8-8-2023

Resource optimization of edge servers dealing with priority-based workloads by utilizing service level objective-aware virtual rebalancing

Amna Shahid
Mississippi State University, amnashahid901@gmail.com

Resource optimization of edge servers dealing with priority-based workloads by utilizing service level objective-aware virtual rebalancing

By

Amna Shahid

Approved by:

Samee U. Khan (Major Professor)
Chaomin Luo
Yu Luo
Jenny Du (Graduate Coordinator)
Jason Keith (Dean, Bagley College of Engineering)

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical and Computer Engineering
Department of Electrical and Computer Engineering

Mississippi State, Mississippi

August 2023

Name: Amna Shahid

Date of Degree: August 8, 2023

Institution: Mississippi State University

Major Field: Electrical and Computer Engineering

Major Professor: Samee U. Khan

Title of Study: Resource optimization of edge servers dealing with priority-based workloads by utilizing service level objective-aware virtual rebalancing

Pages in Study 58

Candidate for Degree of Master of Science

IoT enables profitable communication between sensor/actuator devices and the cloud. Slow network causing Edge data to lack Cloud analytics hinders real-time analytics adoption. VRebalance solves priority-based workload performance for stream processing at the Edge. BO is used in VRebalance to prioritize workloads and find optimal resource configurations for efficient resource management. Apache Storm platform was used with RIoTBench IoT benchmark tool for real-time stream processing. Tools were used to evaluate VRebalance. Study shows VRebalance is more effective than traditional methods, meeting SLO targets despite system changes. VRebalance decreased SLO violation rates by almost 30% for static priority-based workloads and 52.2% for dynamic priority-based workloads compared to hill climbing algorithm. Using VRebalance decreased SLO violations by 66.1% compared to Apache Storm's default allocation.

DEDICATION

Dedicated to my exceptional parents: Shahid Rasheed & Razia Shahid, my beloved sisters and friends whose tremendous support and cooperation led me to this accomplishment.

ACKNOWLEDGEMENTS

The research presented in this thesis is submitted in IEEE Cloud Summit 2023 [52]. The work done in this thesis has been made possible through the financial assistance of the NSF CNS 1911012 grant and the NSF CNS 2124908 grant. Their funding has played a crucial role in facilitating the acquisition of necessary resources, conducting experiments, and disseminating the findings of this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The successful implementation of IoT applications is heavily dependent on the continuous execution of data stream processing from various devices. The fundamental purpose of this processing is to ensure seamless and effective data integration, as well as adequate control operations. Additionally, optimal data integration within intelligent systems is an essential requirement for IoT applications, as highlighted by relevant literature [1], [2]. Cloud-based data processing for Internet of Things (IoT) systems is less than ideal as it is constrained in its ability to support real-time data acquisition because of network jitters and delays [3]. The process of extracting valuable information for subsequent actions from data may entail imposing strict time limitations, which may necessitate processing of data in near real-time to identify relevant patterns. Contemporary business entities are availing themselves to large-scale data processing frameworks to extract substantial insights from voluminous data, otherwise known as "big data." The intended purpose of such a strategy is to ascertain and capitalize on novel business prospects, scrutinize consumer behavior, curtail operating expenses, and accelerate and boost decision-making processes. As per the findings of IBM, an immense quantity of data is produced on a daily basis, with an estimated volume of 2.5 quintillion bytes [4].

In a report presented by SAS [5], it was observed that prominent establishments are employing diverse big data analytics tools for the purpose of formulating customer-centric product and service strategies, enhancing prompt decision-making abilities, and mitigating operational

expenditures. In contemporary data analysis, big data is defined by three integral and interrelated elements, namely Volume, Variety, and Velocity [6]. In the realm of data analysis, the measure of data present is referred to as volume, while the heterogeneous assortment of data types and structures is categorized as variety, and the metrical representation of the rate at which data is received is known as velocity. Stream processing has become a popular computing paradigm for analyzing continuous and high-volume data streams to extract real-time insights from the data [7].

It should be noted that certain prevalent stream processing engines, such as Apache Storm and Apache Spark, are predominantly suited for utilization in resource-abundant cloud environments. Therefore, it is imperative to consider that their performance may not be optimized when operating within edge environments that are resource-constrained. Several batch processing frameworks have effectively tackled the issues of volume and variety in big data processing, such as MapReduce, Spark [8], and Hadoop [9]. However, a novel category of big data processing platforms has arisen to tackle the velocity aspect of big data. Stream processing platforms, such as Apache Storm, are commonly employed for the purpose of processing streaming data that emanates from diverse sources, such as Internet of Things (IoT) applications and social networks. A stream processing application comprises a directed acyclic graph composed of vertices that signify data processing operations and edges that indicate data flow.

Recent research findings as reported by various sources [10], [11] indicate that Internet of Things (IoT) workloads have been extensively investigated. The divergent nature and potential complex variations of distinct application domains can result in significant disparities and variations. It can be affected by a variety of factors, such as external stimuli and internal processes, make event-driven systems complex and challenging to model and analyze. Event-driven systems present a complex challenge in modeling and analysis due to their dynamic characteristics. The

rate at which events arrive is subject to various factors, including external stimuli and internal processes. Over the course of time, there is a significant alteration observed. adaptation is required, thus leading to the necessity of modifying the current approach in order to address the situation accordingly.

The need to leverage the Edge in an agile and efficient way paramount importance. Existing literature on resource scaling for distributed systems [12], [13] has explored the challenges and opportunities associated with the efficient allocation and management of resources in such settings. Stream processing primarily concentrates on mitigating bottlenecks within the data flow. A sole stream processing topology is designed for the purpose of optimal enhancement of mean performance. Additionally, these techniques frequently exhibit limitations. Due to the latency inherent in the scaling mechanism of the SPEs, performance may be affected. In this regard, the current study endeavors to address the issue of end-to-end tail latency. The objectives of concurrent stream processing workloads are targeted by rebalancing the dexterous and granular resources at the system-level.

Related Work

The management of resources is an essential component of Edge computing, which holds significant importance in various domains. The authors in [14] have presented a novel protocol that relies on an auction-based mechanism for the purpose of realizing resource contract establishment while concurrently leveraging a latency-aware scheduling technique to effectively optimize utility for both Edge computing infrastructures and service providers. Araldo et al. [15] have successfully deployed a resource allocation algorithm with polynomial time complexity that enables Edge network operators to optimize their utility. This algorithm has the potential to

enhance a range of performance metrics, such as inter-domain traffic savings, user quality of experience (QoE), and more.

In the study [16], the authors explored the optimization of workload reduction and resource allocation for cloudlet applications as a means of effectively managing the quality of service for such applications within the context of resource contention for cloudlet resources. Within the realm of stream processing, a number of studies have turned their attention towards the auto-scaling of distributed systems through observation and analysis of the performance dynamics inherent to streaming dataflow. This area of research has been explored in various works, including those outlined in [17] and [18].

Load shedding is a crucial strategy implemented in the field of edge computing with the purpose of mitigating system overload, guaranteeing prompt execution of high-priority assignments, and simultaneously adhering to a prescribed latency threshold. Recent scholarly publications [19], [20] have suggested load-shedding frameworks that adopt a probabilistic model to acquire knowledge on the impact of events within a given window based on their position and type. The authors present algorithms that aid in the determination of the appropriate number of events to drop and the optimal timing for such drops, while also facilitating the prediction of the utility threshold required to achieve the desired latency constraint. Input-based load shedding is frequently employed as a technique for handling complex event processing (CEP) queries. However, it must be noted that the effectiveness of this approach with respect to individual events may be greatly impacted by the presence or absence of partial matches, thereby giving rise to potential quality challenges. In order to tackle this issue, pertinent state-based methodologies have been devised to deliberately eliminate incomplete matching instances while upholding superior outcome precision within constrained resources [21].

The study conducted by [22] proposes the implementation of a scheduled program that runs periodically. The present study presents a methodology to develop a systematic timetable by mapping the Storm topology executors onto the worker processes of supervisor nodes in a manner that ensures that no supervisor node exceeds a predetermined threshold for CPU usage. In [23], a scheduling algorithm for Apache Storm which employs graph partitioning has been introduced. The aim of the present scheduling framework is to decrease the overall network burden arising from data migration across nodes within a Storm cluster, whilst ensuring that an equitable level of computational capacity is assigned to each node, thus avoiding any potential imbalances in the workload distribution. It is pertinent to acknowledge that the real-time rescheduling of topologies can be executed with the aid of the online scheduling algorithms.

Stream Processing Engine (SPE)

The present study takes into consideration edge Stream Processing Engines (SPEs) deployed on an Internet of Things (IoT) Gateway, adopting a model akin to that examined in a recent academic publication [24]. The Internet of Things (IoT) Gateways possess a restricted quantity of computational resources when compared to the Cloud. Nonetheless, they offer more resources than those that are within reach of embedded, wireless sensor networks, among other similar networks. As is evident from the illustration presented in Figure 1. The data streams generated by Internet of Things (IoT) devices are subject to processing by an IoT Gateway that runs multiple topologies. The process of stream processing adheres to the dataflow programming model [25], wherein every application is contained within a directed acyclic graph (DAG) data structure, commonly referred to as a topology. The flow of individual data points, referred to as tuples, occurs within a topology whereby such tuples traverse from their sources to corresponding sinks. The process of this data transmission is characterized by its discrete and granular nature.

Each internal node within the system executes computational operations of varying complexity on the data, which can range from basic filtering to intricate processes such as classification algorithms based on Machine Learning (ML) principles. It is posited that every application possesses a Service Level Objective (SLO) target regarding the end-to-end latency of data tuples circulating throughout the topology, specifically within the 95th percentile.

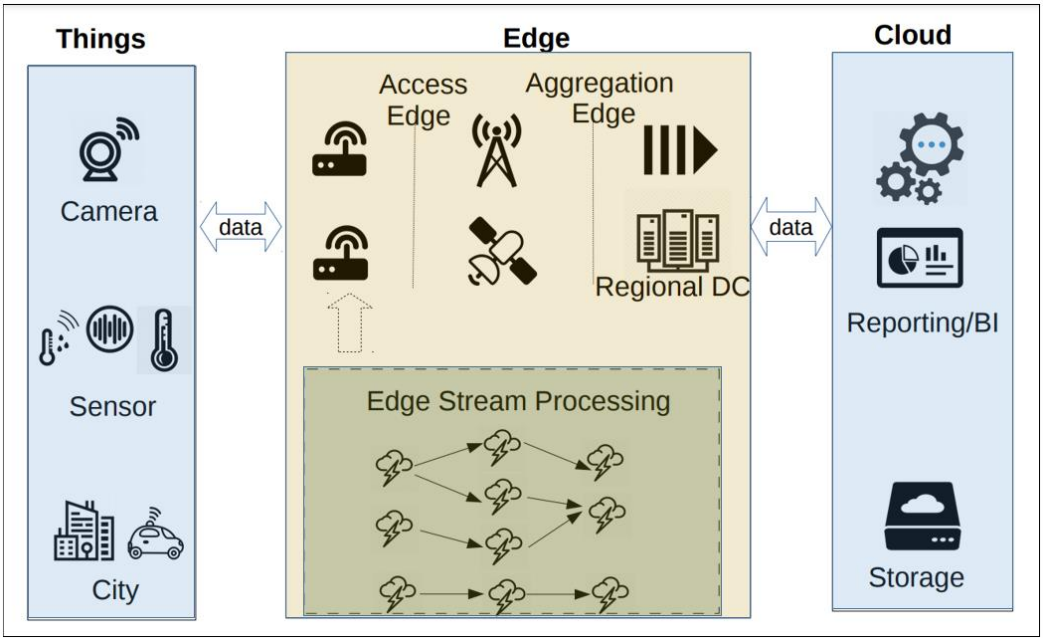

Figure 1 Internet of Things: Sensors to Cloud [51]

We have employed Apache Storm as an exemplar Stream Processing Engine (SPE). Storm is a distributed system designed for real-time computation, specifically for the purpose of processing streams of data that are limitless in size. A Storm topology refers to a directed acyclic graph (DAG) consisting of spouts and bolts. A spout, in this context, serves as a data stream source while a bolt acts as a data processing unit. In a typical deployment scenario, Storm is employed on a clustered system utilizing the master-worker architecture, as illustrated in Figure 2.

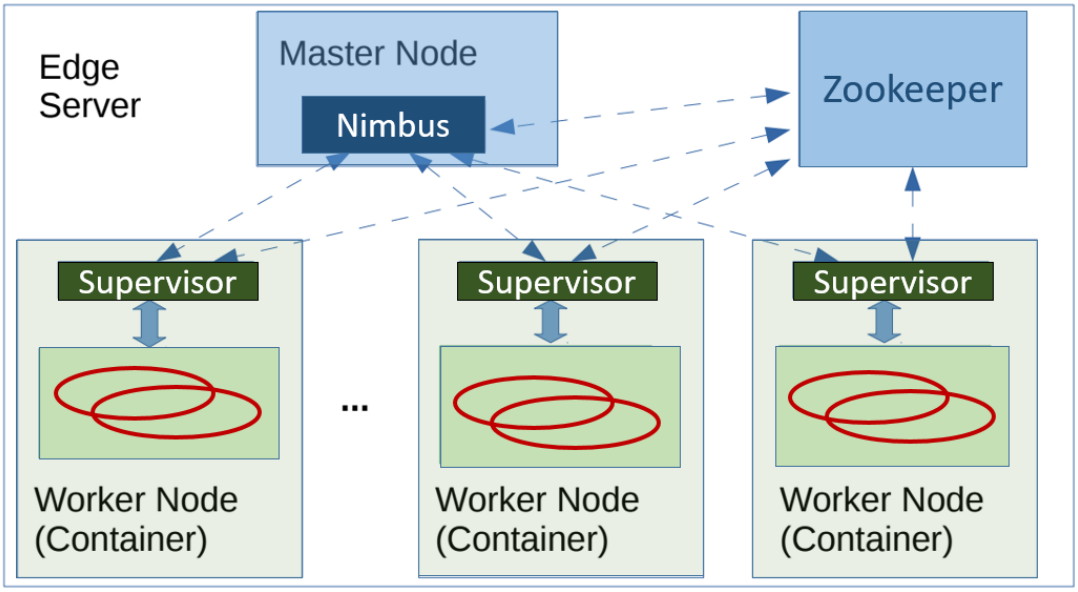

Figure 2 Architecture of Storm Cluster [51]

A Storm cluster comprises two categories of nodes along with zookeeper:

- **Nimbus Node:** The nimbus is a critical component, operating as the master node within a Storm cluster. This daemon, known as the nimbus, is responsible for directing and managing overall cluster activities. In the context of the Storm computing framework, developers submit job requests to the Nimbus component, which subsequently disseminates the tasks of said jobs to the worker nodes through the utilization of a scheduler. Nimbus systematically monitors the state of the cluster and in the event of a failure of a supervisor node, endeavors to either initiate a restart of the supervisor node or reassign the task of processing to alternative supervisor nodes.
- **Supervisor Node:** A Storm cluster has the capability of accommodating a multitude of supervisor nodes, each equipped with a supervisor daemon. Every supervisor node accommodates a predetermined quantity of worker processes in a prearranged manner. Each employed entity executes either a single or a multiple unit of tasks that make up

a specific network structure in Storm. The supervisor daemons bear the responsibility of initiating and terminating the worker processes for the purpose of executing and discontinuing the topologies.

- **Apache Zookeeper:** In addition to the above mentioned nodes, Storm clusters rely on Apache Zookeeper to facilitate cluster synchronization and management of the cluster state. An "ensemble" is formed by multiple nodes executing the Zookeeper daemon, which facilitates replicated synchronization and state management services with eventual consistency to the nimbus and supervisor nodes.

The representation of a stream in Storm comprises an infinite sequence of distinct data structures denoted as tuples. A tuple is an assemblage of named elements of data, consisting of values of various types such as string, integer, float, among others. The processing tasks executed on Storm clusters are referred to as topologies in the field of academic writing on the subject of stream computation. A topology consists of two distinct categories of components and incorporates three principal abstractions:

- **Spout and bolt** are two components of a topology as shown in Figure 3. Spouts facilitate connectivity with external data sources, encompassing diverse message brokers like Apache Kafka. The spout serves to encapsulate the application logic required for establishing connections to message brokers and facilitating the transmission of data streams to downstream processing components. The bolts serve as fundamental processing units in the implementation of a Storm topology. The application logic utilized for stream processing is condensed within bolts. These bolts carry out diverse operations (e.g., filtering and aggregating) on the tuples of the stream in accordance with user-defined logic.

- **Worker processes, tasks and executers** are three main abstractions to run a topology. Each worker process refers to a distinct and independent instance of a Java Virtual Machine (JVM), which executes a specific subset of the constituent elements comprising a given topology. Every supervisory node is equipped with a pre-established quantity of slots to carry out the operations of the worker processes. Executors refer to threads of execution that are created by worker processes, which in turn operate within the Java Virtual Machine (JVM) process of said worker processes. Executors are responsible for the execution of one or multiple specified tasks associated with a given component, as directed by the end-users. A "task" refers to an active instantiation of a Storm component that may include a Spout or a Bolt entity. The processing operation is executed by the assigned tasks, which are operated within their corresponding parent executors. In order to effectively manage a project, developers must ascertain the appropriate quantity of laborers required to accomplish said project.

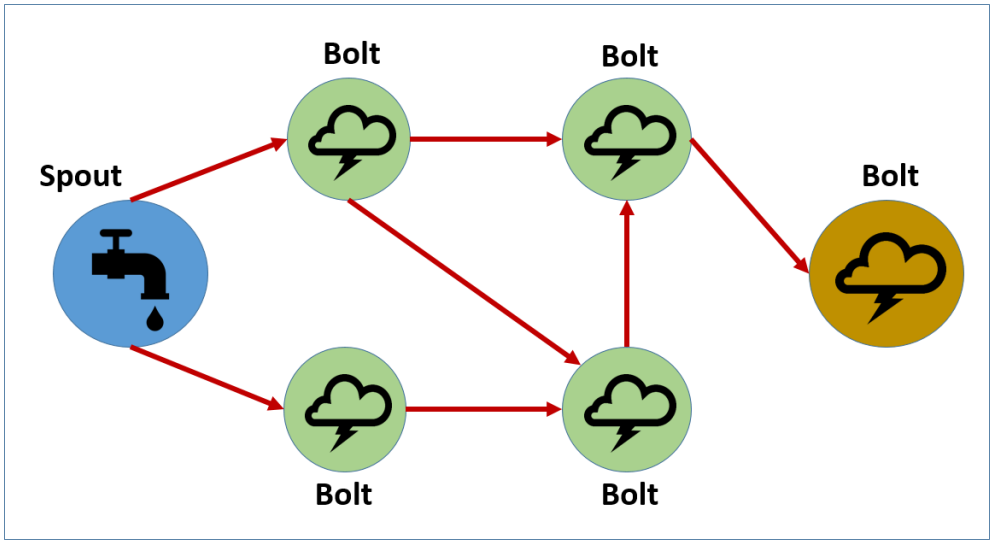

Figure 3 Data Model for Apache Storm

Priority Scheduling

The Apache Streaming framework exclusively employs the first-in, first-out (FIFO) queuing discipline for job processing, without any provision for alternative queueing methodologies. By default, the submitted tasks of an application's threads are processed in a sequential order, following a first-in, first-out (FIFO) approach as shown in Figure 4. If the tasks assigned to a thread exhibit a substantial size, they will expend a commensurate quantity of resources in their processing. This could potentially result in impediments for the remaining threads of application that are currently in line for execution. In the event that the application does not necessitate the complete allocation of resources, the subsequent thread in the queue has the potential to commence execution of its designated tasks.

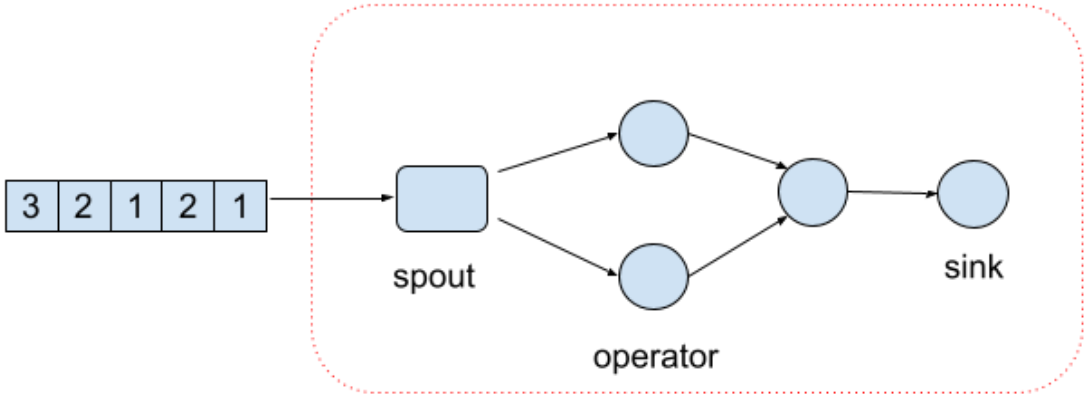

Figure 4 Apache storm stream framework that implements FIFO

There may arise instances where it is deemed desirable to allocate precedence to particular input data elements. In cases where there is a significant influx of incoming data or a sporadic input load, it is possible that the timely processing of high priority data may be impeded. The

capacity to allocate priorities to data inputs serves as a means to guarantee that the most critical data is processed at the outset in such scenarios [10]. Streaming platforms are frequently employed as the preferred instrument for carrying out data analytics. Ideally, adequate resource allocation should be pursued to enable efficient handling of input loads without resulting in queuing delays. The functionality of the system is contingent upon its ability to rapidly process incoming data loads in real-time.

This scheduling methodology necessitates the user to allocate priorities to the various forms of input data in advance. In our proposed system, during program execution, the scheduler assigns a higher precedence to input data that exhibits greater priority. In instances of a significant surge in system load, the latency exhibited by higher priority tasks remains consistent, while the lower priority tasks remain enqueued and thus experience processing delays. When the input load diminishes to a threshold at which the system can accommodate all jobs, it is observed that equitable apportionment of resources is provided to both high and low priority jobs.

This scheduling technique operates without any requisition of supplementary resources. Nevertheless, it is feasible to utilize it alongside dynamic resource allocation techniques. The data-driven priority scheduler has the potential for adoption across multiple micro-batching streaming engines. Micro-batching engines refer to streaming engines that partition the input stream data into diminutive batches, which are subsequently processed at a granular level in batches. This work entails the implementation of a data-driven priority scheduler on the Storm Streaming framework as shown in Figure 5. The Storm Streaming scheduler is adapted to conduct a thorough evaluation of the input dataset such that it can prioritize data elements with higher levels of significance.

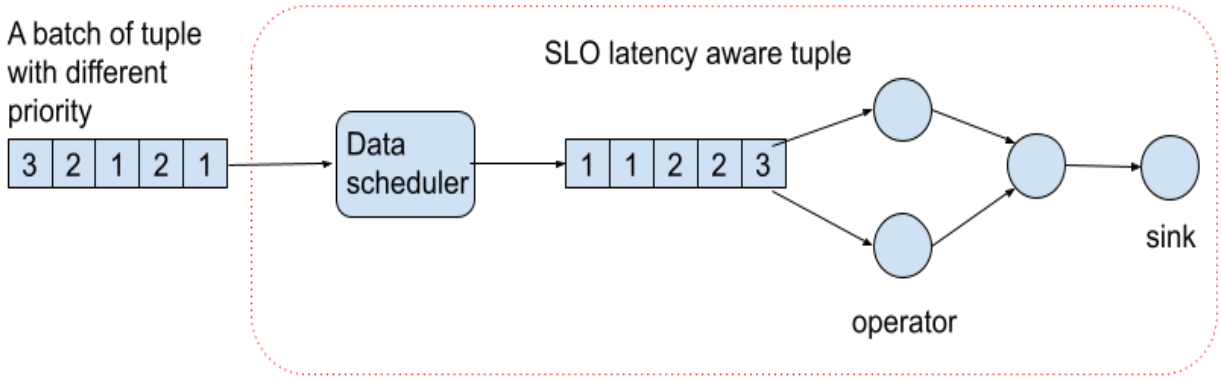

Figure 5 Stream processing with priority-data scheduler

Problem Formulation

In light of an Edge stream processing application and workload, we endeavor to identify an optimal resource configuration that fulfills the specified service level objective (SLO) while minimizing the allocation of resources. The present study endeavors to elucidate the resource configurations that encompass the limits of CPU usage of containerized SPE workers, while simultaneously establishing the SLO target in regard to the end-to-end tail latency of a given application. Memory was excluded as a potential candidate for dynamic resource configuration. The aforementioned observation is attributable to the fact that in order to attain low latency, a processing element (SPE) must possess the capability to execute priority-based message processing procedures devoid of expensive storage operations within the crucial processing pathway, as inferred from a reputable source [20]. When investigating memory configuration via online means, it may give rise to thrashing or other unforeseeable behaviors. The problem is formulated in the following manner:

$$C_{minimize}(x) = \sum \vec{x} \times maximum(\, tail_{latency}, target - tail\_latency) \quad (1)$$

Where

$$\sum \vec{x} \leq Total\_CPU$$

The present study examines the maximum central processing unit resources, denoted as CPU_{Total}, that an edge server can provide, along with the total cost, represented as $CPU(\vec{x})$, that is associated with a given resource configuration $\vec{x}$. The normalized CPU usage limitations of the SPE workers that are containerized are denoted by the vector $\vec{x}$ in academic writing. The cost function, denoted by $CPU(\vec{x})$, accounts for possible over-provisioning of resources for configurations that meet or exceed the Service Level Objective (SLO) target. This consideration is included in order to optimize the allocation of resources and reduce unnecessary expenses. In the event that the SLO objective remains unattained, the tail latency measured exhibits significant influence over the $CPU(\vec{x})$.

Contribution of this Thesis

The following work introduces VRebalance, which is a virtual resource management system designed to offer a comprehensive performance guarantee for concurrent stream processing applications located at the Edge. Bayesian Optimization (BO) is the method utilized by VRebalance to promptly identify resource configurations that minimize the infringement of performance Service Level Objective (SLO) targets, in terms of 95th percentile latency. It is significant to note that VRebalance does not necessitate any prior acquaintance with the application or costly workload profiling. Additionally, it successfully satisfies the performance service level objective (SLO) targets for priority-based workloads involving stream processing,

even in the presence of dynamic changes in system behavior. The experimental findings pertaining to an open-source Internet of Things benchmark, namely RIoTBench, and a stream processing engine representative of its kind, namely Apache Storm, evince the higher efficacy, resource optimization, and adaptability of the resource management system we have developed. The methodologies pursued in this investigation diverge from our own line of inquiry, which centers around analyzing the intricacies of prioritized stream processing in the context of a resource-limited Edge node, utilizing a range of resource allocation methodologies. This study emphasizes the efficacy of adopting a data prioritization approach in conjunction with BO-driven resource allocation techniques, as a means of attaining superior Service Level Objective (SLO) guarantees pertaining to 95th percentile latency.

Key contributions can be enlisted as:

- The present study undertakes an examination of the constraints inherent in extant Stream Processing Engines (SPEs), specifically Apache Storm, with regards to their ability to efficaciously alter the parallelism of extant processing topologies in a swift manner. Additionally, we bring to the fore its suboptimal use of resources within Edge environments.
- In this study, we present the development of a virtual resource orchestrator rooted in BO, which exhibits an awareness of Service Level Objectives (SLOs). The framework aims to efficiently identify and deploy configurations of resources that approach optimality for priority-based stream processing applications when located at the Edge.
- An algorithm for coordinated resource configuration of concurrent priority-based stream processing applications has been developed. The algorithm is based on suggestions gathered from various BO models. In order to manage varying priority-

based workloads, we employ an effective approach by utilizing BO models that are suited to specific intensity ranges of priority-based workloads.

- This study entails the implementation and evaluation of our system through the application of a laboratory-scaled real testbed, utilizing an edge server employing the RIoT benchmark. Through empirical analysis, a comparison between a hill-climbing approach and Bayesian optimization (BO) reveals a reduction in the likelihood of latency service level objective (SLO) violations.

CHAPTER II

BAYESIAN OPTIMIZATION

The Bayesian optimization methodology constitutes an efficient strategy to optimize objective functions that entail a substantial time interval for evaluation, typically ranging from minutes to hours. The optimization technique is most appropriate for continuous domains comprising fewer than 20 dimensions and can effectively accommodate stochastic noise in the assessment of functions. This approach involves the construction of a surrogate model to represent the objective, coupled with the quantitative assessment of uncertainty in this surrogate employing a Bayesian machine learning method known as Gaussian process regression. The surrogate model is further employed to determine potential sampling locations through the utilization of an acquisition function defined from the aforementioned surrogate model. This manuscript expounds on the Bayesian optimization technique, highlighting the fundamental principles governing the approach, namely, Gaussian process regression, and three crucial acquisition functions; expected improvement, entropy search, and knowledge gradient. Subsequently, we expound upon sophisticated methodologies, encompassing the concurrent execution of numerous function evaluations, optimization techniques that incorporate multiple sources of fidelity and information, cost-intensive constraint handling, environmental vagaries, the concurrent pursuit of multiple tasks using Bayesian optimization, and the integration of derivative information. In conclusion, the present study culminates with an in-depth discourse on Bayesian optimization software, as well as the prospective research avenues yet to be explored in this discipline. In our instructional materials,

we present an extension of the concept of expected improvement in the context of evaluations that are subject to noise, as opposed to the typical setting of noise-free evaluations. This assertion is substantiated by a decision-theoretic reasoning, which conforms to a formal paradigm, and is divergent from antecedent makeshift alterations.

Bayesian Optimization (BO) is employed for the purpose of optimizing objective functions that tend to necessitate profound assessment. The possible objective functions that can be optimized utilizing Bayesian optimization (BO) are comprehensively inclusive of but not limited to those that necessitate thorough search, exhibit computational extravagance, lack discernible structure, are treated as opaque entities, are bereft of derivations and may require estimation of gradients. The BO methodology is a multifaceted approach that possesses the capacity to assess black-box problems without requiring access to their derivative information.

In contemporary times, the use of Bayesian Optimization (BO) has gained traction in the context of fine-tuning hyper-parameters in machine learning algorithms and associated methodologies [26]. The origins of Bayesian Optimization (BO) can be traced back to the seminal work of Kushner (1964) [27]. However, the widespread adoption of BO occurred following its generalization and practical implementation by Jones et al. The year 1998 marked a significant point in time. Drawing from the methodology documented in reference [28], subsequent scholarly endeavors pertaining to multi-fidelity optimization [29], multi-objective optimization [30], as well as convergence-rate investigations [31, 32] have advanced the frontiers of numerous research spheres and domains. The studies cited in references [30, 33] were directed towards the training of deep neural networks [34] and the implementation of parallel methodologies [35, 36]. In recent times, the reference cited has been reported. The aforementioned study endeavors to enhance and

formulate simulations utilizing discrete event simulations in an optimal and model-based approach, as evidenced by sources [37, 38].

It should be highlighted that Black-Box Optimization (BO) is not confined to a particular family of algorithms, but rather represents an ideological methodology for optimizing objective functions, which subsequently facilitates the development of numerous algorithms. Bayesian Optimization (BO) employs techniques grounded in the principles of Bayes' Theorem. It is worth noting that BO relies on a probabilistic framework that is closely linked to the objective function.

This procedure works in two parts:

- By using a probabilistic framework offering a surrogate function.
- By offering an acquisition function that is grounded in a surrogate function.

The utilization of a pre-determined acquisition function facilitates Bayesian optimization in exploring multiple neighborhoods within the search space while achieving a harmonious equilibrium between "exploration" of regions with high levels of uncertainty and "exploitation" of non-sampled regions exhibiting high predicted mean values on the surrogate model.

Surrogate Function

Statistical inference presents a structured framework for the derivation of ambiguous parameters within a given system, which can be substantiated by the principles of probability theory. In the context of inference, any quantities that are not known a priori are treated as stochastic variables. The surrogate function provides the most optimal approximation of the input samples to a resultant output score. Various techniques are available for modeling a surrogate function; however, the prevalent approach is regression predictive modeling. This method employs input data samples and generates a score representing the model's output. The most frequently utilized models in statistical and machine learning applications are the Gaussian Process (GP) and

random forest methodologies. This study employed the GP model, which operates as a multivariate-Gaussian for distributions of finite dimensions. The multivariate Gaussian process (GP) model pertains to a statistical framework that entails the estimation of the mean and standard deviation of a distribution. The Gaussian process model demonstrates effectiveness in the optimization of a substantial number of functions, while concurrently facilitating a seamless transition among observations. In the domain of Genetic Programming, the selection of the "kernel" holds significant importance as it represents a critical undertaking. The selection of the kernel is characterized by its ability to ensure minimal distance between any pair of points in the given sample size, a property that is attributable to a strong positive correlation between the pairs in question. As a consequence, the covariance matrix exhibits positive semi-definiteness.

When crafting a surrogate function, one may utilize a Gaussian Process Regression (GPR) model that employs a Radial Basis Function (RBF) kernel of default specifications. The selection of the kernel is of utmost importance as the functional shape at distinct points is contingent upon this decision. Various kernels can be employed, with certain kernels exhibiting superior outcomes contingent upon the dataset utilized. Once the conceptual framework has been established, it may be invoked for the purpose of being configured to suit the given data set. The model is capable of furnishing an estimation of the cost associated with one or more input samples. The outcome of the provided sample will take the format of both the standard deviation and the mean of the distribution. In this illustrative example, the surrogate function is anticipated to execute a rudimentary approximation of the authentic non-corrupted objective function. Figure 6 displays the Ground Penetrating Radar (GPR) analysis on the test function, wherein scattered data points denote the noisy samples, and the continuous curve denotes the outcome of the surrogate function.

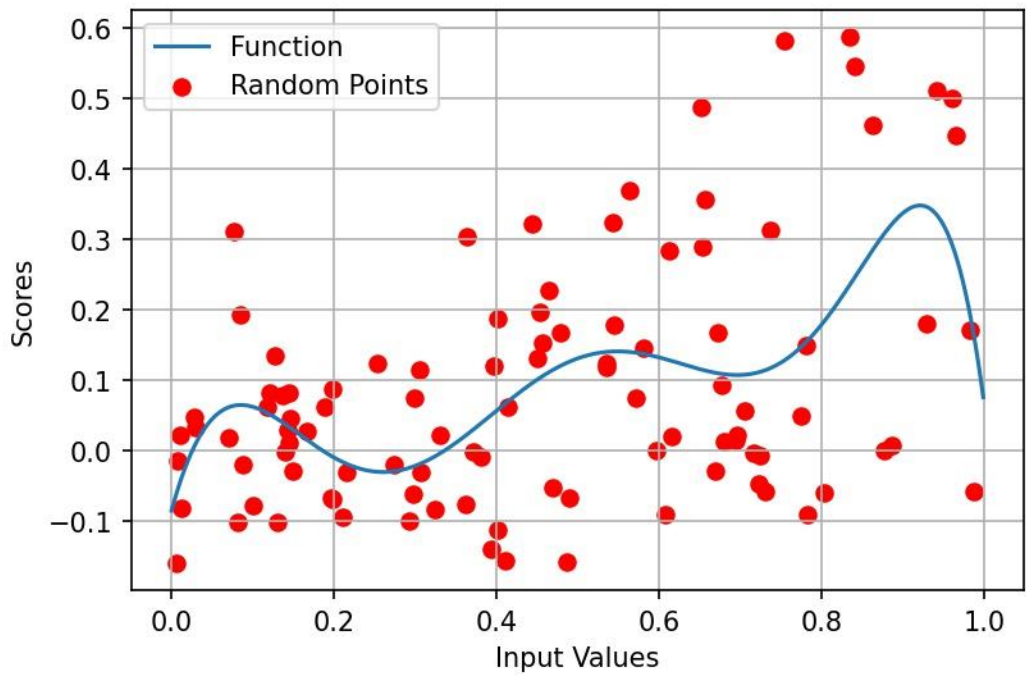

Figure 6 Illustration of Bayesian optimization, using Gaussian regression as a surrogate function

Acquisition Function

The surrogate function assumes the responsibility to examine potential samples, from which one or multiple candidates may be chosen for evaluation based on the resulting outcomes. The acquisition function comprises two main components, namely the selection of a search strategy for domain navigation and the subsequent evaluation and interpretation of score responses from the surrogate function. Effective search techniques for optimization problems include grid-based sampling and random sampling. However, local search strategies such as the BFGS algorithm are the prevailing techniques utilized in these scenarios. The acquisition function leverages the probabilistic data provided by the surrogate function to determine the value of

assessing the presented sample. The preponderantly utilized probabilistic acquisition functions are typically identified as [40]:

- **PI:** Probability Improvement.
- **LCB:** Lower Confidence Bound.
- **EI:** Expected Improvement.

The PI method represents the most uncomplicated option available whilst the EI method stands as the preferred and prevalent technique utilized in various contexts. Employing the principle of Expected Improvement (EI), a decision rule known as an acquisition function is formulated to systematically identify the next sampling point that maximizes the projected increase over the present optimal solution. Due to its ability to provide a practical equilibrium between exploration and exploitation at a reasonable cost of valuation, VRebalance has opted to utilize EI. Equation 2, 3 and 4 show the calculation of each method. Acquisition function consist of exploitation μ(x) and exploration σ(x).

$$PI = cdf\ ((Mean_{Surrogate} - Mean_{Best})/standard\_deviation) \tag{2}$$

$$LCB = \mu(x) + \lambda\sigma(x) \tag{3}$$

$$EI = argmax_x\ \mu(x) - f^{max} \tag{4}$$

The aforementioned illustration employed a rudimentary search methodology whereby a series of random samples were extracted from the candidate samples and subjected to thorough evaluation through the adoption of an acquisition function. The acquisition function attains its

maximum value through the selection of the sample exhibiting the highest score. The acquisition function assumes the responsibility of assessing the probability that a given input sample is meritorious of further evaluation. The PI probability function employs the standard normal cumulative probability distribution to determine the optimal score. The evaluation points generated by the BayesOpt methodology are illustrated in Figure 3. The dots depicted in the plot represent the random samples, while the solid line corresponds to the surrogate function's output.

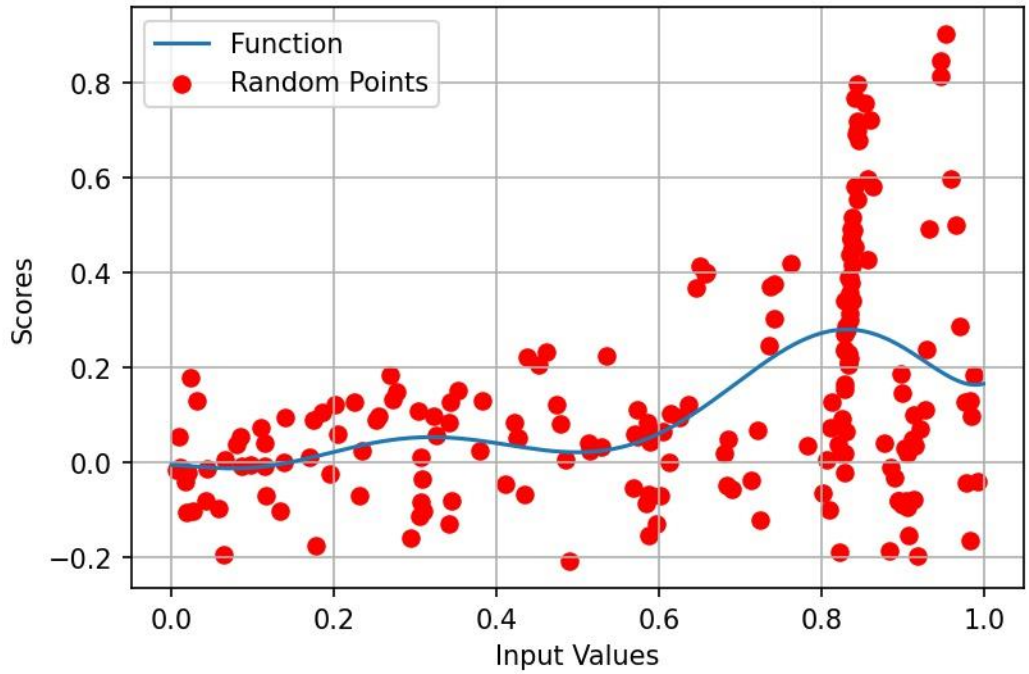

Figure 7 Illustration of Bayesian optimization, using acquisition function to maximize the function

CHAPTER III

METHODOLOGY

This section elucidates an adaptive system known as VRebalance that exhibits a sophisticated awareness of Service Level Objectives (SLOs). This technology is devised with the aim of providing a robust performance assurance for Edge stream processing, particularly in the context of priority-linked workloads. In order to circumvent the constraints presented by the current Systematic Investment Plan Executives (SPEs) and the challenges inherent in the allocation of Edge resources, the VRebalance system employs two significant architectural tenets. To efficiently utilize the limited resources available, a crucial principle is to adopt resource-efficient and agile practices that facilitate the maximum number of applications that can benefit from them. This is accomplished by selectively assigning solely the essential and indispensable assortment of resources mandatory for attaining the service level objective (SLO) of each application. Rather than opting for employee expansion, VRebalance achieves this objective by dynamically configuring the resource utilization thresholds of SPE workers that are containerized. This approach successfully avoids the delay commonly encountered while attempting to rebalance a topology. The utilization of models possessing sufficient precision to differentiate between approximate and suboptimal solutions represents the secondary principle. The proposed VRebalance method proficiently identifies the optimal resource configurations for concurrent stream processing topologies without the requirement of any pre-existing proficiencies or significant workload profiling expenses. Specifically, the VRebalance method utilizes Bayesian

optimization to intelligently analyze optimal resource pairings while simultaneously monitoring their efficacy. Figure 8 shows the entire working of our proposed methodology, it also shows the interplay among different constituents of VRebalance alongside containerized SPE operatives that operate on an Edge node.

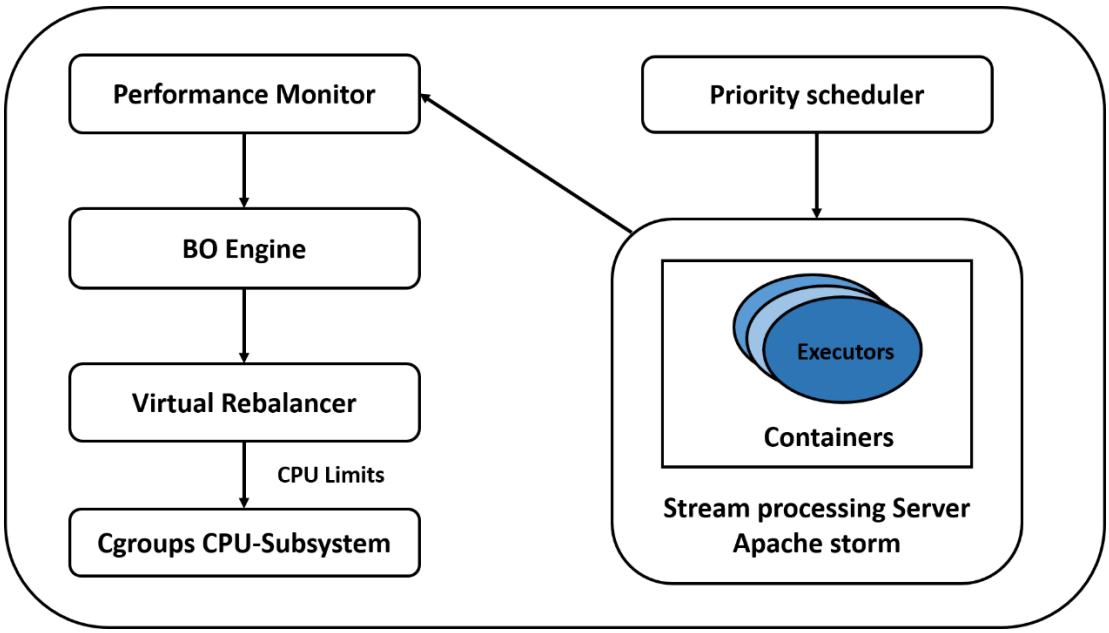

Figure 8 Proposed Methodology

Design Options for BO

The utilization of Bayesian optimization (BO) emerges as a viable approach for enhancing the performance of a costly black-box function, particularly in the context of VRebalance. The term "blackbox" refers to an equivocal association between the input and the objective function. Nevertheless, this association can be inferred through systematic and empirical means such as experimentation, probing, and monitoring techniques. In the VRebalance framework, the assessment of the objective function entails the implementation of a prospective resource

configuration and the subsequent quantification of its effect on the end-to-end tail latency of an application. Undoubtedly, this procedure is characterized by a substantial cost since the repetitive examination of resource configurations poses the risk of non-compliance with the Service Level Objectives (SLOs) owing to under-provisioning, or excessive resource utilization due to over-provisioning. The proficiency of BO lies in the ability to identify nearly optimal solutions with a minimal number of iterations. During the process of search space exploration, Bayesian optimization (BO) conducts an evaluation of the objective function utilizing various input samples, or configurations.

This evaluation leads to the construction of a probabilistic model, also referred to as a surrogate model, which serves to estimate the performance of diverse configurations. Such an approach enables efficient and effective optimization of the objective function within the search space. Subsequently, Bayesian optimization (BO) navigates a range of neighborhoods within the search space by means of a pre-established acquisition function that effectively balances the dual objectives of "exploration" and "exploitation". The former involves venturing into areas of the search space where uncertainty is high, whereas the latter is aimed at leveraging regions that have not been previously sampled, but are expected to yield high mean values based on the surrogate model's predictions.

Surrogate and Acquisition Function

The **Gaussian Process (GP)** is utilized as the surrogate model to establish the prior/posterior distribution for the objective function. The Gaussian process (GP) is a non-parametric model that affords a probability distribution over the entire set of potential functions that align with the available data. In Bayesian optimization, the definition of the prior and posterior distributions can be established through multiple conjugate distributions. However, the Gaussian

process (GP) is selected due to its inherent flexibility. Sufficient data samples enable the approach of the actual function, while maintaining computational feasibility. Moreover, it is widely acknowledged as a substitute model. Following our empirical findings, we have selected the RationalQuadratic kernel [39] as the optimal choice for the covariance kernel function to be utilized within the GP model. The measure of similarity between a pair of resource configurations is determined by the covariance function. The present discourse employs the nomenclatures, surrogate model and BO model, in a mutually exchangeable manner.

There exists multiple approaches to devising an **acquisition function**. Common approaches comprised of probability of improvement (PI), expected improvement (EI), and upper confidence bounds (UCB) have been found to be effective, as indicated by previous studies [24]. The present study employs the concept of expected improvement (EI) in devising an acquisition function aimed at selecting the subsequent sampling point, with the primary objective of achieving maximum improvement over the existing optimum. The selection of EI for VRebalance stems from its capacity to strike a pragmatic equilibrium between the dimensions of exploration and exploitation. The efficient utilization of resources by means of cost-effective evaluation techniques is demonstrated through the implementation of exploitation endeavors. Alternative methods, such as the PI approach, have been observed to frequently encounter local optima, while the utilization of UCB may necessitate supplementary parametric adjustments [24].

Search Space Reduction, Phases, Stop Condition, and Changing Workload

The search space has been restricted through the employment of quantization techniques for resource configuration, thus expediting the process of Bayesian Optimization convergence. A quantization step of 50 millicores, with a conversion factor of 1000 millicores per CPU core, is employed. Furthermore, we have ascertained the threshold values for the CPU usage parameters,

with the minimum and maximum limits currently set at 150 and 4,000 millicores, respectively. The process of quantization results in a notable decrease in the overall search range, specifically from 3, 850 times the number of SPE worker nodes to 77 times the same quantity, whereby the variable 'n' represents the aforementioned number. This is deemed a methodological approach commonly utilized within academic research.

In the current experimental context, Nmin and Nmax were determined empirically to be 5 and 16, respectively. Once VRebalance has gathered data on a minimum of N resource combinations, the exploration phase transitions to the exploitation phase. In the exploitation phase, the optimal configuration is selected by VRebalance, as identified thus far. "If there exist two consecutive infractions of the SLO_target parameter, the VRebalance algorithm initiates the recommencement of the BO search procedure." This process persists until the evaluation of Nmax resource configurations has been completed. Despite the occurrence of Service Level Objective (SLO) violations, the VRebalance system will persist in the exploitation phase, while undertaking an assessment of the Nmax resource configurations. The existing circumstance implies a deficiency in the availability of resources, and further exploration into the configuration of said resources may prove to be deleterious.

We employed discrete Bayesian optimization models to cater to different levels of workload intensity, in order to expedite the adaptation of Bayesian optimization (BO) to evolving workloads. The determination of workload levels depends on the spectrum of workload intensity (l) in academic writing. The VRebalance system is utilized to measure and record the number of resource configurations observed at each value of l. Additionally, it implements the necessary BO model in situations where the workload intensity returns to a previously observed level.

Priority-Based Stream Processing

In this section, we explicate our approach towards facilitating stream processing based on priority. A spout denotes a distinctive component determined by a user which gains access to data from an external origin and subsequently discharges basic unit of data or tuples into a topology designated for analysis. The data processing unit is commonly referred to as a "bolt". Upon the receipt of each record, the system processes it in an individualized manner. However, this particular approach does not consider the differing levels of immediate significance attributed to discrete data entities. Conversely, our priority-oriented stream processing system, depicted in figure n, takes precedence over other competing systems. The use of the method illustrated in Figure 9, is highly advantageous in scenarios where certain sets of data possess a higher level of time-sensitivity or criticality than others, thereby necessitating immediate processing. Conversely, other data items have the ability to accommodate relatively longer processing periods.

The prioritization scheduler has been incorporated into the spout. Upon the ingress of data items into the system, a scanning mechanism is initiated to discern priority levels within the input stream. Each set of data tuples obtained in micro-batches is subjected to priority-based sorting, following which the resultant data is streamed to the remaining components of the topology. In the present study, a batch size of 10, 20, and 100 is employed. In the upcoming chapter, a sensitivity analysis pertaining to the batch size shall be presented in an academic manner. At present, the algorithm has the capability to accommodate three distinct levels of priority, namely low, medium, and high. The processing of high-priority data items is accorded precedence, with a preferential allocation of computing resources, whereas those classified under the categories of medium and low-priority are subsequently handled at a later time. This guarantees the prioritization of high-priority tasks, and maintains their latency at a constant level, despite abrupt surges in input load.

This process aims to enhance the performance of the system, while also ensuring that crucial data pertaining to important tasks are executed punctually. Additionally, the aforementioned algorithm can be expanded to accommodate a larger range of priority levels, leading to enhanced precision in managing the processing of data items and allocation of resources.

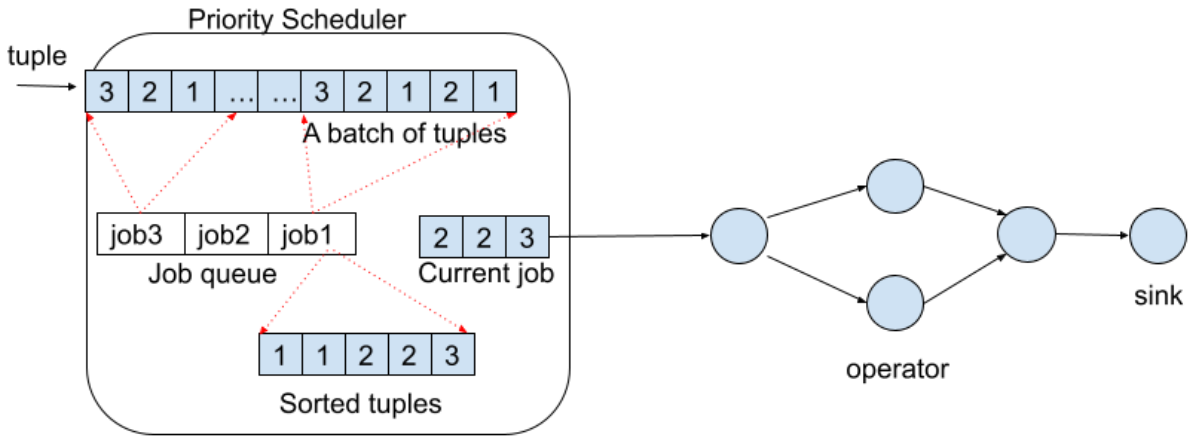

Figure 9 Priority Scheduler designed at the spout

Algorithm

In this particular section, the modality of line numbering derived from Algorithm 1 has been employed to comprehensively explain the functional mechanics of VRebalance. Every program that is operational on the Edge server has its data accumulated by the Performance Monitor, wherein the sampling interval is set to one minute. The computation of the throughput of a topology involves the utilization of the number of tuples generated by the downstream components that are situated at the greatest distance in the system. By assigning a unique identifier to each tuple and calculating the time difference between the recorded timestamps at the source and destination, it becomes feasible to ascertain the end-to-end latency of a topology. During

multiple iterations, VRebalance may shift between the exploration and exploitation phases. The allocation of resources and identification of service level objective violations thus far documented (lines 6-10) will determine future actions. The Bayesian Optimization Engine employs a process of updating the workload-specific BO model for a particular application using observed data gathered during the exploration phase (lines 7-9). Subsequently, the acquisition function of the engine is employed to identify a prospective cpu_a^j value for the desired resource configuration that will be subject to assessment. During the exploitation phase (as depicted in lines 9-10), VRebalance selects the most advantageous configuration that has been identified for an application at a given level of workload, while disregarding the BO models. It is necessary to rephrase the given text in a scholarly and formal manner: "Line 10 of the aforementioned text mandates the need for its rephrasing in the context of academic writing." The Co-ordinated Virtual Rebalancer is responsible for determining the optimal allocation of CPU power for the set of applications App_j' (line 11) in an integrated and systematic manner. Therefore, it is imperative to identify appropriate resource configurations for these applications at an early stage. Subsequently, it discerns the comprehensive central processing unit (CPU) resources that are necessitated by the extant applications, in conjunction with the quantum of residual CPU resources that would subsist following the distribution of the finalized CPU resources (as denoted in lines 12-13). In cases where the need for CPU resources of a program surpasses the current availability of resources, allocation of resources to each remaining program is performed proportionally according to their respective needs. (line 15) The potential need to increase the number of nodes arises if there persists a recurrent occurrence of SLO violations across multiple time periods. By utilizing the cgroup CPU subsystem, it becomes possible to modify the resource configuration of individual

applications. This is achieved by accessing the appropriate directory at (/sys/fs/cgroup/cpu,cpuacct/kubepods/burstable/pod/cpu.cfs quota us).

Algorithm:1 Algorithm for Virtual Rebalancing of Priority-Based Data on Edge Node

1. Resource configuration is performed for each application that is denoted by App_i if finalized for each interval *j*
2. **for** *j* =1 to ∞ **do:**
3: App_j → {Apps executing on the edge server | a} →; App'_j ← ∅;
4: **for all** a ∈ App_i **do:**
5: Priority-based workload (tuple per minute) using priority-scheduler at spout → w_a ;
Performance score calculation (latency, throughput) → p_a ;
6: **if** ($BO\_iteration^{w_a} < N_{min}$) **or** ($N_{max} > BO\_iteration^{w_a}$ **and** two consecutive violations of SLO_target happens) **then:**
7: $BO\_model^{w_a}$ will be updated using p_a *and* cpu_a^{j-1} ;
8: cpu_a^j → Ask for next CPU configurations from $BO_{model}{}^{w_a}$ according to equation 1; **else**
9: cpu_a^j ;Best configuration observed for workload w
App'_j → App'_j ∪ {a}
End
10: **end for**
11: cpu^{final} → $\sum cpu_a^j$;
12: $cpu^{available}$ → cpu^{total} - cpu^{final} ;
13: $\sum_{a\in App_j - App'_j} cpu_a^j$ → cpu^{need} ;
14: **for all** a ∈ App_i **do:**
15: **if** (cpu^{need} > $cpu^{available}$) **then:**
cpu → cpu_a^j / cpu^{need} x $CPU^{available}$
16: **Else**
CPU → cpu_a^i
End
17: Use Cgroup to update the CPU resources for application a;
18: **end for**
19: **end for**

CHAPTER IV

EVALUATION AND RESULTS

The present chapter undertakes an evaluation of the stream processing engine that was developed, leveraging a prototype and measurement methodology, with the goal of ascertaining its performance under two distinct scenarios. The primary scenario entails a static workload, while the secondary scenario involves a dynamic workload.

Experiment Testbed

Dataset

We utilized two Internet of Things (IoT) datasets, namely Sense your City [44] and NY city taxi trips [45]. The present study concerns the data stream of Sense your City (CITY), which represents real-life information gathered from a network of crowd-sourced sensors that were deployed across seven cities in three different continents. Each city was equipped with approximately 12 sensors, according to the report by [46]. The study records six consecutive timestamped observations encompassing temperature, humidity, ambient light, sound, dust, and air quality. Each minute, the sensors capture these measurements alongside metadata on the sensor identifier and its geographic location. The dataset entitled "New York City Taxi Trips (TAXI)" provides a continuous flow of intelligent transportation data stemming from 2 million taxi trips that occurred in 2013 involving 20,355 taxis in New York City, all of which are equipped with GPS technology. Each excursion furnishes information on both the commencement and termination dates, particulars regarding the hired taxi including its license number, as well as

details pertaining to the initial and concluding geographic coordinates alongside the timestamp. Furthermore, the recorded distance, as quantified by the taximeter, as well as the precise amount of incurred taxes and tolls are also cataloged. The benchmark runs in this study utilized aggregated data from January 2013, as outlined in sources [47] and [48]. In this study, the CITY dataset was utilized for generating a static workload, while the TAXI dataset was employed for generating a dynamic workload.

Configurations at Edge Node

A prototype testbed was established in order to simulate an Internet of Things (IoT) Gateway. This was accomplished by employing a Ubuntu 16.04 machine that boasted 4 central processing unit (CPU) cores and 8 gigabytes (GB) of random access memory (RAM). In the present study, Docker container engine (Version 18.06.2-ce) and Kubernetes (Version 1.18.2) container orchestration system were employed for the deployment of an Apache Storm cluster comprised of a total of 12 containers. Two of the containers were allocated for Nimbus and Zookeeper, while the remaining containers were operational as worker nodes. During the course of our empirical investigations, the containers under scrutiny were subjected to a standard initial CPU limit of 400 millicores, which is equivalent to 0.4 CPU cores. Furthermore, the designated CPU requirement for these containers was established at 200 millicores.

RIOTBench Benchmark

The study employed the utilization of the RIoTBench benchmark suite [46], encompassing a total of four distinct IoT applications categorized in accordance with prevalent IoT patterns, geared towards executing tasks within the realm of data pre-processing, statistical summarization, and predictive analytics. The distinct applications of ETL (Extraction, Transform, and Load) and

PRED (Predictive Analytics) were selected to thoroughly dissect the provided datasets, as depicted in Figures 10 and 11. It is necessary to utilize a formal tone and appropriate language in academic writing to convey information effectively to a specific audience. Therefore, the text should be revised to adhere to these conventions. The utilization of the two alternative applications (namely: STATS and TRAIN) is eschewed as a result of their integration with public cloud services, which renders them unsuitable for the prompt Edge stream processing objectives. In order to generate a dynamic workload in the conducted experiments, an adaptation was made to the input generator of RIoTBench as follows. In a recurring interval of one minute, the input generator supplies numerous batches of data, each containing a constant number of tuples (i.e., 10), to the source task (Spout) of the Storm system. The present study incorporates data batches which are subject to random delays that conform to a Poisson distribution, as reported in previous literature [49]. Data batches are adjusted at regular intervals in order to modify the level of workload intensity.

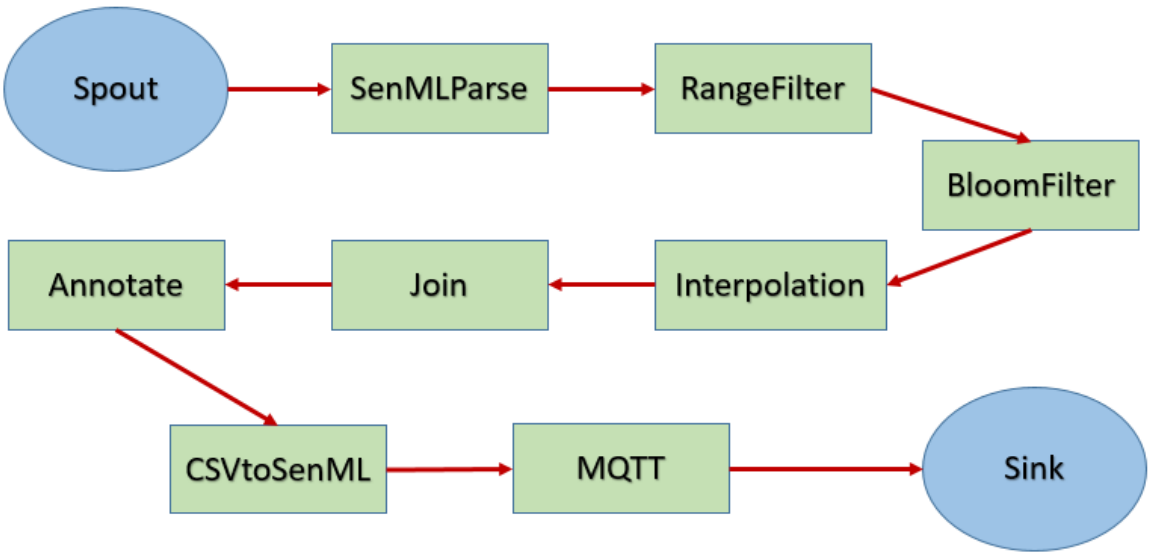

Figure 10 Layout of ETL (Extraction, Transform, and Load) topology.

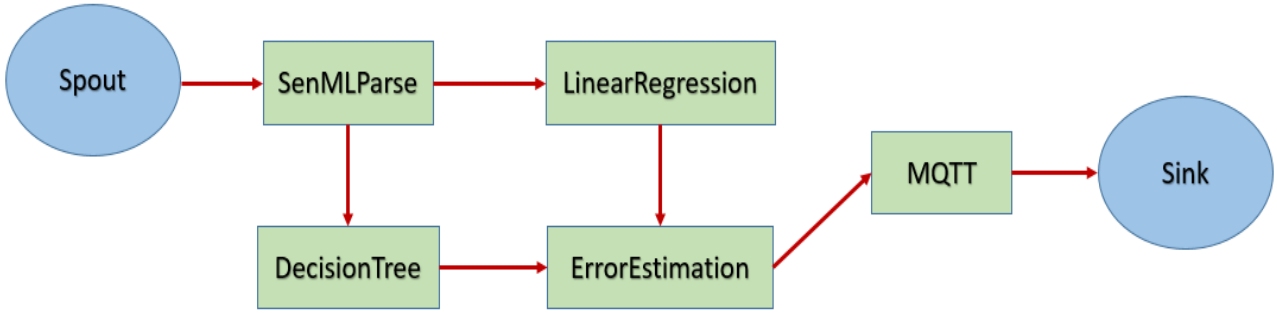

Figure 11 Layout of PRED (Predictive Analytics) topology

Measurement of Throughput and Tail Latency

Determining the 95th-percentile latency can be efficiently achieved through the organization of all recorded latencies per tuple in ascending order, followed by the retrieval of the value located at the 95/100th position. Nonetheless, it is noteworthy that this approach incurs a considerable overhead in situations where the rate of arrival of workload is high. The prevalent concern is effectively tackled by invoking the employment of a histogram approximation methodology as outlined in reference [50]. Rather than retaining and arranging all the values, we categorize them into clusters. The sampling interval employed in this study is one minute, resulting in latency values that will fall within the range of 0 to 60,000ms. The application of histogram bins with logarithmically increasing intervals, ranging from 1ms to 60,000ms, is feasible, including (0-1ms], (1,2ms], (2,4ms], ..., and (512, 1024ms], among others. Increasing the duration to 65536 milliseconds, a value representing 216, would result in the formation of 18 bins. Each bin will register the quantity of values that pertain to its specified range. Therefore, the requirement is

limited to storing only 18 counts as opposed to indefinitely preserving the latency values. Table 1 exemplifies this approach. The initial column denotes the bin's range, while the second column indicates the count of values encompassed by the bin. The third column depicts a cumulative sum of observed counts up until the corresponding row. The eCDF(x) denotes the empirical cumulative distribution function, which is computed by dividing the running total by the aggregate of all counts. In the present illustration, the 95th-percentile latency falls within the range of 128ms and 256ms. Linear interpolation is employed to determine the precise location of the value at 136.7ms within the bin. The efficacy of this aforementioned technique is contingent upon the quantity of bins employed. To augment the precision of our results, we increased the quantity of bins utilized from 18 to 36. The present technique offers a favorable balance between swiftness and precision.

Table 1 Example of histogram bins for tail-latency calculation

Range	Count	Total	eCDF
…	…	…	…
(4ms, 16ms)	1026	5152	44.4%
(16ms, 32ms)	2011	7163	61.8%
(32ms, 64ms)	1346	8509	73.4%
(64ms, 128ms)	409	8918	76.9%
(128ms, 256ms)	984	9902	85.4%
(256ms, 512ms)	356	10258	88.5%
(512ms, 1024ms)	1298	11556	98.1%
…	…	…	…

Evaluation under Static Workload

The present study assesses the performance and resource efficiency of a resource management system based on Bayesian optimization (BO) in the context of a static workload of 90,000 tuples per minute as shown in Figure 12. In order to accomplish this objective, the CITY

dataset and an extract, transform, and load (ETL) topology as depicted in Figure 3.1 are employed. To facilitate a comparison of performance, we apply a hill climbing-based search approach with varying step sizes (i.e., HC-50 and HC-200) in order to determine the optimal configuration of resources. In addition to our utilization of the Storm Rebalance technique in its unaltered form (SR1), we also employ the technique while implementing executor scaling (SR2). Both SR1 and SR2 introduce an additional worker to the Storm cluster every half hour.

In the domain of CPU resource allocation, the BO technique, as expounded in VRebalance [51], is utilized with the central objective of fulfilling the service level objective (SLO) of 200 milliseconds. The results presents evidence of the efficacy of the priority scheduler in the prevention of service level objective (SLO) violations for high priority data that is obtained at two-hour intervals. Based on the findings of Figure 13, it is evident that the absence of the priority scheduler leads to susceptibility of high priority data to Service Level Objective (SLO) violations, owing to the variance observed in the 95th percentile latency. In contrast, the data presented in Figure 14 indicate a significant divergence from the aforementioned results. The findings of Figure 14 indicate that the priority scheduler successfully preserved the 95th percentile latency within the SLO threshold of 100 ms. The results of the study shown in Table 2 and 3 support the notion that implementing a data scheduler based on prioritization can sufficiently guarantee uninterrupted operation for data of higher priority and preclude any potential transgressions in Service Level Objectives (SLOs).

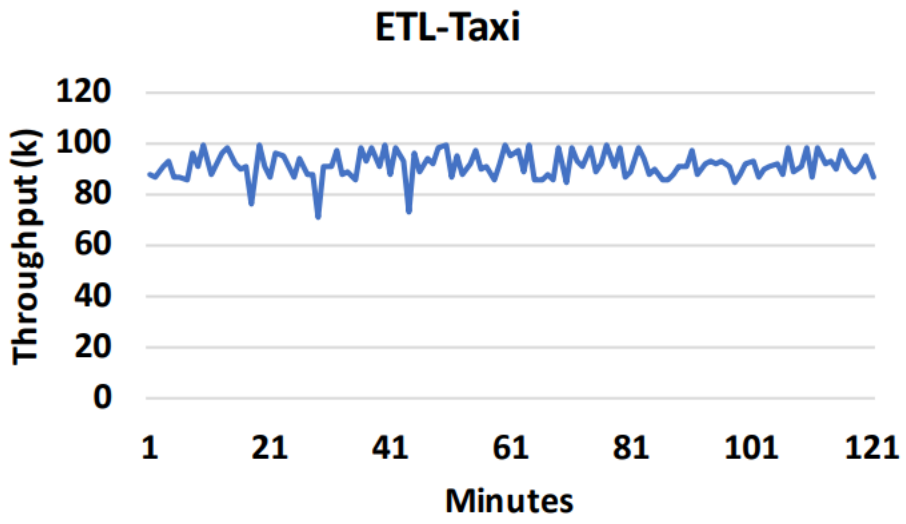

Figure 12 Static Workload

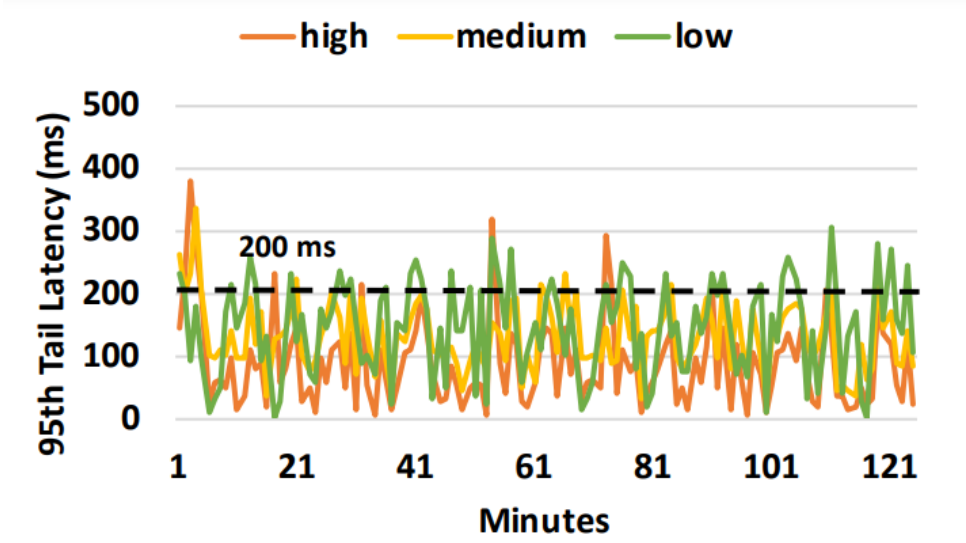

Figure 13 95th percentile latency for ETL topology with TAXI dataset under static workload without priority scheduler

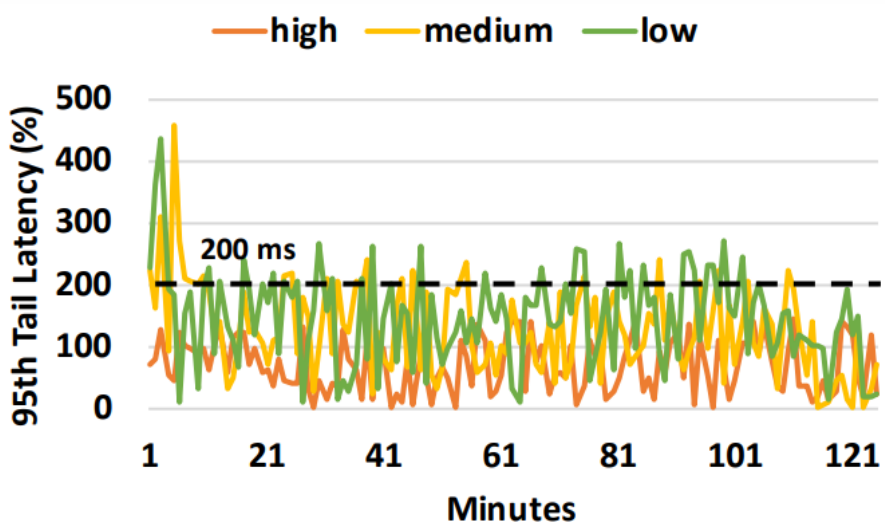

Figure 14 95th percentile latency for ETL topology with TAXI dataset under static workload with priority scheduler

Table 2 Static workload results without priority scheduler (%)

Priority Level	BO	HC-200	HC-50	CPU-400	CPU-4000
Low Priority	3.6	15.11	25.45	29.96	28.67
Medium Priority	4.3	24.63	19.84	26.78	25.46
High Priority	2.1	28.71	23.32	17.79	29.21

Table 3 Static workload results with priority scheduler (%)

Priority Level	BO	HC-200	HC-50	CPU-400	CPU-4000
Low Priority	8.6	28.76	34.43	48.36	54.59
Medium Priority	3.4	16.24	22.32	21.63	25.41
High Priority	0	0	0	0	0

The research findings in Figure 15 and 16, suggests that BO surpasses all alternative methods by achieving the SLO target rapidly. As illustrated in the bar chart presented in Figure 17, the employment of the tactic results in a reduction of the rate of SLO violations by no less than 24% when compared to the utilization of the hill climbing approach. The susceptibility of the hill climbing approach to decision-making primarily grounded in the localized behavior of the objective function is the underlying cause of this phenomenon. On the contrary, Bayesian Optimization (BO) facilitates the making of more informed decisions through the utilization of a data-driven approach that models the global behavior of the objective function based on a sample of available data. The utilization of BO exhibits a notable reduction in the rate of SLO violation, achieving an 85.1% decline in comparison to the Storm Rebalancing approach. The efficacy of tail latency reduction of SR1 and SR2 appears to be impeded by the overheads inherent in the process of rebalancing the ETL topology upon the addition of new workers. It is noteworthy to mention that augmenting the number of workers results in an escalation in the inter-process communication cost. In contrast to Storm Rebalance, our approach of updating Container CPU limits yields an instantaneous effect on the tail latency.

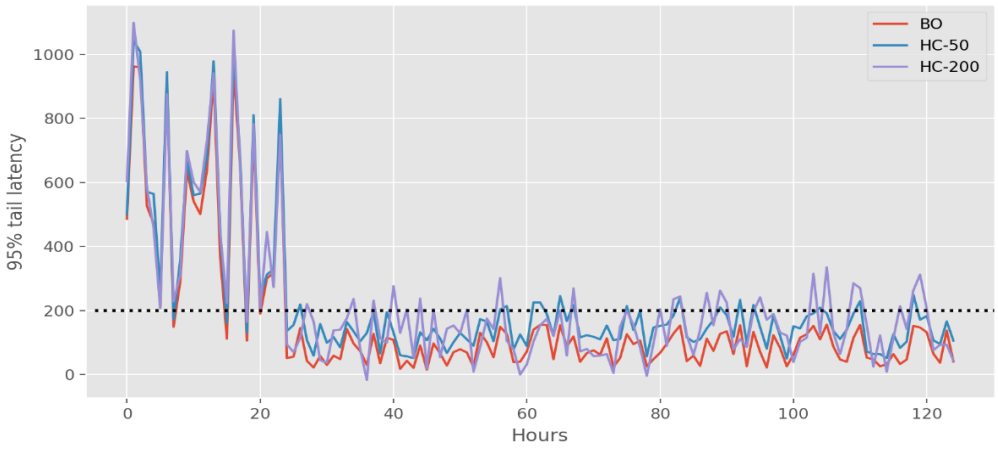

Figure 15 Bayesian Optimization vs Hill Climbing method

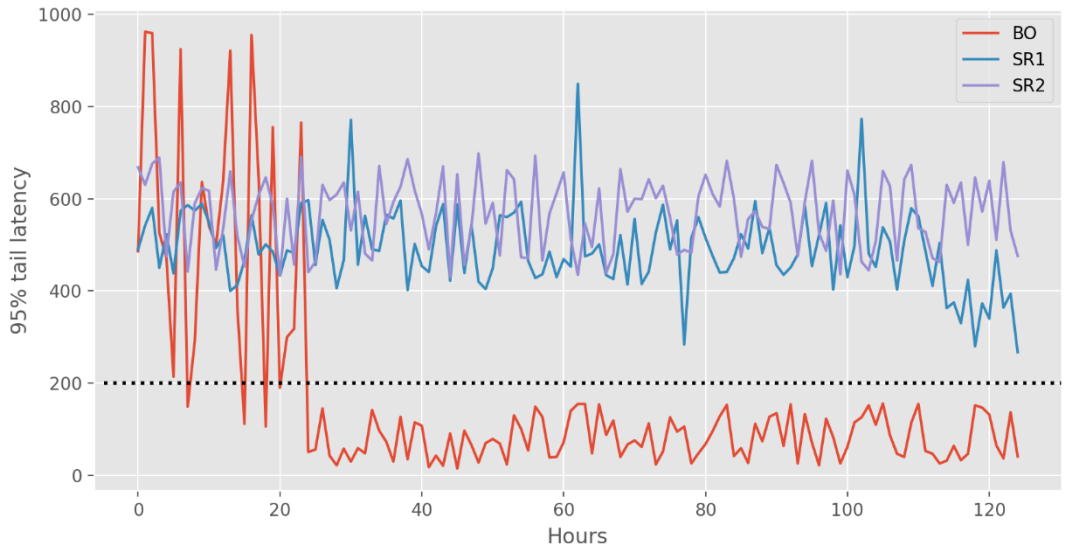

Figure 16 Bayesian Optimization vs Hill Climbing method

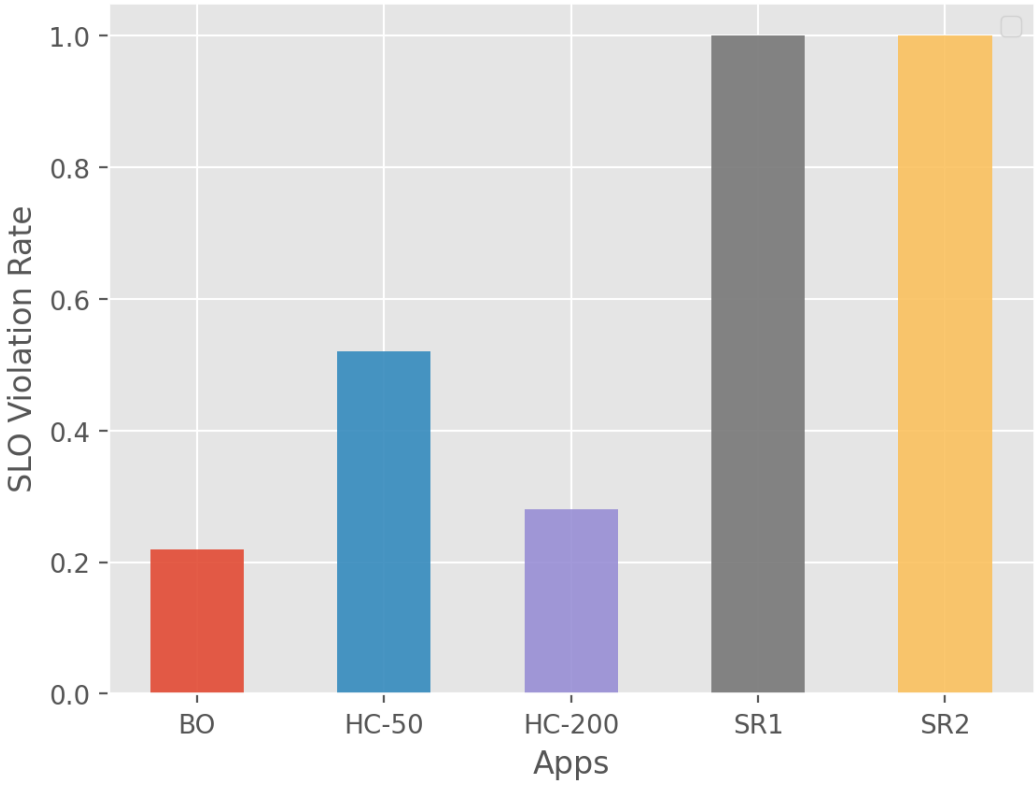

Figure 17 SLO Violation rate under Static Workload

Evaluation under Dynamic Workload

The current study seeks to determine the influence of prioritized scheduling on Service Level Objective (SLO) infractions of numerous concurrent applications operating amidst fluctuating workload dynamics. As evidenced by the illustration in Figure 18. The entities ETL-CITY and PRED-TAXI are subject to fluctuating workloads, in contrast to ETL-TAXI and PRED-CITY, which operate under static workloads. The entirety of the tasks involved spans a duration of 24 hours and comprises recurring patterns. In this study, a comparative analysis was performed on dynamic resource allocation techniques, including VRebalance [4] BO, and Hill Climbing utilizing two distinct step sizes (50 millicore and 200 millicore). Additionally, a static resource allocation approach was also assessed using two varying levels of resource allocation (400 millicore and 4000 millicore), representing partial and full resource utilization, respectively.

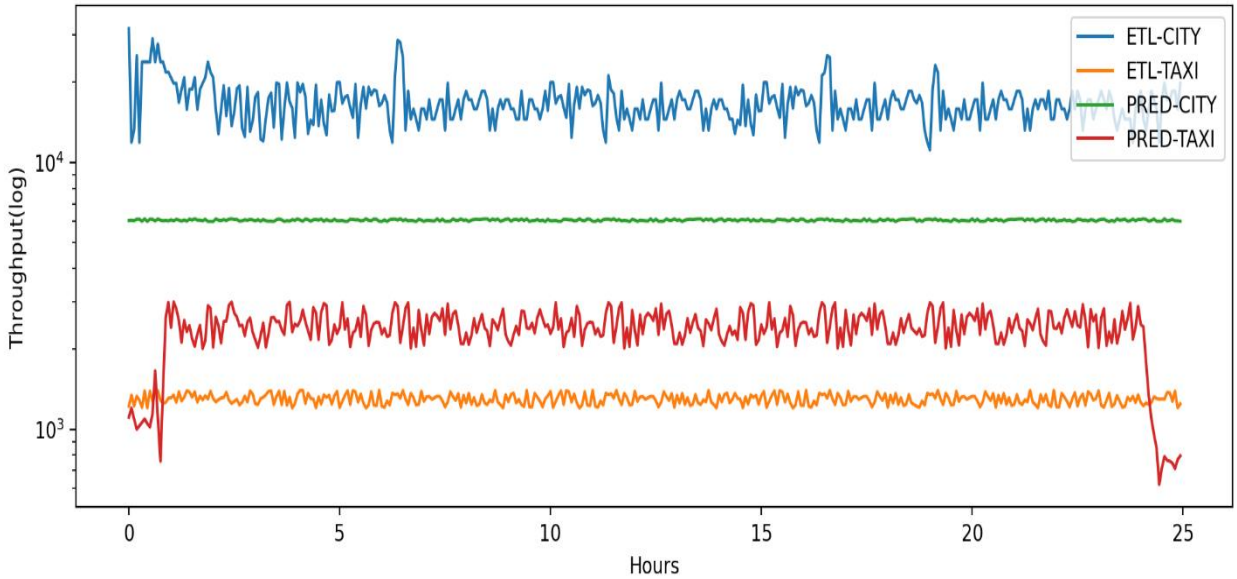

Figure 18 Both static and dynamic workloads. Throughput is measured in tuples per minute and shown in log scale

In Figure 19, a comparison is made of the rates of SLO violations across all methods. It is essential to understand the significant impact that technology has bestowed upon our daily lives. Modern advancements have redefined the way we communicate, work, and access information, rendering traditional methods obsolete. The reliance on technology has reached unprecedented levels, with an increasing number of individuals using it as a fundamental tool for almost all aspects of their lives. As such, it is imperative to recognize the undeniable role technology plays in our society and its ongoing evolution. Table 4 and 5 presents a comparative analysis of the reduction in SLO violation rate brought about by BO in relation to HC-200, HC-50, CPU-4000, and CPU-400 with and without priority scheduler. The utilization of Bayesian optimization (BO) exhibits a noteworthy reduction in the rate of service level objective (SLO) violations, ranging from 53% to 60% in comparison to the hill climbing approach, and ranging from 55.7% to 72% in comparison to the static resource allocation method. The CPU-400 is plagued by significant violations of Service Level Objectives (SLOs) as a result of being inadequately provisioned to handle escalating workloads, as demonstrated by the ETL-taxi software. In relation to the CPU-4000, it can be inferred that every application is granted unhampered utilization of the entire computational resources offered by the Edge server, provided that such capacity is feasible. Consequently, the competition for central processing unit (CPU) resources and the interference that arises between applications under conditions of elevated workloads lead to a number of violations of service level objectives (SLOs). In comparison to BO, the hill climbing approaches (HC-200 and HC-50) exhibit inferior performance as a result of the inadequacy of a comprehensive model for the behavior of the objective function.

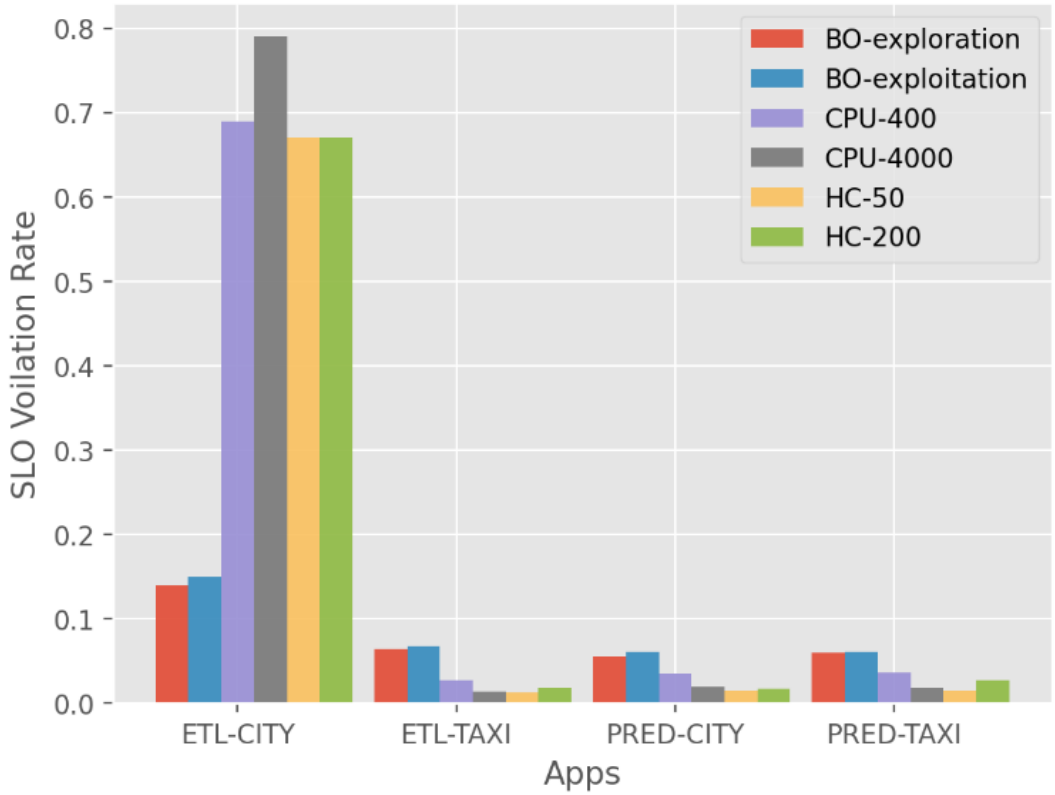

Figure 19 SLO Violation rate for all applications

Table 4 Improvement in SLO violation rate using BO without priority scheduler (%)

vs	HC-200	HC-50	CPU-400	CPU-4000
BO-Exploration	60.1	60.48	64	72.4
BO-Exploitation	59.28	59.18	58.24	70.8

Table 5 Improvement in SLO violation rate using BO with priority scheduler (%)

vs	HC-200	HC-50	CPU-400	CPU-4000
BO-Exploration	53.6	53.1	55	66.4
BO-Exploitation	52.28	52.28	53.24	65.8

The bar charts presented in Figures 20-27 and Tables 6 and 7, we analyzed the incidence of SLO violations associated with various resource allocation techniques, both with and without the incorporation of priority scheduling. The findings presented in Figure 20, 22, 24, and 26 demonstrate that, when priority scheduling is not implemented, all resource allocation techniques result in breaches of service level objectives for all types of data and applications. The ETL-CITY system experiences SLO violations of as much as 30% even for data that is deemed to be of high-priority. On the other hand, it is worth noting Figures 21, 23, 25, and 27. The findings of the study indicate that the priority scheduler demonstrates the capability to effectively diminish the SLO violation rate pertaining to high-priority data for various applications and resource allocation techniques, achieving a rate of 0%. Based on empirical analysis, it has been determined that BO utilizing priority scheduling exhibits superior aggregate performance, leading to SLO violations of no more than 8.5% for low-priority data and 3% for medium priority data.

Table 6 SLO violation for each app without priority (%)

App	BO-exploration	BO-exploitation	HC-200	HC-50	CPU-400	CPU-4000
ETL-City	10.93	9.33	68.45	68.61	74.53	83.34
ETL-Taxi	8	9	3	2	1.8	1.9
PRED-CITY	7.9	11.7	3.5	2.2	1.5	1.7
PRED-Taxi	8	11.1	3	2	1.1	1.5

Table 7 SLO violation for each app with priority (%)

App	BO-exploration	BO-exploitation	HC-200	HC-50	CPU-400	CPU-4000
ETL-City	14.91	14.11	68.11	67.61	69.59	80.91
ETL-Taxi	7.4	6.8	2.7	1.45	1.31	1.87
PRED-CITY	6.6	6.1	3.5	1.9	1.5	1.7
PRED-Taxi	8.9	11.2	3.6	2.8	1.7	2.7

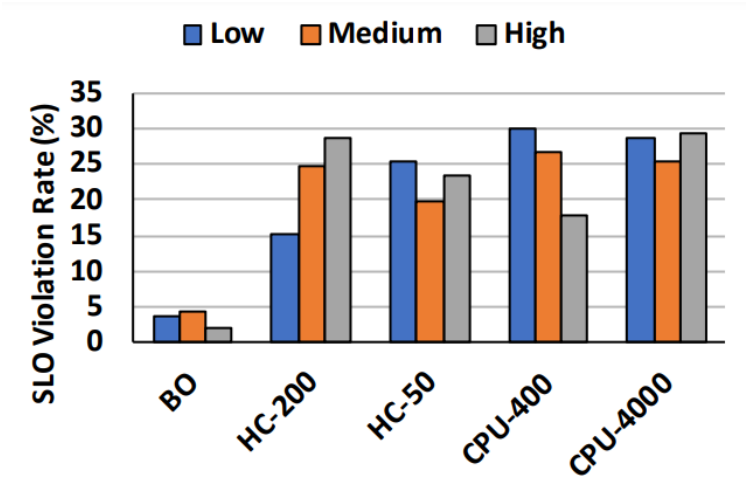

Figure 20 SLO violation without priority scheduler. ETL-CITY

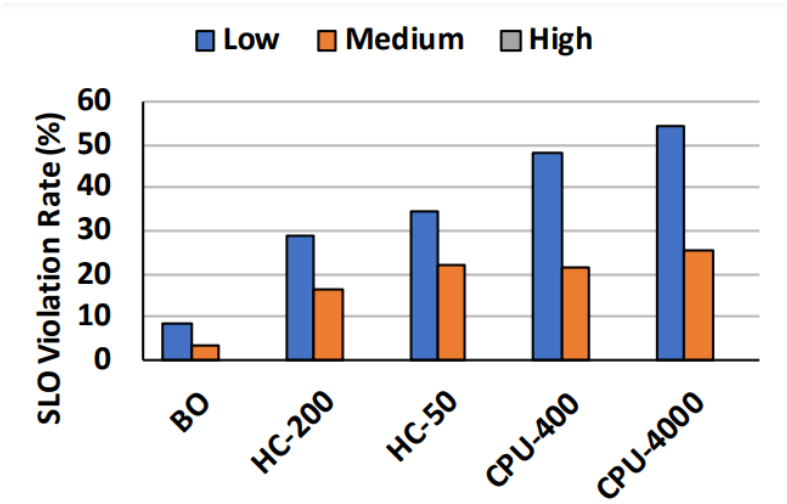

Figure 21 SLO violation without priority scheduler. ETL-CITY

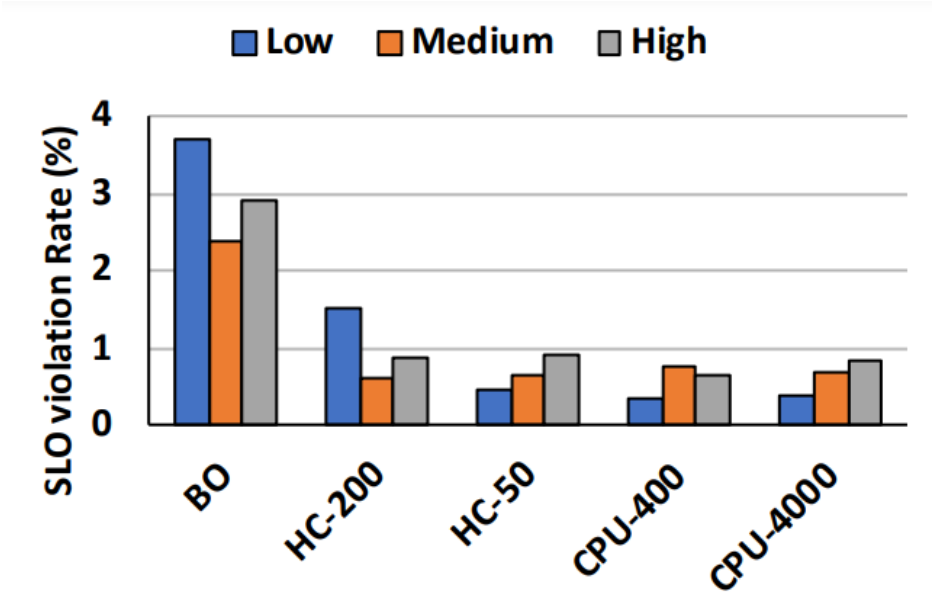

Figure 22 SLO violation without priority scheduler. ETL-TAXI

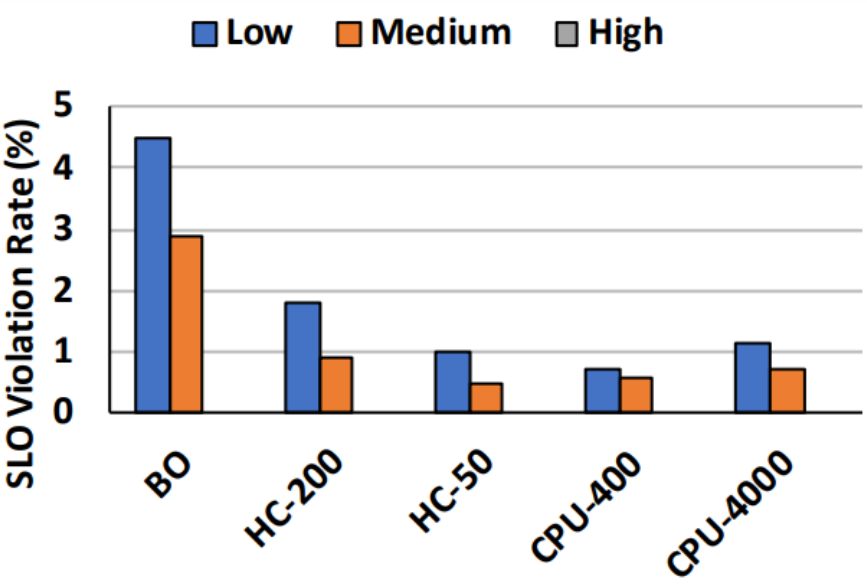

Figure 23 SLO violation with priority scheduler. ETL-TAXI

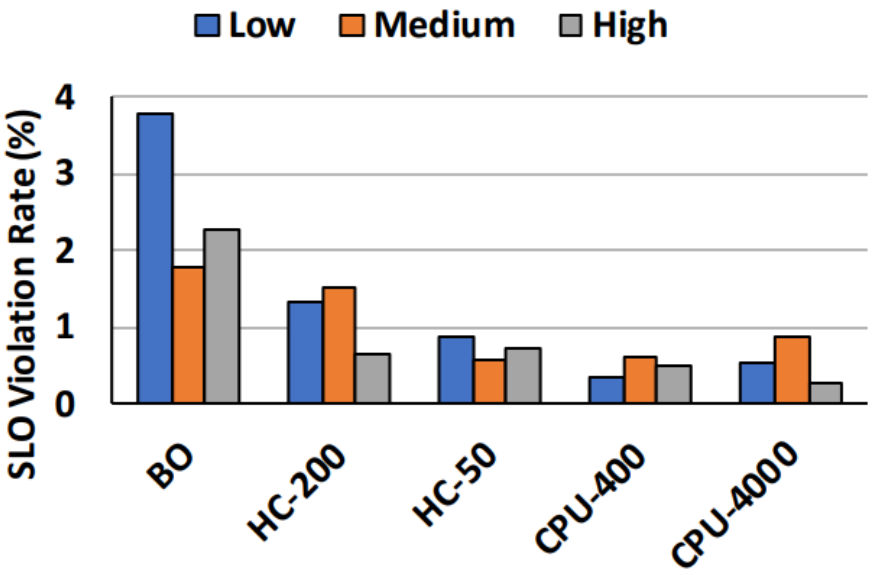

Figure 24 SLO violation without priority scheduler. PRED-CITY

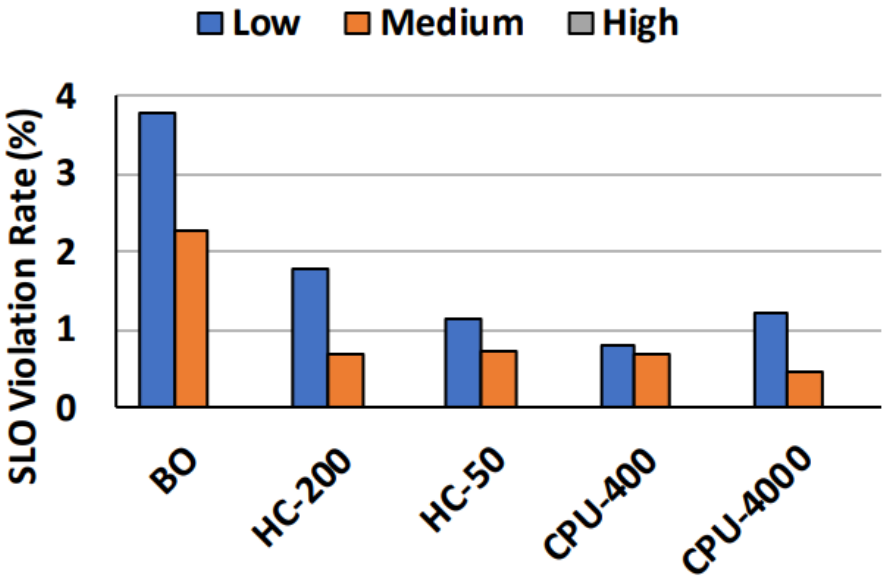

Figure 25 SLO violation with priority scheduler. PRED-CITY

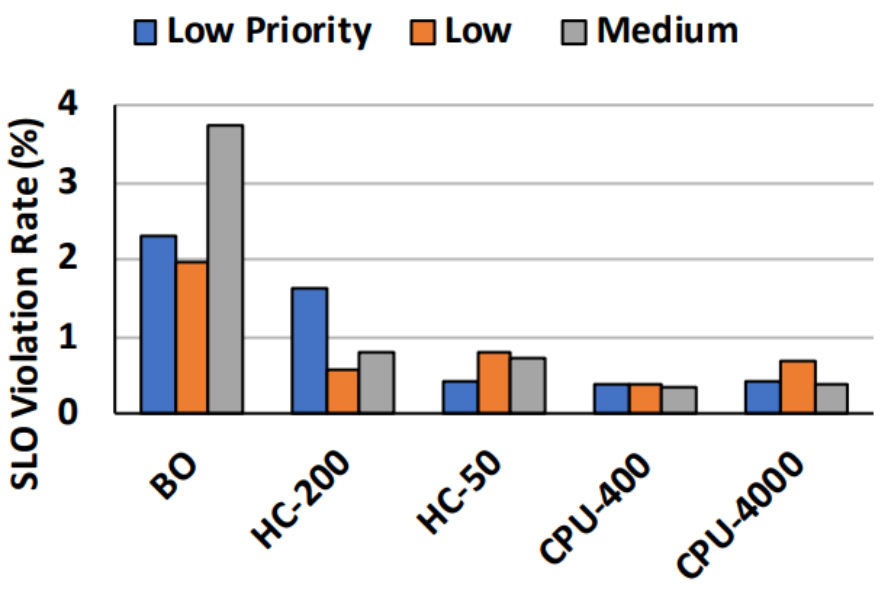

Figure 26 SLO violation without priority scheduler. PRED-TAXI

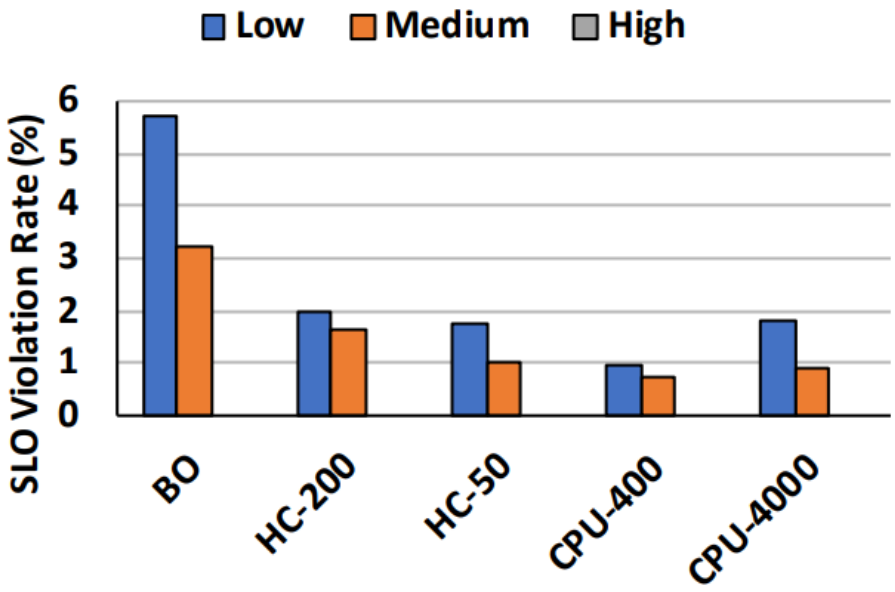

Figure 27 SLO violation with priority scheduler. PRED-TAXI

Evaluation for Different Batch sizes

The selection of batch size is a crucial determinant that could drastically influence the effectiveness and productivity of a stream processing infrastructure. This study aims to analyze the SLO violation rates in the context of BO with priority scheduling using various batch sizes. The present study demonstrates that, across all considered applications, the implementation of a batch size of 10 yields lower rates of SLO violation compared to that of a batch size of 100 as shown in Figures 28 and 29. The employment of a batch size of 10 leads to a considerable decrease in the incidence of Service Level Objective (SLO) violations, with reductions of approximately 55% and 30% observed for low- and medium-priority data correspondingly. When considering high-priority data, the violation rate of the Service Level Objective is found to be 0% for both batch sizes. In general, the reduction of batch sizes enables more frequent system updates, which in turn results in expedited processing times of individual data items. Timely updates play a critical role in real-time processing, highlighting their significance. However, in the case of an insufficient batch size, the efficacy of priority scheduling may be considerably diminished. In the most critical scenario, a singular batch size will have the same effect as the deactivation of prioritization scheduling. Conversely, augmented batch sizes have the potential to yield greater latency within the system as data units necessitate deferral for processing until the whole batch is finished. The present study employs a batch size of 10 as it effectively achieves an optimal equilibrium between prompt responsiveness and differential treatment of data items according to their respective priorities.

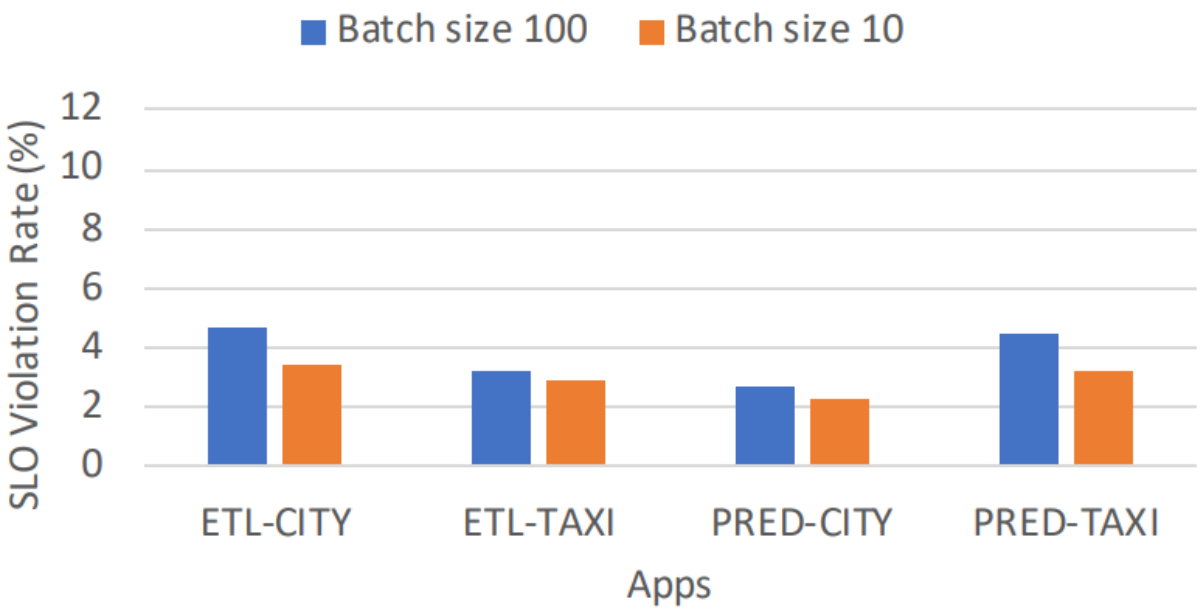

Figure 28 SLO violation rate for medium-priority data for all apps.

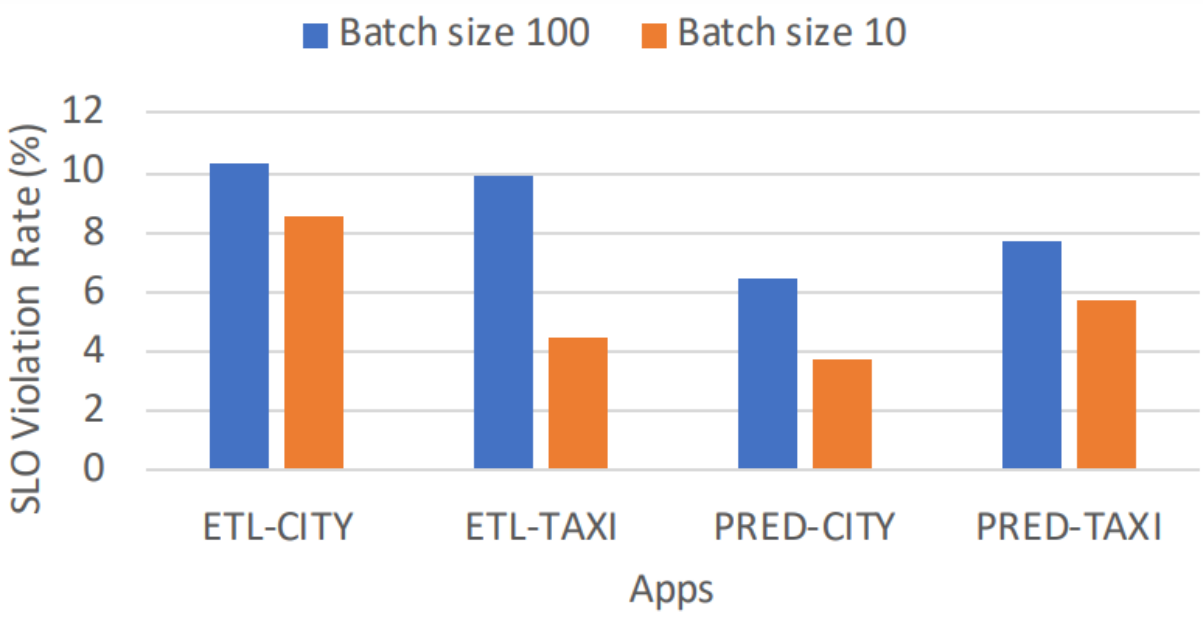

Figure 29 SLO violation rate for low-priority data for all apps

CHAPTER V

CONCLUSION AND FUTURE WORK

The present study introduces VRebalance, a virtual resource orchestrator that offers a comprehensive performance assurance for concurrent stream processing tasks in an Edge computing setting that is resource-limited. The primary focus of our efforts involved devising and enacting a resource management system of high agility and efficacy, drawing inspiration from BO as its foundation. The present study endeavors to examine the influence of data prioritization on the operational efficiency and resilience of pivotal applications within stream processing frameworks that operate under resource limitations at the Edge. The present study indicates that the adoption of a priority-based organizing system for data and subsequent processing of information has the potential to considerably enhance the effectiveness and robustness of the system. This work pertained to the performance analysis of the priority scheduler, with emphasis on its ability to handle high-priority data in RIoTBench applications running concurrently. Of particular interest was the 95th percentile latency, and it was observed that the said scheduler was able to achieve a 0% violation rate of the Service Level Objective (SLO) metric.

In future research, an examination will be conducted on the union of priority scheduling and load shedding in order to mitigate congestion and address increasingly intricate scenarios. The implementation of load shedding can facilitate the discerning exclusion of incoming data during periods characterized by heightened congestion, thereby diminishing the system's burden and averting an overtaxed state. Overall, it is imperative to conduct a thorough evaluation of the

compromises associated with processing efficacy and potential loss of data when incorporating load shedding. It is of critical importance to consider the potential impacts of dropping essential data, as such actions may have substantial repercussions in particular applications. In general, the amalgamation of priority scheduling and load shedding presents a prospect for acquiring significant knowledge on how to effectively manage congestion and enhance the efficacy of stream processing systems. Accordingly, our future endeavors involve the exploration of these techniques through a concentrated lens on assessing their efficacy, discerning potential compromises, and unearthing the most suitable setups for designated utilization scenarios.

REFERENCES

[1] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan, “A serverless real-time data analytics platform for edge computing,” IEEE Internet Computing, vol. 21, pp. 64–71, 01 2017.

[2] C. Qin, H. Eichelberger, and K. Schmid, “Enactment of adaptation in data stream processing with latency implications—a systematic literature review,” Information and Software Technology, vol. 111, pp. 1 – 21, 2019.

[3] S. Ceri, M. Matera, F. Rizzo, and V. Demaldé, “Designing data-intensive web applications for content accessibility using web marts,” Commun. ACM, vol. 50, no. 4, pp. 55–61, 2007.

[4] IBM, "Bringing big data to the enterprise," [Online]. Available: https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html. [Accessed May 2023].

[5] SAS, "Big Data in Big Companies," [Online]. Available: http://www.sas.com/resources/asset/Big-Data-in-Big-Companies-ExecutiveSummary.pdf. [Accessed May 2023].

[6] Gartner, "3D Data Management: Controlling Data Volume, Velocity and Variety," [Online]. Available: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-DataManagement-Controlling-Data-Volume-Velocity-and-Variety.pdf. [Accessed May 2023].

[7] Apache Storm [Online]. Available: http://Storm.apache.org/. [Accessed May 2023].

[8] Apache Hadoop [Online]. Available: http://hadoop.apache.org/. [Accessed May 2023].

[9] Apache Spark [Online]. Available: http://spark.apache.org/. [Accessed May 2023].

[10] S. Di, D. Kondo, and W. Cirne, “Characterization and comparison of cloud versus grid workloads,” in 2012 IEEE International Conference on Cluster Computing, 2012, pp. 230–238

[11] F. Metzger, T. Hoßfeld, A. Bauer, S. Kounev, and P. E. Heegaard, “Modeling of aggregated iot traffic and its application to an iot cloud,” Proceedings of the IEEE, vol. 107, no. 4, pp. 679–694, 2019.

[12] V. Kalavri, J. Liagouris, M. Hoffmann, D. Dimitrova, M. Forshaw, and T. Roscoe, “Three steps is all you need: Fast, accurate, automatic scaling decisions for distributed streaming dataflows,” in Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI’18. USA: USENIX Association, 2018, p. 783–798.

[13] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy, “Dhalion: Self-regulating stream processing in heron,” Proc. VLDB Endow., vol. 10, no. 12, p. 1825–1836, Aug. 2017.

[14] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, “Zenith: Utility-aware resource allocation for edge computing,” in 2017 IEEE International Conference on Edge Computing (EDGE), 2017, pp. 47–54.

[15] A. Araldo, A. D. Stefano, and A. D. Stefano, “Resource allocation for edge computing with multiple tenant configurations,” in Proceedings of the 35th Annual ACM Symposium on Applied Computing, ser. SAC ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1190–1199.

[16] J. Wang, Z. Feng, S. George, R. Iyengar, P. Pillai, and M. Satyanarayanan, “Towards scalable edge-native applications,” in Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, ser. SEC ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 152–165.

[17] J. S. Van Der Veen, B. Van Der Waaij, E. Lazovik, W. Wijbrandi, and R. J. Meijer, “Dynamically scaling apache storm for the analysis of streaming data,” in 2015 IEEE First International Conference on Big Data Computing Service and Applications, 2015, pp. 154–161.

[18] V. Kalavri, J. Liagouris, M. Hoffmann, D. Dimitrova, M. Forshaw, and T. Roscoe, “Three steps is all you need: Fast, accurate, automatic scaling decisions for distributed streaming dataflows,” in Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI’18. USA: USENIX Association, 2018, p. 783–798.

[19] A. Slo, S. Bhowmik, and K. Rothermel, “State-aware load shedding from input event streams in complex event processing,” IEEE Trans. Big Data, vol. 8, no. 5, pp. 1340–1357, 2022.

[20] A. Slo, S. Bhowmik, and K. Rothermel, “Espice: Probabilistic load shedding from input event streams in complex event processing,” in Proceedings of the 20th International Middleware Conference, ser. Middleware ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 215–227

[21] B. Zhao, N. Q. Viet Hung, and M. Weidlich, “Load shedding for complex event processing: Input-based and state-based techniques,” in 2020 IEEE 36th International Conference on Data Engineering (ICDE), 2020, pp. 1093–1104.

[22] L. Fischer and A. Bernstein, "Workload scheduling in distributed stream processors using graph partitioning," in International Conference on Big Data (Big Data), 2015

[23] J. Xu, Z. Chen, J. Tang and S. Su, "T-Storm: Traffic-Aware Online Scheduling in Storm," in International Conference on Distributed Computing Systems (ICDCS 13), 2014.

[24] X. Fu, T. Ghaffar, J. C. Davis, and D. Lee, “Edgewise: A better stream processing engine for the edge,” in Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference, ser. USENIX ATC ’19. USA: USENIX Association, 2019, p. 929–945.

[25] D. Carney, U. C¸ etintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, “Monitoring streams: A new class of data management applications,” in Proceedings of the 28th International Conference on Very Large Data Bases, ser. VLDB ’02. VLDB Endowment, 2002, p. 215–226.

[26] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian optimization of machine learning algorithms,” arXiv [stat.ML], 2012.

[27] H. J. Kushner, “A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise,” J. Basic Eng., vol. 86, no. 1, pp. 97–106, 1964.

[28] D. R. Jones, M. Schonlau, and W. J. Welch, “Efficient global optimization of expensive black-box functions,” Journal of Global Optimization, vol. 13, no. 4, pp. 455–492, 1998.

[29] D. Huang, T. Allen, W. Notz, and R. Miller, “Sequential kriging optimization using multiplefidelity evaluations,” Structural and Multidisciplinary Optimization, vol. 32, pp. 369–382, 2006.

[30] A. J. Keane, “Statistical improvement criteria for use in multiobjective design optimization,” AIAA J., vol. 44, no. 4, pp. 879–891, 2006.

[31] J. M. Calvin and A. ilinskas, “One-Dimensional global optimization for observations with noise,” Comput. Math. Appl., vol. 50, no. 1–2, pp. 157–169, 2005.

[32] J. Calvin and A. Zilinskas, “One-dimensional P-algorithm with convergence rate O(n-3+ ˇ) for smooth functions,” Journal of Optimization Theory and Applications, vol. 106, no. 2, pp. 297–307, 2000.

[33] K. Swersky, J. Snoek, and R. P. Adams, “Multi-task Bayesian optimization,” in Advances in Neural Information Processing Systems, 2013, pp. 2004–2012.

[34] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, “Fast Bayesian optimization of machine learning hyperparameters on large datasets,” arXiv [cs.LG], 2016.

[35] J. Wang, S. C. Clark, E. Liu, and P. I. Frazier, “Parallel Bayesian global optimization of expensive functions,” Oper. Res., vol. 68, no. 6, pp. 1850–1865, 2020.

[36] J. Wu and P. I. Frazier, “The parallel knowledge gradient method for batch Bayesian optimization,” arXiv [stat.ML], 2016.

[37] E. Mehdad and J. P. C. Kleijnen, “Efficient global optimisation for black-box simulation via sequential intrinsic Kriging,” J. Oper. Res. Soc., vol. 69, no. 11, pp. 1725–1737, 2018.

[38] P. Salemi, B. L. Nelson, and J. Staum, “Discrete optimization via simulation using Gaussian Markov random fields,” in Proceedings of the Winter Simulation Conference 2014, 2014.

[39] C. E. Rasmussen and C. K. I. Williams, Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press, 2005.

[40] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” in Advances in Neural Information Processing Systems, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, 2012.

[41] Y. Zhao and H. Zeng, "The Concept of Unschedulability Core for Optimizing Real-Time Systems with Fixed-Priority Scheduling," IEEE Transactions on Computers, vol. 68, no. 8, pp. 926-938, 2019.

[42] J. Mockus, Bayesian approach to global optimization: theory and applications. Springer Science & Business Media, 2012.

[43] A. Shukla, S. Chaturvedi, and Y. Simmhan, “Riotbench: An iot benchmark for distributed stream processing systems,” Concurrency and Computation: Practice and Experience, vol. 29, no. 21, p. e4257, 2017, e4257 cpe.4257.

[44] Data Canvas. Sense your city: Data art challenge. [Online]. Available: http://datacanvas.org/sense-your-city/ [Accessed May 2023].

[45] B. Donovan and D. Work, New York City Taxi Trip Data (2010-2013), [Online]. Available: https://doi.org/10.13012/J8PN93H8 [Accessed May 2023].

[46] A. Shukla, S. Chaturvedi, and Y. Simmhan, “Riotbench: An iot benchmark for distributed stream processing systems,” Concurrency and Computation: Practice and Experience, vol. 29, no. 21, p. e4257, 2017, e4257 cpe.4257.

[47] U. Tadakamalla and D. A. Menasce, “Characterization of iot workloads,” ´ in Edge Computing – EDGE 2019, T. Zhang, J. Wei, and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 1–15

[48] S. Khare, H. Sun, K. Zhang, J. Gascon-Samson, A. Gokhale, and X. Koutsoukos, “Poster abstract: Ensuring low-latency and scalable data dissemination for smart-city applications,” 04 2018, pp. 283–284.

[49] A. C. Cameron and P. K. Trivedi, Regression Analysis of Count Data, 2nd ed., ser. Econometric Society Monographs. Cambridge University Press, 2013

[50] A. Brampton. (2018) Measuring percentile latency. [Online]. Available: https://blog.bramp.net/post/2018/01/16/measuring-percentile-latency/ [Accessed May 2023].

[51] P. Kang, P. Lama, and S. U. Khan, “SLO-aware virtual rebalancing for edge stream processing,” in 2021 IEEE International Conference on Cloud Engineering (IC2E), 2021.

[52] A. Shahid, P. Kang, S. Khan, and L. Palden, “Some New Observations on SLO-aware Edge Stream Processing.” Unpublished.