ContTune: Continuous Tuning by Conservative Bayesian Optimization for Distributed Stream Data Processing Systems

Jinqing Lian[★], Xinyi Zhang[§], Yingxia Shao[★*], Zenglin Pu[★], Qingfeng Xiang[★], Yawen Li[★], Bin Cui[§]
[★] Beijing University of Posts and Telecommunications
[§]School of Computer Science, and National Engineering Laboratory for Big Data Analysis and Applications, Peking University
{jinqinglian,shaoyx,pzl_bupt,xiangqingfeng,warmly0716}@bupt.edu.cn,{zhang_xinyi,bin.cui}@pku.edu.cn

ABSTRACT

The past decade has seen rapid growth of distributed stream data processing systems. Under these systems, a stream application is realized as a Directed Acyclic Graph (DAG) of operators, where the level of parallelism of each operator has a substantial impact on its overall performance. However, finding optimal levels of parallelism remains challenging. Most existing methods are heavily coupled with the topological graph of operators, unable to efficiently tune under-provisioned jobs. They either insufficiently use previous tuning experience by treating successively tuning independently, or explore the configuration space aggressively, violating the Service Level Agreements (SLA).

To address the above problems, we propose ContTune, a continuous tuning system for stream applications. It is equipped with a novel Big-small algorithm, in which the Big phase decouples the tuning from the topological graph by decomposing the job tuning problem into sub-problems that can be solved concurrently. We propose a conservative Bayesian Optimization (CBO) technique in the Small phase to speed up the tuning process by utilizing the previous observations. It leverages the state-of-the-art (SOTA) tuning method as conservative exploration to avoid SLA violations. Experimental results show that ContTune reduces up to 60.75% number of reconfigurations under synthetic workloads and up to 57.5% number of reconfigurations under real workloads, compared to the SOTA method DS2.

PVLDB Reference Format:
Jinqing Lian, Xinyi Zhang, Yingxia Shao, Zenglin Pu, Qingfeng Xiang, Yawen Li, Bin Cui. ContTune: Continuous Tuning by Conservative Bayesian Optimization for Distributed Stream Data Processing Systems. PVLDB, 16(13): 4282 - 4295, 2023.
doi:10.14778/3625054.3625064

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ljqcodelove/ContTune.

Proceedings of the VLDB Endowment, Vol. 16, No. 13 ISSN 2150-8097.
doi:10.14778/3625054.3625064

*Yingxia Shao is the corresponding author.

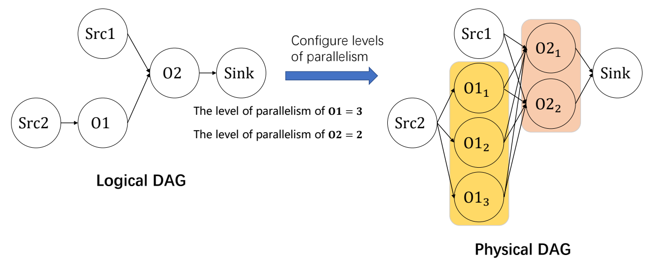

Figure 1: Logical and physical DAG of a stream job.

1 INTRODUCTION

In the past decade, distributed stream data processing systems have been widely used and deployed to handle the big data. Several mature production systems have emerged, including Flink [9], Storm [64], Spark Streaming [74], Heron [37], and Samza [57]. They can timely analyze the unbounded stream data with low latency and high throughput. In these systems, an analytical job is generally abstracted as a Directed Acyclic Graph (DAG) of operators, whose levels of parallelism are configurable. The levels of parallelism refer to the configuration of the number of physical instances used by each operator in a job. These configurations directly determine the allocation of resources for each operator and have a significant impact on the performance of the job, such as latency and throughput [10, 59]. Figure 1 is an example of a job in Apache Flink [9], and the level of parallelism of operator O1 is three and the level of parallelism of operator O2 is two. Therefore, to reduce the Total Cost of Ownership (TCO) while satisfying the Service Level Agreements (SLA), it is critical to set the optimal levels of parallelism.

Given a stream application, configuring the optimal levels of parallelism is non-trivial. First, there is no principled way to manually find the optimal levels of parallelism. Engineers typically try several configurations and pick the one that satisfies the SLA with minimum resource used [20]. Second, considering the dynamic and long-running (i.e., 24/7) stream data, engineers are required to continuously tune the levels of parallelism so as to adapt to variable workloads. As a result, developing effective systems to automatically configure the levels of parallelism has attracted increasing interest from academia and industry [18, 20, 33, 41, 44, 45, 50].

Researchers have put considerable efforts into studying parallelism tuning, which can be classified into three categories. The first category is rule-based methods [5, 11, 20, 24, 28, 29, 71, 74]. Their tuning policy is usually expressed in simple rules, e.g., if CPU utilization is larger than 70% then increase the levels of parallelism until CPU utilization is smaller than 70%. The second category is linearity-based methods (e.g., DS2 [33] and Turbine [50]) which dynamically configure the analytical jobs by linearly increasing or decreasing the levels of parallelism. The third category is Bayesian Optimization-based methods, represented by Dragster [41] and Fischer [18]. They adopt Bayesian Optimization which utilizes a surrogate model to suggest the promising levels of parallelism and updates the surrogate model based on the effect of the suggested levels of parallelism. Owing to these efforts, parallelism tuning can be automated, which largely saves the expensive human-labors. However, when applying these methods to configure real-world analytical jobs, we encounter several issues from the following perspectives:

Inefficiency of tuning under-provisioned jobs. The symptom of under-provisioning (e.g., backpressure) usually occurs when the input load increases (workload spikes) and causes SLA violations, which are often associated with significant financial penalties [27]. To configure the under-provisioned job, there are two types of methods: "find bottleneck and tune it" and "workload estimation". The former [11, 20, 28, 41, 45, 71] enlarges the level of parallelism of the bottleneck operator one by one [33]. Under such an approach, the operator could become a bottleneck repeatedly, influenced by the other tuned operators. This is because the operators of a job follow the producer-consumer model, each operator serves as both a producer for its downstream operators and a consumer for its upstream operators in the DAG. When scaling up a bottleneck operator, it increases the workloads of its upstream operators as consumers and the workloads of its downstream operators as producers. The increased workload can potentially lead to the emergence of new bottleneck operators (as shown in [20, 24, 41]), leading to an increase in the number of reconfigurations. So this approach might interrupt the running job frequently and takes a long time to converge to optimal levels of parallelism. The latter [18, 22, 33, 42, 50] estimates the real upstream data rate and suggests corresponding levels of parallelism to sustain the upstream data. However, the real upstream data rate cannot be accurately estimated when the job is under-provisioned, specifically for jobs containing *stateful* operators (e.g., join and window operators) [43]. Besides, the relationship between the configured levels of parallelism and the sustained datas is non-linear and multi-modal. They adopt simple approximation (e.g., linear function) and cannot characterize the complex relationship [41].

Insufficiency of using previous tuning experience. In front of the long-running stream application with inevitable workload variations, most existing methods treat the successively tuning independently, named as the *one-shot parallelism tuning*. To be concrete, whenever the parallelism tuning of the analytical jobs is triggered according to the changes of workload, such approaches search for the optimal levels of parallelism from the scratch and do not utilize any observations from previous tuning. *One-shot parallelism tuning* is inefficient for the dynamic unbounded stream data and causes a large number of reconfigurations [1]. A bad case is that when the job encounters the historical workload (i.e., the workload of stream data has been processed before), the historical optimal level of parallelism can be reused without tuning from the beginning. As far as we know, Dragster [41] and Turbine [50] are the only two parallelism tuning methods that utilize the historical information, called *continuous tuning* in this paper. Dragster utilizes Bayesian Optimization to find the optimal levels of parallelism for a given upstream data rate. However, Dragster tends to aggressively explore the entire configuration space of the levels of parallelism, resulting in frequent violations of the SLA. Besides, since Dragster establishes separate Bayesian Optimization models to find the optimal level of parallelism for different upstream data rates of each operator, the reuse of previous tuning experience is only possible when the upstream data rates are identical. However, in practice, the range of upstream data rates is wide, making Dragster to rarely reuse the tuning experience. Turbine makes predictions for future workloads by using historical workloads to determine whether the new configuration is optimal for the predicted future workload, and does not use historical data to accelerate tuning itself. In summary, the insufficiency of using the historical tuning experience makes the existing approach inefficient when handling the dynamic workload, and the continuous tuning problem is not well studied yet.

[1]To test a candidate level of parallelism, it requires one reconfiguration which is a time-consuming step. An efficient tuning method finds the optimal level of parallelism with a few (or minimal) number of reconfigurations.

Our approach. To address the above challenges, we propose ContTune, a continuous tuning system for elastic stream processing. ContTune is equipped with a novel Big-small algorithm, in which the Big phase first decouples the tuning from the topological graph by decomposing the job tuning problem into N sub-problems (discussed in Section 4.1). The N sub-problems can be tuned by the Small phase concurrently, largely reducing the number of reconfigurations. On the basis of the Big-small algorithm, ContTune prioritizes SLA – it quickly allocates sufficient resources for the under-provisioned jobs in the Big phase and further improves the resource utilization in the Small phase. Besides, we design a conservative Bayesian Optimization (CBO) technique to speed up the tuning process by utilizing historical observations. Compared with vanilla Bayesian Optimization, CBO leverages SOTA *one-shot parallelism tuning* methods [33, 50] as conservative exploration to avoid the SLA violations caused by aggressive exploration. Specifically, CBO has two modules: (1) conservative exploration, which utilizes SOTA one-shot parallelism tuning methods to avoid aggressive exploration and warm up the tuning when having insufficient historical observations; (2) fast exploitation, which utilizes historical observations to suggests the levels of parallelism according to an acquisition function. When compared to Dragster and Turbine, CBO leverages historical observations to establish the relationship between levels of parallelism and the corresponding processing abilities, which is constant and can be used to deal with different upstream data rates. On the basis of this relationship, ContTune could quickly find the minimum level of parallelism whose processing ability is lager than the upstream data rate. We theoretically prove that ContTune finds optimal levels of parallelism with $O(1)$ average complexity of the number of reconfigurations. Specifically, we make the following contributions:

- We propose the Big-small algorithm to tune levels of parallelism for distributed stream data processing systems. The "Big phase" can decouple the tuning methods from the topological graph and the "Small phase" can concurrently tune

Table 1: Summary of the reconfiguration methods of existing DSDPSs.

Method	Reconfiguration methods
Flink [9]	Redeploy
Heron [37]	Redeploy
Seep [11]	Partial redeploy
Rhino [14]	Partial update
Megaphone [31]	Non-stop partial update
Chi [44]	Partial update
Trisk [45]	Partial update

all operators. Meanwhile, the Big-small algorithm prioritizes SLA to meet online tuning requirements.

- We propose CBO to cope with long-running jobs by using historical observations to fit the relationship between the levels of parallelism and processing abilities for *fast exploitation.* Besides, it first uses *one-shot parallelism tuning* SOTA methods as *conservative exploration* in order to avoid aggressive exploration in vanilla Bayesian Optimization.
- We implement the proposed method and evaluate on standard benchmarks and real workloads. Compared with the SOTA method DS2, ContTune reduced up to **60.75%** number of reconfigurations under synthetic workloads and up to **57.5%** number of reconfigurations under real workloads.

2 PRELIMINARY

We introduce more details of basic concepts such as stream jobs, logical DAG, physical DAG, backpressure, reconfiguration, stateless operators and stateful operators in this section, and formulate the tuning problem.

2.1 Stream Processing Jobs in DSDPS

We target at configuring the job (i.e., a stream processing application) in distributed stream data processing systems (DSDPSs) that are *Data Parallelization* [59]. *Data Parallelization* executes one operator on multiple instances. The count of these instances is called as the level of parallelism of the operator. *Data Parallelization* is commonly supported by DSDPSs, such as Esper [1], Storm [64], Heron [37], Spark Streaming [74], Flink [9], and these systems [23, 34, 35, 47–49, 51–55, 58, 60, 61, 70, 73].

Logical DAG. A job (i.e., a stream processing application) in DSDPSs can be modeled as a *logical* Directed Acyclic Graph (DAG) as shown in the left part of Figure 1, denoted as $G = (vertices, edges)$, where the performance of each operator heavily depends on the others, and $vertices$ indicate the operators of the job and *edges* indicate the passed records (workload) between operators. Specifically, operators that only have outgoing edges are *sources*, and operators that only have incoming edges are *sinks*.

Physical DAG. We denote a job running on the given physical instances as a *physical* DAG. Figure 1 shows a *logical* DAG and its corresponding *physical* DAG for Nexmark Q3 [2, 4, 66] with two sources and one sink. Configuring the levels of parallelism of a job decides the number of physical instances for each operator. In Figure 1, operators O1 and O2 execute with three and two instances, equivalent to their level of parallelism being set three and two.

Table 2: Notations in this paper.

Symbol	Description
G	logical dataflow Directed Acyclic Graph
N	number of operators in G ($N > 1$)
$\hat{\lambda}$	aggregated observed upstream data rates
λ	real upstream data rates
T_u	useful time for an operator
H^t	historical observations in t iterations
o_i	the i^{th} operator in G
PA	the real processing ability
p^j	the level of parallelism at iteration j
p^{max}	the max level of parallelism in H^t
p^{now}	the now level of parallelism of an operator
P_{job}	the levels of parallelism of operators in G given by CBO
GP	Gaussian Process model
α	a threshold for scoring function
$d_{nearest}$	the nearest distance between p^{now} and the observed levels of parallelism in H^t
Sg	the known region segment
len_{all}	the total length of merged region segments
ρ	ρ tuning times
χ	CBO uses χ tuning times in ρ tuning times
ϕ	the maximal number of reconfigurations of SOTA method introduced by CBO of each tuning
ω	ω tuning times for fast exploitation to converge
W_u	the workload unit of synthetic workloads

Backpressure. Backpressure is a mechanism that propagates overload notifications from operators backward to the sources so that data emission rates can be throttled to alleviate overload [10]. It is a symptom observed in under-provisioned jobs. When this happens, workloads that cannot be immediately processed by sources will not be discarded and are usually kept in the queue [3, 43].

Reconfiguration. The job requires reconfigurations to change the levels of parallelism. Each DSDPS enables different reconfigurations methods. Table 1 shows that Flink and Heron need to redeploy (stop and restart), and Trisk adopts a partial pause-and-resume scheme. For all DSDPSs, efficient tuning method finds the optimal level of parallelism using small number of reconfigurations.

The operator of a DSDPS job could be stateless or stateful: (a) **Stateless Operator.** The data processed by the stateless operator is only relevant to the current operator and the stateless operator does not store the state from previous processing. Examples of stateless operators are filter and rescaling. (b) **Stateful Operator.** The data received by the stateful operator will be stored as state information for computation, such as window [26] and join.

2.2 Problem Definition and Terminology

We formulate the parallelism tuning problem and discuss the related terminology. Table 2 summarizes the notations.

Parallelism Tuning Problem. Given a logical DAG of a job with N operators, the source operators generate records at a rate, defined by application data sources (sensors, stock market feeds, etc.) [33]. To maximize system throughput, the physical DAG must sustain the full source rates. This means that each operator must be able to process data without backpressure. Parallelism tuning aims to find the minimal level of parallelism per operator in the physical DAG that sustains all source rates (i.e., satisfying the SLA). Since changing the level of parallelism of the operator requires costly reconfiguration, we additionally want to find the optimal level of parallelism in which each operator can sustain its real upstream data rate via fewer reconfigurations.

Upstream Data Rate. An upstream data rate $\hat{\lambda}$ denotes the aggregated number of observed records (i.e., workload) that an operator receives from its all upstream operators per unit of time. Given a DAG, the observed upstream data rate is affected by the source rate and the processing ability of operators in the DAG (following the producer-consumer model). When all operators can process their upstream data rate (i.e., no backpressure occurs), the upstream data rate is only affected by the source rate. Such an observed upstream data rate is denoted as the real upstream data rate λ.

Useful Time. Useful time T_u is the time that an operator executes in an ideal setting where it never has to wait to obtain input or push output. It differs from the total observed execution time. T_u is the total time that an operator spends in serialization, processing and deserialization [33].

Processing Ability. The processing ability PA denotes how many records an operator can process per unit of useful time. We use the same methodology as DS2 [33] to obtain PA:

$$PA = \frac{\hat{\lambda}}{T_u}. \tag{1}$$

An operator's processing ability is affected by its level of parallelism but does not increase linearly with it [33, 41]. After applying a given level of parallelism (denoted as p_i) for an operator o_i, we could obtain the processing ability of o_i (i.e., $PA(p_i)$) according to Equation 1. We use H_i^t to denote the historical observations in t iterations for operator o_i under different levels of parallelism, i.e., $H_i^t = \left\{\left\langle p_i^j, PA(p_i^j)\right\rangle\right\}_{j=1}^{t}$, where p_i^j denotes the level of parallelism for operator o_i at iteration j.

3 SYSTEM OVERVIEW

Figure 2 presents the overview of ContTune. The controller queries the job-generated metrics and determines whether a job needs tuning based on the conditions. For more details about the controller we refer the reader to our technical report [39]. When a job tuning is triggered, the controller checks the state of the job. It detects symptoms of over-provisioning or under-provisioning (e.g. backpressure). Then under-provisioned jobs go through the Big and Small phases, while over-provisioned jobs directly enter the Small phase. The Big phase enlarges the levels of parallelism of the job, following the *Binary Lifting* method which quickly eliminates the under-provisioned state. The Small phase is executed when the job is not under-provisioned. It finds the minimal level of parallelism of each operator that can sustain the real upstream data rate via conservative Bayesian Optimization (CBO). CBO adopts two strategies to find the optimal levels of parallelism: *fast exploitation* and *conservative exploration*. The fast exploitation utilizes historical observations. It fits Gaussian Processes (GP) on the observations and suggests the levels of parallelism according to a carefully designed

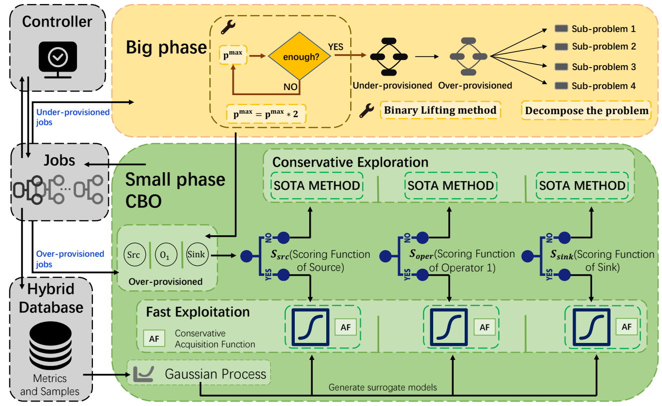

Figure 2: Overall Architecture of ContTune.

acquisition function that guarantees the SLA. The conservative exploration utilizes SOTA *one-shot parallelism tuning* methods to avoid aggressive exploration and warm up the learning of GP. ContTune adopts a scoring function to balance the fast exploitation and conservative exploration. After CBO finds the optimal level of parallelism of each operator, in order to avoid frequent reconfigurations, the controller confirms whether applying the levels of parallelism output by CBO is necessary given the current levels of parallelism. If necessary, the controller reconfigures the job with the levels of parallelism output by CBO, otherwise skips this reconfiguration. At the end of each tuning, the observed levels of parallelism of operators and their corresponding processing abilities will be added to H^t.

4 BIG-SMALL ALGORITHM

In this section, we first discuss the decomposition of the parallelism tuning problem which can be efficiently solved. Then we present the Big phase to decompose the problem and the Small phase to solve the decomposed problem.

4.1 Decomposing the Parallelism Tuning

Given a logical DAG of a job with N operators, parallelism tuning aims to find the minimal level of parallelism per operator that sustains all source rates. Sustaining all source rates is equivalent to that every operator can process their real upstream data rates. Note that the real upstream data rate reflects the real workload of each operator under the producer-consumer model, which is different from the observed one when the job is under-provisioned. The real upstream data rate only can be observed when the job is not under-provisioned. Therefore, if we can obtain the real upstream data rates of each operator, the parallelism tuning of a job can be decomposed to the parallelism tunings of each operator. Each operator can be concurrently tuned to fulfill their corresponding upstream data rate λ_i. And there is no need to tune a single bottleneck operator one by one (requiring several reconfigurations), as "find bottleneck and tune it" used by many existing tuning methods (e.g., Dhalion [20], Dragster [41], IBM Streams [24] and GOVERNOR [13]). Formally, we aim to solving the following equation to achieve the minimal number of reconfigurations:

$$\begin{aligned} &\arg\min_{p_1,\dots,p_N} p_i, \textit{ and } p_i \leq \textit{maximal bound} \\ &\text{subject to } PA(p_i) \geq \lambda_i. \end{aligned} \tag{2}$$

Other existing tuning methods (e.g., DS2 [33]) use instrumentation of bottleneck detection tools (e.g., SnailTrail [30]) to estimate λ via selectivities between operators. We find that these tuning methods face two problems. First, the instrumentation brings additional overhead, which increases the per-record latency (e.g., 13% as shown in [33]). Second, the estimated λ might be inaccurate since selectivities between stateful operators are inaccurate [39, 43]. Federico et. al [43] point out that stateful operators have a large standard deviation of the observed selectivities (discussed in [33, 43, 65]) due to their semantics, and it is inaccurate for "workload estimation" method to use observed selectivities at a specific moment to represent the selectivities of these operators. For example, window, a typical stateful operator, splits the infinite stream into "buckets" of finite size, over which DSDPSs can apply computations. It may obtain a observed selectivity of zero if no "buckets" are computed within the observed time. Then the inaccurate selectivity is used to estimate the upstream data rates to the downstream operators. The inaccuracy is propagated over the topological graph, leading to the non-optimal levels of parallelism of the operators. We use the Big-small algorithm to tackle these two problems. If the job is under-provisioned at the beginning, the Big phase first make the job not in backpressure state and then uses the observed $\hat{\lambda}$ as the real λ. Then the parallelism tuning problem can be decomposed into N sub-problems that find the minimal level of parallelism per operator whose processing ability is not less than its real upstream data rate, i.e., $PA(p_i) \geq \lambda_i$. The Big phase decouples the parallelism tuning from the topological graph by decomposing the parallelism tuning to N sub-problems. Then the Small phase can concurrently tune these N sub-problems. The parallelism tuning problem of the over-provisioned job at the beginning can be directly decomposed into N sub-problems and concurrently tuned by the Small phase.

4.2 Big Phase and Small Phase

Algorithm 1 illustrates the main procedures of the Big-small algorithm. The algorithm has two phases: **Big** and **Small**. Under-provisioned jobs go through these two phases, while over-provisioned jobs directly enter the Small phase. The Big phase increases the efficiency of tuning by first giving sufficient levels of parallelism so that the job is not in backpressure state. For the over-provisioned job, the Small phase aims to quickly find minimal levels of parallelism that the job can sustain all source rates, thereby improving resource utilization.

Big Phase. The Big phase focuses on the fast elimination of operator's backpressure using the *Binary Lifting* method, which can even out the time complexity with the help of historical observations H^t (discussed in Section 6.2). The Big phase maintains the maximal level of parallelism as p^{max}, among all the observations in H^t. All jobs that are at the end of the Big phase, rather than the end of the Small phase, satisfy their SLA, which means the Big phase prioritizes SLA to meet online tuning requirements. Specifically, the Big phase first checks if each operator's current level of parallelism p_i^{now} is equivalent to p^{max} (Line 4 – Line 7). If each p_i^{now} is equivalent to p^{max} and the job is still under-provisioned,

Algorithm 1 Big-small Algorithm

Input: A stream job with N operators, the maximal level of parallelism observed p^{max} in all H^t for the job
Output: The levels of parallelism suggested for the given job P_{job}

```
1: // "Big" phase
2: while the job is under-provisioned do
3:     Flag ← True
4:     for i ← 1...N do
5:         if p^now_i ≠ p^max then
6:             Flag ← False
7:         end if
8:     end for
9:     if Flag then, p^max ← 2 * p^max
10:    end if
11:    for i ← 1...N do
12:        p^now_i ← p^max
13:    end for
14:    P_job ← {p_i^now}_{i=1}^N and apply P_job via a reconfiguration
15: end while
16:
17: // "Small" phase
18: for i ← 1...N do
19:     λ_i ← λ̂_i  // The job is not under-provisioned now
20: end for
21: Use Algorithm 2 to get P_job
```

it indicates that p^{max} is not enough to sustain the upstream data rate, thus ContTune doubles p^{max} (Line 9). Finally, the Big phase sets the current p_i^{now} to p^{max}, $i = 1, .., N$ via one reconfiguration (Line 12). The above process (Line 4 – Line 14) is repeated until the job is not under-provisioned.

Small Phase. In the Small phase, we use CBO (details are discussed in Section 5) to find the optimal levels of parallelism for the over-provisioned jobs to improve resource utilization while satisfying $PA(p_i)) \geq \lambda_i$.

5 CONSERVATIVE BAYESIAN OPTIMIZATION

We adopt Bayesian Optimization to configure the levels of parallelism to improve resource utilization. We present how we adopt the BO to suggest the configuration with minimal resource usage while considering the SLA requirements in Section 5.1. To further avoid the SLA violation, we introduce the conservative Bayesian Optimization (CBO) in Section 5.2, which adopts linearity-based methods to replace the aggressive exploration in the vanilla BO.

5.1 BO for Parallelism Tuning

As discussed in Section 4.1, optimizing the whole DAG can be decomposed to optimizing N sub-problems as Equation 2. As guaranteed by the Big phase, p^{max} is set as the maximal bound in Equation 2. To find the desired p_i, one naive method is to evaluate all possible levels of parallelism, which is prohibitively expensive due to the number of required reconfigurations and the violation of SLA. To this end, we adopt Bayesian Optimization (BO) to guide the search for desired p_i.

BO is a widely-used optimization framework for the efficient optimization of expensive black-box functions. It has two key modules: (1) a *surrogate model* that learns the relationship between configurations and the performances, (2) an *acquisition function* that measures the utility of the given configurations according to the estimation of the surrogate model. In contrast to evaluating the expensive black-box function, the acquisition function is cheap to compute and can therefore be thoroughly optimized [68]. BO works iteratively: it chooses the next configuration to evaluate by maximizing the acquisition function and then updates the surrogate model based on the augmented observations. The main challenge of adopting BO is to set up suitable surrogate model and acquisition function for parallelism tuning. For more details about the techniques of handling noise, we refer the reader to our technical report [39].

Surrogate Model. In our BO method, we adopt Gaussian Process (GP) as the surrogate model. GP is a non-parametric model that can adaptively adjust its complexity to fit the data, which allows GP to capture intricate patterns and adapt to various data distributions. Besides, it provides well-calibrated uncertainty estimations and closed-form computability of the predictive distribution [32]. Other data-intensive techniques, e.g., deep learning may struggle with low data efficiency and interpretability. We use GP to learn the relationship between the levels of parallelism of the operator o_i and its processing abilities, based on H_i^t. Formally, it fits a probability distribution $p(f|p_i, H_i^t)$ of the target function $PA\,(p_i)$ on the observations H_i^t. With the help of GP, given a level of parallelism p_i, we can estimate its processing ability as a Gaussian variable with mean $\mu(p_i)$ and variance $\sigma^2(p_i)$ (indicating the confidence level of the estimation):

$$\begin{aligned} \mu(p_i) &= k_*^T K^{-1} y, \\ \sigma^2(p_i) &= k_*(p_i, p_i) - k_*^T K^{-1} k, \end{aligned} \tag{3}$$

where k is the covariance function, k_* denotes the vector of covariances between p_i and all previous observations, K is the covariance matrix of all previously evaluated configurations and y is the observed function values. To this end, we can utilize the confidence level to obtain the bound of the estimation: $l(p_i) = \mu(p_i) - \beta\sigma(p_i)$ and $u(p_i) = \mu(p_i) + \beta\sigma(p_i)$, where the parameter β controls the tightness of the confidence bounds [63]. The true function value of $PA(P_i)$ falls into the interval $[l(p_i), u(p_i)]$ with a high probability.

Acquisition Function. The sub-problem is essentially a minimization problem with an unknown constraint, as shown in Equation 2. The desired acquisition function should guide the finding of desired p_i as soon as possible, and avoid the SLA violation during tuning. Common acquisition functions such as UCB [62] and Expected Improvement (EI) [56] do not support the constraint conditions. Recently, Constrained EI (CEI) is proposed to optimize a black-box function with unknown constraints for optimizing the resource usage in data management systems [25, 76]:

$$arg\max_{p_i}\left((p_i^* - p_i) \times Pr[f(p_i) \geq \lambda]\right), \tag{4}$$

where p_i^* denotes the minimal feasible level of parallelism and $Pr[f(p_i) \geq \lambda]$ denotes the probability of feasibility. The probability of feasibility guides the search for feasible level of parallelism, while the reduced level of parallelism, i.e., $(p_i^* - p_i)$ encourages improving resource utilization. However, CEI does not consider the constraint safety-critical, and it may suggest infeasible levels of parallelism during tuning (e.g., trying the level of parallelism p_i with large $p_i^* - p_i$ but small $Pr[f(p_i) \geq \lambda]$). Once these levels of parallelism are suggested, additional reconfigurations are required to keep the job from under-provisioned. To prioritize the SLA while tuning, we make the safety constraint of CEI more strict and use the following acquisition function:

$$arg\max_{p_i}(p_i^* - p_i) I\,(\mu(p_i) - \lambda_i) \tag{5}$$

, where $I(x)$ is an indicator function:

$$I\,(x) = \begin{cases} 1 & if\ x \geq 0, \\ 0 & if\ x < 0. \end{cases} \tag{6}$$

In the acquisition function, $I\,(\mu(p_i) - \lambda_i)$ filters the infeasible levels of parallelism based on GP's estimation. Thus, the SLA guarantee is considered the first priority while tuning.

5.2 Trade-off between Conservative Exploration and Fast Exploitation

The above acquisition function filters infeasible levels of parallelism based on GP's estimation. However, in the region with few observations (i.e., unknown region), the estimation will yield large uncertainty (e.g., the cold start case). Exploring the unknown region is inevitable in vanilla BO since it serves as part of learning for the objective functions. However, aggressive exploration is unfavorable in parallelism tuning, since the SLA cannot be guaranteed.

To tackle the problem, we propose to utilize linearity-based tuning methods to warm up the learning of GP and cope with sudden changes in workload. The linearity-based methods estimate the processing ability per instance and essentially suggest the levels of parallelism of operators based on the ratio between the upstream data rate and the processing ability. Since the relationship between the levels of parallelism and processing abilities is non-linear [41], they cannot converge to the optimal level of parallelism in one step. But the linearity-based methods are suitable for warming up the GP and being conservative exploration to avoid aggressive exploration. Since the aggressive exploration is avoided, CBO can be used in real online environments. We refer to the level of parallelism suggested by linearity-based methods for operator o_i as p_i^{lin} and the level of parallelism suggested by the acquisition function as p_i^{acq}. Concretely, CBO applies p_i^{lin}, when GP's estimation has large uncertainty – in other words, when p_i^{acq} falls in the unknown region. Otherwise, CBO applies p_i^{acq}. DS2 is adopted as the linearity-based method in CBO. Intuitively, in CBO, the surrogate model and the acquisition function serve as *fast exploitation*, and the linearity-based method serves as *conservative exploration*.

CBO uses a scoring function to achieve the trade-off between conservative exploration and fast exploitation. Given a level of parallelism, GP's estimation will be more accurate when the given level of parallelism is closer to the observed levels of parallelism. Thus, we use a scoring function to decide whether p_i^{acq} falls in the unknown region by how far p_i^{acq} is from the current observations. Specifically, we use $d_{nearest}^i$ to denote the minimal value of the distance between p_i^{acq} and the observed level of parallelism in H_i^t.

Algorithm 2 CBO algorithm

Input: A stream job with N operators, real upstream data rates for each operator $\lambda_i{}_{i=0}^{N}$,observations for each operator H_i^t, a threshold for scoring function α

Output: The suggested levels of parallelism P_{job} for the given job

1: Initialize P_{job} as an empty list
2: **for** $i \leftarrow 1 \ldots N$ **do**
3:     Fit GP_i based on H_i^t and obtain p_i^{acq} following Equation 5
4:     $d_{nearest}^i \leftarrow +\infty$
5:     **for** $(p, PA(p))$ in H_i^t **do**
6:         $d_{nearest}^i \leftarrow \min\left(d_{nearest}^i, |p_i^{acq} - p|\right)$
7:     **end for**
8:     **if** $d_{nearest}^i \leq \alpha$ **then**
9:         Append p_i^{acq} to P_{job}
10:     **else**
11:         Obtain p_i^{lin} through linearity-based method
12:         Append p_i^{lin} to P_{job}
13:     **end if**
14: **end for**
15: Apply P_{job} via one reconfiguration
16: **for** $i \leftarrow 1 \ldots N$ **do**
17:     Observe $PA(p_i)$ and append $\langle p_i, PA(p_i)\rangle$ to H_i^t
18: **end for**
19: **return** P_{job}

When $d_{nearest}^i$ is smaller than or equal to a threshold, namely α, CBO applies p_i^{acq}. Otherwise, the linearity-based method is adopted and CBO applies p_i^{lin}. And the augmented observation from the linearity-based method is also added to H_i^t, as a training sample for GP, which warms up GP's learning.

Algorithm 2 presents one tuning step of CBO formally. CBO deals with each operator separately (Line 2), since the tuning problem is decomposed into N sub-problems as Equation 2. CBO first obtains true λ as the job is not in backpressure state. Then it fits a GP model on H_i^t and obtain p_i^{acq} and decides whether to use p_i^{acq} or p_i^{lin} by the scoring function (Line 3 - Line 13). After all the sub-problems are solved, CBO applies the suggested levels of parallelism via one reconfiguration (Line 15) and saves the corresponding observations (Line 17).

5.3 Continuous Tuning via CBO

Given a stream job, the workload of stream data (i.e., source rate) is dynamic and the applied levels of parallelism can become inappropriate for the source rate, thus parallelism tuning will be triggered accordingly. Such a scenario is called *continuous tuning*. CBO adopts GP as the surrogate model to fit the relationship between the levels of parallelism and processing abilities. Although the source rate changes, the relationship between the levels of parallelism and processing abilities of an operator is constant. And the surrogate models (i.e., GPs) in CBO can be reused for continuously tuning a stream application in spite of different source rates. Intrinsically, the GPs in CBO enable the speedup of target tuning via historical observations. In contrast to *one-shot parallelism tuning* methods, CBO is specifically designed to undergo continuous improvement. This means that as the number of historical observations increases, the tuning performance of CBO also improves correspondingly. It employs an acquisition function (Equation 5) to identify suitable level of parallelism based on GPs for fast identification of the optimal levels of parallelism.

6 EFFICIENCY OF CONTTUNE

We first discuss the convergence of ContTune in Section 6.1, and analyze its average complexity of the number of reconfigurations in Section 6.2.

6.1 Analysis of ContTune Convergence

Given an observation $\langle p^j, PA(p^j)\rangle$ in H^t, CBO considers the region round p^j with α as the radius to be a known region (i.e., the region with small uncertainty) and the p^{acq} falls in the known region $[\max(p^j - \alpha, 0), \min(p^j + \alpha, p^{max})]$ will be applied. The closer the level of parallelism to the observed level of parallelism in H^t the greater the confidence level in the surrogate model, i.e. the smaller the $d_{nearest}$ the greater the confidence level. As shown in Figure 3, for H^t with size t, therefore there are t known regions, and we refer to the total size for the t regions as len_{all}. And the maximal bound for the configuration space of the level of parallelism is p^{max}. Given a random variable ranging from 0 to p^{max}, the probability Pr_{use} that it falls in the known region is $\frac{len_{all}}{p^{max}}$. As the number of observations increases, the probability that p^{acq} falls in the known region will also increase and the scoring function will be more likely to recommend p^{acq}. The existence of an upper bound on the workloads implies that the p^{max} is smaller than or equal to a constant value. Implying that as tuning proceeds, and the len_{all} is increasing, the probability $\frac{len_{all}}{p^{max}}$ is increasing. And CBO converges to fast exploitation over time. In fact, the real-world workload is not uniformly distributed, and the probability of hitting the fast exploitation is not less than $\frac{len_{all}}{p^{max}}$.

Figure 3 presents a concrete example. For an operator, CBO has its five levels of parallelism (1,4,9,10,15) in H^t with corresponding processing abilities ($PA(1), PA(4), PA(9), PA(10), PA(15)$), and the surrogate model fitted by H^t of this operator is shown in Figure 3 without showing the confidence interval. In this tuning, the *workload* indicates the upstream data rates received by this operator. There are five known regions, and $len_{all} = 12$. Because $p^{max} = 15$, we get $Pr_{use} = \frac{12}{15} = \frac{4}{5}$ based on this H^t. CBO recommends $p^{acq} = 13$ at this time, and the nearest observed level of parallelism in H^t from p^{acq} is 15. The $d_{nearest}$ is 2, and CBO sets α to 2, and $d_{nearest} \leq \alpha$, so CBO recommends p^{acq} in this round of tuning, otherwise, CBO recommends p^{lin}.

6.2 Average Complexity of the Number of Reconfigurations of ContTune

In the scenario of one-shot parallelism tuning, the efficiency of a tuning method is often evaluated based on the convergence speed. However, considering the long-running nature of jobs in distributed stream data processing systems, it is inappropriate to assess the efficiency of tuning solely based on the speed of convergence in a single tuning time. Instead, we use the average complexity of the number of reconfigurations across continuous tuning scenarios as

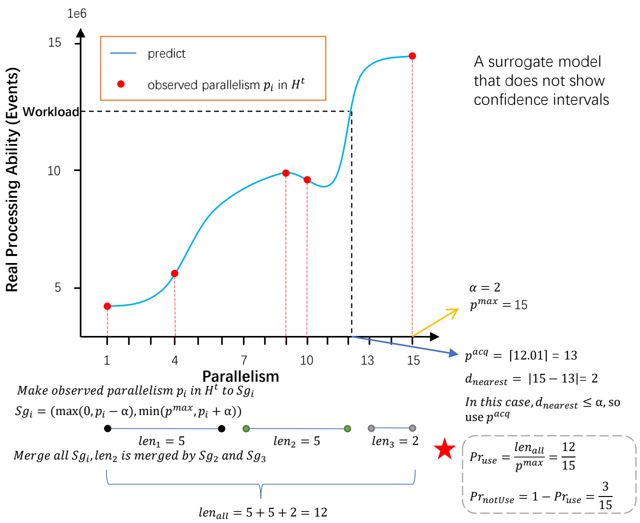

Figure 3: The process of calculating $d_{nearest}$ and probabilities.

a metric to evaluate the efficiency of ContTune. When ContTune tunes long-running jobs, the exploration may delay convergence at a particular tuning time, but it can increase the confidence of the model and yield better results in subsequent tuning times.

We denote the number of tuning for a job as ρ, and the Big phase uses the *Binary Lifting* method, so the complexity of the number of reconfigurations of getting p^{max} (Line 9 in Algorithm 1) is $\log_2 p^{max}$, and the worst-case is that job is under-provisioned at the beginning of each tuning, and needs to reconfigure once (Line 12 in Algorithm 1) at each tuning making job not in backpressure state, and during ρ tuning times, the job needs to be reconfigured ρ times in worst. So the worst-case number of reconfigurations of the Big phase is $\log_2 p^{max} + \rho$, and the worst average complexity of the number of reconfigurations of the Big phase is:

$$O\left(\frac{\log_2 p^{max} + \rho}{\rho}\right). \tag{7}$$

We denote the number of tuning using conservative exploration as χ ($\chi \leq \rho$), then the remaining ($\rho - \chi$) number of tuning is used for fast exploitation. We denote the maximal number of reconfigurations used for tuning of the SOTA method for each tuning as ϕ [2], therefore conservative exploration introduces ($\chi \times \phi$) number of reconfigurations. If CBO employs fast exploitation, CBO uses only one reconfiguration in fast exploitation for each tuning in the best case, a simple example is CBO has every level of parallelism ranging from 1 to p^{max} in H^t. For this best case, fast exploitation introduces ($\rho - \chi$) number of reconfigurations.

In the worst case, if CBO employs fast exploitation when fast exploitation has not yet converged, CBO may find an inappropriate level of parallelism that makes the operator bottlenecked. The suggested inappropriate level of parallelism found by fast exploitation doesn't belong to H^t, because the processing ability in H^t is accurate and not estimated by GP. The worst case would employ CBO [3] once again. We denote the number of reconfigurations introduced by the worst case as ω, and $\omega \leq p^{max}$. For this worst case, fast exploitation introduces $(\rho - \chi) + \omega$ number of reconfigurations.

So the worst average complexity of the number of reconfigurations of CBO is:

$$O\left(\frac{(\chi \times \phi) + (\rho - \chi) + \omega}{\rho}\right). \tag{8}$$

The worst average complexity of the number of reconfigurations of ContTune including the Big phase and CBO is:

$$O\left(\frac{(\log_2 p^{max} + \rho) + ((\chi \times \phi) + (\rho - \chi) + \omega)}{\rho}\right). \tag{9}$$

We assume within ρ tuning times, there is an upper bound on the upstream data rates, and correspondingly, there is an upper bound on the p^{max}, $\phi \leq 3$ and $\omega \leq p^{max}$, so average complexity of the number of reconfigurations of ContTune is $O(1)$.

7 EXPERIMENTAL EVALUATION

In this section, we evaluate ContTune through end-to-end, dynamic scaling experiments with Flink. We verify the efficiency of ContTune in tuning under-provisioned jobs, over-provisioned jobs, stateless jobs and stateful jobs in two scenarios: synthetic workloads and real workloads in Section 7.2 and 7.3. We then validate the design of ContTune by comparing different acquisition functions in Section 7.4. For more experimental results (e.g., the ablation study of Top-K and Mean-reversion), we refer the reader to our technical report [39].

7.1 Setup

Configurations. We run all experiments and use Apache Flink 1.13 configured with 45 TaskManagers, each with 2 slots (maximal level of parallelism per operator = 90) on up to three machines, each with 16 AMD EPYC 7K62 48-Core Processor @2.60GHz cores and 32GB of RAM, running tLinux 2.2 (based on CentOS 7.2.1511).

Queries. We use 6 applications, **WordCount** chosen from original Dhalion publication benchmark [20] and Queries **Q1-3, 5, 8** from Nexmark benchmark (multiple queries over a three entities model representing on online auction system) [2, 4, 33, 66], and 3 real applications **Video streaming**, **ETL** and **Monitoring**.

- **WordCount**, **Q1** and **Q2** contain only stateless operators, such as map and filter and there are 3 operators in WordCount, 3 operators in Q1 and 3 operators in Q2.
- **Q3** contains incremental join, a stateful record-at-a-time two-input operator and there are 5 operators in Q3.
- **Q5** and **Q8** contain two window operators: sliding window, tumbling window join and there are 3 operators in Q5, 4 operators in Q8.
- **Video streaming** contains 3 operators with huge data for Tencent Meeting.
- **ETL** contains 8 operators with complex DAG for Wechat.
- **Monitoring** contains 9 operators with 3 sources and 3 sinks.

Dynamic Workloads Construction. We simulate real-world stream applications by constructing dynamic workloads (i.e., varying their source rate along time). We use the workload unit in

[2] For DS2, $\phi = 3$ [33].
[3] Using CBO rather than ContTune because we have got the real upstream data rate λ, so for under-provisioned jobs, the Big phase is not used.

Table 3: Workload Unit rate (records/s) configuration for WordCount and Nexmark queries on Apache Flink.

	Source		
WordCount	100K	-	-
	Bids	Auctions	Persons
Q1	700K	-	-
Q2	900K	-	-
Q3	-	200K	40K
Q5	80K	-	-
Q8	-	100K	60K

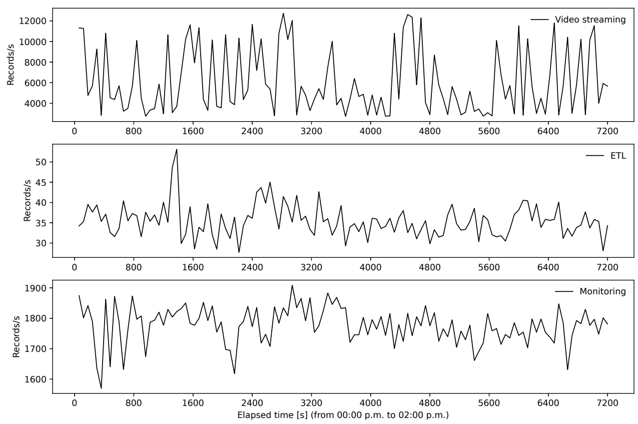

Figure 4: The job workload from Tencent's real cluster from 00:00 p.m. to 02:00 p.m..

Table 3 and simulate the fluctuation using the full permutation of length 10. For example, we generate a period of workloads by varying the source rate as $[9W_u, 2W_u, 3W_u, 10W_u, 1W_u, 4W_u, 5W_u, 8W_u, 6W_u, 7W_u]$, which has 10 tuning times. To simulate the periodicity, we replicate the 10 different source rate, forming a permutation of 20 source rates, which has 20 tuning times. We sample 6 permutations ($per1-6$) for each application, i.e., a total of 120 tuning times for each application. According to the mechanism of applied tuning method, each tuning time may bring different reconfigurations, even zero due to that the tuner is not triggered.

For applications **Video streaming**, **ETL** and **Monitoring**, we collected their real aggregated workloads on Sources from zero p.m. to two p.m. as shown in Figure 4.

Baselines. The baselines are presented below.

- **Dhalion** [20]: Dhalion is a rule-based method which increases the level of parallelism of an operator suffering from backpressure. We adopted the same rule as in its paper.
- **DS2** [33]: DS2 is a linearity-based method and the SOTA parallelism tuning method. We used the same parameters as in its paper.
- **Big + DS2**: The Big phase first ensures that the job is not in backpressure state and get the real upstream data rate λ, and then DS2 tunes the backpressure-free job.
- **Dragster** [41]: Dragster is a Bayesian Optimization-based method, which needs to preset the upper bound of the level of parallelism. Dragster provides two tuning methods, "Online Saddle Point Algorithm" and a "Two-level Online Optimization Framework". The former has shown to accomplish the tuning with fewer reconfigurations, so we used Dragster with "Online Saddle Point Algorithm". For the maximal bound, we use p^{max} in ContTune ($\alpha = 3$) as the maximal bound for each query. Specially, Dragster caches hyper parameters of each tuning for the case that the workload has been processed.
- **ContTune** ($\alpha = 0$): It uses ContTune to tune the levels of parallelism and sets α to 0. Therefore, it will only apply the observed levels of parallelism in H^t or the levels of parallelism suggested by linearity-based tuning methods. Then ContTune ($\alpha = 0$) can be considered as linearity-based tuning methods with cache.
- **ContTune** ($\alpha = 3$): It uses ContTune to tune the levels of parallelism and sets α to 3.
- **Random Search (RS)**: It randomly suggests the levels of parallelism with a given maximal bound. The maximal bound is obtained in the same way as Dragster. The search ends once it finds the same optimal levels of parallelism given by the above methods. Due to the excessive number of reconfigurations required, we enumerate the levels of parallelism and the corresponding processing abilities of all operators beforehand and simulate the search with a program instead (the simulation phase is not accompanied by a real reconfiguration).

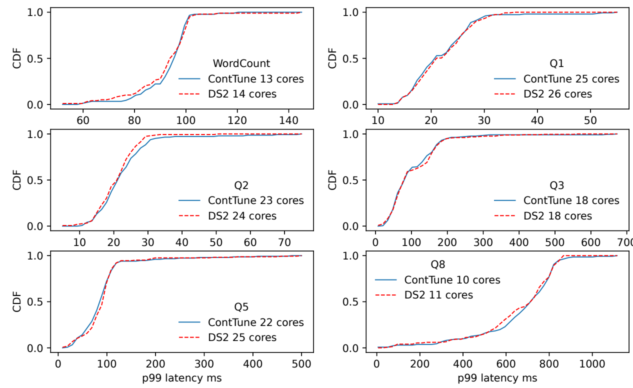

Figure 5: Observed per-record p99 latency CDFs for six quries.

7.2 Evaluations on Synthetic Workloads

We compare ContTune with the baselines on synthetic workloads and make the following observations.

ContTune finds the optimal levels of parallelism via minimal number of reconfigurations. Table 4a shows the average number of reconfigurations per tuning to find the optimal levels of parallelism. In all cases, ContTune ($\alpha = 3$) takes the minimum number of reconfigurations. This shows that ContTune is the most efficient tuning method, and ContTune ($\alpha = 0$) has reduced average **35.42%** ($\frac{(2.40-1.55)}{2.40}$) number of reconfigurations compared to DS2 and ContTune ($\alpha = 3$) has reduced average **46.25%** ($\frac{(2.40-1.29)}{2.40}$) number of reconfigurations compared to DS2. Due to the ability of the GP to fit the processing ability of the level of parallelism near the observed level of parallelism in historical observations H^t,

Table 4: Evaluations on synthetic workloads. Random Search is simulated due to the large number of reconfigurations. Therefore, Random Search is unable to obtain the end-to-end running time, tuning time, and CPU usage. A means the second-best result.

(a) Average number of reconfigurations per tuning.

Baseline	WordCount	Q1	Q2	Q3	Q5	Q8	SUM
Dhalion	3.08	5.36	4.97	3.84	5.59	3.61	4.41
DS2	1.78	2.29	2.29	1.49	3.34	3.21	2.40
Big + DS2	1.73	2.22	2.29	1.66	3.34	3.02	2.37
Dragster	2.75	3.85	3.85	2.75	4.95	3.85	3.67
ContTune ($\alpha = 0$)	1.33	1.61	1.62	1.26	2.01	1.46	1.55
ContTune ($\alpha = 3$)	**1.16**	**1.32**	**1.28**	**1.18**	**1.55**	**1.26**	**1.29**
Random Search	11.16	22.72	17.43	16.18	13.36	8.72	14.93

(b) Maximal number of requested CPU Cores.

Baseline	WordCount	Q1	Q2	Q3	Q5	Q8
Dhalion	17	30	29	22	27	13
DS2	**14**	**26**	**24**	**18**	**25**	**11**
Big + DS2	16	32	32	32	32	16
Dragster	16	32	32	32	32	16
ContTune ($\alpha = 0$)	16	32	32	32	32	16
ContTune ($\alpha = 3$)	16	32	32	32	32	16
Random Search	16	32	32	32	32	16

(c) End-to-end running time (s).

Baseline	WordCount	Q1	Q2	Q3	Q5	Q8
Dhalion	81503.71	87407.70	83633.13	82783.87	88572.63	80546.21
DS2	76658.36	76397.77	76144.72	75148.37	80467.74	79644.56
Big + DS2	76762.78	76236.57	76248.81	75607.61	80230.45	78697.58
Dragster	81062.96	80691.36	80405.64	78828.27	84426.93	81806.26
ContTune ($\alpha = 0$)	75518.32	75050.01	74931.15	74636.20	77506.44	75126.16
ContTune ($\alpha = 3$)	**75213.18**	**74560.88**	**74350.65**	**74439.40**	**76626.16**	**74772.08**

(d) Tuning time (s).

Baseline	WordCount	Q1	Q2	Q3	Q5	Q8
Dhalion	8226.54	9441.17	8888.33	7870.00	11928.58	8032.83
DS2	4624.70	3991.42	4014.15	2955.07	7128.68	7163.79
Big + DS2	4641.78	3901.92	4100.27	3414.64	6941.12	6697.58
Dragster	8569.01	7685.27	7681.32	6757.03	11533.06	9570.16
ContTune ($\alpha = 0$)	3394.57	2725.56	2818.33	2476.41	4209.26	3090.27
ContTune ($\alpha = 3$)	**2947.20**	**2293.70**	**2206.18**	**2304.35**	**3324.88**	**2736.19**

(e) The percentage of backlogged data.

Baseline	WordCount	Q1	Q2	Q3	Q5	Q8
Dhalion	12.79 (%)	28.31 (%)	19.39 (%)	22.03 (%)	27.43 (%)	7.38 (%)
DS2	1.92 (%)	4.60 (%)	2.86 (%)	3.53 (%)	8.03 (%)	1.52 (%)
Big + DS2	1.52 (%)	3.22 (%)	1.96 (%)	2.33 (%)	7.00 (%)	1.40 (%)
Dragster	6.08 (%)	7.97 (%)	7.11 (%)	2.52 (%)	7.49 (%)	5.14 (%)
ContTune ($\alpha = 0$)	**1.01 (%)**	2.04 (%)	**1.31 (%)**	1.76 (%)	6.86 (%)	0.66 (%)
ContTune ($\alpha = 3$)	1.04 (%)	**1.72 (%)**	1.34 (%)	**1.59 (%)**	**6.48 (%)**	**0.63 (%)**

(f) Total CPU cost (Cores × second).

Baseline	WordCount	Q1	Q2	Q3	Q5	Q8
Dhalion	540840.16	**874697.97**	**796300.50**	**692169.32**	**863432.32**	**520995.33**
DS2	550029.28	906065.83	821041.00	698066.74	876788.87	524793.87
Big + DS2	562825.71	929565.85	853817.58	732693.26	899933.87	535530.00
Dragster	570678.66	947174.81	865663.73	736326.66	930962.75	543605.93
ContTune ($\alpha = 0$)	548514.47	903635.23	820388.86	698339.79	878424.08	533789.47
ContTune ($\alpha = 3$)	**532022.10**	905940.99	817178.90	699401.10	878743.96	533849.47

Table 5: CPU cores requested of ContTune and DS2 when they second face the same maximal upstream data rate $10 \times W_u$.

Queries	ContTune ($\alpha = 3$)	DS2
WordCount	**13** CPU cores	14 CPU cores
Q1	**25** CPU cores	26 CPU cores
Q2	**23** CPU cores	24 CPU cores
Q3	18 CPU cores	18 CPU cores
Q5	**22** CPU cores	25 CPU cores
Q8	**10** CPU cores	11 CPU cores

which helps ContTune hit the minimal number of CPU cores. In all experiments, ContTune applies the minimal number of CPU cores at the end of each tuning as shown in Table 5. Figure 5 shows the stable job latency of ContTune and DS2 (SOTA method) for the same maximal workload at the end of tuning is similar in 6 jobs, where ContTune applies 1,1,1,0,3,1 (cf. Table 5) CPU cores less than DS2. Despite using fewer CPU resources, the latency of ContTune tuned jobs is essentially the same as that of DS2 tuned jobs.

ContTune finds the optimal levels of parallelism via minimal running time. The total end-to-end running time as show in Table 4c consists of three parts (1) job ideal running time, 72000 (s); (2) time of processing buffered data as show in Table 6; (3) tuning time (including the time of reconfigurations) as shown in Table 4d. Each source of the job generates data for 600 seconds, so the job ideal running time is 600×120 $(20 \times 6\ pers) = 72000$ (s). The inappropriate configurations will make job under-provisioned and unprocessed data buffered in the queue, and needs time to solve these buffered data. Besides, each method spends time on finding the optimal levels of parallelism by making reconfigurations. In all cases, ContTune achieves both minimum running time in Table 4c and tuning time in Table 4d compared to other methods. This shows that ContTune is the most efficient tuning method, and ContTune ($\alpha = 0$) has reduced average **2.52%** ($\frac{(464461.52-452768.28)}{464461.52}$) end-to-end running times compared to DS2 and ContTune ($\alpha = 3$) has reduced average **3.12%** ($\frac{(464461.52-449962.35)}{464461.52}$) end-to-end running times compared to DS2. ContTune ($\alpha = 0$) has reduced average **37.36%** ($\frac{(29877.81-18714.4)}{29877.81}$) end-to-end tuning time compared to DS2 and ContTune ($\alpha = 3$) has reduced average **47.08%** ($\frac{(29877.81-15812.5)}{29877.81}$) end-to-end tuning time compared to DS2.

Table 6: Total time (s) of processing buffered data.

Baseline	WordCount	Q1	Q2	Q3	Q5	Q8
Dhalion	1277.17	5966.53	2744.8	2913.87	4644.05	513.38
DS2	**33.66**	406.35	130.57	193.3	1339.06	480.77
Big + DS2	121	334.65	148.54	192.97	1289.33	**0.04**
Dragster	493.95	1006.09	724.32	**71.24**	**893.87**	236.1
ContTune ($\alpha = 0$)	123.75	324.45	**112.82**	159.79	1297.18	35.89
ContTune ($\alpha = 3$)	265.98	**267.18**	144.47	135.05	1301.28	35.89

ContTune temporarily requests more CPU Cores and could quickly eliminate the backlogged data. We show the maximal number of CPU Cores requested by each method in Table 4b and the total CPU cost (Core × second) in Table 4f. Dhalion and DS2 request the less maximal number of requested CPU Cores than other methods as shown in Table 4b, because both Dhalion and DS2 tune levels of parallelism from small to big for under-provisioned jobs. And the maximum number of CPU Cores requested by ContTune is more than Dhalion and DS2 due to its Big phase. However, due to the efficiency of finding the optimal levels of parallelism, the total CPU cost is only a little more than DS2 as shown in Table 4f. In all cases, Dhalion achieves the minimum total CPU cost, ContTune and DS2 use almost the same total CPU cost. ContTune ($\alpha = 3$) uses average

Table 7: Tuning WordCount on synthetic workloads, κ means all the number of reconfigurations, θ means the number of reconfigurations for eliminating backpressure and ζ means the number of reconfigurations for over-provisioned jobs.

	κ		θ		ζ	
	DS2	ContTune	DS2	ContTune	DS2	ContTune
$per1$	35	24	22	11	13	10
$per2$	38	23	25	13	13	9
$per3$	35	22	26	13	9	8

0.22% ($\frac{(4376785.59-4367136.52)}{4367136.52}$) total CPU cost smaller than DS2 and ContTune ($\alpha = 3$) uses average **1.84%** ($\frac{(4367136.52-4288435.6)}{4288435.6}$) total CPU cost bigger than Dhalion. So ContTune temporarily requests more CPU Cores.

When the job is reconfigured, the data in the processing queue that have not been processed are the backlogged data as shown in Table 4e. These data must wait until the job completes the reconfiguration before they can be processed (e.g., Flink, Samza and Heron use the kill-and-restart method to execute reconfigurations [45]), and the waiting time will increase the job latency. Table 4e shows that Big + DS2, ContTune ($\alpha = 0$) and ContTune ($\alpha = 3$) have less backlogged data than DS2, Dragster and Dhalion thanks to the Big phase. Both ContTune ($\alpha = 0$) and ContTune ($\alpha = 3$) achieve the best or second-best result as shown in Table 4e, and ContTune ($\alpha = 3$) has reduced average **89.09%** ($\frac{(117.33-12.8)}{117.33}$) number of backlogged data compared to Dhalion and ContTune ($\alpha = 3$) has reduced average **43.01%** ($\frac{(22.46-12.8)}{22.46}$) number of backlogged data compared to DS2. Big + DS2, ContTune ($\alpha = 0$) and ContTune ($\alpha = 3$) also have less time of processing these backlogged data than DS2, Dragster and Dhalion thanks to the Big phase as shown in Table 6, and ContTune ($\alpha = 3$) has reduced average **88.10%** ($\frac{(18059.8-2149.85)}{18059.8}$) time of processing backlogged data compared to Dhalion and ContTune ($\alpha = 3$) has reduced average **16.79%** ($\frac{(2583.71-2149.85)}{2583.71}$) time of processing backlogged data compared to DS2. The Big phase temporarily requests more CPU Cores in order to quickly eliminate these backlogged data.

ContTune prioritizes the job SLA. Table 7 presents the total reconfigurations to find the optimal levels of parallelism and the number of reconfigurations used to eliminating backpressure. We observe that ContTune is faster than DS2 in eliminating the under-provisioned jobs and uses less number of reconfigurations to find the optimal levels of parallelism. This could be contributed to the design of the Big-small algorithm. In the Big phase, ContTune quickly allocates sufficient resources for the under-provisioned jobs. Then, at the beginning of the Small phase, the SLA of the job is satisfied, and the subsequent tuning is used only to improve the CPU resource utilization.

ContTune effectively utilizes the previous tuning observations to speed up the tuning process. The Big phase algorithm can quickly eliminate job backpressure and obtain the λ. DS2 employs other methods (e.g., SnailTrail [30]) to estimate the λ. Therefore, in this experiment, to validate the efficiency of tuning methods after obtaining the λ, we proactively obtained the λ of each operator, and focused on comparing the efficiency of CBO based on BO with the linear search of DS2 to demonstrate the efficiency of

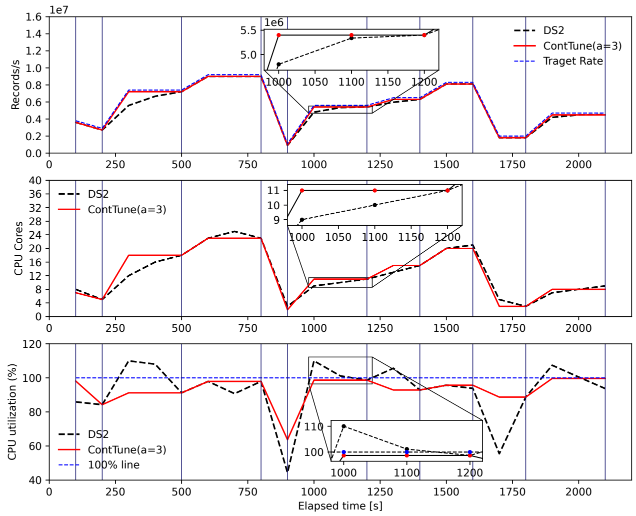

Figure 6: Aggregated Records/s of sources and CPU cores and CPU utilization of Q2 on latter 10 tuning times.

ContTune. Figure 6 presents the performance of latter 10 tuning times for **Q2**. The real CPU utilization does not exceed 100%. Any CPU utilization above the *100% line* in Figure 6 means that the job is *under-provisioned*. We observe that ContTune performs better than DS2 in the latter 10 tuning times. It takes full advantage of historical observations in the face of a workload that has processed before, rather than starting from the scratch like DS2. In all 10 tuning times, the number of reconfigurations used for ContTune is smaller than or equal to the number of reconfigurations used for DS2. And, the CPU cores used is smaller than or equal to the CPU cores used for DS2, and the CPU cores used at the 1^{st}, 5^{th}, 8^{th}, and 10^{th} tuning time smaller than that used for DS2.

7.3 Evaluations on Real Workloads

Figure 7 shows the total number of reconfigurations on real workloads. Since real workloads do not significantly vary as much as synthetic workloads, the number of tuning may also vary depending on the *controller*, for example, *controller* triggers 26, 21 and 15 tuning times for **Video Streaming**, **ETL** and **Monitoring**. Figure 7 shows that compared to DS2, ContTune ($\alpha = 0$) reduced **24.14%** ($\frac{(58-44)}{58}$) number of reconfigurations on **Video streaming**, and **25.64%** ($\frac{(39-29)}{39}$) number of reconfigurations on **ETL**, and reduced **45%** ($\frac{(40-22)}{40}$) number of reconfigurations on the **Monitoring**. And ContTune ($\alpha = 3$) reduced **44.83%** ($\frac{(58-32)}{58}$) number of reconfigurations on **Video streaming**, and **43.59%** ($\frac{(39-22)}{39}$) number of reconfigurations on **ETL**, and reduced **57.5%** ($\frac{(40-17)}{40}$) number of reconfigurations on **Monitoring**. The main reason that ContTune ($\alpha = 0$) and Dragster are not as efficient as the case on synthetic workloads is that the period of real workload we captured do not necessarily contain multiple workload replication, making it unlikely to apply a simple caching mechanism. So the efficiency of the above methods is compromised. In ContTune ($\alpha = 3$) the fitting ability of the GP compensates for this drawback better.

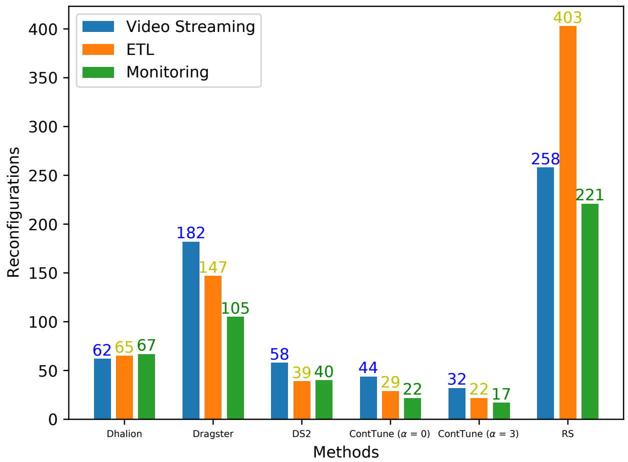

Figure 7: Total reconfigurations on real workloads.

Table 8: Total reconfigurations of Q1 on synthetic workloads with different acquisition function (AF).

Baseline	$per1$	$per2$	$per3$	$per4$	$per5$	$per6$	sum
DS2	51	44	47	41	46	46	275
CBO (AF 4)	26	25	27	28	32	26	164
CBO (AF 5)	22	23	27	28	25	26	151

7.4 Analysis of ContTune

Comparison of Different Acquisition Functions. We propose a carefully designed acquisition function (Equation 5) that allows ContTune to suggest the optimal levels of parallelism while strictly satisfying SLA, and we compare it with CEI (Equation 4). Table 8 shows that CBO with Equation 5 has less number of reconfigurations than CBO with CEI (Equation 4). CEI does not consider the constraint safety-critical, and it may suggest infeasible levels of parallelism during tuning (e.g., trying the level of parallelism p_i with large $p_i^* - p_i$ but small $Pr[f(p_i) \geq \lambda]$). Once these levels of parallelism are suggested, additional reconfigurations are required to keep the job from backpressure. Mean represents exploitation in BO, Equation 5 uses only mean and the surrogate model compose of *fast exploitation*. These designs will avoid re-creating application under-provisioned in the Small phase and reduce the number of reconfigurations.

8 RELATED WORK

Configuration of distributed stream data processing systems. Many distributed stream data processing systems have a wide range of configuration parameters, and tuning these parameters can improve performance and reduce resource utilization. Operator scaling techniques elastically tunes the amount of each operator's needed resource in order to be suitable for workload variations. The user can horizontally or vertically scale operators. Horizontally scaling deploys parallel instances of the same operator leveraging *Data Parallelization*, and each instance processes a share of the input stream. Vertically scaling focuses on tuning computer resource (e.g., CPU time, instance memory) of the existing instance instead of tuning the level of parallelism. In this survey [10], horizontally scaling is more efficient than vertically scaling, so ContTune focuses on horizontally scaling. There are many researches for horizontally scaling. [5, 11, 15, 20, 24, 28, 29, 69, 71, 72] are rule-based tuning methods, their effect depends on the setting of rules and thresholds, and they often propose different rules and thresholds for different systems, so the applicability of their methods is poor. [33, 50] use the performance relation between workload and operator process ability, but they do not know the non-linear relation between the level of parallelism and process ability, so they use other reconfigurations to tune the levels of parallelism. [18, 41] use Bayesian Optimization to tune the levels of parallelism, and the shorts of aggressive exploration brings many reconfigurations, but the use of historical observations is helpful to establish the surrogate model of the level of parallelism and process ability.

Bayesian Optimization. Bayesian Optimization (BO) is a SOTA optimization framework for optimizing of expensive-to-evaluate black-box function. It has been extensively used in many scenarios, including hyperparameter tuning [7, 40, 67], experimental design [21] and controller tuning [8, 17, 19, 46]. BO uses an acquisition function to suggest the next configuration that trades off exploration (i.e., acquiring new knowledge) and exploitation (i.e., making decisions based on existing knowledge) [36]. Instead of evaluating the expensive black-box function, the acquisition function relies on a surrogate model that is cheap to compute, and thus can be efficiently minimized in each iteration. BO has been adopted to configure the parameters of data management systems [6, 12, 16, 38, 75–77]. However, its favor of exploration causes applying configurations in unknown region with potentially bad performance, which is unacceptable for mission-critical applications. For online tuning with SLA requirement, we propose the CBO algorithm that utilizes the safe configurations generated from linearity-based methods as conservative exploration.

9 CONCLUSION

In this paper, we describe and evaluate *ContTune*, a continuous tuning system for elastic stream processing using the *Big-small* algorithm, the Big phase and the Small phase (CBO). ContTune uses the Big phase to quickly eliminate job backpressure and buffered data in the queue, and decouple tuning from the topological graph. The Big phase can quickly satisfy SLA for under-provisioned jobs, and the Small phase can quickly find optimal the level of parallelism for over-provisioned jobs. CBO uses GP as the surrogate model to fit the non-linear relationship for continuous tuning and introduces the SOTA *one-shot parallelism tuning* method as conservative exploration to avoid SLA violations. ContTune performs tuning with $O(1)$ average complexity of the number of reconfigurations. ContTune achieves the best results for benchmarks or real applications, synthetic or real workloads, compared to the SOTA method DS2.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (Nos. 62272054, 62192784, U22B2037, 61832001, 62172056), Xiaomi Young Talents Program and the Fundamental Research Funds for the Central Universities (2023PY11).

REFERENCES

[1] 2019. Esper. Retrieved\from\https://www.espertech.com/.
[2] September 18, 2023. ApacheBeamNexmarkbenchmarksuite. https://beam.apache.org/documentation/sdks/java/nexmark.
[3] September 18, 2023. Network buffer. https://nightlies.apache.org/flink/flink-docs-release-1.15/docs/deployment/memory/network_mem_tuning/.
[4] September 18, 2023. NEXMark benchmark. http://datalab.cs.pdx.edu/niagaraST/NEXMark.
[5] Yanif Ahmad, Bradley Berg, Uğur Cetintemel, Mark Humphrey, Jeong-Hyon Hwang, Anjali Jhingran, Anurag Maskey, Olga Papaemmanouil, Alexander Rasin, Nesime Tatbul, et al. 2005. Distributed operation in the borealis stream processing engine. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 882–884.
[6] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *SIGMOD Conference*. ACM, 1009–1024.
[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *NIPS*. 2546–2554.
[8] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. 2014. An experimental comparison of Bayesian optimization for bipedal locomotion. In *ICRA*. IEEE, 1951–1958.
[9] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).
[10] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. 2022. Runtime Adaptation of Data Stream Processing Systems: The State of the Art. *ACM Computing Surveys (CSUR)* 54, 11s (2022), 1–36.
[11] Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter Pietzuch. 2013. Integrating scale out and fault tolerance in stream processing using operator state management. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. 725–736.
[12] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. 2021. CGPTuner: a Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations Under Varying Workload Conditions. *Proc. VLDB Endow.* 14, 8 (2021), 1401–1413.
[13] Xin Chen, Ymir Vigfusson, Douglas M Blough, Fang Zheng, Kun-Lung Wu, and Liting Hu. 2017. GOVERNOR: Smoother stream processing through smarter backpressure. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 145–154.
[14] Bonaventura Del Monte, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2020. Rhino: Efficient management of very large distributed state for stream processing engines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2471–2486.
[15] S. Desimone. 2017. Storage reimagined for a streamingworld. https://pravega.io/blog/2017/04/09/storage-reimagined-for-a-streaming-world/.
[16] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning Database Configuration Parameters with iTuned. *Proc. VLDB Endow.* 2, 1 (2009), 1246–1257.
[17] Marcello Fiducioso, Sebastian Curi, Benedikt Schumacher, Markus Gwerder, and Andreas Krause. 2019. Safe Contextual Bayesian Optimization for Sustainable Room Temperature PID Control Tuning. In *IJCAI*. ijcai.org, 5850–5856.
[18] Lorenz Fischer, Shen Gao, and Abraham Bernstein. 2015. Machines tuning machines: Configuring distributed stream processors with bayesian optimization. In *2015 IEEE International conference on cluster computing*. IEEE, 22–31.
[19] Lorenz Fischer, Shen Gao, and Abraham Bernstein. 2015. Machines Tuning Machines: Configuring Distributed Stream Processors with Bayesian Optimization. In *CLUSTER*. IEEE Computer Society, 22–31.
[20] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. 2017. Dhalion: self-regulating stream processing in heron. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1825–1836.
[21] Adam Foster, Martin Jankowiak, Eli Bingham, Paul Horsfall, Yee Whye Teh, Tom Rainforth, and Noah D. Goodman. 2019. Variational Bayesian Optimal Experimental Design. In *NeurIPS*. 14036–14047.
[22] Tom ZJ Fu, Jianbing Ding, Richard TB Ma, Marianne Winslett, Yin Yang, and Zhenjie Zhang. 2017. DRS: Auto-scaling for real-time stream analytics. *IEEE/ACM Transactions on networking* 25, 6 (2017), 3338–3352.
[23] Bugra Gedik, Habibe G Özsema, and Özcan Öztürk. 2016. Pipelined fission for stream programs with dynamic selectivity and partitioned state. *J. Parallel and Distrib. Comput.* 96 (2016), 106–120.
[24] Buğra Gedik, Scott Schneider, Martin Hirzel, and Kun-Lung Wu. 2013. Elastic scaling for data stream processing. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (2013), 1447–1463.
[25] Michael A. Gelbart, Jasper Snoek, and Ryan P. Adams. 2014. Bayesian Optimization with Unknown Constraints. In *UAI*. AUAI Press, 250–259.
[26] Lukasz Golab and M Tamer Özsu. 2003. Issues in data stream management. *ACM Sigmod Record* 32, 2 (2003), 5–14.
[27] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. *2010 International Conference on Network and Service Management* (2010), 9–16.
[28] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. 2012. Streamcloud: An elastic and scalable data streaming system. *IEEE Transactions on Parallel and Distributed Systems* 23, 12 (2012), 2351–2365.
[29] 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*. 13–22.
[30] Moritz Hoffmann, Andrea Lattuada, John Liagouris, Vasiliki Kalavri, Desislava C. Dimitrova, Sebastian Wicki, Zaheer Chothia, and Timothy Roscoe. 2018. SnailTrail: Generalizing Critical Paths for Online Analysis of Distributed Dataflows. In *NSDI*. USENIX Association, 95–110.
[31] Moritz Hoffmann, Andrea Lattuada, Frank McSherry, Vasiliki Kalavri, John Liagouris, and Timothy Roscoe. 2019. Megaphone: Latency-conscious state migration for distributed streaming dataflows. *Proceedings of the VLDB Endowment* 12, 9 (2019), 1002–1015.
[32] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2019. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer.
[33] Vasiliki Kalavri, John Liagouris, Moritz Hoffmann, Desislava Dimitrova, Matthew Forshaw, and Timothy Roscoe. 2018. Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 783–798.
[34] Nikos R Katsipoulakis, Alexandros Labrinidis, and Panos K Chrysanthis. 2017. A holistic view of stream partitioning costs. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1286–1297.
[35] 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 *Proceedings of the 2016 International Conference on Management of Data*. 555–569.
[36] Andreas Krause and Cheng Soon Ong. 2011. Contextual Gaussian Process Bandit Optimization. In *NIPS*. 2447–2455.
[37] 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.
[38] Mayuresh Kunjir and Shivnath Babu. 2020. Black or White? How to Develop an AutoTuner for Memory-based Analytics. In *SIGMOD Conference*. ACM, 1667–1683.
[39] Jinqing Lian, Xinyi Zhang, Yingxia Shao, Zenglin Pu, Qingfeng Xiang, Yawen Li, and Bin Cui. 2023. ContTune: Continuous Tuning by Conservative Bayesian Optimization for Distributed Stream Data Processing Systems. arXiv:2309.12239 [cs.DB]
[40] Edo Liberty, Zohar S. Karnin, Bing Xiang, Laurence Rouesnel, Baris Coskun, Ramesh Nallapati, Julio Delgado, Amir Sadoughi, Yury Astashonok, Piali Das, Can Balioglu, Saswata Chakravarty, Madhav Jha, Philip Gautier, David Arpin, Tim Januschowski, Valentin Flunkert, Yuyang Wang, Jan Gasthaus, Lorenzo Stella, Syama Sundar Rangapuram, David Salinas, Sebastian Schelter, and Alex Smola. 2020. Elastic Machine Learning Algorithms in Amazon SageMaker. In *SIGMOD Conference*. ACM, 731–737.
[41] Yang Liu, Huanle Xu, and W. Lau. 2022. Online Resource Optimization for Elastic Stream Processing with Regret Guarantee. *Proceedings of the 51st International Conference on Parallel Processing* (2022).
[42] Björn Lohrmann, Peter Janacik, and Odej Kao. 2015. Elastic stream processing with latency guarantees. In *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 399–410.
[43] Federico Lombardi, Leonardo Aniello, Silvia Bonomi, and Leonardo Querzoni. 2017. Elastic symbiotic scaling of operators and resources in stream processing systems. *IEEE Transactions on Parallel and Distributed Systems* 29, 3 (2017), 572–585.
[44] Luo Mai, Kai Zeng, Rahul Potharaju, Le Xu, Steve Suh, Shivaram Venkataraman, Paolo Costa, Terry Kim, Saravanan Muthukrishnan, Vamsi Kuppa, et al. 2018. Chi: A scalable and programmable control plane for distributed stream processing systems. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1303–1316.
[45] Yancan Mao, Yuan Huang, Runxin Tian, Xin Wang, and Richard TB Ma. 2021. Trisk: Task-Centric Data Stream Reconfiguration. In *Proceedings of the ACM Symposium on Cloud Computing*. 214–228.
[46] Alonso Marco, Felix Berkenkamp, Philipp Hennig, Angela P. Schoellig, Andreas Krause, Stefan Schaal, and Sebastian Trimpe. 2017. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization. In *ICRA*. IEEE, 1557–1563.
[47] Ruben Mayer, Christian Mayer, Muhammad Adnan Tariq, and Kurt Rothermel. 2016. Graphcep: Real-time data analytics using parallel complex event and graph processing. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. 309–316.

[48] Ruben Mayer, Ahmad Slo, Muhammad Adnan Tariq, Kurt Rothermel, Manuel Gräber, and Umakishore Ramachandran. 2017. SPECTRE: Supporting consumption policies in window-based parallel complex event processing. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. 161–173.

[49] Ruben Mayer, Muhammad Adnan Tariq, and Kurt Rothermel. 2017. Minimizing communication overhead in window-based parallel complex event processing. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. 54–65.

[50] Yuan Mei, Luwei Cheng, Vanish Talwar, Michael Y Levin, Gabriela Jacques-Silva, Nikhil Simha, Anirban Banerjee, Brian Smith, Tim Williamson, Serhat Yilmaz, et al. 2020. Turbine: Facebook's service management platform for stream processing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1591–1602.

[51] Gabriele Mencagli, Massimo Torquati, and Marco Danelutto. 2018. Elastic-PPQ: A two-level autonomic system for spatial preference query processing over dynamic data streams. *Future Generation Computer Systems* 79 (2018), 862–877.

[52] Gabriele Mencagli, Massimo Torquati, Marco Danelutto, and Tiziano De Matteis. 2017. Parallel continuous preference queries over out-of-order and bursty data streams. *IEEE Transactions on Parallel and Distributed Systems* 28, 9 (2017), 2608–2624.

[53] Gabriele Mencagli, Massimo Torquati, Fabio Lucattini, Salvatore Cuomo, and Marco Aldinucci. 2018. Harnessing sliding-window execution semantics for parallel stream processing. *J. Parallel and Distrib. Comput.* 116 (2018), 74–88.

[54] Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, David Garcia-Soriano, Nicolas Kourtellis, and Marco Serafini. 2015. The power of both choices: Practical load balancing for distributed stream processing engines. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 137–148.

[55] Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, Nicolas Kourtellis, and Marco Serafini. 2016. When two choices are not enough: Balancing at scale in distributed stream processing. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 589–600.

[56] Vu Nguyen, Sunil Gupta, Santu Rana, Cheng Li, and Svetha Venkatesh. 2017. Regret for Expected Improvement over the Best-Observed Value and Stopping Condition. In *ACML (Proceedings of Machine Learning Research)*, Vol. 77. PMLR, 279–294.

[57] Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. 2017. Samza: stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1634–1645.

[58] Nicoló Rivetti, Emmanuelle Anceaume, Yann Busnel, Leonardo Querzoni, and Bruno Sericola. 2016. Online scheduling for shuffle grouping in distributed stream processing systems. In *Proceedings of the 17th International Middleware Conference*. 1–12.

[59] Henriette Röger and Ruben Mayer. 2019. A comprehensive survey on parallelization and elasticity in stream processing. *ACM Computing Surveys (CSUR)* 52, 2 (2019), 1–37.

[60] Omran Saleh, Heiko Betz, and Kai-Uwe Sattler. 2015. Partitioning for scalable complex event processing on data streams. In *New Trends in Database and Information Systems II: Selected papers of the 18th East European Conference on Advances in Databases and Information Systems and Associated Satellite Events, ADBIS 2014 Ohrid, Macedonia, September 7-10, 2014 Proceedings II*. Springer, 185–197.

[61] Scott Schneider, Joel Wolf, Kirsten Hildrum, Rohit Khandekar, and Kun-Lung Wu. 2016. Dynamic load balancing for ordered data-parallel regions in distributed streaming systems. In *Proceedings of the 17th International Middleware Conference*. 1–14.

[62] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. 2009. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995* (2009).

[63] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. 2010. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *ICML*. Omnipress, 1015–1022.

[64] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. 2014. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 147–156.

[65] Yi-Cheng Tu, Song Liu, Sunil Prabhakar, and Bin Yao. 2006. Load shedding in stream databases: a control-based approach. (2006).

[66] Pete Tucker, Kristin Tufte, Vassilis Papadimos, and David Maier. 2008. *Nexmark–a benchmark for queries over data streams (draft)*. Technical Report. Technical Report. Technical report, OGI School of Science & Engineering at ....

[67] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Sequential Model-Free Hyperparameter Tuning. In *ICDM*. IEEE Computer Society, 1033–1038.

[68] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. 2019. Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology* 17, 1 (2019), 26–40.

[69] Sai Wu, Ying Li, Haoqi Zhu, Junbo Zhao, and Gang Chen. 2022. Dynamic index construction with deep reinforcement learning. *Data Science and Engineering* 7, 2 (2022), 87–101.

[70] Yingjun Wu and Kian-Lee Tan. 2015. ChronoStream: Elastic stateful stream computation in the cloud. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 723–734.

[71] Le Xu, Boyang Peng, and Indranil Gupta. 2016. Stela: Enabling stream processing systems to scale-in and scale-out on-demand. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 22–31.

[72] Qiang Yin, Jianhua Wang, Sheng Du, Jianquan Leng, Jintao Li, Yinhao Hong, Feng Zhang, Yunpeng Chai, Xiao Zhang, Xiaonan Zhao, et al. 2022. An Adaptive Elastic Multi-model Big Data Analysis and Information Extraction System. *Data Science and Engineering* 7, 4 (2022), 328–338.

[73] Nikos Zacheilas, Nikolas Zygouras, Nikolaos Panagiotou, Vana Kalogeraki, and Dimitrios Gunopulos. 2016. Dynamic load balancing techniques for distributed complex event processing systems. In *Distributed Applications and Interoperable Systems: 16th IFIP WG 6.1 International Conference, DAIS 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings 16*. Springer, 174–188.

[74] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*. 423–438.

[75] Xinyi Zhang, Zhuo Chang, Yang Li, Hong Wu, Jian Tan, Feifei Li, and Bin Cui. 2022. Facilitating Database Tuning with Hyper-Parameter Optimization: A Comprehensive Experimental Evaluation. *Proc. VLDB Endow.* 15, 9 (2022), 1808–1821.

[76] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. ResTune: Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. In *SIGMOD Conference*. ACM, 2102–2114.

[77] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. Towards Dynamic and Safe Configuration Tuning for Cloud Databases. In *SIGMOD Conference*. ACM, 631–645.