers 13. pp. 98–118. Springer (2022)
33. Isah, H., Abughofa, T., Mahfuz, S., Ajerla, D., Zulkernine, F., Khan, S.: A survey of distributed data stream processing frameworks. IEEE Access **7**, 154300–154316 (2019)
34. Jiang, Y.: Real Time Anomaly Detection Framework. `https://github.com/yxjiang/stream-outlier` (2013), [Online; accessed 25-04-2024]
35. Kalavri, V., Liagouris, J., Hoffmann, M., Dimitrova, D., Forshaw, M., Roscoe, T.: Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows. In: USENIX OSDI. pp. 783–798 (2018)
36. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V.: Benchmarking distributed stream data processing systems. In: ICDE. pp. 1507–1518 (2018)
37. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V.: Benchmarking distributed stream data processing systems. In: 2018 IEEE 34th international conference on data engineering (ICDE). pp. 1507–1518. IEEE (2018)
38. Kulkarni, S., Bhagat, N., Fu, M., Kedigehalli, V., Kellogg, C., Mittal, S., Patel, J.M., Ramasamy, K., Taneja, S.: Twitter heron: Stream processing at scale. In: Proceedings of the ACM SIGMOD international conference on Management of data. pp. 239–250 (2015)
39. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T.: How good are query optimizers, really? **9**(3), 204–215 (2015)
40. Li, P., Rao, X., Blase, J., Zhang, Y., Chu, X., Zhang, C.: Cleanml: A study for evaluating the impact of data cleaning on ml classification tasks. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). pp. 13–24. IEEE (2021)
41. Library, G.: GeoTools. `https://www.osgeo.org/projects/geotools/` (2020), [Online; accessed 25-04-2024]
42. Liu, X., Buyya, R.: Resource management and scheduling in distributed stream processing systems: a taxonomy, review, and future directions. in CSUR **53**, 1–41 (2020)
43. Lu, R., Wu, G., Xie, B., Hu, J.: Stream bench: Towards benchmarking modern distributed stream computing frameworks. In: IEEE UCC. pp. 69–78 (2014)
44. Luthra, M., Koldehofe, B., Danger, N., Weisenberger, P., Salvaneschi, G., Stavrakakis, I.: TCEP: Transitions in operator placement to adapt to dynamic network environments. Journal of Computer and System Sciences **122**, 94–125 (2021)
45. Mathioudakis, M., Koudas, N.: Twittermonitor: trend detection over the twitter stream. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. pp. 1155–1158 (2010)

46. Nambiar, R.O., Poess, M.: The making of tpc-ds. In: VLDB. vol. 6, pp. 1049–1058 (2006)
47. Neumeyer, L., Robbins, B., Nair, A., Kesari, A.: S4: Distributed stream computing platform. In: 2010 IEEE International Conference on Data Mining Workshops. pp. 170–177. IEEE (2010)
48. Poess, M., Floyd, C.: New tpc benchmarks for decision support and web commerce. ACM Sigmod Record **29**(4), 64–71 (2000)
49. Poess, M., Nambiar, R., Kulkarni, K., Narasimhadevara, C., Rabl, T., Jacobsen, H.A.: Analysis of tpcx-iot: The first industry standard benchmark for iot gateway systems. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE). pp. 1519–1530. IEEE (2018)
50. Reddi, V.J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C.J., Anderson, B., Breughe, M., Charlebois, M., Chou, W., et al.: Mlperf inference benchmark. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). pp. 446–459. IEEE (2020)
51. Röger, H., Mayer, R.: A comprehensive survey on parallelization and elasticity in stream processing. ACM Computing Surveys **52**(2), 37 (2019)
52. Shukla, A., Chaturvedi, S., Simmhan, Y.: Riotbench: An iot benchmark for distributed stream processing systems. CCPE **29**(21), 42–57 (2017)
53. Simmhan, Y., Cao, B., Giakkoupis, M., Prasanna, V.K.: Adaptive rate stream processing for smart grid applications on clouds. In: Proceedings of the 2nd international workshop on Scientific cloud computing. pp. 33–38 (2011)
54. Solazzo, D.: Storm Log Processing. `https://github.com/domenicosolazzo/click-topology` (2013), [Online; accessed 25-04-2024]
55. Taheri, H.R., Little, G., Desai, B., Bond, A., Johnson, D., Kopczynski, G.: Characterizing the performance and resilience of hci clusters with the tpcx-hci benchmark. In: Performance Evaluation and Benchmarking for the Era of Artificial Intelligence: 10th TPC Technology Conference, TPCTC 2018. pp. 58–70. Springer (2019)
56. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: IROS. pp. 23–30. IEEE (2017)
57. Tucker, P., Tufte, K., Papadimos, V., Maier, D.: Nexmark–a benchmark for queries over data streams (draft). Technical report, Technical Report. Technical report (2008)
58. Van Dongen, G., Van den Poel, D.: Evaluation of stream processing frameworks. IEEE TPDS **31**(8), 1845–1858 (2020)
59. Wakou, N.: Dell Reinforces its TPCx-AI Benchmark Leadership using the 16G PowerEdge R6625 Hardware Platform at SF1000. `https://shorturl.at/uv5qU` (2023), [Online; accessed 03-08-2024]
60. Wang, L., Zhan, J., Luo, C., Zhu, Y., Yang, Q., He, Y., Gao, W., Jia, Z., Shi, Y., Zhang, S., et al.: Bigdatabench: A big data benchmark suite from internet services. In: IEEE HPCA. pp. 488–499 (2014)
61. Wang, Y., et al.: Stream processing systems benchmark: Streambench. Master's thesis (2016)
62. Wu, Z., Marcus, R., Liu, Z., Negi, P., Nathan, V., Pfeil, P., Saxena, G., Rahman, M., Narayanaswamy, B., Kraska, T.: Stage: Query execution time prediction in amazon redshift. In: Companion of the 2024 International Conference on Management of Data. p. 280–294 (2024)
63. Yoon, K.A., Kwon, O.S., Bae, D.H.: An approach to outlier detection of software measurement data using the k-means clustering method. In: First International

Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, Manisha Luthra. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. To Appear in Proceedings of the Performance Evaluation and Benchmarking - 16th TPC Technology Conference, TPCTC 2024, Springer, 22 pages

Symposium on Empirical Software Engineering and Measurement (ESEM 2007). pp. 443–445. IEEE (2007)
64. Zapridou, E., Mytilinis, I., Ailamaki, A.: Dalton: Learned partitioning for distributed data streams. In VLDB **16**(3), 491–504 (2022)
65. Zeuch, S., Monte, B.D., Karimov, J., Lutz, C., Renz, M., Traub, J., Breß, S., Rabl, T., Markl, V.: Analyzing efficient stream processing on modern hardware. Proceedings of the VLDB Endowment **12**(5), 516–530 (2019)