National College of Ireland

Optimization of Multi-Cloud Workload Placement for Performance and Cost Efficiency

MSc Research Project
Masters in Cloud Computing

Rishabh Sinha
Student ID: X21171203

School of Computing
National College of Ireland

Supervisor: Dr Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rishabh Sinha
Student ID:	X21171203
Programme:	Masters in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr Aqeel Kazmi
Submission Due Date:	24/01/2024
Project Title:	Optimization of Multi-Cloud Workload Placement for Performance and Cost Efficiency
Word Count:	XXX
Page Count:	25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	27th January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimization of Multi-Cloud Workload Placement for Performance and Cost Efficiency

Rishabh Sinha
X21171203

Abstract

This research project aims to optimize the performance and cost efficiency of deploying multiple application components on various cloud providers in a multi-cloud environment. The project proposes a solution based on Stochastic Hill Climbing and Simulated Annealing algorithms to search for cost-efficient and optimal configuration parameters in Azure and AWS cloud environments. The solution retrieves current pricing and instance-specific configuration data from AWS and Azure APIs, and searches configuration parameters for up to 20 application components. The project also conducts experiments with various advanced machine learning algorithms to predict the optimal CPU and memory requirements of a workload for optimal performance. In addition, the project implements a simulation script for the new workload to be tested by executing the workload on a Docker container and estimating the CPU and memory requirement of the workload based on application characteristics. Both of these are implemented for performance optimization. The proposed solution suggests which application components should be deployed to which cloud service provider, and aims to provide an optimal solution for cost optimization in a real-world scenario.

1 Introduction

Current organizational production workloads often include diverse application components running on varied infrastructures, each with distinct computational parameters tailored to the workload's memory or compute requirements. For example, a workload can consist of "web_server", "database_server", "monitoring_server", "cache_server" and many other application components. These application components are integrated with various monitoring capabilities that provide insights into application usage which is a useful data asset that enables estimation of optimal workload performance.
The use of Containers as a service for executing such application components makes it easy to select desired underlying infrastructure on any given CSP(Cloud Service Provider) platform given the application portability advantages Liu et al. (2023).
In the current landscape of organizational modernization, there's a significant emphasis on adopting Cloud services, containerization, and orchestration. This involves configuring specific parameters such as instance types, allocating CPUs, and memory (In this project we term it as configuration parameters), which is important for optimizing the underlying infrastructure to ensure efficient workload execution. According to Gartner's forecast, global spending on public cloud services is projected to surge by 21.7%, reaching a total of $597.3 billion, a notable increase from the $491 billion recorded in 2022 Gartner

(2023)DeLisi and Howley (2023). Effectively harnessing the offerings of CSPs' cloud services and IaaS(Infrastructure as a service) resources while minimizing costs and optimal workload performance emerges as a significant challenge in this evolving landscape.

1.1 Research Background/ Objective

The main objective of this research is to thoroughly compare the offerings of Azure and AWS, aiming to understand the cost differences between these two cloud platforms and use them to formulate a solution. The objective is to develop a solution that helps users choose the best configuration parameters, considering both cost and performance for the various application components of a workload. The study explores different types of algorithms, focusing on predictive and search-based, to come up with practical solutions. The aim is to efficiently pick the right configuration settings for multiple application components of a workload in less time. This approach allows users to estimate the configuration parameters of multiple application components of a workload collectively, making it easier to estimate pricing and deploy different parts of an application across various cloud providers, like Azure and AWS. This not only addresses concerns about being tied to a specific provider but also helps in reducing costs. The overall objective is to improve both performance and cost-effectiveness by choosing the most suitable cloud service provider (CSP) with the best configuration settings for running various application components. The research tackles challenges related to avoiding vendor lock-in, reducing costs, and optimizing performance in the context of placing workloads with multiple application components across multiple cloud environments. These issues can all fall under the umbrella of "problems related to placing workloads in multi-cloud environments" which will be referred to here as "multi-cloud workload placement problems"Chen et al. (2021). The objective is to provide a real-world practical guide for decision-making when selecting cloud providers and configuration settings based on the specific needs of a workload.

1.2 Research Question

The growing dependence of enterprises on cloud services from major providers like AWS and Azure highlights the need for careful selection of cloud service providers and configuration of multiple application components. Through the use of search-based algorithms and prediction techniques, this research aims to remove the complexities surrounding the allocation of workloads across multiple cloud environments by answering the research question, **"How can search-based algorithms and predictive methods be used to select the cloud service provider and optimal configuration for multiple application components of a complex workload in the multi-cloud settings?"** that directs this research project.

1.3 Document/Report Structure

Each section of the research project is thoughtfully structured to offer an exhaustive overview: Section 2, Related Work, explores existing practices, predictive and search-based strategies, and challenges in optimizing multi-cloud workloads. Section 3, Methodology, defines the approach to tackle the multi-cloud workload placement problem by clearly defining the issue, specifying data requirements, and introducing predictive and

search algorithms. Section 4, Design Specification, provides a detailed two-phased solution design, providing a clear roadmap for the realization of research objectives. Section 5, Implementation, guides through tools and software, presenting a systematic process for implementing predictive and search phases. Section 6, Evaluation assesses and validates the proposed solution. Finally, Section 7, Conclusion and Future Work, summarizes findings, and contributions, and outlines future research directions.

2 Related Work

There are several various algorithms and approaches implemented to solve the problem of optimal configuration parameter selection and this paper explores them in two phases to propose a two-phase solution in selecting optimal CSP for better workload performance at less cost.

2.1 Prediction-based Machine Learning strategies

This research explores predictive strategies utilizing statistical models to estimate workload performance across diverse cloud settings. By projecting workload performance based on input cloud computation requirements (vCPU, RAM), these predictive models facilitate the identification of optimal cloud computational parameters. Some literature proposes predictive algorithms leveraging offline data from benchmarking various frameworks, enhancing prediction accuracy and reducing time and cost for model training.
Venkataraman et al. Venkataraman et al. (2016) introduce the framework Ernest, forecasting large-scale analytics systems in multi-tenant setups. Despite scalability issues, Ernest accurately predicts execution time, demonstrating high accuracy on Amazon EC2. In a related study Hou et al. (2022), predictive models for HPC systems are presented, utilizing random forests, SVM, neural networks, and decision trees for forecasting task performance, and optimizing resource allocation. Witt et al. (2019) addresses distributed batch processing systems using random forest-based models for performance forecasting, aligning with Mahgoub et al. (2020). In Mohapatra and Oh (2023), Smartpick predicts query completion time and cost, comparing favorably to Cocoa and SplitServe. Mahgoub et al. (2020) predicts VM performance using a random forest model, considering constraints for optimal resource utilization. Newaz and Mollah (2023) employs a two-stage technique for high-memory jobs, reducing prediction errors and training costs. The paper by De Gooijer and Hyndman (2006) explores ARIMA and ANN models, while Devi and Valli (2023) introduces a hybrid ARIMA–ANN model, both for cloud workload prediction. In Saxena et al. (2023), cloud workload prediction models are systematically evaluated, and categorized into quantum learning, ensemble learning, hybrid learning, deep learning, and evolutionary neural networks, with key performance indicators assessed. The literature review highlights diverse approaches, emphasizing predictive models' role in workload performance prediction based on existing workload usage data, optimizing cloud configurations, and resource allocation.

2.2 Challenges in Predictive-based Machine Learning strategies

Predictive techniques offer cost-effective assessments using historical data but have limitations of potential noise from substantial offline data Mahgoub et al. (2020). In a survey by Masdari and Khoshnevis (2020), AdaBoost and Random Forests stand out for their

accuracy and stability in handling noisy data. This project will access similar predictive models to predict CPU and Memory requirements of a workload, comparing methods like Gradient Boosted, Linear Regression, SVM, Random Forest, ARIMA, and an ensemble model. The results guide pricing decisions through search-based algorithms on publicly available datasets from CSPs like Azure and AWS.

Algorithms	**Type**	**Paper**
Linear Regression	Predictive	Venkataraman et al. (2016)
Random Forest	Predictive	Mahgoub et al. (2020), Witt et al. (2019)
ARIMA	Predictive	Devi and Valli (2023)
SVM	Predictive	Hou et al. (2022)
Ensemble	Predictive	Masdari and Khoshnevis (2020)

Table 1: Summary of Predictive Algorithms and Related Papers

2.3 Search Based Black box strategies

In the study by Alipourfard et al. Alipourfard et al. (2017), CherryPick is introduced as a search-based black-box methodology for iterative assessing cloud configurations. It effectively combines optimization strategies, such as genetic algorithms and hill climbing, and adapts the exploration budget dynamically. CherryPick outperforms strategies used in Venkataraman et al.'s work Venkataraman et al. (2016), which relies on predictive models based on historical data. Bilal et al. Bilal et al. (2020) extend Bayesian Optimization to cloud deployment parameters, evaluating eight black-box optimization algorithms. Their focus on gradient-boosted regression trees (GBT) in conjunction with Bayesian Optimization showcases the efficiency of this approach in determining optimal cloud computational requirements. They also compare their method with simulated annealing, a stochastic hill-climbing algorithm that probabilistically accepts worse solutions to escape local optima. Hsu et al. Hsu, Nair, Freeh and Menzies (2018) address the VM selection challenges with the Arrow strategy, using Bayesian optimization and low-level performance data. The experiments reveal that Arrow outperforms other optimization strategies in terms of cost-effectiveness. Arrow also incorporates simulated annealing to improve the exploration of the search space and avoid premature convergence. In a related context, Hsu et al. Hsu, Nair, Menzies and Freeh (2018b) introduces SCOUT, leveraging previous data and low-level performance parameters to improve search performance and reduce costs. SCOUT outperforms Random Search and CherryPick Alipourfard et al. (2017). SCOUT employs a stochastic hill-climbing algorithm that iterative selects the best neighbor of the current solution until no improvement is possible. Hsu et al. Hsu, Nair, Menzies and Freeh (2018a) formulate the cloud instance selection challenge as a multi-armed bandit problem and propose the Micky collective optimization technique. Micky achieves near-optimal solutions with significantly reduced measuring costs compared to state-of-the-art methods like CherryPick Alipourfard et al. (2017). Micky also uses simulated annealing to balance the exploration and exploitation trade-off in the multi-armed bandit setting.

In Ansari Shiri et al. (2023) the author introduces the Filter-Wrapper Binary Equilibrium Optimizer Simulated Annealing (FWBEOSA) method for feature selection, showcasing enhanced performance in classification accuracy, selected features, and convergence speed compared to the Binary Equilibrium Optimizer (BEO) and other algorithms. whereas

in Yan et al. (2023) the author focuses on intelligent scheduling algorithms, revealing that Simulated Annealing surpasses genetic algorithms in terms of runtime and total resource cost, aligning better with user needs for cloud services. Similarly in Sa'ad et al. (2023) the author proposes a Cuckoo-based Discrete Symbiotic Organisms Search (C-DSOS) strategy for task scheduling, where Simulated Annealing enhances convergence and solution quality within the Symbiotic Organisms Search (SOS) algorithm. C-DSOS outperforms the Simulated Annealing Symbiotic Organism Search (SASOS), especially in mitigating the degree of imbalance for large-scale tasks. Collectively, the reviewed literature underscores the effectiveness of Simulated Annealing across diverse cloud computing challenges, emphasizing its robustness and superiority.

In the comparative exploration of dynamic resource allocation strategies in cloud computing, two distinct papers, Singh and Choudhary (n.d.) and Achar (2023), present innovative approaches employing various optimization algorithms. The Singh and Choudhary (n.d.) introduces a dynamic priority-based spill-over technique and short life/long life containers to tackle fragmentation challenges, incorporating algorithms such as round robin, AMLB, TLB, ant colony, honey bee, firefly, and notably, stochastic hill climbing (SHC). The study highlights stochastic hill climbing as a local optimization algorithm that outperforms others in resource utilization, response time, and throughput, emphasizing its stability and ability to evade local optima traps. Experimental evaluations, utilizing the CloudSim simulator and Google workload trace data, affirm the effectiveness and efficiency of this approach. Similarly Achar (2023) introduces Neural-Hill, a novel algorithm that integrates a Deep Neural Network (DNN) with a Random Restart Hill Climbing (RRHC) approach for scheduling IoT-Cloud resources with scalability in mind. The DNN predicts VMs' computational loads in the upcoming scheduling cycle, while RRHC optimizes task allocation based on the predicted state space landscape. RRHC's application successfully identifies underloaded and overloaded VMs, contributing to effective load balancing and improved service quality. Comparative experiments showcase substantial enhancements over existing solutions concerning optimal solution-finding time, execution time, routing overhead, and throughput. Both studies contribute valuable insights into resource allocation challenges, the emphasis on stochastic hill climbing in Singh and Choudhary (n.d.) underscores its significance as a local optimization technique with superior performance characteristics. This comparison highlights the distinctive strengths and applications of stochastic hill climbing in the context of dynamic resource allotment in cloud computing.

2.4 Research Niche

Upon reviewing the literature, it becomes evident that optimization methods, notably simulated annealing, and stochastic hill climbing, hold great promise in efficiently addressing the intricate challenges associated with optimizing cloud setup parameters. It is important to note that a significant portion of past research has primarily focused on single-cloud scenarios and with a focus on single application components. In such cases, search-based strategies, particularly those employing simulated annealing, prove to be highly effective. The study underscores the effectiveness of these strategies over predictive methods, showcasing substantial benefits in terms of both cost and performance efficiency.

There is a noticeable research gap, as the majority of studies have concentrated on single-cloud providers, leaving multi-cloud deployment scenarios underexplored. This gap em-

phasizes the need for further investigation and advancement in optimizing multi-cloud environments, specifically for multiple application components of workloads across various cloud platforms.
This research aims to bridge this gap by introducing and utilizing optimal algorithms in 2 phase solution (Evaluating Predictive Algorithms for workload performance estimation and Search based algorithms for selecting cloud service providers with optimal configuration parameters), specifically simulated annealing and stochastic hill climbing, to select optimal cloud configurations in a multi-cloud environment. The significance of these algorithms lies in their robustness and adaptability, essential attributes for addressing the distinctive challenges inherent in complex and dynamic multi-cloud settings. Despite advancements in single-cloud optimization and the predominant focus on Bayesian Optimization in existing work, the clear conclusion drawn from this research is that, for selecting the best computational parameters and cloud service providers for various application components, search-based black-box strategies outperform predictive strategies in terms of both cost and performance efficiency.

3 Methodology

This section will outline the proposed methodological approach to address the multi-cloud workload placement problem by adapting promising Predictive and search-based black-box optimization algorithms from single-cloud environments to multi-cloud environments and solving the problem of vendor lock-in, Cost Optimisation, and Performance Optimisation.

3.1 Problem Formulation:

Achieving the best performance and cost efficiency for a workload involves understanding its computational needs. This is influenced by factors like whether the workload runs in parallel or serial, adding a layer of complexity. Predicting these performance requirements is a challenge and depends on cases like predicting the performance with the help of past available usage data of the workload and alternative cases like newly introduced applications/workloads that have never been evaluated need to be evaluated on a simulation environment for performance estimation. Once we can predict how a workload will perform, the next challenge is choosing the right configuration parameter from multiple cloud providers (like Azure and AWS). This involves finding the optimal cloud configuration from the large CSP offering data sets that align with the estimated performance requirement of the workload, ultimately reducing the cost of running that workload while providing better performance.
The complexity and wide variability of available options in a cloud environment make multi-cloud performance and costs even more challenging. Several virtual machines with diverse instance types are available on public cloud platforms, dispersed across multiple locations and operating systems. Determining the best location to distribute workloads gets more challenging. For instance, for applications requiring a large number of application components, the conventional "brute force" approach of evaluating all options and selecting the least expensive one is ineffective and needs to employ algorithms that can overcome the given issue.
This research uses two approaches to address these issues. First, to achieve optimal performance, the project uses historical workload utilization data to estimate CPU and

Memory usage. This paper uses a variety of prediction models, such as Random Forest, SVM Models, Gradient Boosting Tree, Linear Regression, and Ensemble Learning, and finds the best algorithm for estimating CPU and Memory consumption percentages by doing a comparative analysis. We also investigate performance estimation for recently introduced workloads for which there is no available historical data. This involves running workloads on the Docker engine in a container while taking into account variables such as the number of tasks, the number of instructions per task, the elements of the parallel part of the application programme, processes, and threads per CPU.
Finding the ideal cloud configuration using multi-cloud pricing data is the next step after obtaining the CPU and Memory utilization requirements. To create a search database gathered from several cloud providers' public APIs and endpoints, the research assesses price and different instances across Azure and AWS cloud environments. A comprehensive analysis of search-based algorithms is carried out in the study, which includes traditional Brute Search, tuned, Greedy Search, Stochastic Hill Climbing, and Simulated Annealing algorithms. This analysis determines the most efficient algorithm, which is then integrated into the research's ICT solution. This exhaustive method allows us to effectively control expenses in the ever-changing multi-cloud environment while also forecasting workload performance, including newly introduced workloads hence answering the proposed research question.

3.2 Data Requirements

3.2.1 Historical Data for CPU and Memory Requirement Prediction

This study uses a dataset sourced from Bitbrains IT Services Inc. to assess various predictive algorithms using historical workload data Shen et al. (2015). The dataset encompasses performance metrics from virtual machines (VMs) within Bitbrains' distributed data center. The dataset features 500 VMs designated as the Rnd trace. The Rnd directory categorizes files into three sub-directories based on the month of metric collection Shen et al. (2015). The dataset comprises metrics related to the number of CPU cores, supplied CPU capacity (measured in MHz), CPU utilization (measured in MHz), and CPU usage percentage. Memory-related metrics encompass provided memory (measured in kilobytes) and currently used memory (measured in kilobytes). Disk-related metrics include disk read and write throughput, both measured in kilobytes per second (KB/s). These metrics play a fundamental role in the research, aiding in the accurate selection of algorithms for performance forecasting based on various features.

3.2.2 AWS and Azure (Multi-Cloud) Pricing Data

This paper constructs the AWS and Azure dataset using the Price List Bulk API [1], utilizing the AWS Bulk API for AWS services and the Azure Retail API for Azure services. The AWS Price List Bulk API[2] is accessed through the endpoint `https://api.pricing.us-east-1.amazonaws.com`, offering a means to efficiently gather substantial pricing information for EC2 services. Similarly, the Azure Retail API, accessible through `https://shorturl.at/fgzE2`. These APIs facilitate the exploration of prices for

[1]AWS Bulk API: `https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/using-the-aws-price-list-bulk-api.html`

[2]Azure Retail API: `https://shorturl.at/lwCFU`

Azure services in different regions and SKUs, offering programmatic examples for seamless integration. This dataset forms a crucial component in the decision-making process for selecting the most cost-effective and performance-oriented cloud configurations for specific workload demands. These acquired data are parameterized into categories such as family, type or size, physical processor, processor architecture, spot price, and other pertinent details. The inclusion of these low-level details is crucial for accurately selecting cloud configuration parameters that align with desired workload requirements, ensuring optimal cost and performance. The resulting dataset, encompassing a comprehensive exploration of available cloud configurations and SKUs, is analyzed by a custom function incorporating the available APIs and saved in JSON format for utilization by the ICT solution.

3.3 Algorithms in consideration

3.3.1 Predictive Algorithms

This research uses a comprehensive set of time-series analysis and advanced regression techniques, employing Python with the sklearn library, to discover resource usage patterns and identify crucial predictive features. On the BitBrains Inc dataset Shen et al. (2015), the approach includes initial modeling with popular time-series models such as ARIMA, SARIMAX, and Holt-Winters (smoothing). To enhance model performance, standardization is employed using the standard scaler from sklearn, and stationarity tests are conducted to ensure data stability. For the predictive analysis of CPU and Memory usage, this paper explores a variety of regression models. Linear Regression, GradientBoostingRegressor, Support Vector Machine (SVM), and Random Forest are individually explored to note their efficiency. Ensemble techniques are implemented, combining RandomForestRegressor, AdaBoostRegressor, and VotingRegressor. These varieties of models have their unique advantages and suitability for time series data. A foundational understanding of the linear relationships in the data is provided by linear regression. GradientBoostingRegressor adapts to the subtleties of the information and does a fantastic job of capturing intricate patterns and correlations. SVM is very helpful in managing non-linear relationships, and Random Forest's ensemble of decision trees adds robustness. The goal of the ensemble approaches, which combine models, is to maximize the combined strengths of separate algorithms. A comprehensive analysis is done to compare and validate these models' performance. For time series data, the study evaluates how effective they are in predicting CPU and Memory utilization, offering insights into the best models for precise workload predictions.

3.3.2 Search Algorithms

For searching the best cloud configuration based on the required CPU and RAM this paper uses various heuristic search algorithms for solving combinatorial optimization problems, with a focus on placing multiple application components of a workload in a multi-cloud environment. The algorithm uses a combination of strategies, including simulated annealing, Stochastic hill-climbing, and greedy selection, to explore and exploit the solution space. The optimization problem involves finding the best configuration of components and considering various requirements. The proposed solution employs a hybrid approach, combining brute-force and local search algorithms to address the complex problem of optimizing multiple application configurations. The brute-force algorithm is

utilized for its ability to exhaustively explore the entire solution space by generating all possible combinations of groups of components. This approach ensures that every potential configuration is considered, contributing to the identification of potentially optimal fleet offers. The algorithm meticulously evaluates each combination, taking into account affinity and anti-affinity conditions, and subsequently sorts and presents the results. The search algorithms based on simulated Annealing1 and Stochastic Hill Climbing2 introduce efficiency into the optimization process Delahaye et al. (2019) Stubbs et al. (2020). Beginning with an initial solution, the local search iteratively refines it by exploring nearby solutions. The optimization process is driven by parameters such as the number of desired results, a price calculator function, and the initial separated parameters. This algorithmic strategy is particularly advantageous for large-scale problems where exhaustive exploration of all possibilities becomes impractical. It adapts to the solution space, continually seeking improvements in the fleet offer configurations.

Algorithm 1 Simulated Annealing

Initialize ($i := i_{\text{start}}$, $k := 0$, $c_k = c_0$, $L_k := L_0$);

repeat

  for $l = 0$ to L_k **do**

    Generate j from S_i; {Neighborhood of i}

    if $f(j) < f(i)$ **then** $i := j$;

    else j becomes i with prob. $e^{\frac{f(i)-f(j)}{c_k}}$;

    end if

  end for

  $k := k + 1$; Compute(L_k, c_k);

until $c_k \leq 0$

3.4 Tools/Softwares Used

Computational scalability benefits for executing large datasets and applying predictive algorithms with changing demands are offered by AWS Cloud9. The main programming language used was Python3, for developing ICT solutions. BitBrains Inc., Azure, and AWS datasets were used to explore machine learning techniques in the Jupyter Notebook environment. Scalable analysis on large VM SKUs and Pricing datasets from Azure and AWS APIs were made possible in large part by PySpark. Docker Desktop was utilized as an alternative tool to forecast CPU and Memory consumption in highly complex workloads by taking advantage of its containerization characteristics. These tools are useful for solving the multi-cloud workload placement problem

4 Design Specification

This solution to the research objective of this paper is designed in two phases, which we will discuss in the section below:

Algorithm 2 Hill Climbing Procedure

procedure HILLCLIMBING(S)

  for each $e_0 \in E$ **do** $ncov(e_0) \leftarrow 0$;

  end for{Initialize coverage count}

  for each $e \in S$ **do**

    for each $e_0 \in \text{Path}(e)$ **do** $ncov(e_0) \leftarrow ncov(e_0) + 1$;

    end for{Update coverage counts}

  end for

  while not all edges $e \in S$ are processed **do**

    Select a yet unprocessed edge $e \in S$;

    if $\forall e_0 \in \text{Path}(e) : ncov(e_0) \geq 2$ **then**

      $S \leftarrow S \setminus \{e\}$; {Remove edge e from S}

      for each $e_0 \in \text{Path}(e)$ **do** $ncov(e_0) \leftarrow ncov(e_0) - 1$;

      end for{Update coverage counts}

    end if

  end while

end procedure

4.1 Predictive Phase

4.1.1 Predicting CPU and Memory requirement based on Historical Data

The first phase of the paper makes use of the VM traces dataset from Bitbrain INC Shen et al. (2015) by downloading and unzipping rnd zip file in any Jupyter environment on Jupyter Notebook to understand the available data characteristics for workload performance estimation. This phase as mentioned in the Figure 1 is centered on thorough data preparation, exploratory analysis, and creating predictive models for workload forecasting in a multi-cloud setting. First, the solution imports necessary packages, leveraging well-known Python libraries like NumPy, Pandas, Matplotlib, Seaborn, and scikit-learn. After importing and analyzing several CSV files, a cohesive data frame is produced in the final stages. In the feature engineering phase, similar relevant information is extracted from the timestamp column, such as the 'weekday, weekend, month, and day' features. Over time, these characteristics help to provide a more complex picture of workload patterns. To improve the model's capacity to represent temporal relationships, the research additionally produces new features based on the variations between successive values in the dataset. The study carefully considers data quality by using methods like mean-based imputation and z-score normalization to standardize characteristics and impute missing values. The algorithm moves on to model training and evaluation after preprocessing the dataset and feature engineering. The study uses a variety of machine learning algorithms, including ensemble methods like the combination of Random Forest and AdaBoost, and other advanced algorithms like Support Vector Machines, Linear Regression, and Gradient Boosting. Every model is evaluated, and one important metric for judging how well a model predicts is its Root Mean Squared Error (RMSE). A bar graph, which is a visual depiction of the RMSE values across various models, makes it easier to compare them and

offers insightful information about the relative effectiveness of each model. Preprocessing, feature engineering, and model evaluation are carefully integrated to guarantee that the research is prepared to tackle the complexities of workload performance prediction and hence predicts the compute configuration requirement of the different application components of the workload. This predicted configuration parameter is useful in the next phase to apply which can help in cost optimization and for searching for the optimal configuration and the suitable cloud service provider for every application component of a workload. This is one of the methods that has been implemented to explore the predictive algorithms and the type of time-series data that can be available in any existing infrastructure environment of an organization with various monitoring capabilities.

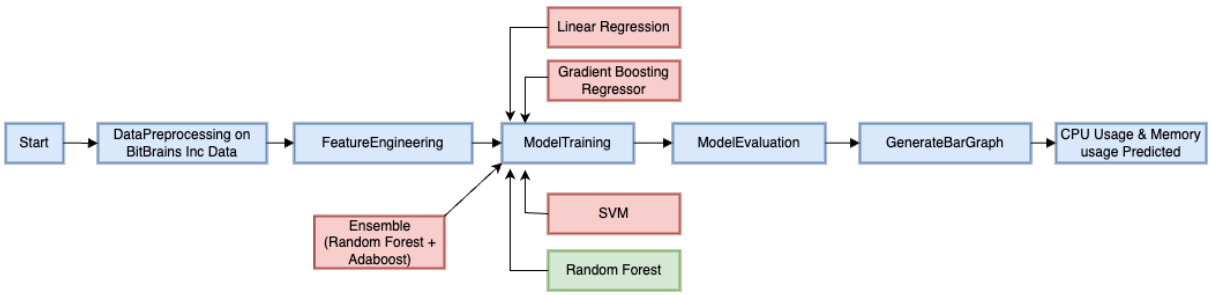

Figure 1: Analysing different Predictive models in Predicting CPU and Memory Usage requirement of a workload based on past Grid Workload trace data made available by BitBrain Inc.

4.1.2 Predicting CPU and Memory requirement of a new workload by profiling method

The implemented research introduces an innovative approach utilizing the Docker engine and containerized environments Figure 2. The core of this method involves a Python script designed to emulate a custom workload, with a primary focus on evaluating resource utilization metrics through the integration of the psutil library for comprehensive system monitoring within the Dockerized environment. The script encompasses two fundamental functions: complex_task(task_id, instructions) and custom_workload(...). The former is responsible for simulating a task with intensive computational demands using the NumPy library, while the latter orchestrates a workload featuring a specified number of tasks, both in parallel and serial, through a combination of threading and multiprocessing. During the execution of parallel tasks, the script employs a ThreadPoolExecutor, and for serial tasks, it utilizes a ProcessPoolExecutor. Systematic collection of resource metrics, encompassing CPU usage per CPU, memory usage, disk I/O, and network I/O, occurs throughout the execution of tasks. The script undertakes a comprehensive analysis of these metrics, facilitated by dedicated functions such as analyze_cpu_metrics, analyze_memory_metrics, analyze_disk_io_metrics, and analyze_net_io_metrics. The final component of the script, the print_resource_usage() function, retrieves and prints information regarding the system load. In the main function, main(), crucial workload parameters are defined, the workload simulation is executed, and resource usage information is printed. This Docker-based Python script serves as a valuable tool for profiling and gaining insights into the intricate resource demands of a workload within a containerized system other various functions can be added to the script based on different workload requirements and can be helpful in

profiling and estimating the workloads execution characteristics that will enable us with performance optimization of a specific workload.

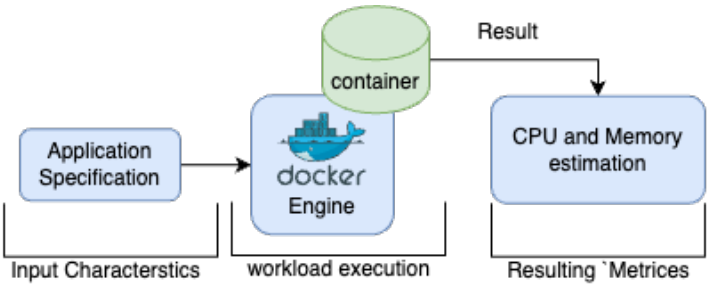

Figure 2: Analyzing New Workload Characteristics on Docker Container for Estimating CPU and Memory Requirements.

4.2 Search Phase: Searching for the Optimal Cloud Configuration and Cloud Service Provider

4.2.1 Analysing Azure/ AWS Dataset

This research project aims to provide a comprehensive understanding of the intricate pricing structures and Stock Keeping Unit (SKU) details within AWS and Azure cloud services. The project leverages Jupyter Python notebooks to conduct an in-depth analysis of AWS and Azure datasets, resulting in a comprehensive dataset encompassing low-level Virtual Machine (VM) parameters. This dataset offers transparent insights into selected configurations, enhancing resource utilization efficiency. The project generates a .json file containing vital parameters like onDemandPrice, spot_price, region, CPU, family, memory, network, operating system, instance type, storage, architecture, discounts, and more, categorized region-wise for both AWS and Azure cloud environments.
To tackle challenges caused by dynamic pricing updates from cloud service providers, the project designs a function employing boto3 to fetch existing cloud data. It introduces a data parser module that functions as a tool for handling calls and parsing data related to EC2 instances from AWS Price List Bulk API and Azure Retail API. This module interacts seamlessly with these APIs, retrieving crucial information on instance pricing and SpotAdvisor data. This contains essential functionalities for collecting and processing data, aligning with the research project's goals of comprehending cloud service pricing dynamics and optimizing configurations in multi-cloud environments.
The analysis effectively addresses configuration optimization challenges by extracting detailed information, resulting in the creation of valuable Python functions and classes. This offers significant insights to make informed decisions regarding cloud service provider selection that takes the instance type, region, and pricing model (on-demand or spot) as inputs and returns the corresponding price. These functions and classes can be used to optimize cloud configurations and reduce costs.

4.2.2 Searching for best Cloud Service Provider Solution Design

In this research, the algorithm is designed to help find the best cloud service provider and configuration parameters at the lowest cost. It works by breaking down the process into

smaller parts called epochs, each of which has two phases. In the first phase, the algorithm searches for the best combination of cloud service provider and configuration parameters by using two techniques called Simulated Annealing and Stochastic Hill Climbing. These techniques help the algorithm explore different options and find the best solution. In the second phase, the algorithm selects the next starting point for the search and introduces an element of randomness to prevent getting stuck in suboptimal solutions. During the searching phase, the algorithm starts with an initial node and develops its "children partitions" based on a decision on the proportion of children to develop. The algorithm then splits the child into two groups based on whether they improve or deteriorate the start node's price. A critical decision point follows, determining whether to improve. If yes, the algorithm selects one of the good nodes stochastically, weighted by the difference in performance; if no, it selects one of the bad nodes stochastically based on the same criteria. The selected node then becomes the starting point for the next iteration of the search process, contributing to the overall refinement of the algorithm.
Parallely the algorithm employs various hyperparameters to fine-tune its performance. These hyperparameters include the Candidate List Size, representing the maximum capacity of the Candidate list from the Reset Selector. Additionally, the Time per Region hyperparameter sets the maximum time (in seconds) the algorithm is allowed to run on each region. Another crucial hyperparameter is the Proportion of node child to Develop, defining the initial proportion to develop at each epoch. The algorithm's decision-making process is influenced by two bias factors: Exploitation Score Price Bias, determining the proportion between price score and subtree score, and Exploration Score Depth Bias, indicating the proportion between depth score and uniqueness score. The Exploitation Bias hyperparameter establishes the proportion between the exploitation score and the exploration score. These hyperparameters collectively govern the algorithm's behavior, allowing for flexibility and adaptability in selecting the best cloud configuration parameters for the given set of application components of a workload.

In Figure 3 the optimization algorithm, focuses on selecting cost-effective cloud service configurations. The process begins with the initialization of the initial node. The algorithm then moves to the decision phase, where it determines the proportion of nodes to develop. These children represent potential configurations of cloud service components. In the development phase, a proportion of nodes is developed. This leads to a split in the group of nodes, differentiating between those that improve the start node's price and those that deteriorate it and a decision is made to improve the start node. If the decision is negative, the algorithm selects one of the "bad" nodes stochastically, weighted by the difference in their characteristics. In parallel, if the decision is positive, the algorithm selects one of the "good" nodes stochastically. The process then evaluates whether a node is selected. If no node is selected, the algorithm concludes the search. If a node is selected, the algorithm either finishes the search or continues from the selected node, creating a loop back to the decision phase. This search-based algorithm is iterative and dynamic in nature of the optimization process, where the algorithm explores various configurations, makes decisions based on their impact on the objective function, and refines the search over multiple epochs. The use of the term "children" emphasizes the cloud configuration and exploration of potential solutions in the algorithmic process.

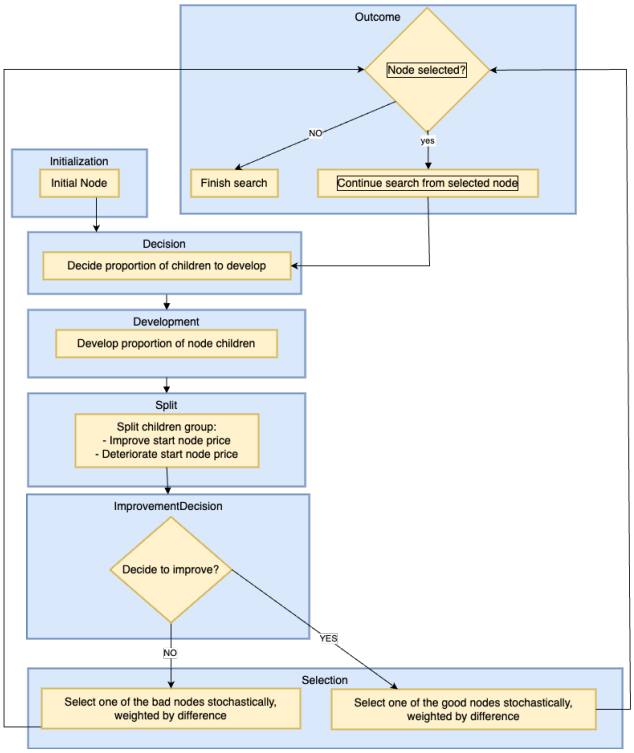

Figure 3: Cloud Configuration Selection Algorithm Flow Chart

5 Implementation

5.1 Implementing Predictive Phase:

5.1.1 Predciting CPU and Memory usage based on historical data

For predicting CPU and memory usage based on historical data, a Jupyter notebook with a Python3 runtime and T4 GPU hardware accelerator was employed. The analysis included various predictive machine learning algorithms, including an ensemble method combining Random Forest and AdaBoost, and advanced models like Support Vector Machines, Linear Regression, Random Forest, and Gradient Boosting from the sklearn module. The experimentation took place on Google Colab, a cloud-based Jupyter environment providing memory-optimized computational power for model performance testing. To facilitate the training of predictive machine learning models, the dataset from the GWA-T-12 Bitbrains Shen et al. (2015) workload trace was utilized. To align the data with the desired inputs and outputs criteria for predicting CPU and memory utilization based on past resource utilization trends, a series of filters were applied to the raw input data. The initial step involved an averaging filter applied to the raw data, computing the 9-second (30 rows) average resource utilization. The selected CSV files are then concatenated into a unified Pandas DataFrame for file selection based on a predefined pattern. After data cleaning and conversion steps the 'Timestamp [ms]' column is converted into a datetime format and all columns are converted to numeric types. Feature engineering operations are conducted, including the extraction of weekdays, the creation of a binary 'weekend' column indicating weekends, and the derivation of additional features related to 'month' and 'day' from the 'Timestamp' column. The 'Timestamp' is set as the index of the DataFrame. The creation of additional features involves generating new columns based

on the differences between consecutive values, including CPU usage, network received throughput, and network transmitted throughput. The training columns 'CPU capacity provisioned [MHZ]', 'Memory capacity provisioned [KB]', 'Memory usage [KB]', 'Disk read throughput [KB/s]', 'Disk write throughput [KB/s]', 'Network received throughput [KB/s]', 'Network transmitted throughput [KB/s]', 'CPU usage prev', 'CPU_diff', 'received_prev', 'received_diff', 'transmitted_prev' are selected and column 'CPU usage [MHZ]' is selected as a column to be predicted. A similar method can be implemented in the prediction of Memory utilization by selecting the 'Memory usage [KB]' as the column to be predicted and including the 'CPU usage [MHZ]' column in the training columns. These preprocessing steps were essential to transform the dataset, align it with the desired predictive model criteria, and prepare it for the application of machine learning algorithms in predicting CPU and memory usage based on historical data. For Linear Regression, Gradient Boosting, SVM, Ensemble, and Random Forest models, the dataset was split into training and testing sets. Missing values were handled using SimpleImputer for features and the target variable, followed by standardization using StandardScaler. The RMSE value was then calculated for each model, providing a measure of predictive accuracy. Linear Regression involved fitting the model to training data, predicting the test set, and evaluating RMSE. Gradient Boosting utilized 100 base learners, a learning rate of 0.3, and a random seed for initialization. SVM applied a linear kernel, trained on the provided data, predicted on the test set. Random Forest involved initializing and fitting the model with 100 decision tree estimators. Ensemble techniques combined RandomForestRegressor and AdaBoostRegressor models into a VotingRegressor. The ensemble model was trained on the data, predicted on the test set, and evaluated using RMSE. The bar graph comparing RMSE values for all models was generated using Matplotlib, offering a visual comparison of predictive accuracy across different machine learning models applied to CPU and memory usage prediction enabling performance optimisation of workload by estimating its configuration requirements.

5.1.2 Implementing Docker Profiling method: New Workload

In this docker profiling method for profiling and estimating the memory and CPU consumption this paper used Docker Engine to run a container and execute the workload. In this research an executable python script is created that assumes applications complex scenarios. This Python script has been developed to simulate a custom workload, facilitating an in-depth analysis of resource usage during the execution of parallel and serial parts of a workload. The script incorporates essential libraries such as os, time, psutil, numpy, and memory_profiler. Two key functions, complex_task that simulates a computationally intensive task through matrix operations and custom_workload, have been defined that orchestrate the parallel and serial execution of tasks, collecting metrics on resource usage. The @profile decorator from memory_profiler enables the profiling of memory usage during function execution, offering insights into the memory footprint of the tasks. The workload simulation begins by determining the number of parallel and serial tasks based on user-specified percentages. Parallel tasks are then executed using threading Classes (concurrent.futures.ThreadPoolExecutor), and metrics such as CPU and memory usage are recorded. Serial tasks are also executed using multiprocessing (concurrent.futures.ProcessPoolExecutor), and corresponding metrics are collected. Metrics contain various aspects, including CPU usage per CPU, memory usage, disk I/O (read and write), and network I/O (sent and received). The

analyze_metrics function processes and prints these metrics, providing a detailed overview of resource utilization. Resource usage analysis is further detailed through subfunctions such as analyze_cpu_metrics, analyze_memory_metrics, analyze_disk_io_metrics, and analyze_net_io_metrics. These subfunctions calculate average usage metrics and present the results and the print_resource_usage function fetches and prints system load metrics, including 1-minute, 5-minute, and 15-minute averages. The main function defines parameters for the workload, such as the number of tasks, instructions per task, parallel percentage, threads, processes, and iterations that can be defined as workload characteristics to fetch the CPU and Memory usage details.

Dockerfile is specified that create a Docker image for running a Python script that simulates a workload. An official Python runtime environment based on the OpenJDK 11 slim image is used. The image is configured with necessary build dependencies, including gcc, python3-dev, python3-pip, and openjdk-11-jre. This Docker image is designed to provide an environment for executing the specified workload simulation script with its dependencies. this implementation help in estimating CPU and Memory requirement of a new workload that has no previous usage data therefore a useful tool for performance optimisation.

5.2 Implementing Search Phase

In this phase of implementation and experimentation, this research project employs the AWS Cloud9 Environment. The objective is to develop a solution capable of taking a list of application components as input and delivering optimal results in terms of pricing, region, and Cloud Service provider for deploying each application in a multi-cloud environment, considering workload requirements. Users provide input in JSON format through the appcluster_offer.json file, specifying details such as the operating system, pricing preference (spot/on-demand), region choices, and application specifics. The output is presented in a JSON file and contains configurations, each denoting an application component placed on multiple cloud environments. This output includes information like total price, instance specifications, cloud service provider, and assigned components. The implementation offers flexibility and customization, enabling users to fine-tune optimization parameters and choose their preferred cloud provider. This research project aims to empower users to make informed decisions about deploying applications in a multi-cloud environment based on their unique requirements while optimizing cost.

The solution introduces essential functions, including fetching historical spot prices using Boto3, serializing groups and instances for analysis, and running an optimizer to obtain optimal configurations. The use_boto3 function employs Boto3 to retrieve spot price history, while serialize_group and serialize_instance focus on structuring data for readability. The run_optimizer function orchestrates the optimization process, parsing input parameters, processing application details, and calling the Application Cluster optimizer to provide a list of recommended configuration parameters. A Component class is defined to represent individual application components, capturing details such as memory, vCPUs, network, affinity rules, behavior, and storage specifications.

The GroupedParam class aggregates parameter values within a group, summing vCPUs, memory, and network values while calculating overall behavior and interruption frequency within the group. A get_info method is created to retrieve group parameters. The GroupedInstance class represents an offer for each combination of instances, encompassing spot prices, discounts, components, and total prices. It calculates the total price considering

either spot or on-demand pricing and includes a get_info method to retrieve component information. The Offer class is created that encapsulates an overall offer, considering multiple partitions and their respective sizes, tracking remaining partitions, total prices, and instance groups. This class includes methods like get_info for obtaining information about remaining partitions and copy_group function for creating a deep copy of the group. The Calculation Class serves as the core component for fleet calculations, featuring methods for calculating CPU and memory limits for a given region, creating component offers, and matching instances to groups of components. The get_offers method generates appcluster offers based on the provided parameters.

Several helper functions are dveloped, such as price_calc_lambda, check_anti_affinity, check_affinity, affinity, anti_affinity, and compare_sublists, are employed to perform checks and calculations within the cost calculation process. CombOptim class is created that represents the main optimization algorithm that includes stochastic hill climbing and simulated annealing, initializing with various parameters like the number of results, candidate list size, and price calculation function which are configurable after the experimentation. The run method executes the optimization algorithm. The Node class represents a node in the search space, used to denote different configurations. The hashCode method generates a hash based on the partitions of the node. OptimumSet method keeps track of the best nodes seen so far based on their prices. The resetSelector function is responsible for selecting a node to start the next run, maintaining a list of candidate nodes, and calculating scores based on exploration and exploitation. SearchAlgorithm defines the search algorithm used to explore the solution space, including methods for calculating scores, distance, and selecting nodes within the ResetSelector and SearchAlgorithm classes. There are some database-related methods created (finish_stats_operation, insert_stats, create_stats_table) for storing results during optimization, and enumeration classes (DevelopMode, GetNextMode, and GetStartNodeMode) defining different modes used in the algorithm.

This comprehensive implementation provides a fully-fledged solution for selecting optimal cloud configuration parameters and cloud service providers based on given specifications answering the research question.

6 Evaluation

The evaluation of the proposed solution is carried out for both the Predictive phase and the search phase.

6.1 Case Study 1: Comparing different predictive models in predicting CPU and Memory Utilisation based on historically available data.

In the predictive phase, the Root Mean Square Error (RMSE) value is fetched from all the models to compare their performance and select the suitable predictive algorithm that can be used to find the CPU and Memory usage estimates from the past usage data of the workload. This research employs various predictive algorithms such as Linear Regression, GradientBoostingRegressor, Support Vector Machine (SVM), and Random Forest, which are individually explored to note their efficiency. The ensemble technique is implemented

by combining RandomForestRegressor, AdaBoostRegressor, and VotingRegressor. The experimentation was performed on an extensive dataset from BitBrains Inc. The dataset highlights differences in the types of VMs based on their usage. It's a valuable resource for understanding how different VMs perform in terms of speed and storage requirements. Standard Scaler is used for the standardization of the dataset.
The result in Figure 4 showcases that Random Forest performed well in comparison to other models to predict the optimum CPU and Memory usage of the workload based on the past usage time-series data.

- Random Forest achieves close to an optimum solution with the Root Mean Square (RMSE) of '0.07770' with n_estimators set to 100. n_estimators hyperparameter set to 100 is chosen to due to the large dataset size and to provide Computational performance and accuracy.
- Linear Regression achieves an RMSE value of '0.13523'.
- Gradient Boosting Regressor also achieves an RMSE value of '0.13523' with n_estimators set to 100 and learning_rate of 0.3.
- Support Vector Machine (SVM) achieves an RMSE value of '0.23813' with a 'linear' kernel setting due to the large number of feature columns than training data.
- Ensemble method by combining RandomForestRegressor, AdaBoostRegressor, and VotingRegressor achieves the worst RMSE of '1.02034' with n_estimator of both the combining model of 100.

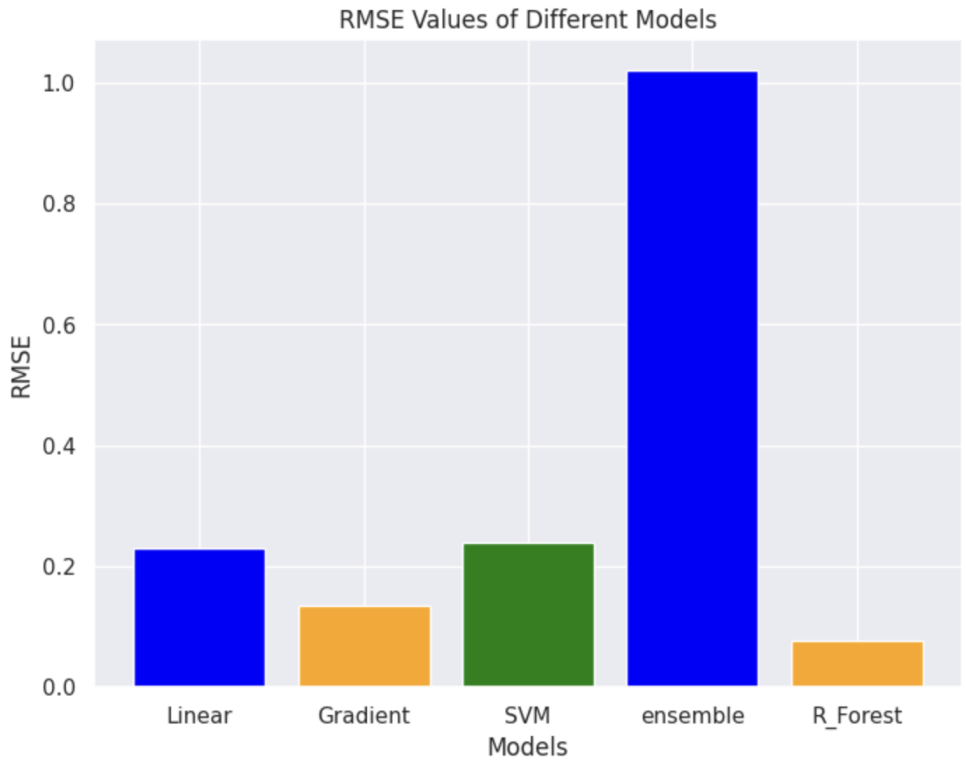

Figure 4: Comparison of different predictive algorithms on Bitbrains Inc dataset.

For predictive modeling tasks like regression and classification, Random Forest is a popular machine learning technique. When working with structured data sets, such as the historical consumption data of a workload, it is especially useful. Time series forecasting can also be done with Random Forest, however, it has to convert the time series dataset into a supervised learning issue first.
This experimentation proves that the Random Forest model works well in analyzing time series workload usage data for the prediction of CPU and Memory Usage and has proved to be a crucial algorithm for the performance optimization of the specific workload.

6.2 Experiment: Evaluating the performance of the proposed search-based solution for selecting cloud service provider

The experimentation Python script is developed for conducting systematic experiments to assess the performance of a search-based local optimization algorithm under diverse configurations. It consists of a series of functions and procedures designed for experimentation within a larger project focused on optimizing cloud configuration parameters. The code primarily revolves around implementing and testing a search-based optimization algorithm. A generic creator function is created that constructs instances of an Experiment class with specified parameters. The gen_ function for generating a list of experiment creators that iterates through sets of algorithm parameters such as experiment names, and other configurations, utilizing the GenericCreator for each combination. Functions, such as restart_algs and develop_proportion, tailor algorithm parameters for experiments involving algorithm resets and proportional development, respectively. The trail function amalgamates configurations from restart_algs and develop_proportion, introducing a "brute_force" configuration. The experimentation script sets specific values for multiprocessing processes (mp), and the target region (region), and generates experiments for two scenarios with different application component counts of a workload to test the effectiveness of the search algorithm in case of multiple application component workload. The gen_exp function is employed to create experiment instances, and the Series class, along with the run method, orchestrates the execution of these experiments. The experimentation setup for evaluation and optimization of a workload with nine and twenty application components of a workload to evaluate the effectiveness of fewer and more application components using different algorithmic approaches to the evaluate efficiency of selecting a cost-efficient configuration parameter and cloud service provider for multiple application components scenario. The parameters are outlined for three distinct modes: "Develop Proportion," "Brute Force," and specific modes named "Random Reset," "Greedy," and "Root." Each mode is configured with control parameters, search algorithm parameters, and reset algorithm parameters.
Each Experimentation phase consists of a total 60 random input samples which consist of the configuration requirement of each application component, this sample input is evaluated against brute force search and the combination of Simulated annealing and stochastic hill climbing algorithm. The restart_algs function is defined as a part of the experimentation framework and defines configurations for restarting optimization algorithms. It sets up three distinct algorithm configurations during evaluation: "Random_Reset," "Greedy," and "Root." Each configuration includes reset algorithm parameters, specifying factors like candidate list size, exploitation score bias, exploration depth bias, and exploitation bias. The search algorithm parameters are also defined, determining the optimization approach, and include parameters such as development mode, proportion amount of node

children to develop, the mode for getting the next node, and the mode for selecting the starting node.
For the "Develop Proportion" mode, the experiment is conducted with different proportions and the associated parameters include the component count, time per region, significance, develop mode, proportion amount, get next mode, get starting node mode, candidate list size, exploitation scores, and various biases. The "Brute Force" mode represents a straightforward exhaustive search with a longer time per region. The main focus centers on the "Develop Proportion" parameter, which undergoes variations ranging from 0.1 to 100. The experiment maintains consistent control parameters, such as a fixed component count of 9, a time allocation of 4 units per region, and a significance level of 1 during the experimentation phase. Within the search algorithm parameters, the "Develop Mode" is set to 2, while the "Proportion Amount Node to Develop" is systematically adjusted. The "Get Next Mode" remains constant at 1, and the "Get Starting Node Mode" varies between 1 and 3. The reset algorithm parameters include a fixed candidate list size of 64, with exploitation and exploration biases set at 0.5. The experiment introduces distinct scenarios, including a brute force approach with an extended time per region (10000 units) and various modes like Random Reset, Greedy, and Root, each configured with their unique settings. This compared algorithmic behaviors under diverse conditions analyzing the impact of the "Develop Proportion" parameter and other influencing factors on system performance. Similar parameters have been used in experimentation performance with twenty application components simulating a larger workload with different configuration requirements.
The experimentation found that when dealing with a small number of application components in this case 9 application components, the proposed search algorithm can't find the best solution within five seconds, the straightforward method of Brute search takes less than a second. There is a 0.1 costing gap difference between the two methods Figure 5. When similar experimentation was conducted with 20 application components, the search algorithm performed better than the brute search method producing better results in less than two seconds and achieving results twice as good within five seconds. These results evaluate that the search algorithm is more practical when dealing with a larger number of parts in predicting the desired configuration parameter and cloud service provider given various application components of a workload for cost optimization and optimum cloud service provider selection.Figure 6

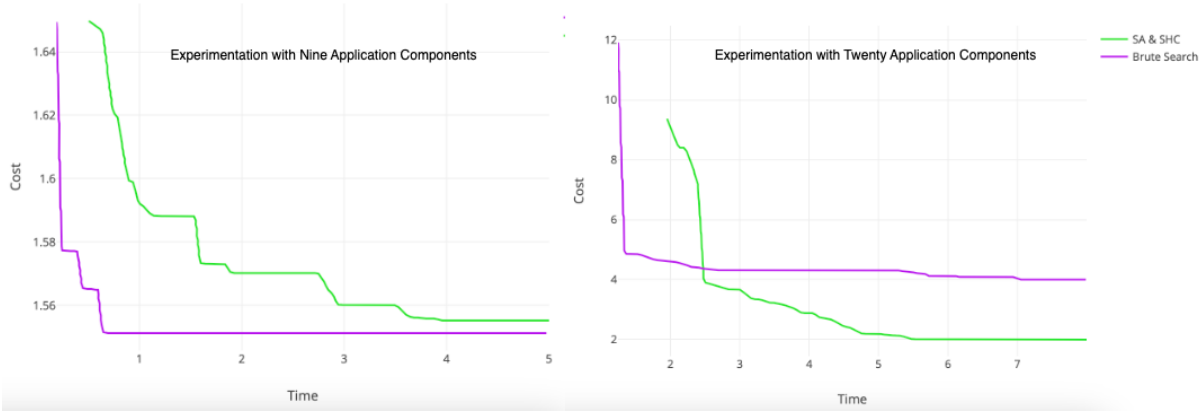

Figure 5: Algorithm Performance with different Application Components

S.No	Brute Search				SA & SHC			
	9 Application components		20 Application Components		9 Application components		20 Application Components	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	0.219981906	1.648669788	1.253258376	11.91166862	0.549862896	1.649722834	1.961180096	9.381015913
2	0.219464015	1.646986642	1.25318487	11.74211373	0.601294627	1.648566503	1.997371555	9.187434811
3	0.218946124	1.645303496	1.253111363	11.57255885	0.643315351	1.647426121	2.030265025	8.99332877
4	0.218428233	1.643620349	1.253037857	11.40300396	0.668201032	1.64567994	2.063160499	8.803846953
5	0.220213388	1.641771674	1.25296435	11.23344908	0.676655998	1.643576917	2.099351067	8.608210641
6	0.219695497	1.640088528	1.254197254	11.03461189	0.683916675	1.641672686	2.155362377	8.491393092
7	0.219177606	1.638405381	1.262702285	10.87510511	0.68804568	1.639791093	2.21468842	8.41372243
8	0.218659715	1.636722235	1.262628778	10.70555023	0.692896333	1.637704711	2.254185729	8.238796163
9	0.216689374	1.634908886	1.262555272	10.53599534	0.704896761	1.63590381	2.287077194	8.040065898
10	0.220846628	1.633119101	1.262481766	10.36644046	0.708956377	1.633432399	2.316677073	7.855582519
11	0.226204447	1.630640898	1.262408259	10.19688557	0.72069375	1.631140872	2.340776888	7.662345933
12	0.225580624	1.628613472	1.260348408	10.00959781	0.727028762	1.628778373	2.359389613	7.490285145
13	0.225062733	1.626930325	1.272131715	9.805144436	0.738507114	1.626300781	2.38678803	7.298986456
14	0.224544842	1.625247179	1.275340383	9.5996345	0.7477103	1.623860746	2.402741603	7.078619307
15	0.22402695	1.623564033	1.275266877	9.430079614	0.759803875	1.621633963	2.398710554	6.908537253
16	0.223509059	1.621880886	1.273207359	9.243562557	0.790231664	1.61997577	2.409182773	6.722576321
17	0.222991168	1.62019774	1.283094359	9.011724476	0.812383282	1.617650068	2.417217474	6.49242019
18	0.229987743	1.617799483	1.281581749	8.782710506	0.824109818	1.615323324	2.419527804	6.243234056
19	0.225326723	1.609208677	1.279528246	8.610066122	0.832990475	1.61358374	2.428033503	6.085268687
20	0.222169604	1.606817056	1.292417269	8.420516738	0.841871132	1.611844155	2.434547177	5.896468475
21	0.230983171	1.615570029	1.295603662	8.163626533	0.847552241	1.609906614	2.43380543	5.706873057
22	0.230465279	1.613886883	1.294415308	7.958101747	0.854382912	1.608134948	2.447569514	5.507307491
23	0.23000624	1.612395003	1.294341801	7.788546861	0.868460136	1.606453261	2.445512335	5.326185363
24	0.22260275	1.61128495	1.294268295	7.618991975	0.870244507	1.604602035	2.451788895	5.0975647
25	0.232489763	1.605165595	1.297489882	7.443282594	0.879194608	1.603088145	2.462461587	4.866903677
26	0.231971872	1.603482449	1.307305667	7.264512672	0.892812792	1.601444665	2.465001569	4.640325603
27	0.231453981	1.601799302	1.30723216	7.094957786	0.919266438	1.599439072	2.473492471	4.448229056
28	0.23093609	1.600116156	1.307158654	6.925402901	0.954820845	1.597939318	2.475357292	4.185645867
29	0.230418198	1.59843301	1.3103769	6.741986479	0.968577918	1.596310059	2.509554742	3.956652407
30	0.229900307	1.596749863	1.314246012	6.538533981	0.980188097	1.594698073	2.582073718	3.883330226
31	0.227927613	1.594928864	1.311514245	6.322727944	1.000796046	1.59282004	2.654603022	3.833829806
32	0.232875828	1.592649531	1.32110112	6.123062637	1.045043909	1.591267429	2.727132325	3.784329385
33	0.231312738	1.590629661	1.32585335	5.928176486	1.096298016	1.58953382	2.799661325	3.734128325
34	0.236390762	1.58833504	1.326430709	5.738585523	1.147731886	1.588384444	2.87220126	3.709150306
35	0.240154553	1.585703668	1.326357203	5.569030637	1.199454663	1.588174013	2.944748788	3.701688287
36	0.240046266	1.583821652	1.326283697	5.399475752	1.251228801	1.588130506	3.017284774	3.667601947
37	0.239528375	1.582138505	1.32621019	5.229920866	1.30301792	1.588135685	3.089783399	3.547336885
38	0.239010484	1.580455359	1.327538079	5.032845609	1.354807038	1.588140864	3.162280202	3.422867982
39	0.23706133	1.578710866	1.371582717	4.914856614	1.406574756	1.588076492	3.234810113	3.374768842
40	0.263897081	1.577182073	1.441485623	4.892602181	1.458361734	1.588074715	3.30735187	3.353994663
41	0.315617718	1.576964687	1.514034973	4.889344002	1.510150852	1.588079894	3.379881478	3.305194882
42	0.367368316	1.576844673	1.586569744	4.852455103	1.549985316	1.587484594	3.452413819	3.262700862
43	0.399789213	1.575106573	1.659095099	4.793846362	1.561845809	1.585228903	3.524955576	3.241926683
44	0.408553933	1.572990193	1.731619847	4.733836341	1.571313016	1.582845466	3.597490954	3.206439063
45	0.416576538	1.571012084	1.804153403	4.694144882	1.578474824	1.580619907	3.670022992	3.163244403
46	0.423760317	1.568857931	1.87669516	4.673370702	1.581772849	1.578587773	3.742548346	3.104635663
47	0.426450997	1.56689204	1.949238132	4.655399083	1.587581234	1.576553849	3.815066108	3.028510922
48	0.45368922	1.565280291	2.021781408	4.638128104	1.595496965	1.574738429	3.887577794	2.93837338
49	0.504924066	1.564875074	2.094322558	4.615952645	1.623267275	1.573186778	3.960116817	2.911293441
50	0.556657543	1.564699419	2.166852165	4.567152865	1.675000752	1.573011123	4.032666472	2.908735902
51	0.594114344	1.564027144	2.239380253	4.514849884	1.72677061	1.572953705	4.105189093	2.843821401
52	0.597431538	1.561693002	2.31190895	4.463948184	1.778546888	1.572917153	4.177703513	2.75998962
53	0.60967462	1.559514947	2.384451618	4.445275925	1.830306046	1.57282496	4.250240107	2.72730456
54	0.607953024	1.55799999	2.456989426	4.415393425	1.872298032	1.571591181	4.322773966	2.68831374
55	0.616228297	1.556333023	2.52953088	4.393918606	1.914268619	1.57028785	4.39529112	2.6107877 19
56	0.623273972	1.55424006	2.602073852	4.375946987	1.965989255	1.570070464	4.467801288	2.517146977
57	0.632894977	1.552356467	2.67461652	4.357274728	2.017778374	1.570075643	4.540323909	2.452232477
58	0.66073013	1.551015556	2.747166782	4.356118469	2.069567492	1.570080822	4.61285139	2.398528216
59	0.712457187	1.550819036	2.819717652	4.356363491	2.12135661	1.570086	4.685366721	2.316798355
60	0.764246306	1.550824215	2.892268521	4.356608512	2.173145729	1.570091179	4.757883571	2.238571694
61	0.816035424	1.550829394	2.964819391	4.356853534	2.224934847	1.570096358	4.830424114	2.214994955
62	0.867824542	1.550834573	3.03737026	4.357098555	2.276723965	1.570101537	4.902974072	2.213138056
63	0.919613661	1.550839752	3.10992113	4.357343577	2.328513084	1.570106716	4.975524334	2.211981797
64	0.971402779	1.550844931	3.182472	4.357588598	2.380302202	1.570111895	5.048071255	2.203118499
65	1.023191897	1.550850109	3.255022869	4.35783362	2.43209132	1.570117074	5.120614227	2.185146879
66	1.074981016	1.550855288	3.327573739	4.358078641	2.483880439	1.570122253	5.193159021	2.171379101
67	1.126770134	1.550860467	3.400124608	4.358323663	2.535669557	1.570127432	5.265704727	2.159713242
68	1.178559252	1.550865646	3.472675478	4.358568684	2.587458675	1.570132611	5.338247396	2.141040982
69	1.23034837	1.550870825	3.545226347	4.358813706	2.639247793	1.57013779	5.410778522	2.095744402
70	1.282137489	1.550876004	3.617777217	4.359058727	2.691036912	1.570142968	5.48330904	2.049046542
71	1.333926607	1.550881183	3.690328086	4.359303749	2.742791789	1.570036865	5.55585505	2.038081323

S.No	Brute Search				SA & SHC			
	9 Application components		20 Application Components		9 Application components		20 Application Components	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
72	1.385715725	1.550886362	3.762878956	4.35954877	2.784822297	1.568928278	5.628405919	2.038326345
73	1.437504844	1.550891541	3.835429825	4.359793792	2.817277932	1.567303075	5.700956789	2.038571366
74	1.489293962	1.55089672	3.907978569	4.355134333	2.849625953	1.565328127	5.773507659	2.038816388
75	1.54108308	1.550901899	3.980528223	4.352576795	2.886812857	1.563778687	5.846058528	2.039061409
76	1.592872199	1.550907078	4.053079093	4.352821816	2.921576957	1.562005564	5.918609398	2.039306431
77	1.644661317	1.550912256	4.125629051	4.350964918	2.961061523	1.560273097	5.991160267	2.039551452
78	1.696450435	1.550917435	4.19817749	4.345604819	3.012767179	1.560007025	6.063711137	2.039796474
79	1.748239554	1.550922614	4.27072836	4.345849841	3.064554157	1.560005249	6.136262006	2.040041495
80	1.800028672	1.550927793	4.34327923	4.346094862	3.116343276	1.560010427	6.208812876	2.040286517
81	1.85181779	1.550932972	4.415830099	4.346339884	3.168132394	1.560015606	6.281363745	2.040531539
82	1.903606908	1.550938151	4.488380969	4.346584905	3.219921512	1.560020785	6.353911881	2.0344708
83	1.955396027	1.55094333	4.560931838	4.346829927	3.271710631	1.560025964	6.426462143	2.033314541
84	2.007185145	1.550948509	4.633482708	4.347074948	3.323499749	1.560031143	6.499013013	2.033559563
85	2.058974263	1.550953688	4.706033577	4.34731997	3.375288867	1.560036322	6.571563882	2.033804584
86	2.110763382	1.550958867	4.778584447	4.347564991	3.427077985	1.560041501	6.644114752	2.034049606
87	2.1625525	1.550964046	4.851135316	4.347810013	3.478867123	1.559997994	6.716665621	2.034294627
88	2.214341618	1.550969224	4.923686186	4.348055034	3.530521399	1.559613684	6.789216491	2.034539649
89	2.266130737	1.550974403	4.996237055	4.348300056	3.581880368	1.558220878	6.86176736	2.03478467
90	2.317919855	1.550979582	5.068787925	4.348545077	3.633222217	1.556772431	6.93431823	2.035029692
91	2.369708973	1.550984761	5.141338794	4.348790099	3.684818731	1.556151646	7.006869099	2.035274713
92	2.421498091	1.55098994	5.213889664	4.34903512	3.736562908	1.556010767	7.079419969	2.035519735
93	2.47328721	1.550995119	5.286440533	4.349280142	3.788307086	1.555869887	7.151970838	2.035764756
94	2.525076328	1.551000298	5.358982898	4.329907243	3.840049123	1.555722053	7.224520493	2.033207218
95	2.576865446	1.551005477	5.431525263	4.310534343	3.891831821	1.555706367	7.297069236	2.028547759
96	2.628654565	1.551010656	5.504067324	4.290460804	3.943528917	1.555412474	7.369620106	2.028792781
97	2.680443683	1.551015835	5.576605739	4.261979585	3.995230294	1.555132492	7.442170975	2.029037802
98	2.732232801	1.551021014	5.649148712	4.244007966	4.047017272	1.555130716	7.514721845	2.029282824
99	2.78402192	1.551026192	5.721676496	4.191004345	4.09880639	1.555135894	7.587272714	2.029527845
100	2.835811038	1.551031371	5.794216431	4.166026326	4.150595509	1.555141073	7.659823584	2.029772867
101	2.887600156	1.55103655	5.866767301	4.166271347	4.202384627	1.555146252	7.732374453	2.030017888
102	2.939389275	1.551041729	5.93931817	4.166516369	4.254173745	1.555151431	7.804925323	2.03026291
103	2.991178393	1.551046908	6.01186904	4.16676139	4.305962864	1.55515661	7.877476192	2.030507931
104	3.042967511	1.551052087	6.084419909	4.167006412	4.357751982	1.555161789	7.950027062	2.030752953
105	3.094756629	1.551057266	6.156963489	4.150436073	4.4095411	1.555166968	8.002791331	2.03093115
106	3.146545748	1.551062445	6.229508283	4.136668294	4.461330219	1.555172147		
107	3.198334866	1.551067624	6.302059153	4.136913315	4.513119337	1.555177326		
108	3.250123984	1.551072803	6.374610023	4.137158337	4.564908455	1.555182505		
109	3.301913103	1.551077982	6.447160892	4.137403358	4.616697574	1.555187684		
110	3.353702221	1.551083161	6.519711762	4.13764838	4.668486692	1.555192862		
111	3.405491339	1.551088339	6.592262631	4.137893401	4.72027581	1.555198041		
112	3.457280458	1.551093518	6.664813501	4.138138423	4.772064928	1.55520322		
113	3.509069576	1.551098697	6.73736437	4.138383444	4.823854047	1.555208399		
114	3.560858694	1.551103876	6.80991524	4.138628466	4.875643165	1.555213578		
115	3.612647813	1.551109055	6.882466109	4.138873487	4.927432283	1.555218757		
116	3.664436931	1.551114234	6.955014549	4.133513389	4.979221402	1.555223936		
117	3.716226049	1.551119413	7.027543852	4.084012969	5.014532164	1.555227467		
118	3.768015167	1.551124592	7.100078927	4.047824709				
119	3.819804286	1.551129771	7.172629493	4.04736909				
120	3.871593404	1.55113495	7.245180362	4.047614112				
121	3.923382522	1.551140129	7.317731232	4.047859133				
122	3.975171641	1.551145307	7.390282101	4.048104155				
123	4.026960759	1.551150486	7.462832971	4.048349176				
124	4.078749877	1.551155665	7.535383841	4.048594198				
125	4.130538996	1.551160844	7.60793471	4.048839219				
126	4.182328114	1.551166023	7.68048558	4.049084241				
127	4.234117232	1.551171202	7.753036449	4.049329262				
128	4.285906351	1.551176381	7.825587319	4.049574284				
129	4.337695469	1.55118156	7.898138188	4.049819305				
130	4.389484587	1.551186739	7.970689058	4.050064327				
131	4.441273705	1.551191918						
132	4.493062824	1.551197097						
133	4.544851942	1.551202275						
134	4.59664106	1.551207454						
135	4.648430179	1.551212633						
136	4.700219297	1.551217812						
137	4.752008415	1.551222991						
138	4.803797534	1.55122817						
139	4.855586652	1.551233349						
140	4.90737577	1.551238528						
141	4.959164889	1.551243707						
142	4.9921216	1.551247002						

Figure 6: Experimentation Result table

6.3 Discussion

In the first case study evaluating predictive models for estimating CPU and Memory Utilization, various machine learning algorithms, including Linear Regression, GradientBoostingRegressor, Support Vector Machine (SVM), Random Forest, and an ensemble method, were applied to historical workload data. The evaluation metric, Root Mean Square Error (RMSE), was utilized to compare their performance. The findings revealed that Random Forest outperformed other models, achieving an RMSE of '0.07770' compared to Linear Regression ('0.13523'), Gradient Boosting Regressor ('0.13523'), SVM ('0.23813'), and the ensemble method ('1.02034'). This suggests that Random Forest is a robust algorithm for predicting CPU and Memory Usage in the given workload. This aligns with previous research emphasizing Random Forest's effectiveness in handling time series data for predictive modeling. However, it's essential to note that model evaluation is a continuous process, and future work could explore additional algorithms or fine-tune

parameters to enhance predictive accuracy further.
The second case study focused on evaluating the performance of a search-based local optimization algorithm in selecting a cloud service provider. The experimentation involved various configurations, including different modes like "Develop Proportion," "Brute Force," "Random Reset," "Greedy," and "Root." Based on the evaluation, the search algorithm found the optimal solution after five seconds when dealing with nine application components, whereas the brute force method found the solution in less than a second, albeit at a minor cost difference of 0.1. The search algorithm beat the brute force approach when there were twenty application components involved. It produced better results in two seconds and doubled the efficiency in five seconds.Figure 5
These findings provide valuable insights into the practicality of the search algorithm for larger workloads. The observed performance gap between the two methods underlines the importance of considering the scale of the workload when choosing the optimization approach. While the brute force method may excel in scenarios with a limited number of components, the search-based algorithm proves more efficient and scalable for larger workloads. It's important to acknowledge the limitations of the search algorithm for smaller configurations and explore potential modifications or alternative algorithms for further improvements. Future research could investigate the algorithm's behavior under different cloud environments and service providers, considering the evolving landscape of cloud computing.
The result evaluation aligns with the general understanding that the choice of optimization algorithm depends on the specific characteristics and scale of the problem. While search-based algorithms have demonstrated effectiveness in various optimization scenarios, their performance can vary based on the complexity and size of the workload. The research findings add to the continuing conversation on the necessity of flexibility and scalability in optimization techniques by highlighting the complex interplay between algorithmic choice and workload parameters.

7 Conclusion and Future Work

This research project effectively addresses the research question "How can search-based algorithms and predictive methods be employed to select the cloud service provider and optimal configuration for multiple application components of a complex workload in the multi-cloud setting?" The project involves analysis with predictive algorithms designed for workload performance prediction, utilizing historical workload usage data. A simulation script has been developed, configurable and extensible based on workload characteristics, enabling the execution of workloads in the Docker engine. This simulation provides insights into the overall required CPU and memory usage by an application during test runs of the workload hence helpful in accessing the workload performance. The core of this research project lies in the implementation of a solution based on Stochastic Hill Climbing and Simulated Annealing. This solution aims to search for cost-efficient and optimal configuration parameters and CSP in a multi-cloud environment, specifically Azure and AWS. By retrieving current pricing and instance-specific configuration data from AWS and Azure APIs, the implemented solution can search configuration parameters for up to 20 application components. In a real-world scenario, the project proposes a practical approach to suggest the deployment of specific application components on suitable service providers. The solution further accommodates various filters that can

be adjusted based on application or workload requirements. The key findings underscore the project's success in achieving performance and cost optimization in a multi-cloud environment. There is a further need for exploration into the dynamic nature of cloud environments and the potential challenges associated with real-time data accuracy. For future work, the integration of reinforcement learning techniques, such as Trust Region Policy Optimization or Deep Deterministic Policy Gradient algorithms, holds promise for enhancing the optimization process, also terraform automation can be further employed to extend the functionality of the proposed solution that will be able to deploy the application components of the workload to the selected cloud service provider.

References

Achar, S. (2023). Neural-hill: A novel algorithm for efficient scheduling iot-cloud resource to maintain scalability, *IEEE Access* **11**: 26502–26511.

Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M. and Zhang, M. (2017). {CherryPick}: Adaptively unearthing the best cloud configurations for big data analytics, *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 469–482.

Ansari Shiri, M., Omidi, M. R. and Mansouri, N. (2023). A new hybrid filter-wrapper feature selection using equilibrium optimizer and simulated annealing, *Journal of Mahani Mathematical Research* **13**(1): 293–332.

Bilal, M., Serafini, M., Canini, M. and Rodrigues, R. (2020). Do the best cloud configurations grow on trees? an experimental evaluation of black box algorithms for optimizing cloud workloads, *Proceedings of the VLDB Endowment* **13**(12): 2563–2575.

Chen, J., Du, T. and Xiao, G. (2021). A multi-objective optimization for resource allocation of emergent demands in cloud computing, *Journal of Cloud Computing* **10**(1): 1–17.

De Gooijer, J. G. and Hyndman, R. J. (2006). 25 years of time series forecasting, *International journal of forecasting* **22**(3): 443–473.

Delahaye, D., Chaimatanan, S. and Mongeau, M. (2019). Simulated annealing: From basics to applications, *Handbook of metaheuristics* pp. 1–35.

DeLisi, M. and Howley, C. (2023). Gartner forecasts worldwide public cloud end-user spending to reach nearly $600 billion in 2023, *Gartner* .

Devi, K. L. and Valli, S. (2023). Time series-based workload prediction using the statistical hybrid model for the cloud environment, *Computing* **105**(2): 353–374.

Gartner (2023). Public cloud container services reviews and ratings: Peer insights, *Gartner PeerInsight* .

Hou, Z., Shen, H., Zhou, X., Gu, J., Wang, Y. and Zhao, T. (2022). Prediction of job characteristics for intelligent resource allocation in hpc systems: a survey and future directions, *Frontiers of Computer Science* **16**(5): 165107.

Hsu, C.-J., Nair, V., Freeh, V. W. and Menzies, T. (2018). Arrow: Low-level augmented bayesian optimization for finding the best cloud vm, *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 660–670.

Hsu, C.-J., Nair, V., Menzies, T. and Freeh, V. (2018a). Micky: A cheaper alternative for selecting cloud instances, *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, pp. 409–416.

Hsu, C.-J., Nair, V., Menzies, T. and Freeh, V. W. (2018b). Scout: An experienced guide to find the best cloud configuration, *arXiv preprint arXiv:1803.01296* .

Liu, Y., Xu, H. and Lau, W. C. (2023). Cloud configuration optimization for recurring batch-processing applications, *IEEE Transactions on Parallel and Distributed Systems* **34**(5): 1495–1507.

Mahgoub, A., Medoff, A. M., Kumar, R., Mitra, S., Klimovic, A., Chaterji, S. and Bagchi, S. (2020). {OPTIMUSCLOUD}: Heterogeneous configuration optimization for distributed databases in the cloud, *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 189–203.

Masdari, M. and Khoshnevis, A. (2020). A survey and classification of the workload forecasting methods in cloud computing, *Cluster Computing* **23**(4): 2399–2424.

Mohapatra, A. D. and Oh, K. (2023). Smartpick: Workload prediction for serverless-enabled scalable data analytics systems, *Proceedings of the 24th International Middleware Conference on ZZZ*, pp. 29–42.

Newaz, M. N. and Mollah, M. A. (2023). Memory usage prediction of hpc workloads using feature engineering and machine learning, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, pp. 64–74.

Sa'ad, S., Muhammed, A., Abdullahi, M. and Abdullah, A. (2023). An optimised cuckoo-based discrete symbiotic organisms search strategy for tasks scheduling in cloud computing environment, *arXiv preprint arXiv:2311.15358* .

Saxena, D., Kumar, J., Singh, A. K. and Schmid, S. (2023). Performance analysis of machine learning centered workload prediction models for cloud, *IEEE Transactions on Parallel and Distributed Systems* **34**(4): 1313–1330.

Shen, S., Van Beek, V. and Iosup, A. (2015). Statistical characterization of business-critical workloads hosted in cloud datacenters, *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, IEEE, pp. 465–474.

Singh, D. and Choudhary, S. J. (n.d.). Dynamic resource allotment in cloud computing with predetermined waiting queue.

Stubbs, R., Wilson, K. and Rostami, S. (2020). Hyper-parameter optimisation by restrained stochastic hill climbing, *Advances in Computational Intelligence Systems: Contributions Presented at the 19th UK Workshop on Computational Intelligence, September 4-6, 2019, Portsmouth, UK 19*, Springer, pp. 189–200.

Venkataraman, S., Yang, Z., Franklin, M., Recht, B. and Stoica, I. (2016). Ernest: Efficient performance prediction for {Large-Scale} advanced analytics, *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pp. 363–378.

Witt, C., Bux, M., Gusew, W. and Leser, U. (2019). Predictive performance modeling for distributed batch processing using black box monitoring and machine learning, *Information Systems* **82**: 33–52.

Yan, X., Zhao, J. and Han, J. (2023). Research on intelligent scheduling algorithm based on cloud computing, *2023 15th International Conference on Computer Research and Development (ICCRD)*, IEEE, pp. 100–104.