University of Nevada, Reno

Solutions for Improving Model Simulation in the Virtual Watershed Platform

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science and Engineering

by

Jose Thomas Painumkal

Dr. Sergiu Dascalu, Dr. Frederick C. Harris, Jr /Co-advisors

August, 2017

THE GRADUATE SCHOOL

We recommend that the thesis

prepared under our supervision by

JOSE THOMAS PAINUMKAL

entitled

Solutions for Improving Model Simulation in the Virtual Watershed Platform

be accepted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

Sergiu Dascalu, Ph.D., Advisor

Frederick C. Harris, Jr, Ph.D., Co-advisor

Janet Usinger, Ph.D., Graduate School Representative

David W. Zeh, Ph. D., Dean, Graduate School

August, 2017

Abstract

This thesis is a collage of the works implemented to enhance the modeling capabilities of the NSF EPSCoR-supported Western Consortium for Water Analysis, Visualization and Exploration (WC-WAVE) Virtual Watershed Project. The core components of this work are a web-based tool to conduct scenario-based studies with watershed models, a proposed server-usage optimization strategy to enable cost-effective deployment of model containers to reduce the waiting time of jobs in a cloud environment, and a web tool to minimize the prediction errors of computer simulated models using a generic machine learning based approach. The developed prototype application includes an elastic hybrid server cluster comprising owned and rented servers that can facilitate on-demand provisioning of the computing resources based on job arrivals and ensures reduced waiting time for the modeling jobs within the allocated budget amount. The prototype contains a dashboard to track the progress of model run jobs and a user feedback monitoring module to generate auto alerts during severe performance issues. The model-scenarios component in the application could help hydrologists in simulating user-defined model scenarios using the PRMS (Precipitation Runoff Management System) model. The tool facilitates the download of watershed datasets available in the Geographic Storage, Transformation and Retrieval Engine (GSToRE) and enables the insertion of model simulations to GSToRE. The proposed model accuracy component uses a generic machine learning approach to process the predictions made by a computer-simulated model and helps in improving the accuracy of the model by minimizing the prediction errors. The prototype system supports the processing of the model data set using four machine learning regression techniques and enables the fine tuning of the model predictions.

Acknowledgments

I would never have been able to finish my thesis without the guidance of my committee members, help from friends, and support from my family.

I would like to express my deepest gratitude to my advisor Dr. Dascalu and my co-advisor Dr. Harris for the excellent guidance, caring, patience, and engagement throughout the course of my study at University of Nevada, Reno. Thank you for all your intellectual and emotional support throughout this thesis. I also thank my committee member Dr. Janet Usinger for her valuable intellectual inputs in improving my thesis work.

I would also like to thank all the members of the WC-WAVE team with whom I had a chance to work in close collaboration, including Rui Wu, Chao Chen, Dr. Karl Benedict, John Erickson, John Savickas and Hays Barrett.

I would like to thank my beloved parents for the love, care, and support throughout my life. I would also like to thank my siblings for the tremendous encouragement and support throughout my studies at UNR.

Finally, I am thankful to the National Science Foundation for the support provided during my Masters study. The material presented in this thesis is based upon work supported by the National Science Foundation under grant numbers IIA-1329469 and IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modeling has become an indispensable tool for environmental scientists to understand how natural systems react to changing conditions. It sheds light on complex environmental mysteries and helps researchers in formulating policies and decisions on future scenarios. Environmental modeling is highly challenging as it involves complex mathematical computations, rigorous data processing, and convoluted correlations between numerous parameters. However, the emergence of software tools has played a significant role in minimizing the complexities associated with model processing. Software platforms contain sophisticated tools to facilitate model simulation and interpretation of their results. The Virtual Watershed [10, 7] is such a platform to help watershed scientists in their research. It is a collaboration between the universities in New Mexico, Idaho, and Nevada funded by NSF EPSCoR [14] to study the impacts of climate change on high mountain catchments. The Virtual Watershed (VW) offers a computing platform for environmental scientists to share data with fellow researchers, run different models and visualize results in a cloud environment through web services.

The goal of this thesis was to enhance the modeling capabilities of the Virtual Watershed and thereby offer an improved platform to environmental scientists for their modeling research activities. The core components of this work include 1) a web-based tool to perform scenario-based studies with watershed models 2) a new server-usage optimization approach to improve the efficiency of model simulation jobs in Virtual Watershed and 3) a web-based tool to enhance the accuracy of computer-simulated models using a generic machine learning based approach by minimizing the

prediction errors.

Modification of the existing model simulations is a complex activity that hydrologists often have to deal with while analyzing complicated environmental scenarios. The simulation of the desired environmental situation demands numerous modifications on the underlying model input files and also requires many model re-runs. Programming languages could come handy in many ways while dealing with large scale file modifications. However, while working with our collaborators, we noticed that how challenging it is for the hydrologists to write their own programs to handle file modifications for scenario-based studies. The majority of the hydrologists we worked with were not very well familiar with programming languages and lacked the technical expertise to leverage the benefits of programmatic handling of file modifications. Therefore, they usually depend on the third party file manipulation softwares such as Microsoft Excel, Notepad, Sublime Text, etc. to do the modifications manually. Manual editing is not efficient when dealing with model scenarios which require large-scale changes on the existing model simulation. To help hydrologists on this problem, we developed a web-based tool which enables researchers to conduct scenario-based studies without manually editing the model inputs.

The computational limitations in the modeling server imposes unnecessary delays in finishing model simulations and the researchers have to wait longer period to get their simulation jobs processed in the Virtual Watershed platform. To improve the efficiency of running model simulation jobs through VW platform, we developed a self-managed elastic scale hybrid server system using an improved server-usage optimization approach and integrated it with the Virtual Watershed platform. The developed system uses a modified queuing theory to facilitate on-demand provisioning of computing resources based on budget constraints and user feedback. The proposed approach promises a better user experience on VW platform by efficiently distributing the model simulation jobs on a cluster of owned and rented servers from cloud providers. The system can ensure the continuous monitoring of user feedback and generate auto-alerts on predefined feedback scenarios. Through this work, we

also introduced a way to relate budget constraints directly with the desired Quality of Service (QoS) targets (here, waiting time of the jobs), which could help managers in making important budget-policy decisions more easily.

Model accuracy becomes a decisive factor when it comes to making crucial decisions based on model outcomes. A slight improvement in accuracy percentages can make a huge impact on the decision making capability. One of the goals of this thesis was to examine how machine learning techniques can be utilized to improve the accuracy of predictions made from a computer-simulated model. As a proof of concept, a web-based tool was developed to analyze the variations between the model output and the observed values. Applying machine learning regression techniques on a sufficiently large enough dataset, the tool could predict the difference between the model predicted values and the actual observation (i.e. the delta value), which was later used to fine-tune the model outputs to improve the accuracy of the underlying model.

All the above works are further detailed in this thesis. In its remaining parts, the thesis is structured as follows: Chapter 2 provides an overview of the current Virtual Watershed platform, Precipitation Run Off Modeling System (PRMS), the problems of the current watershed portal, and the works related to the developed components. Chapter 3 presents the specification and requirements of each component implemented as part of this thesis work; Chapter 4 depicts the details about the design and implementation along with screenshots of the components; Chapter 5 describes the results of the evaluation of the components and, finally, Chapter 6 completes the thesis with details on planned future work and the main highlights of our work.

Chapter 2

Background and Related Work

This thesis is a collage of the works implemented to improve the modeling capabilities of the Virtual Watershed platform. The objectives of this work are:

1. Provide an intuitive and efficient way for the creation of user-defined model scenarios.

2. Improve the efficiency of model running in VW platform using a modified server-usage optimization approach

3. Evaluate the application of machine learning techniques in improving the accuracy of predictions made from a computer-simulated model.

This chapter gives an overview of the design and functionality of the Virtual Watershed platform, a brief description of the popular Precipitation Run Off Modeling System (PRMS), the problems faced by researchers in the current Virtual Watershed platform, brief description about some of the related works, and an overview of on-demand computation and its key players.

2.1 Overview of the Virtual Watershed platform

The Virtual Watershed (VW) [53] offers a computing platform for environmental scientists to study the mysteries hidden in watershed science. It is a collaboration between the universities in New Mexico, Idaho, and Nevada and is funded by NSF

EPSCoR [14]. The platform contains a collection of software tools to handle the processing, simulation, and visualization of different environmental models, by utilizing the underlying architecture and web services of the portal.

The VW platform was designed with a micro-service architecture [34], in which each service is highly independent in nature and is designed to perform a particular task. In a micro-service architecture, the services can be easily scaled out based on its resource requirements. To leverage the flexibility and power of micro-service architecture, the latest software containerization technology, Docker [31] was used to implement various services in the VW platform. Docker simplifies the development and management of distributed systems [48] by supporting the deployment of a service in a self-contained lightweight container. The Docker container offers an isolated running environment with all the necessary libraries, dependencies and packages to run the service. Therefore by using Docker, multiple applications developed using different languages and frameworks can be run autonomously on a single server or multiple servers, where each application runs on its own isolated environment not interfering with the resources (processes, threads) of other applications. In the Virtual Watershed platform, the modules are packaged and deployed into separate Docker containers. User authentication, model dashboard, model data processing, data converter tools are some of modules in VW platform. All the containers reside on a single host machine and the communication across containers was facilitated through a restful web service [42].

Figure 2.1 shows the components of the current Virtual Watershed platform. VW-PY provides python wrappers for different environmental models to run programmatically on the platform. These wrappers take care of the runtime complexities such as model's input data conversions, event triggering on model progress, etc. VW-WEB acts as the front end for the end users. The users can log into the system, track model run progress, upload/download simulation files and run models. VW-AUTH handles the authentication of users during login, user registration, generation of access token for using the modelrun API service, and resetting of passwords. VW-SESSION is a

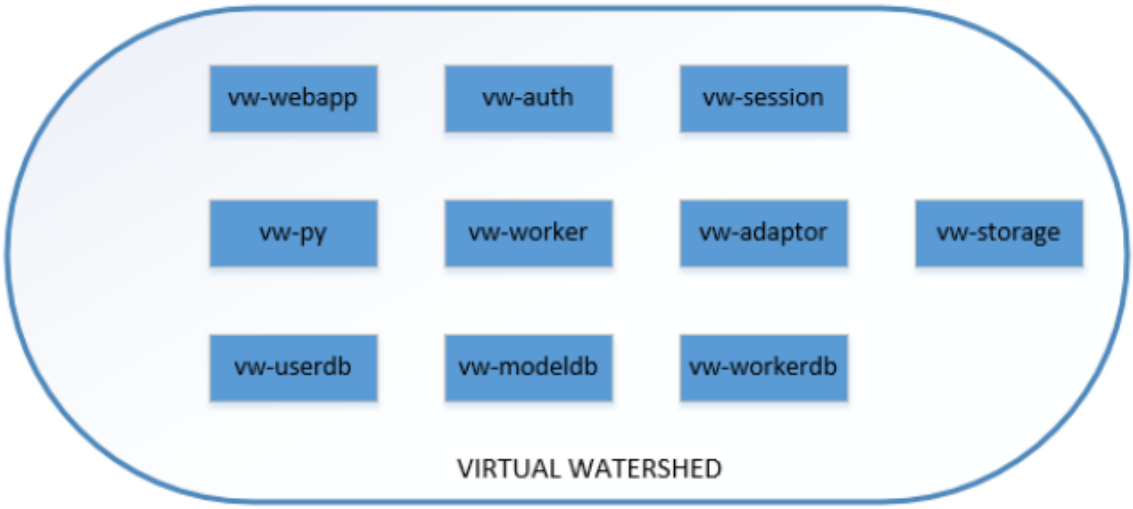

Figure 2.1: Components of the Virtual Watershed platform

common session backend, implemented with key-value data store, Redis [6], which is shared with different applications on the VW platform. VW-STORAGE is a generic wrapper for data storage which can be configured with different storage providers in cloud or host machine. VW-WORKER is worker service which is configured to a messaging queue to handle the processing of the model execution. VW-ADAPTER contains the model-run API service to perform the upload and download of model simulation files in the platform, creation and deletion of the model runs, starting and stopping of model run execution. VW-MODELDB uses a Postgres [46] backend to store the model run file details and progress updates of model run. VW-WORKERDB is configured as Redis store backend which is used as the messaging queue between VW-WORKER and VW-ADAPTER. VW-USERDB uses a Postgres storage backend to store the login credentials and other details of the registered users in the VW platform.

To run a model in Virtual Watershed portal, the user need to upload all the input files required by the specific model into the system using the restful API service. The uploaded files were stored at a common storage location. A model id was generated to uniquely identify the model run and placed it on an asynchronous job-queue. When the worker container become available, they grab the model id from the job-queue

and fetches the required input files from the storage location to start the execution of the model run. After finishing the model run, the output files were stored at the storage location along with the input and log files, providing the user an option to download the files later for further studies.

2.2 Precipitation Runoff Modeling System

Precipitation Runoff Modeling System (PRMS) is one of the environmental models currently supported in the VW platform. PRMS is a deterministic, distributed-parameter, physical process based modeling system used by hydrologists to study how the variations in combinations of precipitation, climate, and land use affects streamflow, sedimental yields and general basin hydrology [50]. Environmental scientists use the PRMS model to simulate the hydrologic processes at a watershed scale and analyze the effects of climate changes and human activities on water resources. The model can be used to estimate daily streamflow, snowmelt from input time-series data of daily precipitation, minimum and maximum air temperature, short-wave solar radiations and pan evaporation. To estimate hydrologic responses of the watershed, PRMS divides the model area into discrete Hydrologic Response Unit (HRU) of any shape. Each HRU is identified by an index and the numerical index starts from 1. HRU is assumed to be uniform on the physical properties. According to [26], HRU can be composed either of land, lake, swale or inactive. A HRU will have numerous parameters [13] and its values would be defined in one of the input file called parameter file. For example, the vegetation type of a HRU is represented using the variable 'cov_type' in the parameter file and its value would be either 0 (bare soil), 1 (grasses), 2 (shrubs), 3 (trees) or 4 (coniferous) [25].

PRMS model requires three input files to run the simulation. They are parameter file, data file and control file. Parameter file specifies the parameters and its dimensions required by the model. The dimension can be space related, time-related, and the combination of both or others. The data file contains the time series historical records of climate variables. The control file specifies the control parameters.

The simulation of PRMS model results in the generation of four output files - water budget file, statistic variables file, animation file and map results file.

2.3 Problems of Current VW Platform

The following section will briefly discuss the problems faced by researchers while performing modeling activities with VW platform.

2.3.1 Wait time of model simulation jobs

When the VW platform is flooded with model simulation jobs, the users were forced to wait for a long time to get their job started. This problem could be handled by the addition of more worker containers to the VW platform. However, the addition of more workers on the same host machine could cause serious performance issues. The addition of more workers drastically increases the memory and resource consumption on the host server, which could ultimately slow down the model run execution. Therefore a feasible solution to this approach is to distribute the workers across multiple machines. This way, all the workers could simultaneously grab jobs from the queue, process the jobs and store the output files at the common storage location.

We could rent instances from cloud providers and integrate them with the platform to increase the job processing rate. Currently, cloud providers offer various types of machine instances to host a service for a fixed time period. The cloud host providers follow pay-per-use [29] payment structure in which the user has to pay only for the time the service is being used. This gives us the liberty to develop a hybrid server system forming a cluster of owned and rented machine instances to reduce the waiting time of users on model execution jobs in the VW platform. However, there are several concerns we need to address to develop a budget based hybrid server system. What is the trade-off between the rental budget and the server performance? How can we ensure effective utilization of resources by avoiding over-provisioning/under-provisioning of the resources? How can we make sure the allocated budget is used judiciously over the assigned time period?

The cloud providers could provide budget notifications and offer surveys to help project managers in efficiently utilizing the allocated budget amount. Cloud providers offer several mechanisms to scale the machine instances automatically to maintain the right amount of resources to handle the load on the application. This is facilitated by imposing a set of rules over a cluster of machine instances. These rules can be of two types: 1) Metric-based and 2) Time-based. Using metric-based rules, the instances are scaled up when the CPU utilization crosses a threshold level and are scaled down when the CPU utilization comes down. The user can specify a minimum and a maximum number of instances to run. With time-based rules, the user can define a rule to decide when to auto scale. For example, the user can state a rule that triggers an instance every 7 am on Friday in a given time zone. However, these auto scale services have their own flaws. Assume a budget amount is allocated for a time period of 20 hours and the manager wants the amount to be spent judiciously taking into account the entire 20 hours time frame. With the current auto scaling features in cloud providers, the auto scale up happens whenever there is a need. Due to the stochastic nature of the job arrivals, the allocated budget may get utilized within the first few hours of the allocated time period and the later customers have to wait longer time to get their service done. Furthermore, there is less control over the budget, and the chances are high that the system exceeds the predefined budget. Moreover, the CPU utilization may not be always considered a right indicator to scale up or down the instances. For example, on performing complex jobs, the utilization could become unstable and may reach or go beyond the threshold memory utilization level. At such moments, instead of spinning new instances instantaneously, if we give a few more seconds for the utilization to come down, the over-provisioning of the resources could be avoided.

2.3.2 Creation of user defined scenarios

Hydrologists often need to recreate different environmental scenarios to analyze the impacts of climate changes on watersheds. Simulating user-defined scenarios would

help researchers immensely in understanding the situation better and formulating an effective action plan for handling unexpected scenarios. To simulate a user-defined scenario, hydrologists need to make modifications on the parameter values in the underlying model input files. Most of the hydrologists may not be familiar with programming and handling of complex data processing tools. Therefore, researchers are forced into a situation, where they have to manually edit the model input files (plain text file) by copying the original data into third party text editors such as Sublime Text, Atom or Notepad++, and then replace the original data with the modified content. Sometimes, the changes to be made are spread over the entire input file (in multiple pages) depending on what type of scenario the researcher wants to simulate. Also, the modification of the model parameter values on large input files could become a painful and tedious process as the user have to be extremely careful while selecting the values in the file because an incorrect button click could deselect all the previous selections and the user would have to start over again.

2.3.3 Accuracy of model predictions

The accuracy of the model prediction is very crucial as it helps us in deciding the correct action plan to be taken during a critical situation. With the increasing popularity of Machine Learning (ML) concepts, the demand for tools utilizing ML techniques to work with large data sets is becoming high. However, many of the tools available in this field are either still in their early stage of development or not experimented well enough to evaluate their true potential on different scenarios. More studies have to be conducted in this area to explore the hidden capabilities of machine learning tools in finding solutions to research problems. Even though many efficient strategies and libraries have been introduced to apply machine learning algorithms in clusters or distributed systems, very few have been adopted to build real world applications. Apache Spark [30] is an open source cluster-computing framework for large-scale data processing with built-in modules for machine learning and graph processing. Though the methodologies in Apache Spark have been subjected to many research studies,

there are very few machine learning research applications which use Apache Spark.

The performance of hydrologic model simulations can be improved in different ways by reducing the uncertainties from various sources. They are pre-processing of model inputs, data assimilation, model calibration, and model result post-processing. Model input pre-processing deals with the uncertainties from model input variables, whereas data assimilation handles the uncertainties in model initial and boundary conditions. The uncertainties in model parameterization are taken care by model calibration techniques and uncertainties related to model results are taken care through post-processing techniques. Due to the increased complexities associated with hydrological models, it is difficult and highly challenging to apply model input pre-processing, data assimilation and model calibration to improve the model predictions. Considering all these, post-processing of model results is considered to be the one of the most feasible option to deal with the accuracy of model predictions. Also, post-processing requires less computational resources when compared to data assimilation and model calibration approaches.

Through this thesis, we also focus on how effectively machine learning techniques can be utilized in the post-processing of model results to improve the accuracy of predictions made from computer-simulated models. A web-based prototype application, leveraging the data processing capabilities of Apache Spark, was built for the experimental study. The details of the proposed approach and the prototype application is discussed in Section 4.3

2.4 Related Work

There are numerous studies conducted in the field of dynamic provisioning of computing resources in a cloud environment. Some of the successful works are briefly discussed in this section.

Rodrigo *et al.* [5] proposed an adaptive provisioning of computing resources based on workload information and analytical performance to offer end users the guaranteed Quality of Services (QoS). The QoS targets were application specific and were

based on requests service time, rejection rate of requests and utilization of available resources. The proposed model could estimate the number of VM instances that are to be allocated for each application by analyzing the observed system performance and the predicted load information. The efficiency of the proposed provisioning approach was tested using application-specific workloads, and the model dynamically provision resources to meet the predefined QoS targets by analyzing the variations in the workload intensity.

Qian Zhu *et al.* [62] proposed a dynamic resource provisioning algorithm based on feedback control and budget constraints to allocate computational resources. The goal of the study was to maximize the application QoS by meeting both time and budget constraints. The CPU cycles and memory were dynamically provisioned between multiple virtual machines inside a cluster to meet the application QoS targets. The proposed approach worked better compared with static scheduling and work conserving approach way of resource provisioning. The flaw with this approach was that it requires the reconfiguration of computing resources within the machine instances, which is not well recommended in the current cloud environment where resources could be efficiently managed by the addition and removal of virtual machines from the cloud host providers. Moreover, the dynamic allocation of resources based on CPU cycle and memory usage could go inaccurate more often, as the parameters cannot truly indicate the need for more resources. There are chances that the virtual machine is just busy with some low-CPU or low network jobs.

Bi *et al.* [3] proposed a dynamic provisioning technique to optimize provisioning in cluster-based virtualized multi-tier applications using a hybrid queuing model. The model could predict the number of virtual machines needed for each virtualized multi-tier application to meet the QoS target on request service time. Request rate, service rate, and end-to-end response time were used to estimate the number of virtual machines required.

In this thesis, we proposed and implemented a self-managed elastic scale hybrid server system using budget input and user feedback. The system is referred as a

hybrid, as it constitutes a cluster of owned and rented servers. The proposed system facilitates on-demand provisioning of computing resources to ensure reduced waiting time for jobs consistently over a predefined period of time within the allocated budget constraints. A modified queuing model was used to build the system. The system could make estimations on waiting time and queue length based on the budget amount. The system also contains a user feedback monitoring module to generate auto alert emails to notify the manager on performance issues. In our approach, the waiting time of the job requests is linked directly to the allocated budget amount. Since cloud providers follow a flexible pay-as-you-go payment model [51], our approach is more useful as it relates the budget amount with the desired waiting time and thereby helps the managers in making budget decisions more easily.

2.5 On-demand Computation

On-demand computing (ODC) is an enterprise level computing model in which the computing resources are made available to the users based on their demands. The computing resources referred here are elastic in nature and are either owned by the enterprise or rented from third party cloud service providers. The resources are said to be elastic because they can be easily scaled up or scaled down based on the requirements, without disrupting the underlying operations. In the current highly volatile computational environment, the demand for computing resources vary drastically from time to time. Considering the increasing expenses on computing infrastructures, it is not economical for an enterprise to buy resources more than what are needed on average. Buying more than what required just to meet the peak requirements could lead to the under-utilization of the available resources. ODC comes handy in dealing with such scenarios as it has the flexibility to deal with fluctuating demands. In ODC, the resources are made available to the user, only if they are really needed at a specific time. In this way, ODC ensures the maximum efficient use of the computational resources. With the emergence of on-demand computation, usage-based payment structure becomes more popular. In usage-based payment structure, the computing

resources are made available to the users as needed and the users have to pay based on their specific usage rather than a flat rate. Currently Amazon EC2, Google GEC and Microsoft Azure are the most prominent cloud host service providers.

2.6 Key players in On-demand Computation

Amazon Elastic Cloud Compute (EC2) is a web-based service, offered by Amazon, to facilitate the running of business applications in the Amazon Web Services (AWS) public cloud. EC2 provides a web service to facilitate the scalable deployment of applications. The virtual machines (called as instances) in EC2 can be created, launched and terminated based on the user's requirements. The service is termed elastic, as it facilitates the scale up/down of resources based on needs and follows a pay per usage payment structure. Amazon EC2 [43] offers instances of different sizes, types, and pricing patterns to satisfy users of different needs. The three pricing options provided by EC2 are: 1) On-demand 2) Reserved and 3) Spot. The EC2 on-demand instances are charged hourly. For using this plan, no commitment is needed and the user can quit the service at any time. For EC2 reserved instances, the user has a 1 year/3 year contract commitment and a 75 % discount on the hourly rate is awarded. With EC2 spot instances, the user names the price and bid for spare EC2 instances. The price for spot instances changes in real time based on supply and demand. The user can use the instances as long as the bid price is higher than the spot price.

Microsoft Azure Container Service (ACS) provides an on-demand container hosting environment for supporting the development and deployment of the container-based applications [2]. ACS facilitates the creation, configuration and management of a cluster of virtual machines with the help of popular open source scheduling and orchestration tools. Depending on the committed level of usage, ACS gives discounts ranging from 10-45 percent to its enterprise customers. Like EC2, Azure also provides instance types under different categories. General purpose, compute optimized, memory optimized, GPU and High-Performance Compute are the five container service

categories offered by Azure Container Service.

Google Container Engine (GKE) [16] is an on-demand container hosting service offered by Google to support the management and orchestration of container clusters in Google's public cloud. It offers various tools to create, re-size, and debug container clusters. Like EC2 and ACS, GKE also provides both command-line and web console to deploy and manage containers. Google charges a flat fee for container services based on the number of instances in a cluster. Google provides a Sustained Use Discounts (SUD) feature, in which the discounts are given based on how long a specific instance type, RAM size, and the number of CPUs being used. With more usage, the discounts will be more. Google revises the prices with SUD and updates the subscription details so that users don't need to explicitly make any adjustments in the payment plan. Google supports custom instance types, with which the user can specify what configuration required for the instances in the cluster. The customization feature in GKE is very flexible such that the user is not forced to over-provision the instances with more RAMs and CPUs to get the desired configuration.

Comparing the pricing structure of Microsoft Azure [32], Amazon EC2 [43] and Google Container Engine [17], ACS and GKE offers more flexibility to the users with the billing time period. With Amazon EC2, the instances are charged hourly, whereas ACS and GKE support per-minute-billing. For example, if a Amazon EC2 instance is run for five minutes and turned it off, EC2 will charge for a full hour of use. Whereas, with Azure and Google Container Engine, the user has to pay only for the exact five minutes of computing usage.

Chapter 3

Specification and Requirements

3.1 Requirement Specification

This section describes the functional and non-functional requirements of the system. Functional and non-functional requirements are helpful to understand the overall design of the system.

3.1.1 Functional Requirements

Functional requirements help us to understand what all functions the system should do. Table 3.1 describes the functional requirements of PRMS model modification component. Table 3.2 describes the functional requirements of the self-managed elastic hybrid server. And, Table 3.3 shows the functional requirements of model accuracy enhancer for computer-simulated models. The functional requirements are listed with a brief description and their priority in this work. Label 1 denotes high priority requirements that are implemented. Label 2 are those requirements that may be implemented in this work. Label 3 denotes those requirements that have not yet been implemented, but would be useful in the future work.

Table 3.1: Functional requirements of PRMS model scenarios component

Id	**Priority**	**Description**
R 01	1	The system should allow the user to choose the selection method while choosing HRUs for scenario creation
R 02	1	The system should allow user to select HRUs based on its parameter values
R 03	1	The system should allow user to select HRUs through simple drag and drop operation of HRU cells on a 2D map
R 04	1	The system should apply the specified modifications to the parameter values of chosen HRUs in the underlying model input files
R 05	2	The system should alert the user about the allowable maximum and minimum values of the parameter being modified
R 06	1	The system should update the HRU grid map instantly on modifying the parameter values of the HRU cells
R 07	1	The system should allow user to modify different parameters of different HRU cells at a time (manual selection of HRUs)
R 08	1	The system should highlight the selected HRU cells on the grid map and mark them with their latest updated parameter value, during the modification of parameter values of HRU cells
R 09	1	The system should overlay the 2D grid map at the correct geological location on a Google Map. The user should be able to zoom-in/zoom-out the HRU grid map overlay on the Google map
R 10	1	The system should allow the insertion of model runs to GSToRE [54]
R 11	1	The system should allow searching through the datasets stored in GSToRE and also facilitates the downloading of the datasets
R 12	3	The system should support the model scenario creation with ISNOBAL model [28]

Table 3.2: Functional requirements of self-managed elastic hybrid server system

Id	**Priority**	**Description**
R 01	1	The system should accept budget amount, instance rate, etc and provide an estimation on queue length and waiting time for the jobs.
R 02	1	The system should allow users to modify the budget, instance rate, etc during the execution of the system
R 03	1	The system should update the budget amount and recalculate the number of jobs that can be processed with available budget at the completion of each job.
R 04	1	The system should create new worker containers at designated machines when there is a need.
R 05	1	The system should remove the worker containers upon the completion of the assigned job.
R 06	1	The system should allow users to submit feedback based on the performance of the service.
R 07	2	The system should provide a facility for the manager to visualize the feed backs submitted by the users.
R 08	1	The system should automatically send an alert mail to the designated person, when the feed backs from users crosses a predefined threshold level.
R 09	1	The should allow the manager to track the completion of the submitted jobs
R 10	1	The system should support the creation of worker containers in rented machine instances from cloud providers
R 11	3	The system should support the creation of worker containers in actual instances of cloud service providers.
R 12	2	The system should allow the manager to update the interval at which the alert email to be sent if the user feedback score crosses the threshold.
R 13	2	The system should allow the manager to subscribe and unsubscribe users from receiving alert emails.
R 14	2	The system should allow the manager to set/update the threshold feedback score, reaching which an alert email would be sent to the subscribed users.

Table 3.3: Functional requirements of model accuracy enhancer component

Id	Priority	Description
R 01	1	The system should allow user to upload the model data file with as many columns(features), where the first two columns should contain the observed values and model predicted values respectively.
R 02	1	The system should allow the user to choose a machine learning model from four different ML models available in the system
R 03	1	The system should use the chosen machine learning model to train the model and generate a report showing the existing and improved RMSE, PBIAS, CD and NSE values for the given input data.
R 04	1	The system should provide an option to visualize and compare the model predictions and improved model predictions with the actual observed values.
R 04	3	The system should support data classification functions along with regression techniques available in Apache Spark.
R 04	3	The system should enable user to set complex rules that can be applied while improving the model predictions.

3.1.2 Non Functional Requirements

Non Functional Requirements describes how the system works. It tells us the global constraints on the system. The requirements are listed with an identifier and a brief description. Table 3.4 describes the non-functional requirements the works implemented as part of this thesis.

Table 3.4: Non Functional Requirements

Id	Description
R 01	The system should be implemented with Python Flask micro framework, JavaScript and Bootstrap.
R 02	The system should be implemented with a micro-service architecture
R 03	The system should support the modification of model inputs in .netCDF [41] file format
R 04	The system should run on Linux operating system
R 05	The system should maintain a user friendly web interface
R 06	The system should ensure only one worker container is allocated per machine and the number of jobs to be processed by the worker should be limited to one

3.2 Use Case Modeling

Use cases capture user requirements for a system by describing how a user uses a system to accomplish a particular goal. Use cases can be demonstrated both graphically and textually. This section includes a use case diagram for each component developed as part of this work. Figure 3.1 shows the use case diagram of our proposed self-managed elastic hybrid server system. Figure 3.2 shows the use case diagram of the model accuracy enhancer component. And, Figure 3.3 shows the use case diagram for the PRMS model scenario tool.

3.2.1 Self-managed Elastic Hybrid Server

* **UC 01 - View expected queue length and waiting time:**

  The user can view the estimated expectations on queue length and waiting time of the jobs based on the given inputs - budget amount, instance cost, average job execution time, expected job arrival rate, and budget period. The tool also includes a slider component to help the manager make easy adjustments on the budget amount and see how it affects the waiting time of jobs.

* **UC 02 - Track job completion:**

  The user can view the details of the completed jobs in real time. The details of the job include the waiting time of the job in the queue, the duration of job execution, the name of the server which executed the job, the category of the server (owned or rented), and the cost for the job execution.

* **UC 03 - Activate the settings on self-managed elastic hybrid server system:**

  The manager can initiate the hybrid elastic server prototype after inputting the budget amount, expected job arrival rate, instance cost and the budget period. Once the system is activated, the incoming jobs will be executed at the rented servers, if the owned servers are not available. For each job executed at

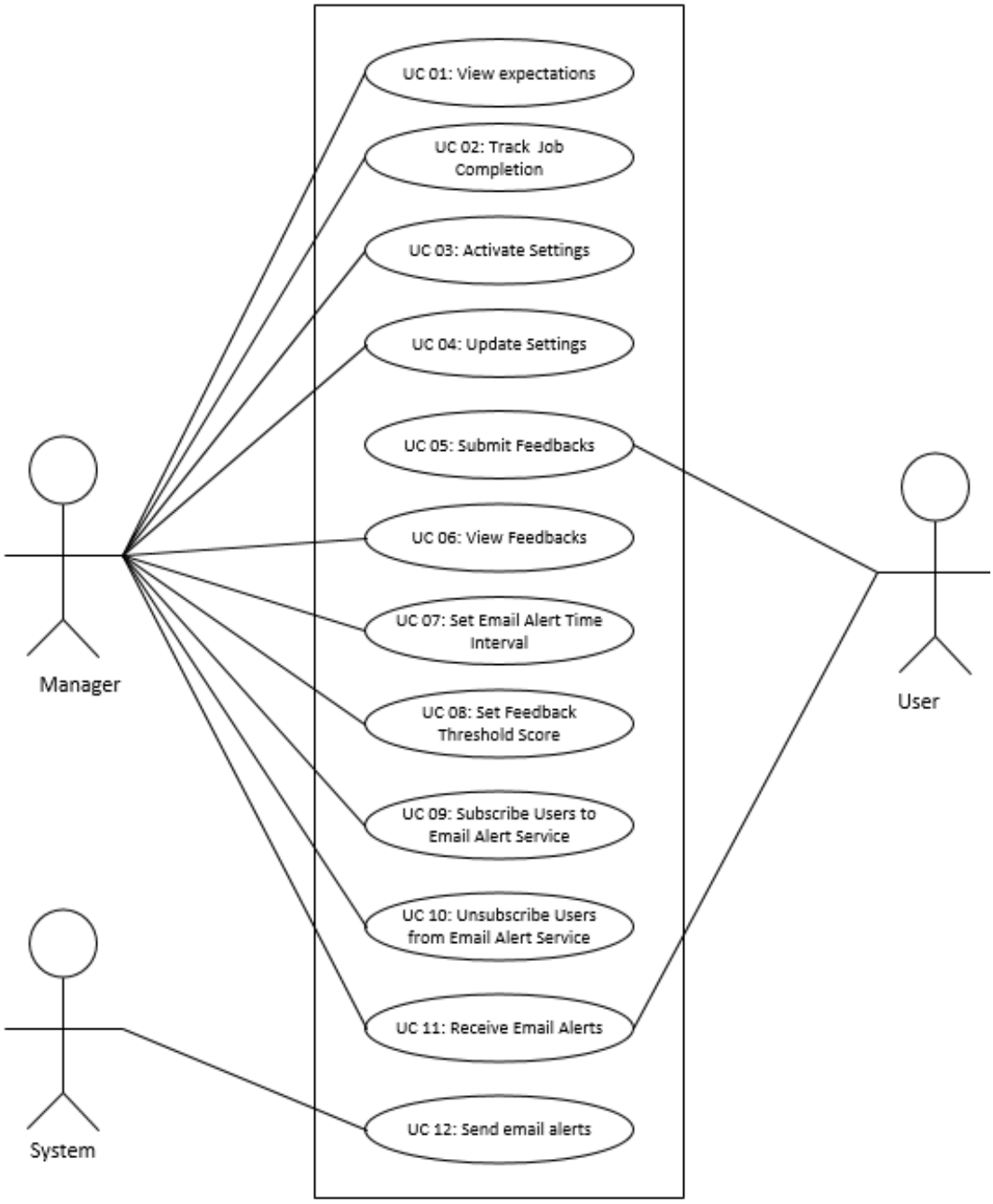

Figure 3.1: Use-case diagram of self-managed elastic hybrid server system

the rented server, the cost for executing the job is deducted from the budget amount.

* **UC 04 - Update the settings on self-managed elastic hybrid server system:**

  The manager can update the budget amount, instance cost, job arrival rate at any time during the budget period. The system recalculates the total allowable rented jobs and time interval based on the modified inputs.

* **UC 05 - Submit feedbacks:**

  The user can submit a survey form about the performance of the system. The survey form contains multiple choice questions. There are three choices given for each question, from which the user has to choose one to record the feedback. Each question is weighed between 0.0 and 1.0. The choices for the questions are weighed either -1, 0 or 1.

* **UC 06 - View user feedbacks:**

  The manager can view the feedbacks submitted by the users. The system will display separate bar charts for each question in the survey form and displays how many users opted a particular option in the question. The system should also display the current overall feedback score of the system.

* **UC 07 - Set email alert time interval:**

  The manager can customize the time duration (in seconds) at which the email alert to be sent when feedback threshold condition is met.

* **UC 08 - Set feedback threshold score:**

  The manager can set the feedback threshold score for the system. On crossing the threshold score, the system will start sending email alerts at regular time interval.

* **UC 09 - Subscribe to email alerts:**

  The manager can enroll a user for email notification alerts. The system should show the list of users subscribed to the auto email alerts. An auto-generated email will be sent to the user to notify about the subscription.

* **UC 10 - Unsubscribe from email alerts:**

  The manager can remove a user from the email alert subscription. An auto-generated email will be sent to notify the user about cancelling the subscription.

* **UC 11 - Receive email alerts:** The users who have enrolled for the email notification service will receive email alerts at regular time intervals when the user feedbacks about the service crosses the threshold level.

* **UC 12 - Send email alerts:**

  The system will generate auto email alerts when the overall feedback score of the system crosses a predefined threshold score. The users subscribed to the alert notification service will receive the email alerts at regular time intervals. The auto generated alert email should contain a visualization of the user feedbacks.

3.2.2 Model Accuracy Enhancer

* **UC 01 - Choose ML model:**

  The system should support more than one machine learning model in the prototype. The user should be able to choose the desired machine learning model to train the input data set.

* **UC 02 - Upload Input file:**

  The user should be able to upload the model data file to perform machine learning processing on the input data set. The uploaded file should be in CSV format and the first two columns in the file should be observed data and model predicted values respectively.

* **UC 03 - Generate evaluation report:**

  The system should display a detailed report based on the machine learning processing on the uploaded input data set. The report should include results of the various quantitative statistics to help the user understand the improvement in accuracy predictions which has gained through the ML approach.

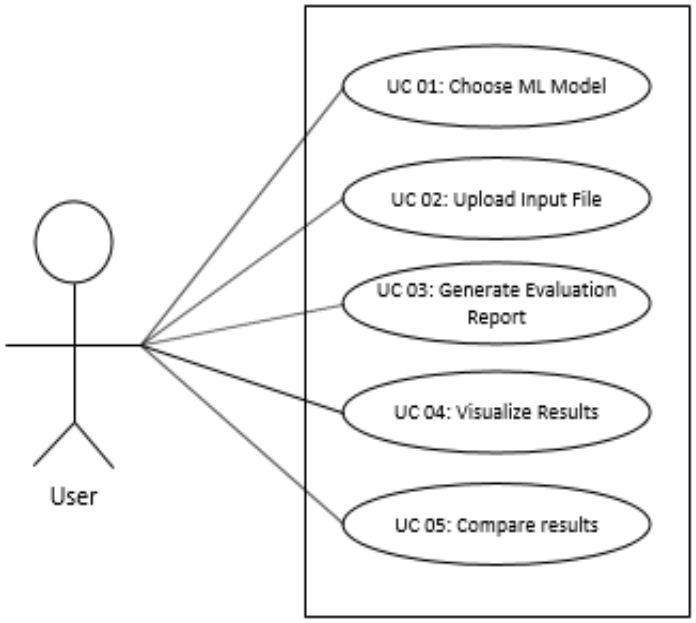

Figure 3.2: Use-case diagram of the model accuracy enhancer component

* **UC 04 - Visualization of the results:**

  The system should display a line chart displaying the actual observations, model predicted values and the improved predictions.

* **UC 05 - Comparison of the results:**

  The user should be allowed to compare the actual observations between model predicted values and the improved predictions obtained using the proposed ML based approach.

3.2.3 PRMS Model Scenarios Tool

* **UC 01 - Choose the HRU selection method:** The user should be allowed to choose from two HRU [13] selection methods to perform the model scenario creation. The HRUs could be selected either based on its parameter values or through manual selection by performing drag and drop operation on 2D grid map.

* **UC 02 - HRU selection using drag and drop:** The user should be allowed to choose desired HRU grids through simple drag and drop operation on a 2D grid map.

* **UC 03 - HRU selection using parameter constraints:** The user should be allowed to choose desired HRU grids by specifying constraints on the parameter values of the HRU's.

* **UC 04 - Choose the model parameter for modification:** The system should display the complete list of parameters for the input dataset and allow the user to choose the parameter that needs to be modified. On choosing the parameter, the maximum and minimum values for the chosen parameter should be displayed to the user. This will prevent the user from inputting erroneous values for the model parameters while creating model run scenarios.

* **UC 05 - Set new value for the model parameter:** The user should be allowed to set new value for the chosen model parameter to create the desired model scenario.

* **UC 06 - Add parameter constraints for HRU selection:** The user should be able to add any number of parameter constraints to fine tune the selection of HRU grids for creating a desired model run scenario. The user should be allowed to specify upper and lower limits on parameter values to filter out the desired HRUs from the model area.

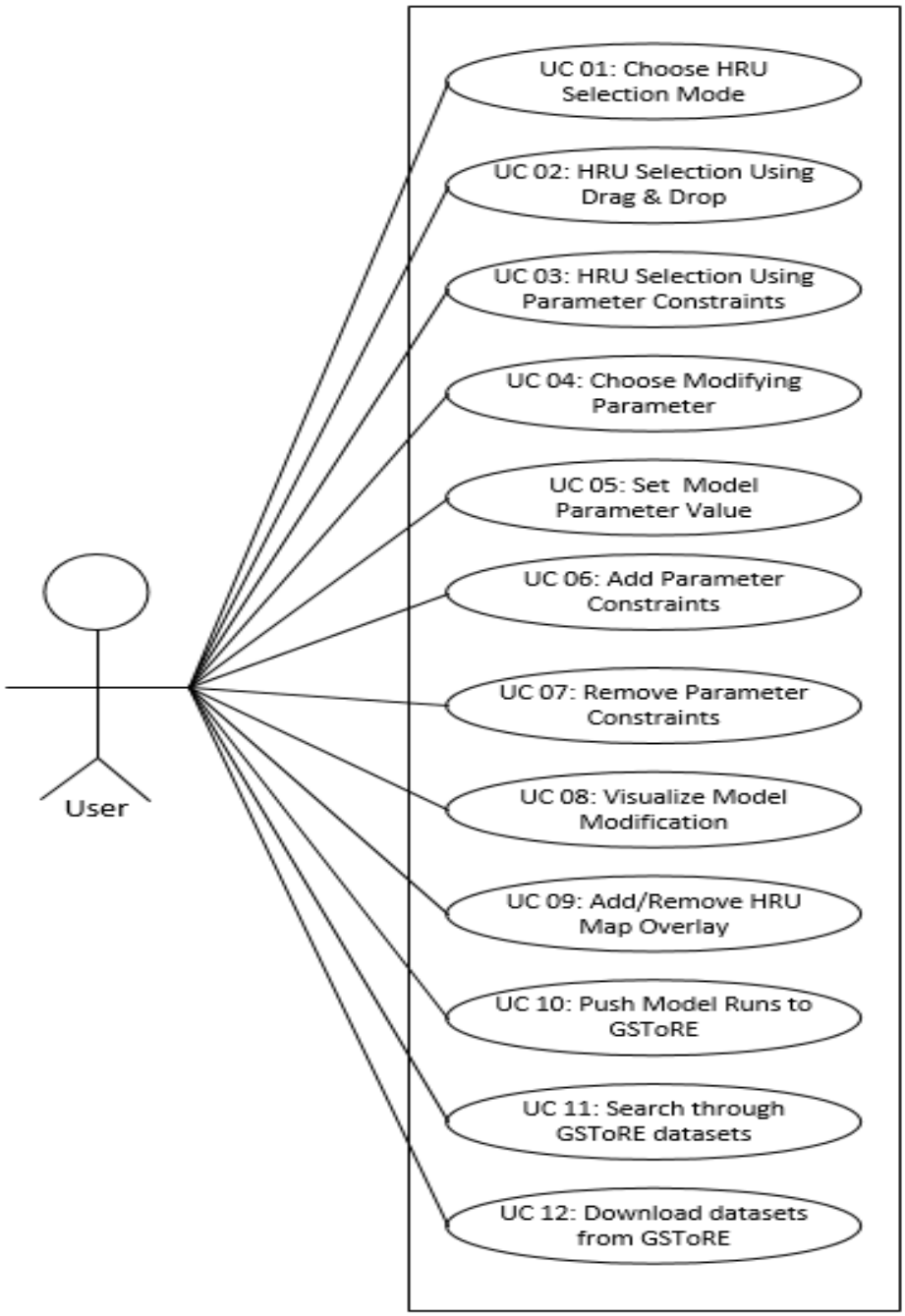

Figure 3.3: Use-case diagram for the PRMS Model Scenarios Tool

* **UC 07 - Remove parameter constraints:** The system should allow the user to remove parameter constraints to filter out desired HRUs.

* **UC 08 - Visualize model modification** The user should be able to visualize the modification of the model input file on the 2D HRU grid map. On applying the modifications, the cells on the HRU grid map should undergo a color change to give the user a visual feedback about the change on the parameter values.

* **UC 09 - Add/remove HRU map overlay on Google map:** The user should be able to add or remove the 2D HRU grid map overlay on the Google map. Also, the user should be allowed to change the HRU map transparency using mouse click.

* **UC 10 - Push model runs To GSToRE:** The user should be allowed to choose the model simulations and push them to GSToRE data storage server. The push operation will upload the model files and its associated metadata to GSToRE.

* **UC 11 - Search through GSToRE datasets:** The user should be allowed to search through the vast GSToRE datasets with the help of suitable filters.

* **UC 12 - Download datsets from GSToRE:** The user should be able to download datasets from GSToRE

Chapter 4

Design and Implementation

This chapter will introduce the implementation details of the three components that are build as part of this thesis work. Section 4.1 explains the details of the developed elastic hybrid server for improving the efficiency of handling model simulation jobs in the VW platform. Section 4.2 describes the model modification component implemented for assisting environmental scientists in scenario-based studies. Section 4.3 explains the details of the generic ML-based approach for minimizing the prediction errors in computer-simulated models.

4.1 Self-managed Elastic Hybrid Server System

We built a prototype of the self-managed elastic hybrid server [39] to perform on-demand provisioning of model containers in the Virtual Watershed platform. To implement the prototype, we incorporated new functionalities, restructured and reconfigured the components of the Virtual Watershed project described in [21]. The developed system uses an innovative server-usage optimization approach to facilitate the dynamic provisioning of the resources based on budget constraints. The design and implementation details of the elastic server component are described below.

4.1.1 Proposed Approach

The M/M/1/1/∞/∞ is one of the simplest queuing models which is widely used in many real world applications. As the Kendall notation denotes, there is one server,

the queue length can be infinite and the population (maximum number of jobs at the same time) can also be infinite. The queuing model represents the scenario where the job arrival follows a Poisson distribution [9] and the jobs come at a rate of λ/hour, whereas the server processes the jobs at a rate of μ/hour. The rate of job arrival should be less than the rate of job process. i.e. $\lambda < \mu$. Then based on [22], the expected queue length (L) and average wait time (T) of the job in the queue will be as follows,

$$L = \frac{\lambda^2}{\mu^2 - \lambda\mu} \tag{4.1}$$

$$T = \frac{\lambda}{\mu^2 - \lambda\mu} \tag{4.2}$$

To implement a prototype, we modified the M/M/1/1/∞/∞ queuing model and used it in the proposed self-managed elastic hybrid server system. The basic idea was to incorporate budget amount, budget period, cost of rented instances, and the average time for job execution into a generic formula to estimate the average queue length and job waiting time in an hybrid server environment. The modified queuing model uses a cluster of owned and rented servers to process the jobs.

Let B be the allocated budget amount, T_{own} be the average job execution time in an owned server, T_{rent} be the average job execution time in a rented server, and N_0 be the total number of owned servers in the hybrid cluster, then $(N_0 * T_b)/T_{own}$ would be the maximum number of jobs that could be processed by owned servers during the budget period T_b. Now if a rented instance costs \$P/hour, then B/ (P * T_{rent}) would be the total number of jobs that can be processed with rented servers for the allocated budget amount B. Therefore, the total number of jobs processed by the hybrid cluster during the budget period would be the sum of $(N_0 * T_b)/T_{own}$ and B/ (P * T_{rent}). Since our goal was to utilize the rented resources judiciously, the usage of rented servers is distributed uniformly across the budget time period T_b. To achieve this, the manager should rent a job at every time interval, $T_{int} = (T_b * T_{rent} * P)/B$,

if the owned servers are not available at T_{int} to process the job. If the owned servers are available during the inspection at T_{int}, then the system need not spin up a rental server for the incoming job. The system will increment a counter variable to record such occasions, so that, later if a job comes in and the owned servers are busy, the system will rent a server immediately, instead of waiting till T_{int}. In this manner, the proposed approach ensures that the rented servers are utilized judiciously during the budget period, T_b.

After incorporating the above details into equation 4.2, the expected average queue length (L_H) and the expected average wait time of job (T_H) in the queue would be estimated as follows:

$$L_H = \frac{\lambda^2}{(\frac{N_0}{T_{own}} + \frac{B}{P*T_{rent}})^2 - \lambda(\frac{N_0}{T_{own}} + \frac{B}{P*T_{rent}})} \tag{4.3}$$

$$T_H = \frac{\lambda}{(\frac{N_0}{T_{own}} + \frac{B}{P*T_{rent}})^2 - \lambda(\frac{N_0}{T_{own}} + \frac{B}{P*T_{rent}})} \tag{4.4}$$

Figure 4.1 shows the work flow of our proposed self-managed hybrid server system. Let consider an example to illustrate the proposed approach. Assume a manager has 5 owned servers and the servers in the hybrid cluster (owned and rented servers) takes an average of 10 minutes to finish the job. If the manager can spend $100 for a budget period of one hour, and each rented instance cost $1/hour, then a maximum of 600 jobs can be rented during the budget period of 1 hour. To ensure uniform usage of rented servers during the one hour budget period, the system should rent a job every 0.1 minutes (i.e. T_{int}), if the job queue is not empty at T_{int}. If at T_{int}, the owned servers are available, then a counter variable is incremented and wait for the next job. With the owned and rented servers, the hybrid server system could process at most 630 jobs during the budget period of one hour. Now, assume that 500 jobs arrive per hour (i.e. $\lambda = 500$), then based on equation 4.3 and equation 4.4, the expected queue length would be estimated as 3.0525 and the average waiting time of the job in the queue would be 0.0061.

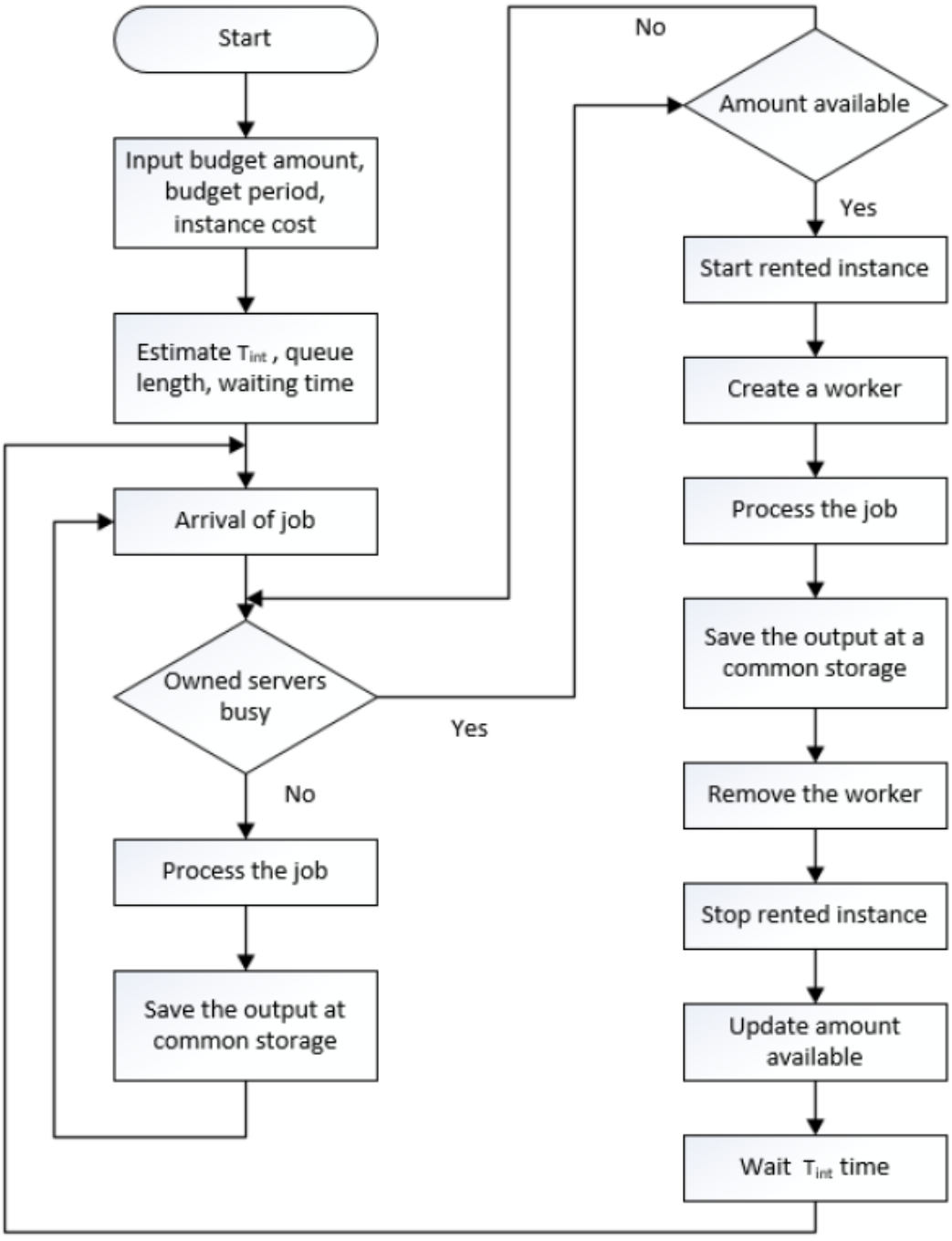

Figure 4.1: Work flow of the proposed self-managed elastic hybrid server system

4.1.2 Prototype System

Figure 4.2 shows the architecture of the proposed hybrid server system. A Docker multi-host swarm cluster [12] was used to build the proposed system. In a swarm cluster, one machine will act as the swarm master and any number of instances can be added to the cluster. As seen in the Figure 4.2, Host0 acts as the swarm master and contains the owned worker. Besides the worker container, all the Docker containers for handling various services in the VW platform are also placed in the host0 machine. The system maintains a worker pool of rented machines and the rented workers are distributed across the machines. To keep the prototype system simple, the number of worker containers allocated per machine is limited to one and each worker is allowed to process exactly one job at a time. The containers across the host instances are connected using a Docker overlay network [12] and hence can communicate with each other. All the worker containers are configured to listen to a job queue. The job queue in the prototype system was implemented using the popular distributed task queue, Celery [49], which handles the execution of the jobs asynchronously. The workers are designed in such a manner that whenever a job is placed into the queue, the worker will pick the job and start processing the job. Host0 instance holds a task manager container which can start, stop, and terminate instances in the worker pool. Task manager takes the budget amount and instance cost details from the manager and handles the creation and removal of worker containers in rented instances. The creation and removal of Docker containers are facilitated using the python library for Docker engine API, docker-py [11].

One of the objectives of this study was to implement a self-managed hybrid server that utilizes the rented resources effectively to reduce the waiting time of the jobs for a specific time period under a predetermined budget limit. On inputting a budget amount, the proposed system calculates how much time the rented resources could be used to host the worker containers. With the rented time, the system estimates the number of jobs that can be processed with rented workers. Based on these estimations,

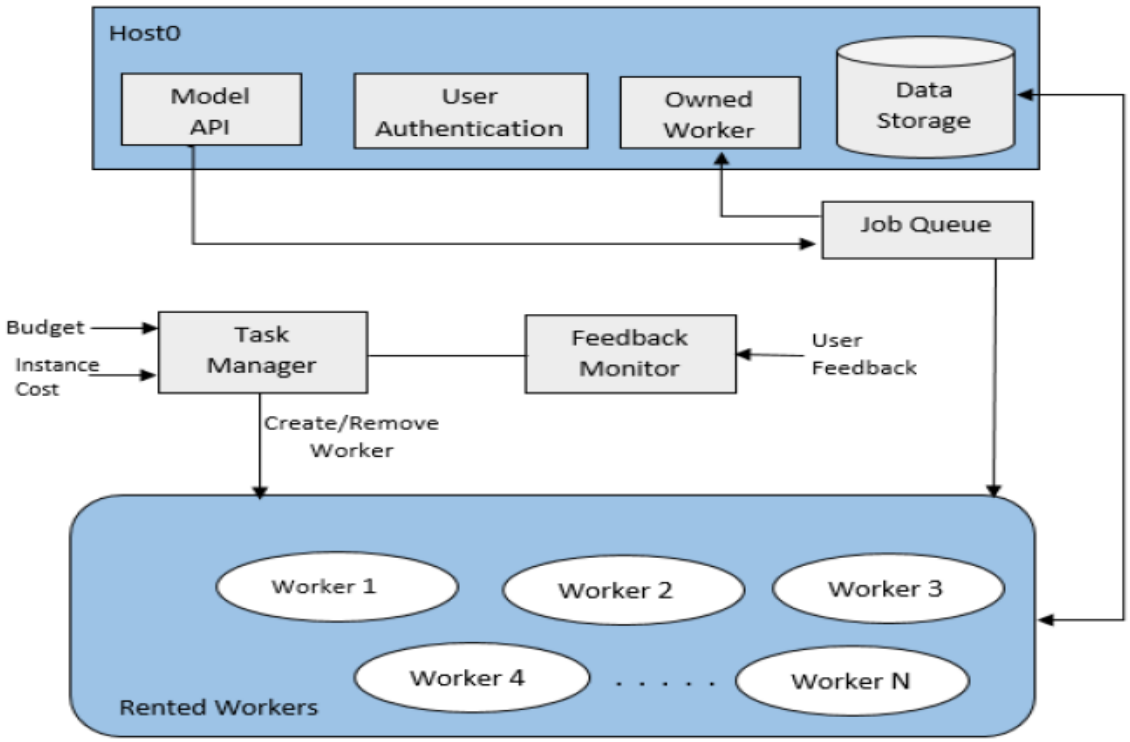

Figure 4.2: Architecture of self-managed elastic hybrid server system [39]

the system formulates a job execution plan using the owned and rented workers to bring down the overall waiting time of the jobs. The proposed system can prevent the unnecessary wastage of computing resources by shutting down the instances once the job execution is finished. This is done by sending a signal to the rented server after it finishes the execution of its job. However, the prototype is not fully self-managed as it requires the manager to change the budget based on the users' feedback.

4.1.2.1 Rented Worker Creation

Figure 4.3 shows the logic for the creation of new rented workers in the proposed system. On inputting the budget amount (B) and the cost of rented instances ($P/hr), the system estimates the total available rented time (T) from the cloud provider. The average execution time of the job (T_{avg}) is already available in the system from previous job execution details.

The total number of jobs that could be processed with rented workers is N =

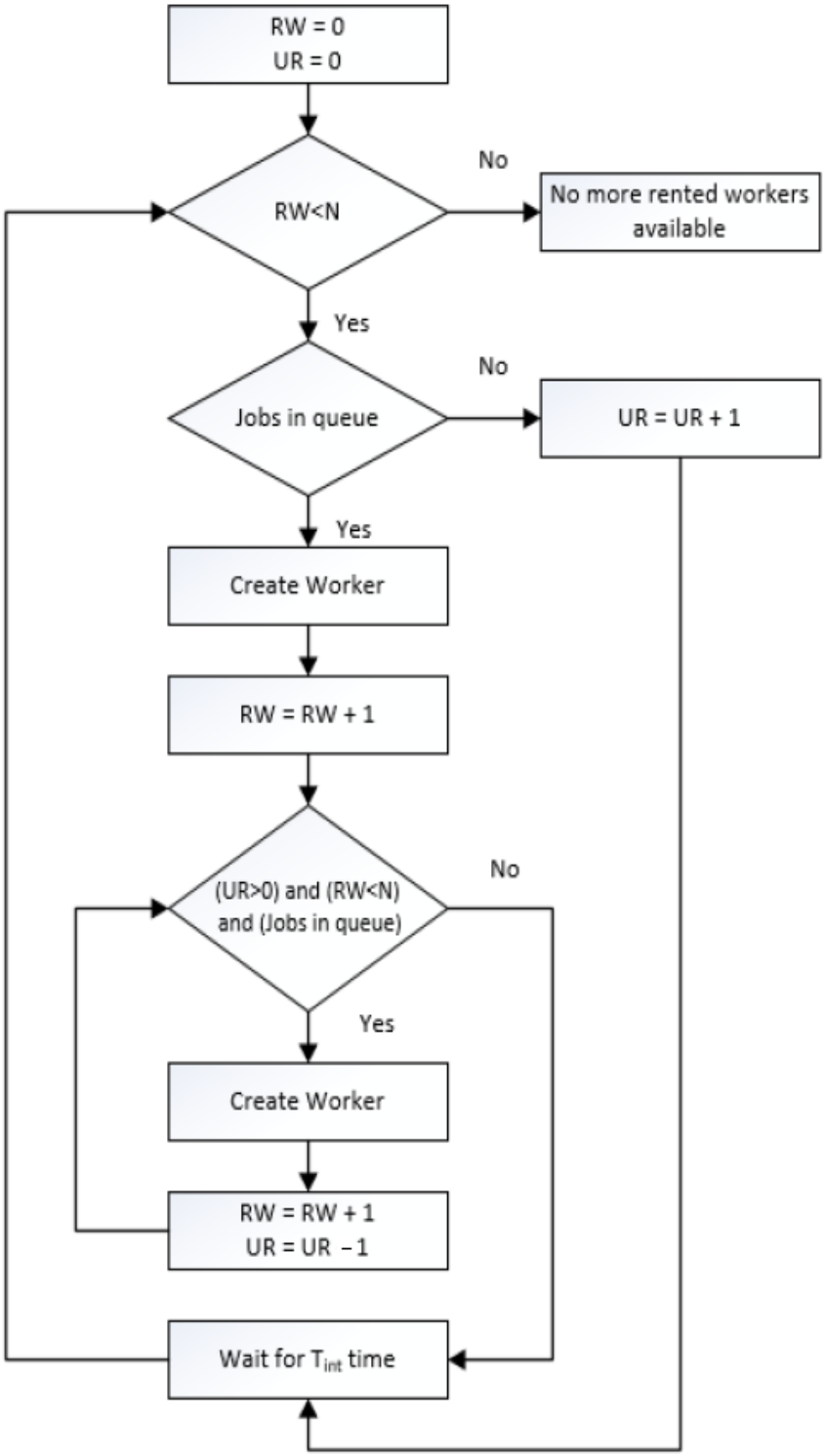

Figure 4.3: Logic for rented worker creation, UR =Unused Rentals, N = maximum number of models processed with rented workers for the input budget

T/ T_{avg}. The counter variable RW (Rented Workers) would keep track of the total number of jobs rented. Every T_{int} interval, the system would inspect whether there is a necessity for new workers. At T_{int}, if owned workers are not available (i.e. there are jobs waiting in the queue), then the system will create a new worker in one of the rented machines in the worker pool and increments the counter variable RW by one. If at T_{int}, an owned worker is available, then the system would record such occasions on to a counter variable UR (Unused Rentals), so that later, if a job comes in and the owned workers are not available, the system could immediately create a new worker to handle the job instead of waiting for the next T_{int} interval. The project manager can increase or decrease the budget amount in the middle of the execution and the system updates N accordingly with the changes in the budget amount. The execution time of the job varies with the workload on the host machine. The value of N is constantly updated on completion of each job based on the actual running time each job has taken for its execution. This process will be repeated until the number of jobs rented equals N, i.e. the maximum number of jobs that could be processed with rented containers for the given budget.

4.1.2.2 Configuration Module

The configuration module helps the manager in taking decisions over the budget amount to be allocated. Figure 4.4 shows a screenshot of show-info page in the configuration module. The user can input the budget amount, budget period, instance cost, average job execution time and expected job arrival rate to the system. The system will display the expectations on the waiting time and queue length based on the input parameters. The time interval at which rented servers will be used will also be shown to the manager. The module also includes a slider tool to adjust the budget amount. Therefore, instead of inputting different values for the budget amount, the user can simply use the slider to tune the budget amount to get the desired queue length, wait time and time interval. The interface will also provide error messages to alert the user on inputting wrong values for the input parameters. For example,

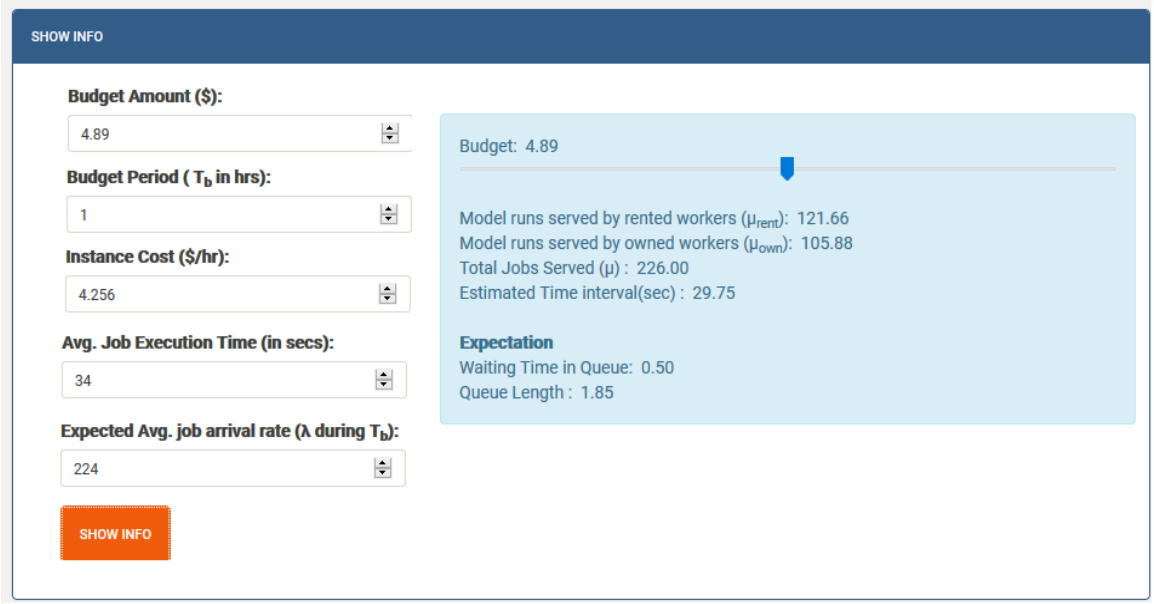

Figure 4.4: Screenshot of the configuration module

according to M/M/1/1/∞/∞ queuing model, the value of λ should be always less than μ. So whenever the user violates this condition, an error message would be displayed to alert the user.

Figure 4.5 shows a screenshot of the activate page in the configuration module, where the user can specify the budget amount, the budget period, the cost of instances, the expected job arrival rate and the average job execution time to activate the elastic server system. The system also supports updating the input parameters (budget amount, time period, instance cost and job arrival rate) at any time during budget period. Figure 4.6 shows the update page in the prototype system. In the update page of the configuration module, the current configuration setting will be displayed and the user can modify the settings by clicking the update button after inputting new values for the parameters.

4.1.2.3 Feedback Module

The prototype system includes a feedback survey form, where the user can provide feedback on the performance of the service. Figure 4.7 shows a screenshot of the survey form in the prototype system. The survey is intended to help managers to

Figure 4.5: Screenshot of activate page in the configuration module

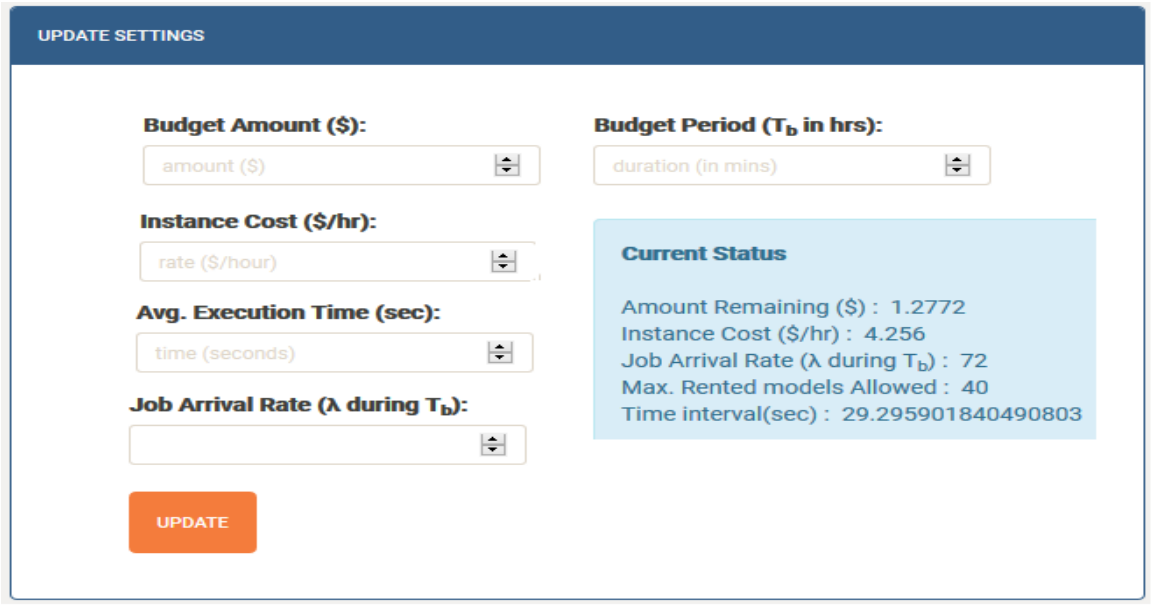

Figure 4.6: Screenshot of update page in the configuration module

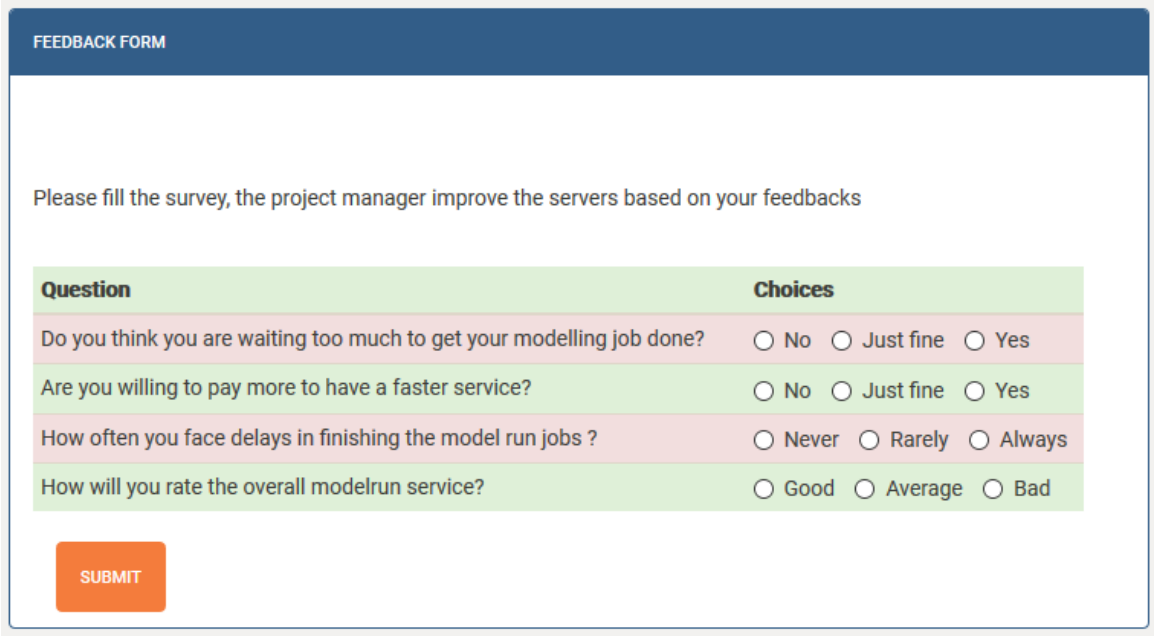

Figure 4.7: Screenshot of the survey form in the prototype system

make policy decisions on the budget amount to be allocated. Each question contains three answer options. The questions in the survey and their options are set with varying weight factor based on the behaviour of the question. The questions are weighed between 0.0 and 1.0 and the options are weighed either -1, 0 or +1. For example, the question "Are you willing to pay more to have a faster service?" holds a weighing factor of 0.8. Therefore if the user answers a "Yes" to this question, then the feedback score will be incremented with (0.8*1). The system will start sending alert emails at regular time intervals, once the feedback score passes the threshold level. Figure 4.8 shows a screenshot of the alert email sent from the prototype system.

The manager can also view the results of the feedback survey from the users in Survey Results page. The page will display separate bar graphs for each question in the survey questionnaire. The bar graphs shows the total votes obtained for each option of the question. This will help the manager to easily understand how efficiently the system can serve its users and help the manager in making decisions on increasing or decreasing the budget amount. Figure 4.9 shows a screenshot of the feedback visualization page in the prototype system.

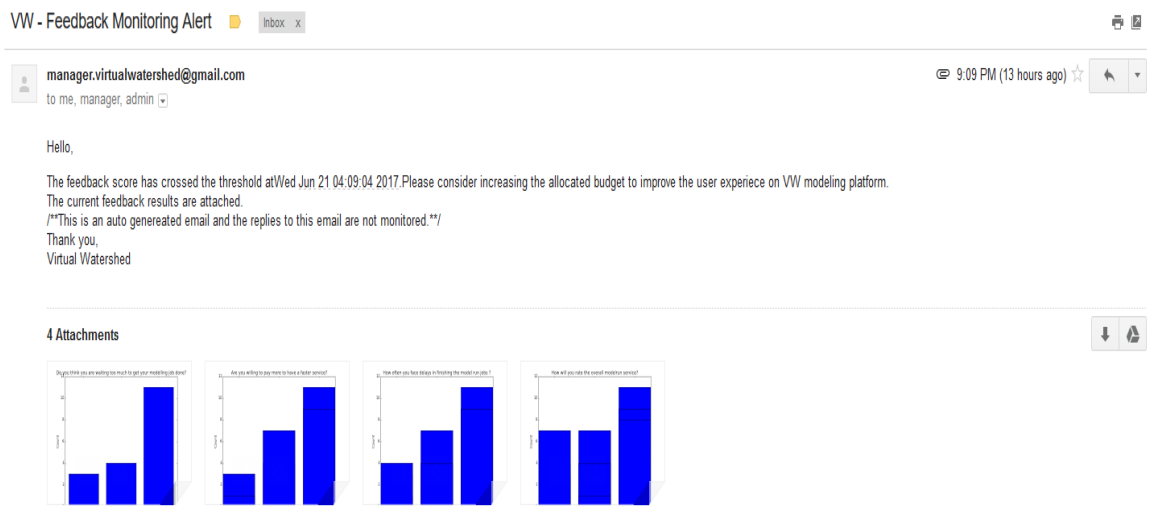

Figure 4.8: Screenshot of the alert email sent from the prototype system

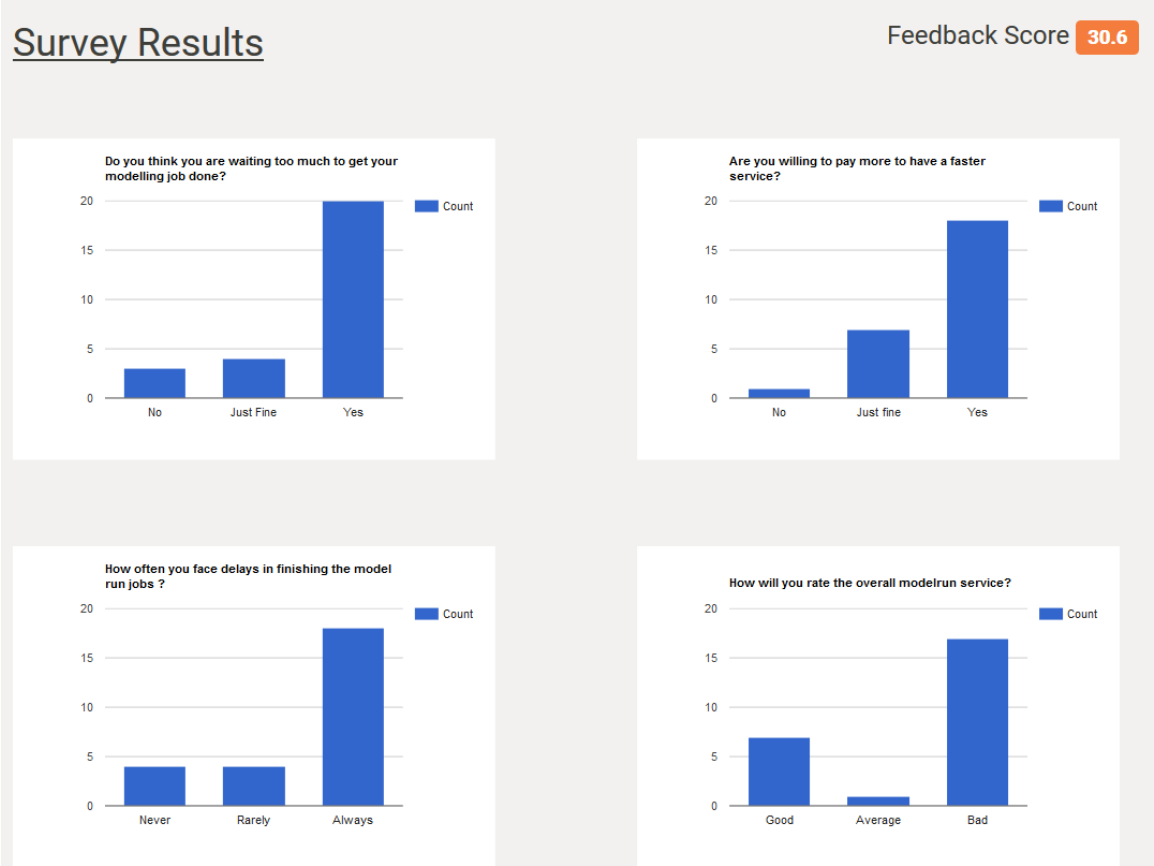

Figure 4.9: Screenshot of feedback visualization page in the prototype system

The prototype system includes an email settings page to manage the setting of

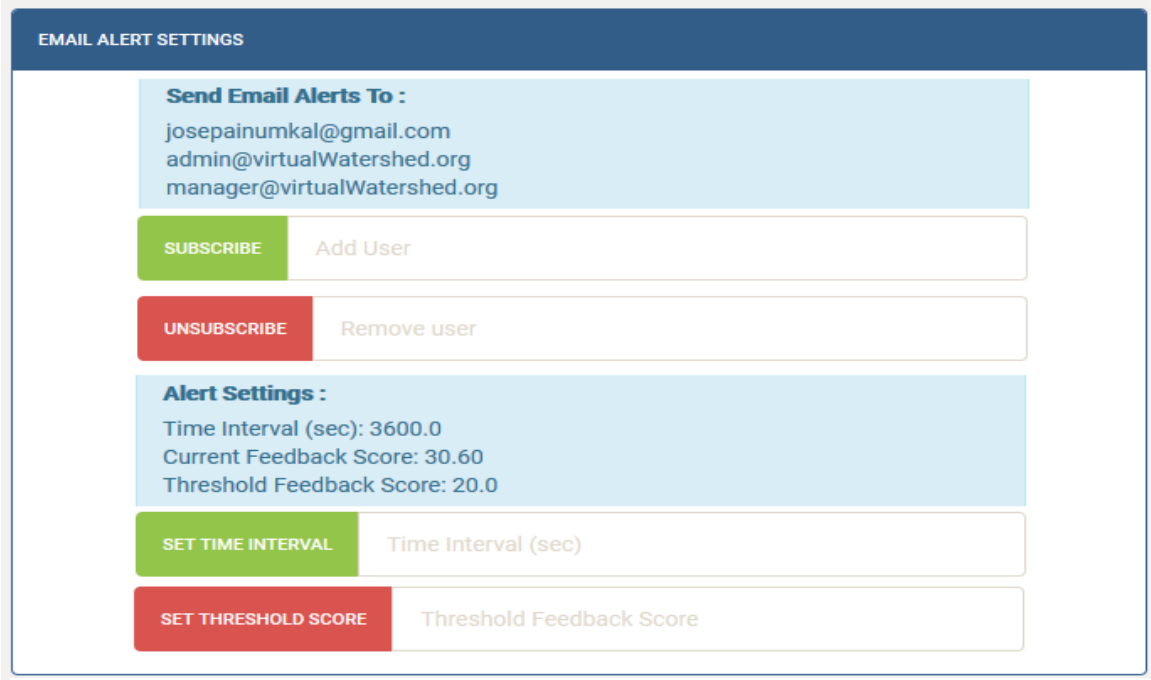

Figure 4.10: Screenshot of the email settings page

the email alerts. Figure 4.10 shows a screenshot of the email settings page. The manager can set a threshold level for the feedback score and also set a time interval at which emails to be sent. The system offers an easy interface to add/remove users from the email alert notification.

4.1.2.4 Task Dashboard

The system also includes a dashboard where the user can view the details of the finished jobs at real time. Figure 4.11 shows a screenshot of the task dashboard page in the prototype system. On finishing a model simulation job, the dashboard will display the details of the job such as the task id, the cost for the job execution, run-time of the job, waiting time in the queue, the name of the worker that processed the job and the category to which the worker belongs (owned or rented). The dashboard also shows the total number of jobs finished, the number of jobs processed with rented and owned workers, and also the remaining amount available to spend.

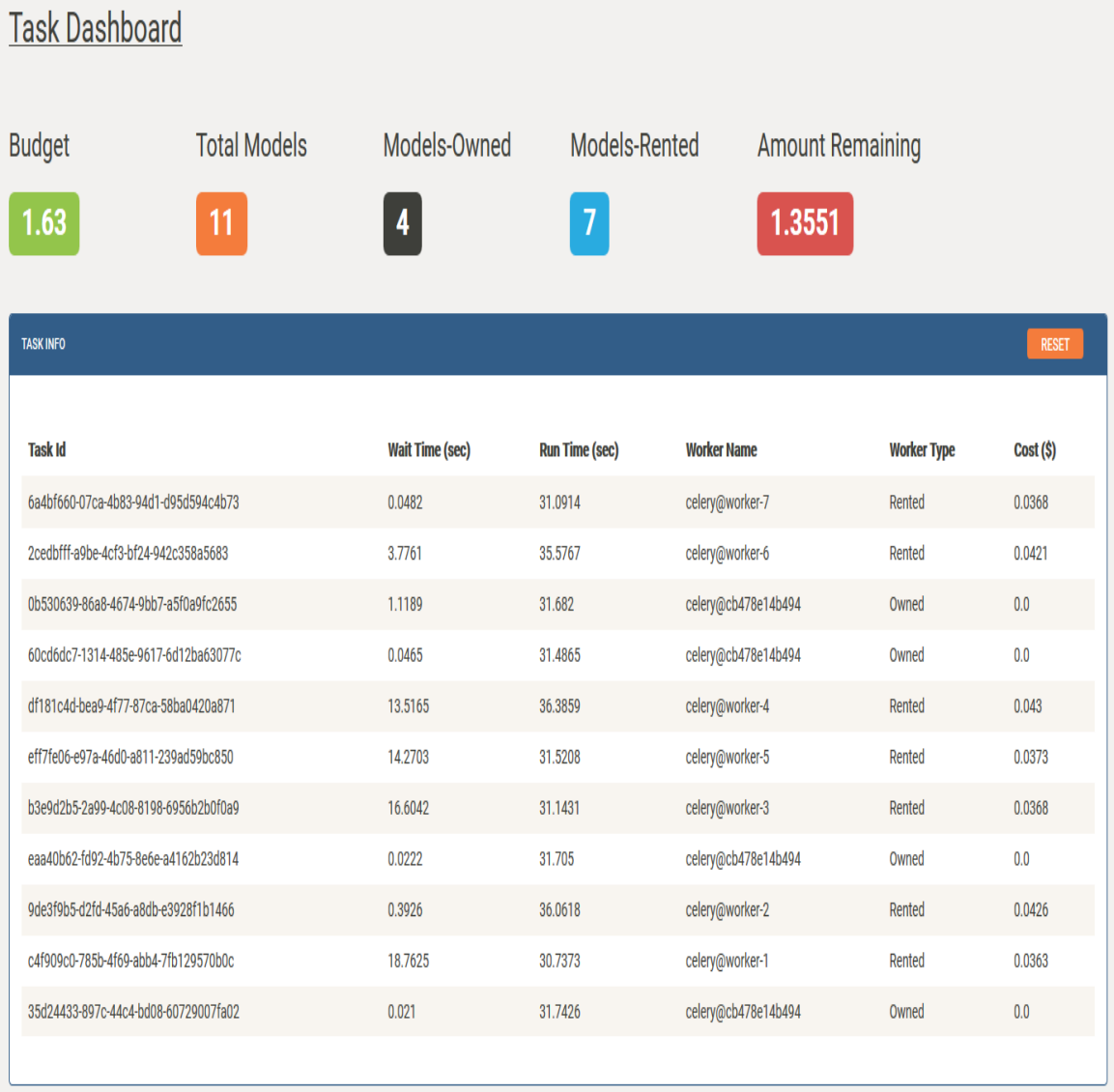

Figure 4.11: Screenshot of task dashboard page in the prototype system

4.2 PRMS Model Scenario Component

PRMS Model Scenario component enables researchers to modify existing model simulations and re-run models with modified input files to analyze user-defined model scenarios. The user need not have programming skills to use our model modification component. The user interface is made extremely simple and user-friendly so that the user can perform the model modification activities through simple mouse clicks.

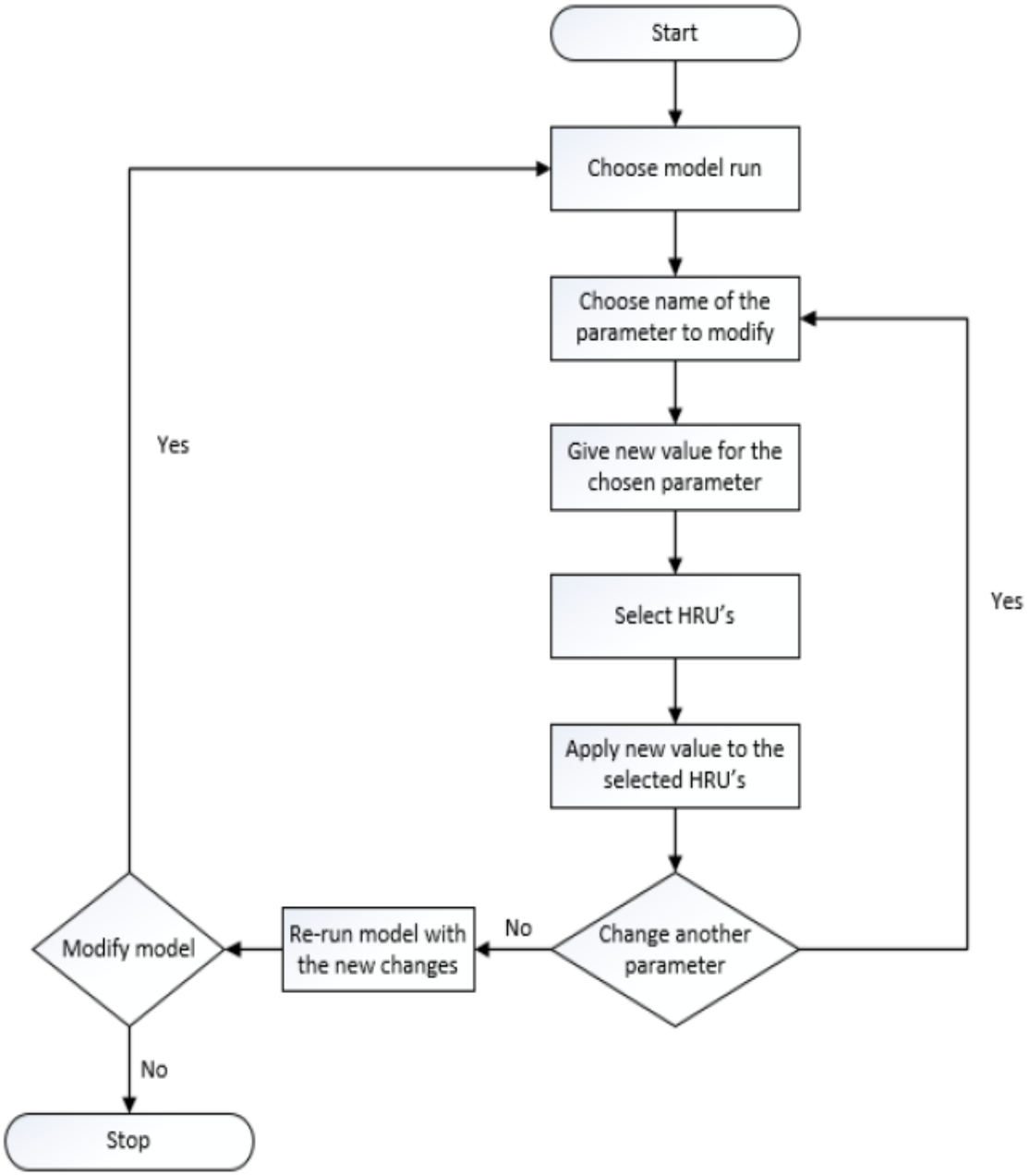

Figure 4.12: Workflow of model modification component

Figure 4.12 shows the work flow of PRMS modification component. To create a user-defined simulation scenario, first the user has to choose one of the existing model simulations that has to be modified. Then the user must determine what parameters are to be modified to get the desired model scenario. Once the modifying parameters are decided, then the user chooses the HRUs whose parameters have to be modified.

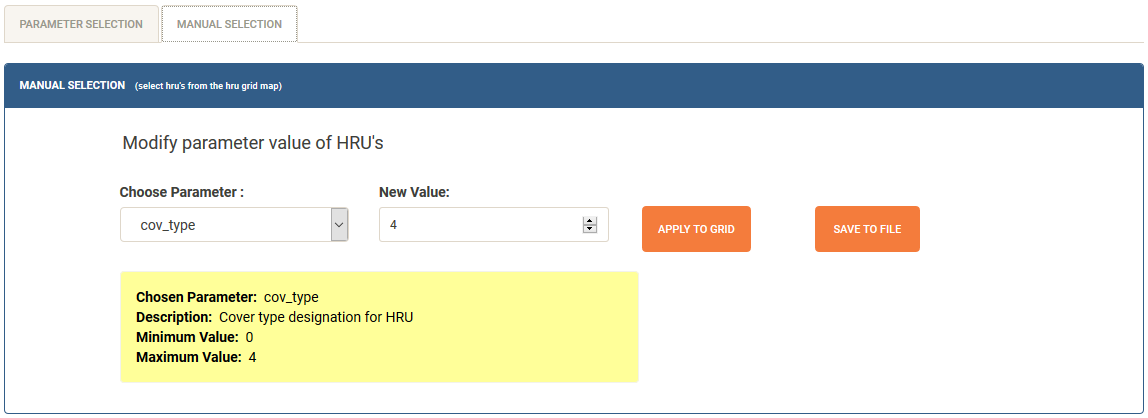

Figure 4.13: Screenshot of the model modification component in PRMS Scenarios Tool

4.2.1 HRU Selection Methods

The PRMS modification component offers two different ways to select the HRU's for parameter modification. They are 1) parameter selection and 2) manual selection. In parameter selection, the HRUs can be selected based on their parameter values, and in manual selection, the user can manually choose the desired HRUs from a 2D HRU grid map. After finishing the modfication of HRUs, the user can re-run the model with the modified inputs. Figure 4.13 shows a screenshot of the model modification component in PRMS Scenarios Tool. The modification component of PRMS scenarios tool has a tabbed interface, where the user can choose the desired HRU selection method by clicking on the corresponding tab.

4.2.1.1 Parameter Selection

Using parameter selection, the user can specify the parameter constraints for the HRUs to be filtered out from HRU set. To define a parameter constraint, the user needs to specify the name of the parameter, the operator condition (greater than, less than or between), and the parameter value. For example, Figure 4.14 displays the scenario where the user wants to change the vegetation type to trees (Type 3) for HRUs whose elevation is between 2000 and 4000 and whose vegetation type is grass (Type 1). Here, the parameter to be modified would be 'cov_type' (i.e. vegetation),

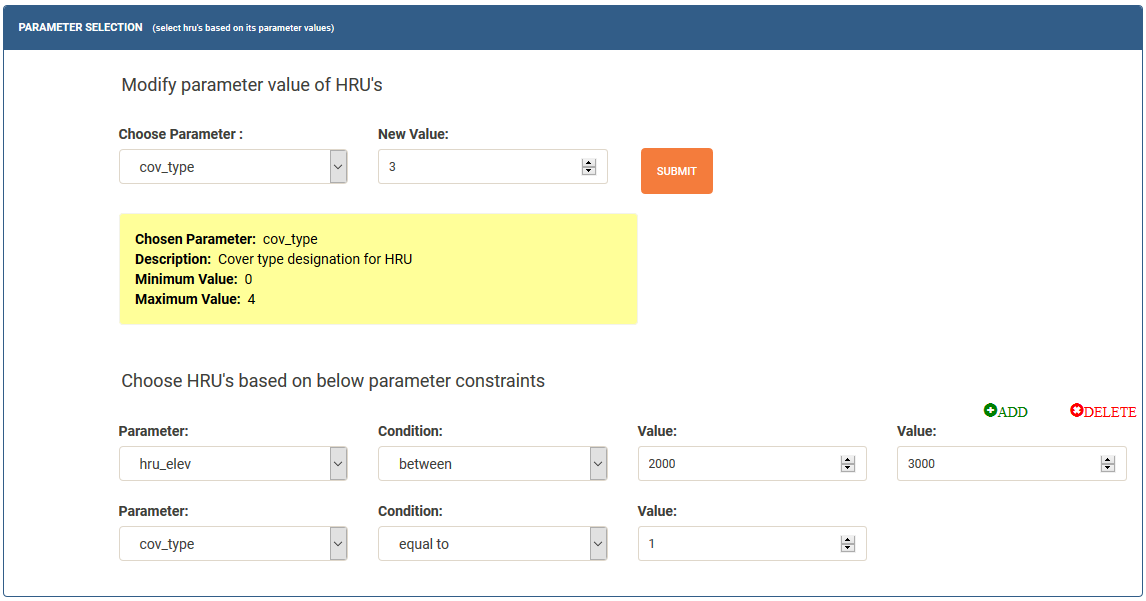

Figure 4.14: Model modification using parameter selection of the HRUs

and the modified value is '3'. To define the parameter constraint that elevation should be between 2000 and 4000, the user needs to choose the parameter name as 'hru_elev' (i.e. elevation), the operator condition as 'between', and then input the values 2000 and 4000. Multiple parameter constraints can be defined to fine-tune the selection of HRUs. 'Add' button can be used to add more parameter constraints and 'Delete' button can be used to remove an unwanted parameter constraint from the HRU selection process. Here, to define the second parameter constraint that the vegetation type should be grass, choose 'cov_type' as the parameter name, the condition should be 'equal' and the value should be given as '1'. On clicking 'Submit' button, the system would filter out the HRUs that satisfy all the given parameter constraints and then update the parameter which is to be modified with the new given value. The modifications could be visualized at real time on a 2D HRU grid map. On the HRU grid map, the color intensity of the HRU cells varies with the values of the parameter. The higher and lower values of the parameter are represented using dark and light colors respectively. Figure 4.15 shows the change in HRU grid map on performing the modifications on model parameter values.

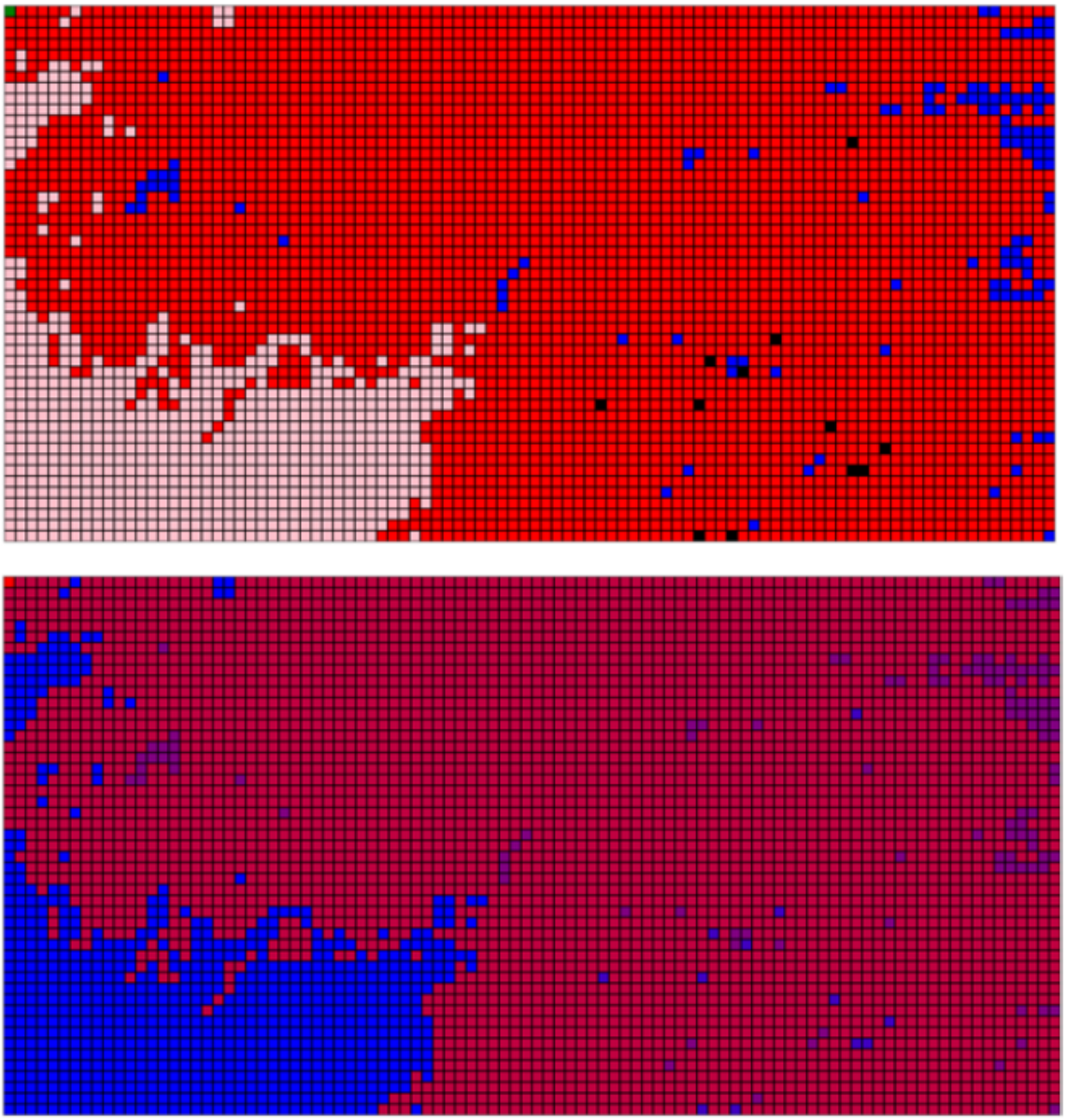

Figure 4.15: Visualization of parameter modifications on HRU grid map (Before and after the modification of HRU parameters)

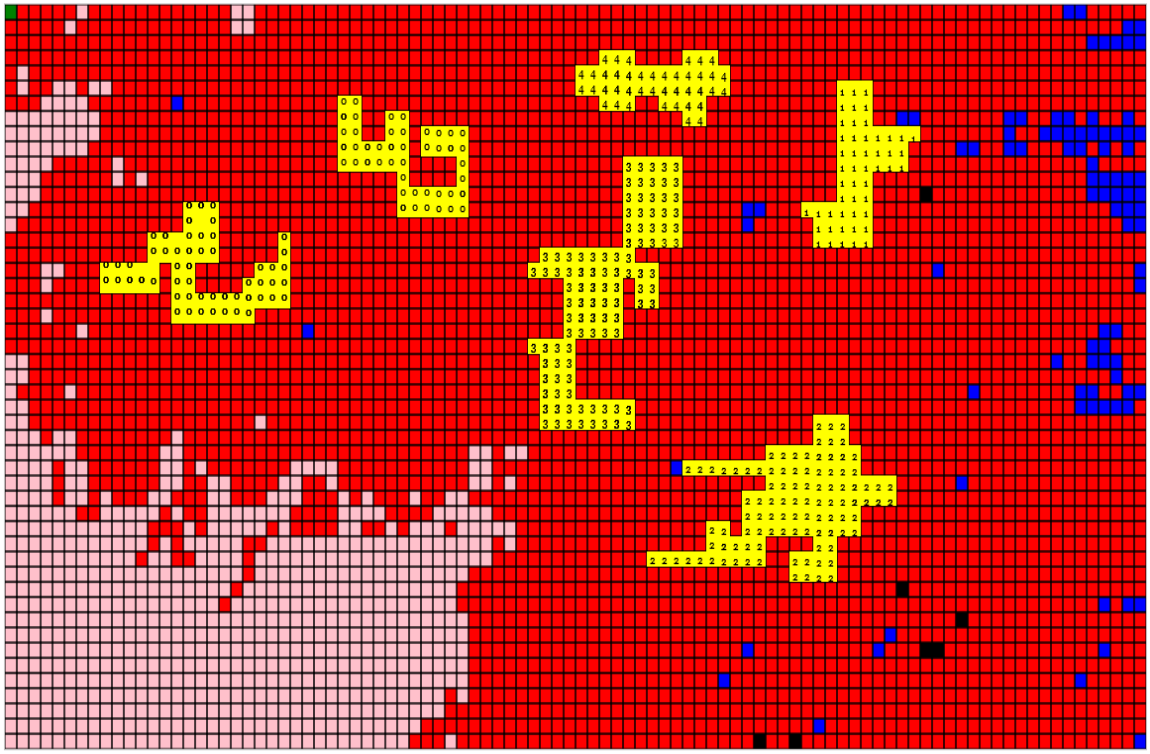

Figure 4.16: Model modification using manual selection. Modified the vegetation type of chosen HRUs to bare soil (0), grasses(1), shrubs(2), trees(3), coniferous (4)

4.2.1.2 Manual Selection

Using manual selection, the user could select the HRU cells directly on the 2D grid map through simple drag and drop mouse click operation. To select a HRU, the user places the mouse cursor over the desired HRU cell on the 2D grid map and performs a left click. To select multiple HRUs, the user left clicks on the HRU cell, drags along the desired direction, and then releases the mouse button. The chosen HRUs will be then highlighted with yellow color. On clicking 'Apply to Grid' button, the underlying HRU grid map will be updated with the new value for the selected HRUs. On clicking 'Save To File' button, the chosen parameter value of the selected HRUs would be updated with the new value in the underlying model input file. Figure 4.16 shows a screenshot of model modification using manual selection, where the user is changing the vegetation type of selected HRUs to bare soil (Type 0), grasses (Type 1), shrubs (Type 2), trees (Type 3), and coniferous (Type 4).

The model modification component of PRMS Scenarios Tool is very convenient and intuitive. It allows users to modify different parameters at the same time and avoids the unnecessary rerunning of the model. The tool also gives instant alerts while making modification to the parameters. On selecting the parameter, an alert box would be displayed with the details of the chosen parameter. The displayed details include the name of the parameter, description, and the allowed minimum/maximum value for the parameter. This alert mechanism is very helpful and effective, as it warns the user on inputting wrong value for the modifying parameter. This saves the time and effort of the researchers while performing scenario-based studies.

As described in section 2.2, PRMS divides the model area into discrete Hydrologic Response Unit (HRUs), where each HRU is composed either of land, lake, swale or inactive. The PRMS Scenarios Tool displays vegetation types of each HRU with a 2D grid map and then overlays the 2D map on a Google Map. Google Map gives the user geological information, which can be used to verify the data veracity. The user can add/remove the 2D grid map overlay and change the 2D grid map transparency by clicking on the respective buttons. Figure 4.17 shows the HRU grid map overlay on the Google Map.

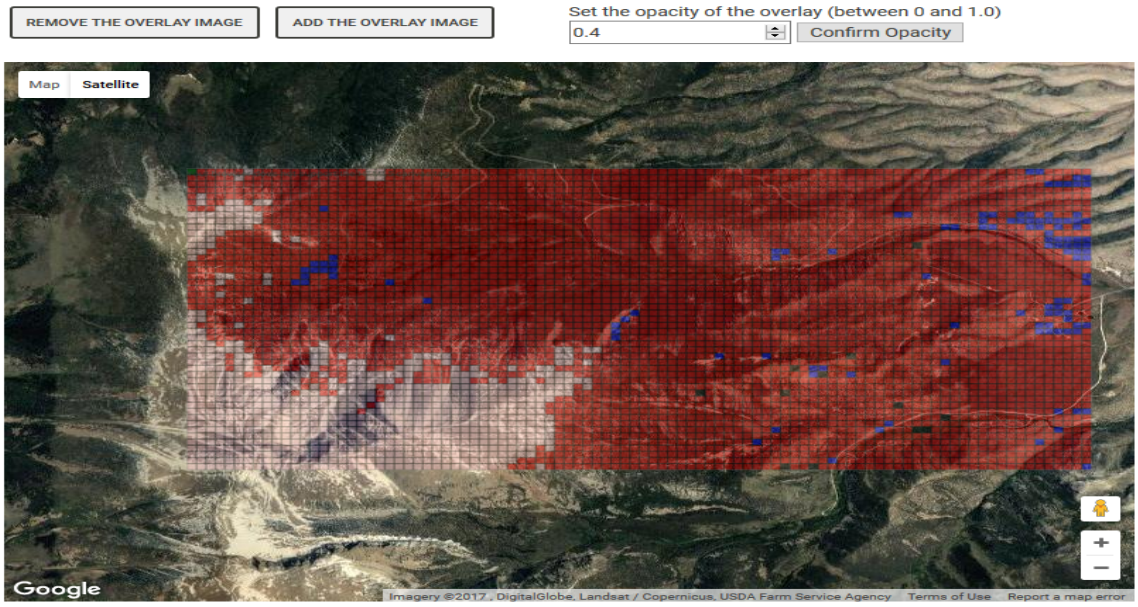

Figure 4.17: HRU grid map overlay on the Google Map

4.2.2 GSToRE Integration

Geographic Storage, Transformation and Retrieval Engine (GSToRE) [54] is a data management framework to support the storage, management, discovery and sharing of scientific and geographic data, which is similar to the data management system introduced in [20]. It was developed at Earth Data Analysis Center, University of New Mexico with a goal to offer a flexible and scalable data management platform for scientific research. GSToRE offers a REST API [8] to facilitate the storage, retrieval and removal of geospatial data and associated metadata.

4.2.2.1 Insertion and Removal of Model Runs

The PRMS Model Scenarios component developed as part of this work supports the insertion of model runs from the Virtual Watershed portal [53] to GSToRE platform. The existing model dashboard of the Virtual Watershed was modified to facilitate the GSToRE integration. In the modified dashboard, the users will be given an option to choose the model simulation files that need to be pushed to GSToRE. Figure 4.18 shows a screenshot of the model dashboard in the Virtual Watershed, before and after performing GSToRE model push. As shown in Figure 4.18, each model resource will be associated with a check box utility. The user can select or unselect the resources and click on 'GSTORE-PUSH' button to push the files to GSToRE. The check box component of all the model resources will be checked by default.

The GSToRE push operation involves two stages. The first stage is to upload the model simulation files to the GSToRE file system. The second stage involves the uploading of metadata information for each simulation file uploaded to the GSToRE server. The metadata information includes all the details of the uploaded dataset. Some of the information present in the metadata are: the name and description of the dataset; the taxonomy to which the dataset belongs (i.e. file, vector, table, or geoimage); the model set taxonomy (i.e. input, output or reference); the information about the researcher who created the dataset; the model run (UUID); and model name to which the dataset is associated. Figure 4.18 shows a screenshot of the model-run

dashboard after pushing a model run to GSToRE. Once a model run is pushed, a tick mark symbol will be shown beside the model resources that are pushed to the GSToRE. Also, a cross mark symbol will be displayed adjacent to the resources that are not pushed to the GSToRE platform. In this way, the user can keep track of the resources pushed or not pushed to the GSToRE from the modeling platform of the Virtual Watershed portal. To remove a pushed model run, the user can use the 'GSTORE-REMOVE' button. The remove functionality will delete the model run, the pushed files and its associated metadata from the GSToRE server.

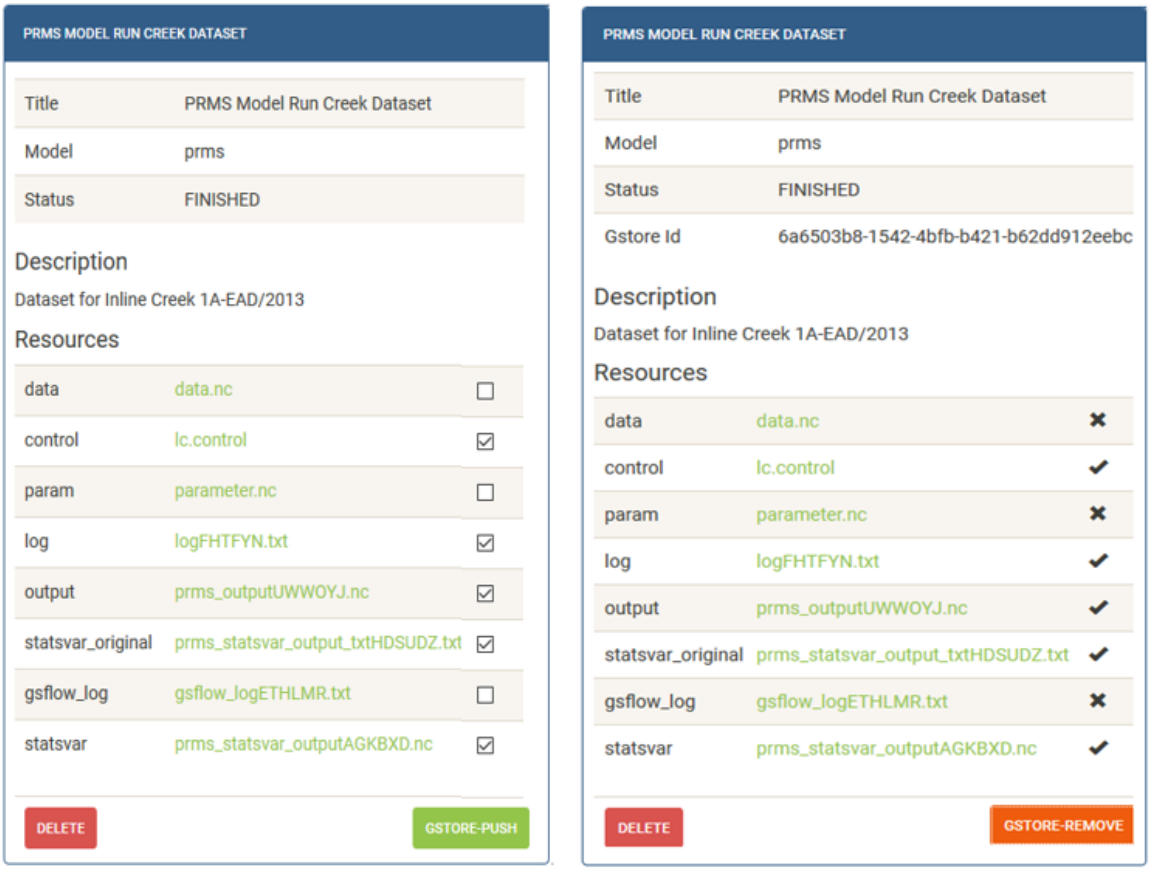

Figure 4.18: Screenshot of the model dashboard before and after GSToRE push

4.2.2.2 Search Utility

The PRMS Model Scenario Tool offers a facility to search through the vast number of GSToRE datasets. The search functionality gives several options to fine tune the search results and thereby help the user in easily finding the desired dataset in

GSToRE. The user can filter the GSToRE search results based on the model name, model run UUID, model set taxonomy, user UUID, model set, taxonomy, model set type, and type of service. The interface also provides a facility to sort the results from GSToRE in ascending or descending order. Figure 4.19 shows a screenshot of the GSToRE search functionality in Virtual Watershed. The results section displays the model run name, description, parent model UUID, model set, and taxonomy for each entry in the search results. The results section also includes a downloadable link to the dataset, which can be used to download the file directly from the GSToRE server. An API client was created in Python to facilitate the storage and retrieval of geospatial data and associated metadata using GSToRE's API service.

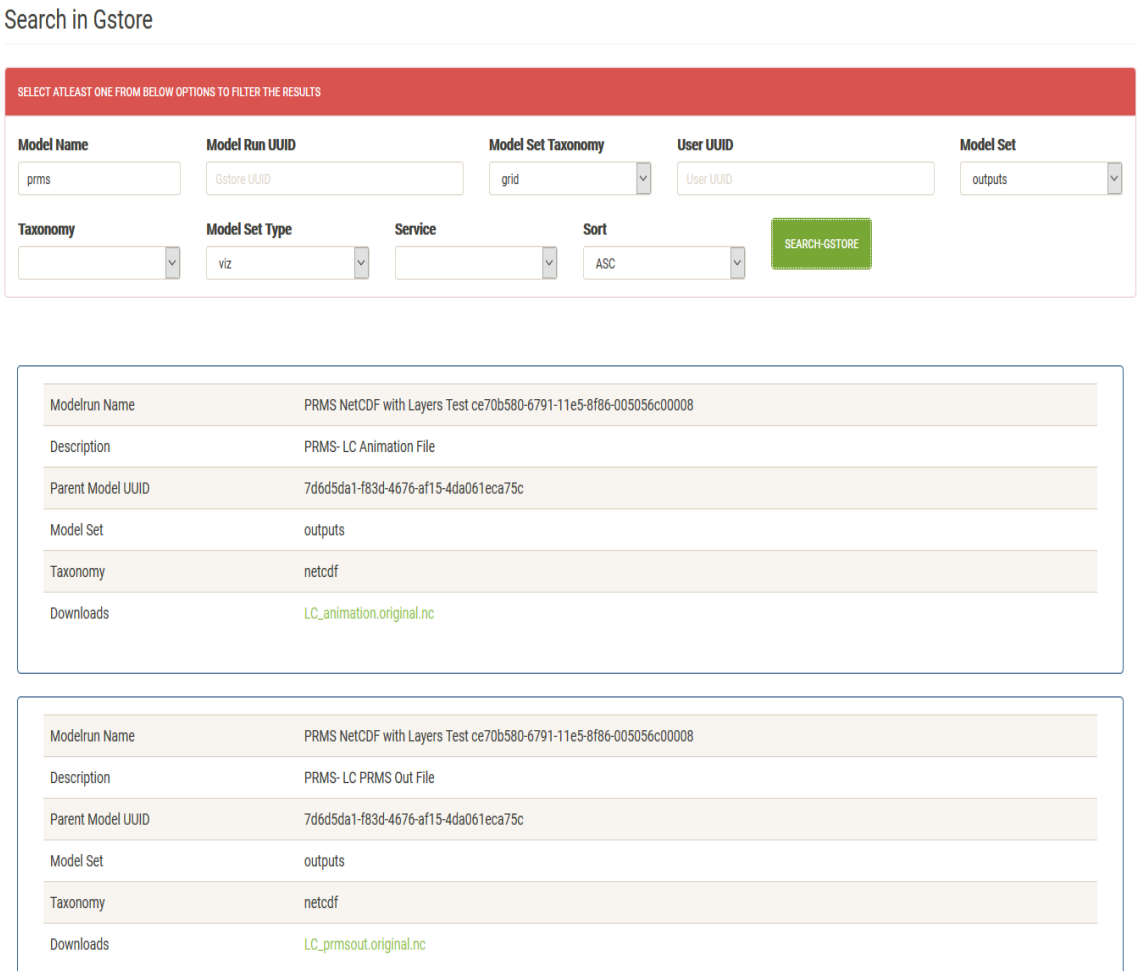

Figure 4.19: Screenshot of GSToRE search functionality in Virtual Watershed

4.3 Accuracy Enhancer For Computer Simulated Models

4.3.1 Proposed Approach

We proposed a hybrid model using machine learning algorithms to improve the accuracy of results produced from a physically based computer model. To implement the proposed approach, we used the difference between the model predictions and observed values (i.e. Δ_{error}) to further tune the actual model results. Figure 4.20 shows the structure of the proposed hybrid model. We developed a generic delta model which makes use of different machine learning models to predict the difference between the actual observation and model predictions(Δ_{error}). Once Δ_{error} is computed, the sum of model predictions and Δ_{error} will give us the final improved results.

$$\begin{aligned} \Delta_{\text{error}} &= Observed_{\text{value}} - Model_{\text{value}} \\ Observed_{\text{value}} &= Model_{\text{value}} + \Delta_{\text{error}} \end{aligned} \tag{4.5}$$

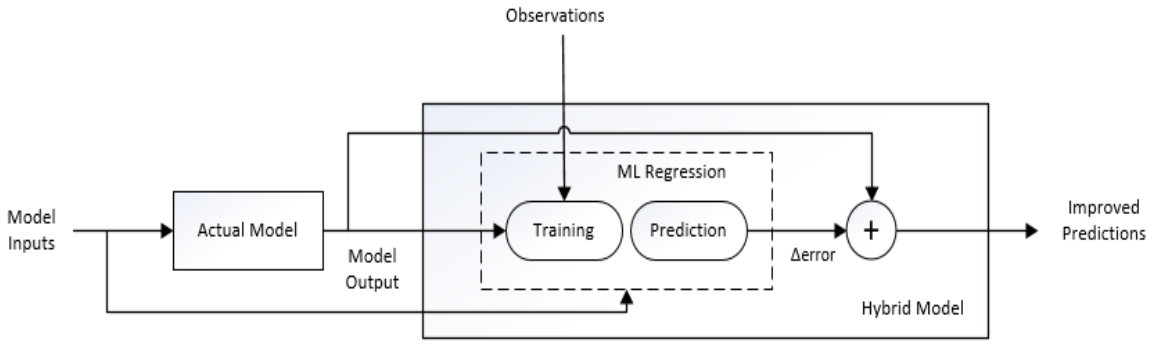

Figure 4.20: Proposed hybrid model

To predict Δ_{error}, the machine learning models have to be trained with valid, sufficiently large data sets. The model input parameters and model predictions, along with actual observed results, are used as the feature set to train the ML model. We used 70% of the data for training, 30% of the data for testing and performed the

cross-validation 30 times to conduct the tests. To implement the proposed hybrid model, four different machine learning regression models were used.

4.3.2 Machine Learning models used

Generalized linear regression, gradient boosted tree regression, decision tree regression and random forest regression are the machine learning models used in our system.

Linear regression [24] is a popular modeling technique used to estimate values for an unknown parameter. The data for the known variables (features) are used to map a linear relationship with the parameter to be estimated. Linear regression is not suited for problems which maintain a nonlinear relationship between predicted parameter and features. Generalized linear regression [33] is more accurate than linear regression, as it allows transformation between predictors and interactions. Decision tree regression [35] uses decision tree as the predictive model and is widely used in data classification research [23]. It breaks down data into smaller datasets, by incrementally developing an associated decision tree. Random forest regression [4] is similar to decision tree regression, where random forest regression uses multiple decision trees to improve the regression results. Gradient boosted tree regression [15] is another machine learning technique which follows a stage-wise fashion to build an additive prediction model using the combination of other predictive models [25]. It is a popular technique which is used by Google and Yahoo for page ranking in search engine.

4.3.3 Prototype System

A prototype system was developed to evaluate the feasibility of the proposed approach. The developed system offers four different ML techniques to train the data and make predictions based on past observations. The four ML regression techniques offered in the prototype are generalized linear regression, decision tree regression, random forest regression and gradient boosted tree regression. Apache Spark [30] was used to implement the machine learning method for the prototype system. Apache

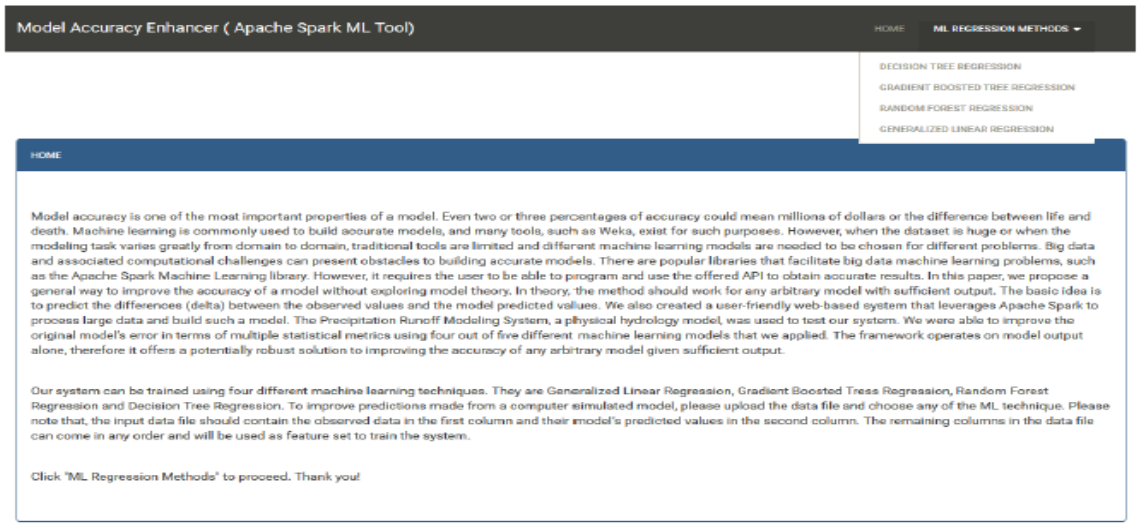

Figure 4.21: Screenshot of the home page of Accuracy Enhancer

Spark is an open source cluster computing framework which enables faster application development by providing in-memory processing for large-scale data applications. The proposed application utilizes the advanced machine learning techniques offered by Sparkś machine learning library, Mlib [1] to implement the regression functions. Mlib contains utility modules for performing various common learning algorithms including classification, regression, collaborative filtering, clustering, etc. The Mlib is customized based on the specific requirements of the proposed system to create the machine learning models that are discussed in Section 4.3.2.

The prototype system was developed as a web-based application. The backend of the prototype was built using Flask. Flask [18] is a popular light-weight micro web framework used for building web applications with Python. The system manages datasets with a file system and the MongoDB. The data are stored in the file system; the file index and the location of the file in the file system are stored in MongoDB. The front-end of the application was created using jQuery, HTML and Bootstrap. The results are visualized using Google Chart [63] library. The user can use any client-side device to send requests to the server side and view the results.

Figure 4.21 shows the screenshot of the home page in the proposed web application. On clicking the 'ML Regression Methods' tab on the header section, the user

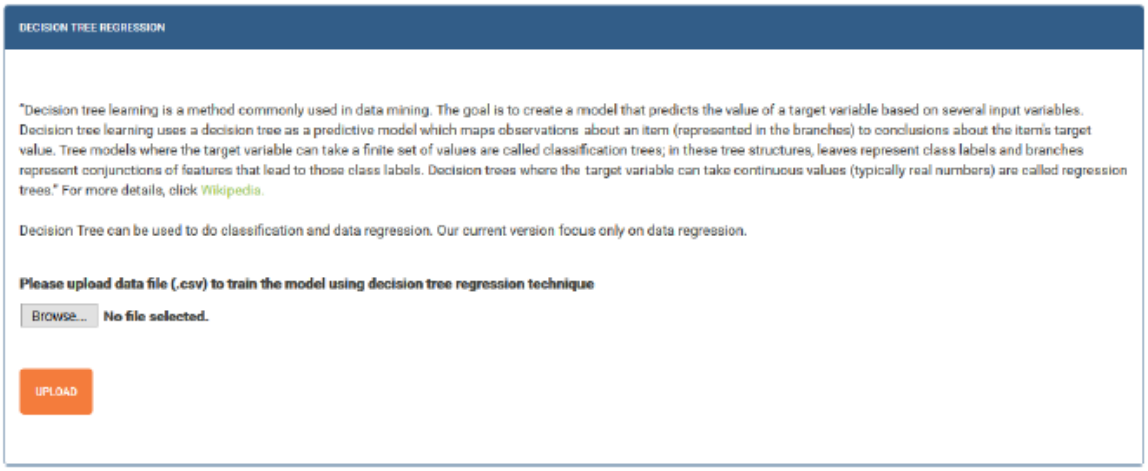

Figure 4.22: Screenshot of atset upload page in the web application

will be shown all the available ML regression techniques. Currently, our application supports four regression techniques for training the model. After choosing the regression technique, the user will be asked to upload the data file or choose an available data set stored on the server side.

Figure 4.22 shows the screenshot of the upload page in the prototype system. The uploaded data file will be used to train the model for making predictions. The data file should contain the observed data in the first column and their model predicted values in the second column. The other columns in the data file would be used as the feature set for training the ML models and could come in any order in the data file.

Once the data file is uploaded, the user can view the results by clicking 'Get Error Report and Line Chart' button. The results section includes the root mean square error (RMSE), percent bias (PBIAS), coefficient of determination (CD), and Nash-Sutcliffe efficiency (NSE) values of the final model results. Along with the accuracy statistics, a line chart depicting the differences in the prediction and observed values will also be shown. The user can understand the variation in the observed, predicted, and improved values by hovering over the points on the graph. Figure 4.23 shows the screenshot of the result page of the proposed web application. The system enables the user to modify the machine learning model with different methods. For example, the user can split the chosen dataset into training data and test data with different

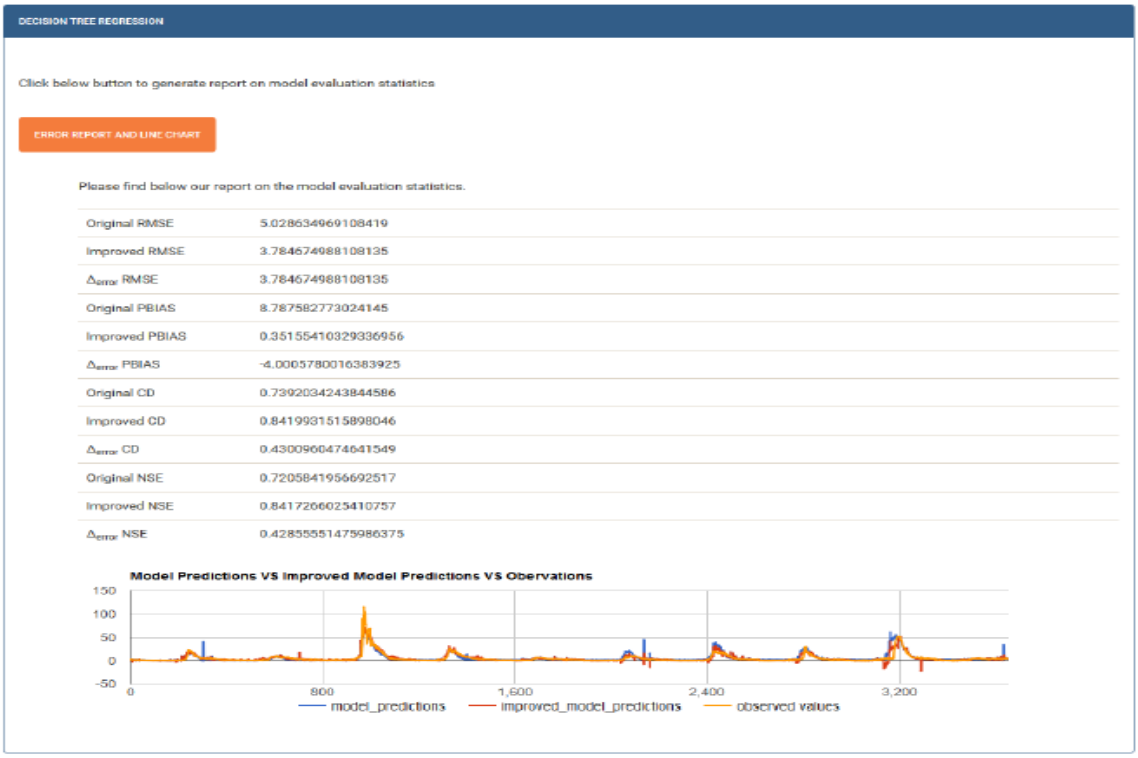

Figure 4.23: Screenshot of the home page of Accuracy Enhancer

ratios. The user can also set simple rules, such as the predicted values must be greater than zero, so the system will replace the negative values into zeros.

There are two reasons that our system does not build machine learning models directly on the raw input data: First, it is complex to build a machine learning model on a certain problem. Noise reduction and raw data QA/QC are basic first steps of modeling for fields such as hydrology. These steps are hard to implement in a general use system; Second, it is our opinion that the model should be treated as a black box if the user wants to improve the model accuracy in a general way. However, every model has its own applicable uses and using a model simply for its accuracy improvement is not always the goal of scientific modeling. For example, a scientist may wish to understand interactive processes within a physical model as opposed to simply using it to make accurate predictions. In this case, our framework would not be applicable.

Chapter 5

Evaluation

This chapter discusses the results of the evaluation of the components developed as part of this thesis work. Section 5.1 describes the results of the evaluation of the elastic hybrid server explained in Section 4.1. Section 5.2 talks about the evaluation of the proposed generic machine learning approach described in Section 4.3.

5.1 Self-managed Elastic Hybrid Server System

The performance of the proposed approach was evaluated by simulating a Poisson job arrival stream on the job queue. A PRMS model simulation with one month climate data of Lehman Creek constitutes a job for the system. In the experiment, the worker took an average of 34 seconds to process one model simulation job. In the experimental study, an approximate job execution time was initially collected from the user and later it was replaced with the average job execution time of the jobs after the server starts the processing. The experimental study was conducted with four machines with Intel i7 CPU, 16 GB DDR4 RAM, and 256 GB SSD. Since the experiments were conducted with real machines instead of machine instances from cloud providers, we used a dummy price as the cost of a machine instance. The cost of AWS instances available at [44] was used to represent the price of rented instances in our prototype system.

In the experiment, the system was allocated with a budget amount of $1.63 for a budget period of 20 minutes and a rented machine cost $4.256/hour (current cost

for a high end compute node on AWS). With the allocated budget and given instance cost, the proposed system can rent a maximum of 40 jobs. To promise a uniform distribution of rented jobs across the given budget time period, the system should rent a job every 30 seconds, provided the owned workers are not available to process the job at T_{int}. We compared our proposed approach with FIFO [23] approach. In FIFO, whenever a new job comes in, the system will immediately rent the job if the owned servers are not available. Due to this fashion of job processing, the FIFO approach could not ensure the availability of rented workers until the end of budget duration and may utilize all the rented time well before the end of the budget period. As a result, once the rented jobs are over, the later jobs have to wait longer in the queue, causing a steep increase in the waiting time. This would result in customer dissatisfaction and could eventually lead to users isolating the platform.

Figure 5.1 shows the comparison of the waiting time of jobs in FIFO approach and our approach. As shown in the figure, using our approach, the waiting time of the jobs are maintained in a controlled level throughout the entire budget time, whereas in FIFO approach, the waiting time became drastic once the rented jobs were finished. Figure 5.2 shows the comparison of number of jobs waiting in the queue between the proposed approach and FIFO. As seen in the figure, the available rented time was completely utilized around the 13^{th} minute, and therefore the queue length shows a steep increase thereafter. In the proposed approach, since the available rented time was utilized judiciously, the queue length was maintained at a controlled level through out the budget period.

The experiment was repeated by giving different sets of model inputs with different combinations of budget amount, instance rate, and job arrival rate. The waiting time of a job was calculated as the time taken by the worker to start the job processing, once the job was placed into the queue. While observing the Figure 5.1, we could see several fluctuations in the waiting time of the jobs in the proposed approach compared to FIFO. This is because, in the proposed approach, the system maintains a time interval T_{int}, only at which the availability of the owned servers are inspected

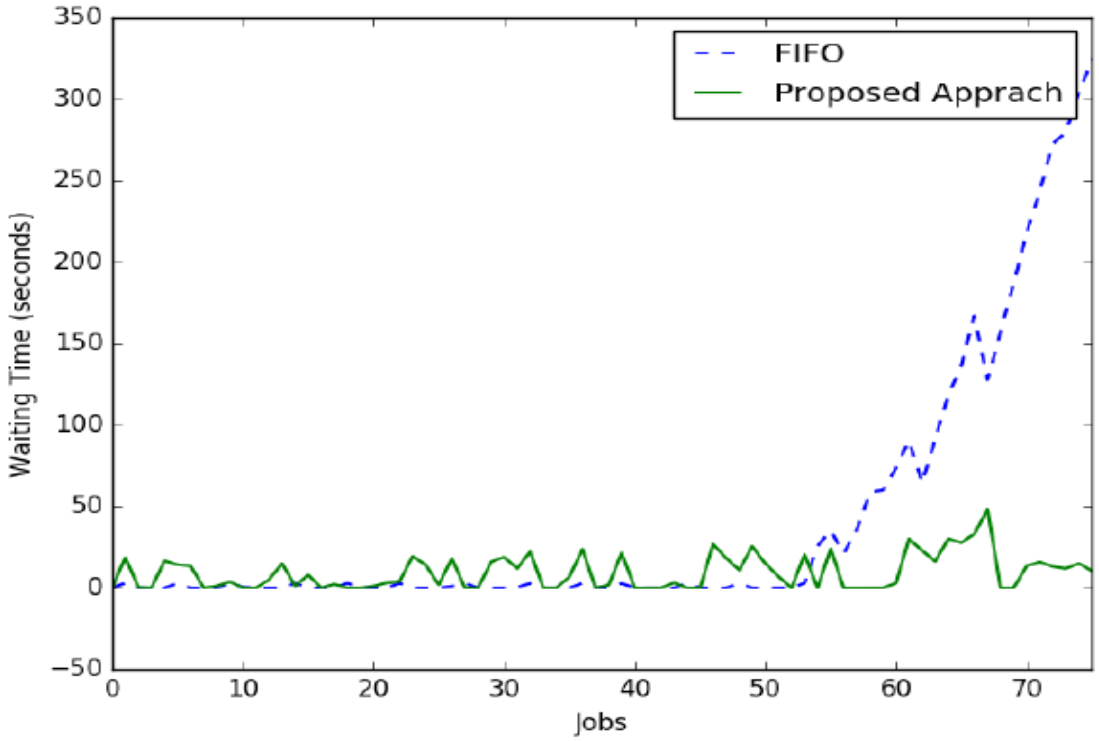

Figure 5.1: Comparison of waiting time of jobs in FIFO and proposed approach

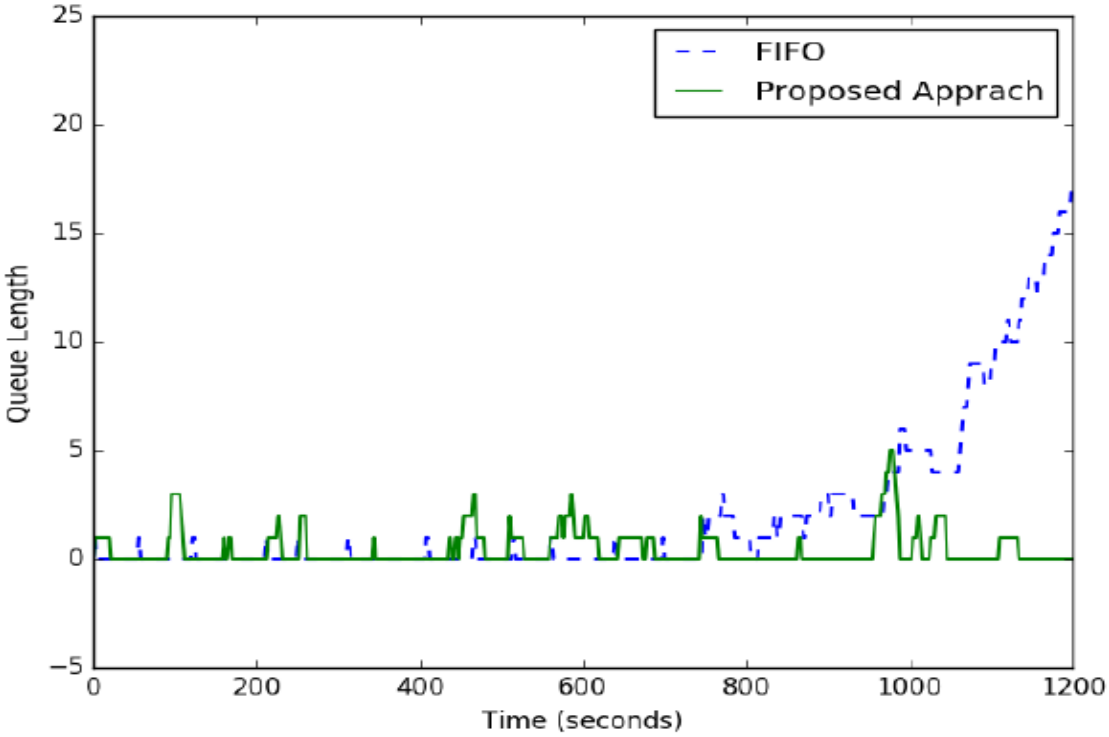

Figure 5.2: Comparison of job-queue length in FIFO and proposed approach

and the jobs are grabbed by the rented workers. Also, the rented worker containers were created from scratch using a base image stored at the Docker Hub [52]. The creation of worker container takes several milliseconds to seconds, depending on the resource utilization on the host machine. After the creation of the container, an additional few more seconds will be taken to establish connection with the configured job queue and pick a job from it. Different thread processes were used to handle the simulation of job arrival stream, creation of rented workers, monitoring of queue length and job status. The time slicing between the thread processes could also be a reason for observing fluctuations in the observed queue length and waiting time.

Apart from maintaining queue length and waiting time at a controlled level, the proposed approach offers other benefits also. The experimental results showed that the proposed approach ensured better resource utilization. During the experiment, the FIFO approach ran out of rented workers in 747.1 seconds. During this time frame, the system had finished 54 jobs, out of which 14 of them were processed by owned workers. Therefore, 25.93% was the utilization rate of owned workers with FIFO. On the other hand, during the same time frame (747.1 seconds), the proposed system finished 40 jobs, out of which 21 jobs were processed by owned servers, resulting in a utilization rate of 52.5% for owned servers. It is evident from the results that the proposed method ensured a better utilization rate for the owned workers and saved more rented workers for later use.

Based on Equation 4.3 and Equation 4.4, the expected queue length (number of job arrivals in the queue) was 1.46 and the real queue length was 0.622. Theoretically, each job needed to wait 0.39 minute and in fact each job waited 0.34 minute on average. This shows that the proposed method worked well in this job queue case. The experiment was repeated by giving different sets of model inputs with different combinations of budget amount, instance rate, and job arrival rate.

The time consumption for starting and stopping a rented instance varies with the work load and the cloud hosting service. Normally, the starting time of an instance ranges between 30 seconds to 6 minutes. Since our goal was to prove the applicability

of the proposed approach, the experiment was conducted with comparatively shorter jobs and hence the time interval T_{int} value was also relatively small (less than a minute). However in real world scenarios, while dealing with high time consuming jobs, the estimated T_{int} value would be sufficiently large to accommodate the varying VM start time.

5.2 Accuracy Enhancer For Computer Simulated Models

The proposed approach was evaluated on two different computer simulated models. Precipitation-Runoff Modeling System (PRMS) model described in Section 2.2 and the genetic algorithm tuned nitrate prediction model described in [60] were used to evaluate our proposed approach. To perform the statistical evaluation on model accuracy, we used the following quantitative statistics.

5.2.1 Quantitative Statistics Used:

Root Mean Square Error (RMSE): RMSE measures how close the observed data points are to the predicted values of the model, while retaining the original units of the models output and observed data. Lower values of RMSE indicate a better fit of the model. Since the main purpose of our model is to predict, RMSE is one of the important standards that defines how accurately the model predicts the response and it is commonly used in the machine learning fields.

Percent Bias (PBIAS): PBIAS is a measure to check the behavior of the predictions made by the model simulations. It determines whether the predictions are underestimated or overestimated to the actual observations. If the PBIAS values are positive, the model overestimates the results, whereas if the values are negative, the model underestimates the results by the given percentage. Therefore, values closer to zero are preferred for PBIAS.

Nash-Sutcliffe efficiency (NSE): NSE is a normalized statistic which is used to determine the efficiency of the model. NSE values show the models ability to make predictions that fit 1:1 line with the observed values. The values for NSE range between -∞ and 1.0. To consider a model has acceptable levels of performance, the values of NSE should lie close to 1.0, and the higher NSE indicates the better results.

Coefficient of determination (CD): CD stands for coefficient of determination, calculated as the square of the correlation between the observed values and the simulated values. The values for CD range between 0.0 and 1.0 and correspond to the amount of variation in the simulated values (around its mean) that is explained by the observed data. Values closer to 1.0 indicate a tighter fit of the regression line with the simulated data. Similar to NSE, the higher CD values indicates the better results. Therefore, the improved models are better than the original based on the higher CD values.

The statistical parameters are defined by the following:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - A_i)^2} \tag{5.1}$$

$$PBIAS = \frac{\sum_{i=1}^{N}(A_i - P_i)100}{\sum_{i=1}^{N}A_i} \tag{5.2}$$

$$NSE = 1 - \frac{\sum_{i=1}^{N}(A_i - P_i)^2}{\sum_{i=1}^{N}(A_i - \bar{A})^2} \tag{5.3}$$

$$CD = \left\{ \frac{\sum_{i=1}^{N}(A_i - P_i)(P_i - \bar{P})}{\left(\sum_{i=1}^{N}(A_i - \bar{A})^2\right)^{\frac{1}{2}}\left(\sum_{i=1}^{N}(P_i - \bar{P})^2\right)^{\frac{1}{2}}} \right\}^2 \tag{5.4}$$

where P_i and A_i represent the simulated and observed values respectively, and $\bar{A}$ is the mean of the observed values and $\bar{P}$ is the mean of simulated values for the entire evaluation period.

5.2.2 Evaluation on PRMS Model

Table 5.1 shows the results of the evaluation of proposed approach on PRMS model. The table contains the results of the four quantitative statistical parameters evaluated. Each quantitative statistic was evaluated with three different scenarios: 1) between model predictions and actual observations, 2) between improved model predictions and actual observations, and 3) between predicted Δ_{error} and actual Δ_{error}, where Δ_{error} is the difference between the model predictions and actual observations.

Table 5.1: Results of evaluation on PRMS model

Statistic	**Regression Techniques**			
	Decision Tree Regression	**Gradient Boosted Tree Regression**	**Random Forest Regression**	**Generalized Linear Regression**
Original RMSE	5.028	5.028	5.028	5.028
Improved RMSE	3.849	3.154	3.666	4.808
Delta RMSE	3.849	3.154	3.666	4.808
Original PBIAS	8.787	8.787	8.787	8.787
Improved PBIAS	-0.571	0.569	-0.581	7.466
Delta PBIAS	6.500	-6.477	6.620	-8.482
Original CD	0.739	0.739	0.739	0.739
Improved CD	0.838	0.892	0.852	0.759
Delta CD	0.409	0.604	0.495	0.077
Original NSE	0.720	0.720	0.720	0.720
Improved NSE	0.836	0.890	0.851	0.744
Delta NSE	0.408	0.603	0.463	0.077

It is evident from Table 5.1 that the proposed approach has improved the accuracy of the predictions from PRMS model by minimizing the extent of the prediction error. For the evaluation, the PRMS simulation dataset for Inline Creek Watershed was used. The original RMSE value of the watershed data set was 5.028. All four ML

regression techniques succeeded in reducing the RMSE value. Our approach using gradient boosted tree regression resulted in the best RMSE value of 3.154. The proposed approach improved the CD value for all the four regression methods used. The best CD value obtained was 0.892 on applying the proposed approach with gradient boosted tree regression. As seen in the results, all the four regression methods gave better PBIAS and NSE values compared to the original PBIAS and NSE of the watershed dataset. Figure 5.3 shows the comparison between the actual observations, model predictions and improved model predictions.

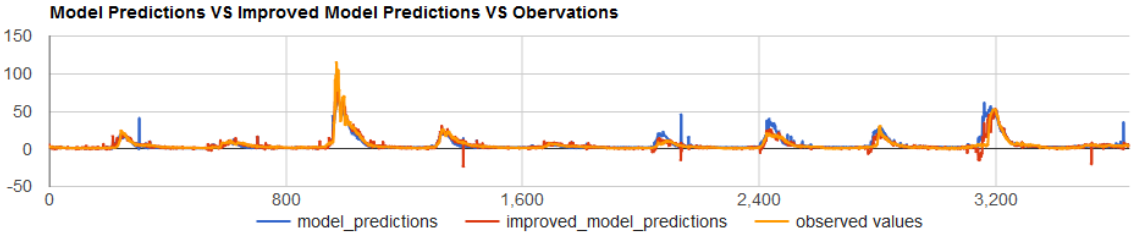

Figure 5.3: Comparison between the actual observations, model predictions and improved model predictions with PRMS model

5.2.3 Evaluation on Nitrate Prediction Model

Figure 5.4 shows the comparison between the actual observations, model predictions and improved model predictions.

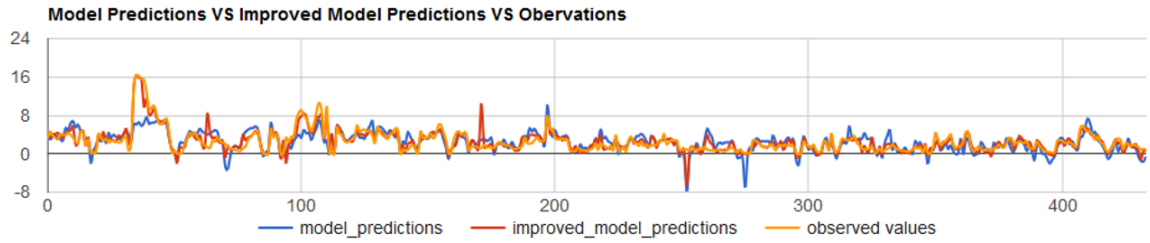

Figure 5.4: Comparison between the actual observations, model predictions and improved model predictions with nitrate prediction model

Table 5.2 shows the results of the evaluation of the proposed approach on nitrate prediction model. The proposed approach resulted in improving the prediction

accuracy of the model by minimizing the prediction errors. All four machine learning regression techniques were successful in improving the RMSE, PBIAS, CD and NSE statistics of the nitrate prediction model. Among the four regression techniques, Gradient Boosted Tree Regression gave the best RMSE, PBIAS, CD and NSE value.

Table 5.2: Results of the evaluation on nitrate prediction model

Statistic	**Regression Techniques**			
	Decision Tree Regression	**Gradient Boosted Tree Regression**	**Random Forest Regression**	**Generalized Linear Regression**
Original RMSE	1.8966	1.8966	1.8966	1.8966
Improved RMSE	1.5183	1.1004	1.26321	1.7832
Delta RMSE	1.5183	1.1004	1.26321	1.7832
Original PBIAS	-3.7017	-3.7017	-3.7017	-3.7017
Improved PBIAS	-2.7454	1.1585	3.3113	7.1756E-13
Delta PBIAS	-74.1668	31.2969	89.4546	1.9306E-11
Original CD	0.4158	0.4158	0.4158	0.4158
Improved CD	0.6201	0.7933	0.7230	0.4517
Delta CD	0.3677	0.6628	0.6138	0.1197
Original NSE	0.3660	0.3660	0.3660	0.3660
Improved NSE	0.5937	0.7866	0.7192	0.4395
Delta NSE	0.3575	0.6625	0.5561	0.1138

Overall, by only concerning accuracy, Table 5.1 and Table 5.2 suggest that the framework we present (build machine learning models on model error) does indeed improve the model accuracy in a robust manner. The implemented prototype could support linear regression along with the other regression techniques. However, the results of linear regression are excluded from the shown results. For PRMS model evaluation, the delta RMSE was 112 with linear regression. By our guess, this was due to systematic model errors such as auto-correlation in the model delta that a linear model could not effectively model. This does not mean that the linear regression is meaningless. Some model outputs follow linear relations. Also, the linear regression is fast and easy to understand.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we introduced three components to enhance the modeling capabilities of the NSF EPSCoR-supported Western Consortium for Water Analysis, Visualization and Exploration (WC-WAVE) Virtual Watershed Project. The developed components would help the watershed researchers significantly in their model processing and research activities.

The major contribution of this work are, a web-based tool to conduct scenario-based studies with watershed models, a proposed server-usage optimization strategy to enable cost-effective deployment of model containers, and a web tool to minimize the prediction errors of computer-simulated models using a generic machine learning approach. The prototype facilitates on-demand provisioning of computing resources based on job arrivals and ensures reduced waiting time for the modeling jobs within the allocated budget amount. The model-scenarios component in the application could help hydrologists in simulating user-defined model scenarios using Precipitation Runoff Management System (PRMS) model. The tool also facilitates the download of watershed datasets available in the Geographic Storage, Transformation and Retrieval Engine (GSToRE) and enables the insertion of model simulations to GSToRE. The developed model accuracy component uses a generic machine learning approach to process the predictions from computer simulated models and helps improve the accuracy of the model by minimizing the prediction errors. The tool al-

lows users to upload model datasets and enables the fine tuning of the prediction data using a generic approach with the help of four different machine learning regression techniques.

The three components combined contain approximately 9000 lines of code. The codes are published through the GitHub repository [36, 37, 38].

6.2 Future Work

The elastic server component developed as part of this thesis was evaluated by simulating job arrivals on physical machines. However, the elastic server components used in the prototype were designed with a micro-service architecture to leverage the potential of cloud computing. The work can be further extended and evaluated by implementing the elastic server component on actual instances from cloud host providers. Also, in the current work, the proposed approach was evaluated with comparatively small tasks due to time and resource limitations. The feasibility of the approach can be further evaluated in a cloud environment with high time consuming real world jobs.

A detailed user study can be conducted to evaluate the usability of the developed elastic server system. The study would be carried out with two types of users. 1) the manager/admin and 2) the researchers/modelers. The manager allocates the budget amount and takes decision on the cost of the rented instances. The modelers run the model simulations in the Virtual Watershed platform after uploading the required model inputs. The user study would involve two stages. In the first stage, the user study can be carried out without allocating any budget amount to the system. The modelers should be asked to place their model simulation jobs, and then collect their opinion about the performance of the modeling platform using the feedback survey form. The manager would verify the waiting time of the placed jobs and the overall feedback score of the system during the evaluation. In the second stage of the study, the manager allocates a budget amount to the system. The feedback should be collected from the modelers after finishing their model simulation jobs. The manager

can verify how effectively the budget amount is utilized and the time interval at which the jobs are rented. After finishing the jobs, a comparison can be done between the expected waiting time, queue length and the observed waiting time, queue length. The tests must be repeated after increasing/decreasing the allocated budget amount and modifying the cost of the rented instance.

The model accuracy enhancer component was evaluated with two different computer simulated models in the current work. The component uses a generic machine learning based approach to improve the accuracy of the model predictions. We believe that our approach will work for various types of models and will not be limited to just the two models (described in Section 5.2). However, it is important to keep in mind that improvements in model accuracy do not reflect improvements in the functional (or in this case, physical) representations of the underlying model. Therefore, the domain expert who uses our system will need to understand the implications and limitations of any improved results in a case-by-case manner. For example, in our hydrologic application, it may be useful to simulate stream-flow with increased accuracy for water resources management and prediction. However, to understand which hydrologic processes (what climatic conditions) are responsible for the model's prediction error, we would likely need to analyze the inputs and outputs of the original model as well as other components of the model that might relate to the question at hand. Also, the efficiency of the proposed approach can be further analyzed and extended by comparing it with Temporal Differences algorithm described in [47].

The model scenarios component developed in this work supports the modification of model simulations for PRMS model only. The component can be further extended to support model scenario creation for other environmental models like ISNOBAL [28], MODFLOW [19], etc. We also want to improve the data visualization of this component by incorporating the works introduced in [56, 57, 58, 59] and provide more toolsets described in [40, 55].

Bibliography

[1] Apache Spark 2.1.0. Mllib: rdd-based api. URL: `https://spark.apache.org/docs/2.1.0/mllib-guide.html`. Accessed on 31 May 2017.

[2] Microsoft Azure. Azure container service. URL: `https://azure.microsoft.com/en-us/services/container-service/`. Accessed on 18 May 2017.

[3] Jing Bi, Zhiliang Zhu, Ruixiong Tian, and Qingbo Wang. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In *Cloud Computing (CLOUD), 2010 IEEE 3rd international conference on*, pages 370–377. IEEE, 2010.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Rodrigo N. Calheiros, Rajiv Ranjan, and Rajkumar Buyya. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *Proceedings of the 2011 International Conference on Parallel Processing*, ICPP '11, pages 295–304, Washington, DC, USA. IEEE Computer Society, 2011.

[6] Josiah L Carlson. *Redis in Action*. Manning Publications Co., 2013.

[7] Chase D. Carthen, Thomas J. Rushton, Christine M. Johnson, Aaron Hesson, Daniel Nielson, Bryan Worrell, Donna M Delparte, W. Joel Johansen, John W. Anderson, Roger Lew, Nicholas R. Wood, Matthew Ziegler, Sergiu M. Dascalu, and Frederick C. Harris Jr. Design of a virtual watershed client for the wc-wave project. In *2015 International Conference on Collaboration Technologies and Systems (CTS)*, pages 90–96, 2015.

[8] Earth Data Analysis Center. Gstore stable api. URL: `http://vwp-dev.unm.edu/docs/stable.html`. Accessed on 1 June 2017.

[9] Prem C. Consul and Gaurav C. Jain. A generalization of the poisson distribution. *Technometrics*, 15(4):791–799, 1973.

[10] Sergiu M. Dascalu. Scientific collaboration in virtual environments: the western consortium watershed analysis, visualization, and exploration (wc-wave) project. In *2014 International Conference on Collaboration Technologies and Systems (CTS)*, pages 560–561, 2014. DOI: `10.1109/CTS.2014.6867625`.

[11] Docker. Docker sdk for python. URL: `https://docker-py.readthedocs.io/`. Accessed on 18 May 2017.

[12] Docker. Overlay networking and swarm mode. URL: https://docs.docker.com/engine/userguide/networking/get-started-overlay/. Accessed on 18 May 2017.

[13] David A Dunnette and Antonius Laenen. *River quality: dynamics and restoration*. CRC Press, 1997.

[14] NSF EPSCoR. Experimental program to stimulate competitive research. URL: https://www.nsf.gov/od/oia/programs/epscor/. Accessed on 1 June 2017.

[15] Jerome H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*:1189–1232, 2001.

[16] Google. Google container engine. URL: https://cloud.google.com/container-engine/. Accessed on 18 May 2017.

[17] Google. Google container engine pricing and quotas. URL: https://cloud.google.com/container-engine/pricing. Accessed on 18 May 2017.

[18] Miguel Grinberg. *Flask web development: developing web applications with python*. O'Reilly Media, Inc., 2014.

[19] Arlen W. Harbaugh, Edward R. Banta, Mary C. Hill, and Michael G. McDonald. Modflow-2000, the u. s. geological survey modular ground-water model-user guide to modularization concepts and the ground-water flow process. *Open-file Report. U. S. Geological Survey*, (92):134, 2000.

[20] Moinul Hossain, Hannah Munoz, Rui Wu, Eric Fritzinger, Sergiu M. Dascalu, and Frederick C. Harris Jr. Becoming dataone tier-4 member node:steps taken by the nevada research data center. In *Electrical Electronic Equipment Aegean Conference on Electrical Machines Power Electronics (OPTIM 2017)*. IEEE, 2017.

[21] Moinul Hossain, Rui Wu, Jose T. Painumkal, Mohamed Kettouch, Cristina Luca, Sergiu M. Dascalu, and Frederick C. Harris Jr. Web-service framework for environmental models. In *2017 Internet Technologies and Applications (ITA)*.

[22] Samuel Karlin and James McGregor. Many server queueing processes with poisson input and exponential service times. *Pacific J. Math*, 8(1):87–118, 1958.

[23] Robert L. Kruse, Bruce P. Leung, and Clovis L. Tondo. *Data structures and program design in C*. Pearson Education India, 2007.

[24] Michael H. Kutner, Chris Nachtsheim, and John Neter. *Applied linear regression models*. McGraw-Hill/Irwin, 2004.

[25] Jacob H. LaFontaine, Lauren E. Hay, Roland J. Viger, R. Steve Regan, and Steven L. Markstrom. Effects of climate and land cover on hydrology in the southeastern u.s.: potential impacts on watershed planning. *JAWRA Journal of the American Water Resources Association*, 51(5):1235–1261, 2015. ISSN: 1752-1688. DOI: 10.1111/1752-1688.12304. URL: http://dx.doi.org/10.1111/1752-1688.12304.

[26] George H. Leavesley, R.W. Lichty, B.M. Thoutman, and L.G. Saindon. *Precipitation runoff modeling system: User's manual*. USGS Washington, DC, 1983.

[27] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[28] Danny Marks, James Domingo, Dave Susong, Tim Link, and David Garen. A spatially distributed energy balance snowmelt model for application in mountain basins. *Hydrological Processes*, 13.

[29] Peter Mell and Tim Grance. Pay-per-use, 2011.

[30] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, and Sean Owen. Mllib: machine learning in apache spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[31] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[32] Microsoft. Azure pricing. URL: `https://azure.microsoft.com/en-us/pricing/`. Accessed on 18 May 2017.

[33] John A. Nelder and R. Jacob Baker. Generalized linear models. *Encyclopedia of statistical sciences*, 1972.

[34] Sam Newman. *Building microservices*. O'Reilly Media, Inc., 2015.

[35] Cristina Olaru and Louis Wehenkel. A complete fuzzy decision tree technique. *Fuzzy sets and systems*, 138(2):221–254, 2003.

[36] Jose T. Painumkal. Model accuracy enhancer. URL: `https://github.com/josepainumkal/Model-Accuracy-Enhancer`. Accessed on 13 August 2017.

[37] Jose T. Painumkal. Prms model scenario tool. URL: `https://github.com/josepainumkal/prms-vegetation-scenarios`. Accessed on 13 August 2017.

[38] Jose T. Painumkal. Self-managed Elastic Hybrid Server. URL: `https://github.com/josepainumkal/Elastic-Hybrid-Server`. Accessed on 13 August 2017.

[39] Jose T. Painumkal, Rui Wu, Sergiu M. Dascalu, and Frederick C. Harris Jr. Self-managed elastic scale hybrid server using budget input and user feedback. In *Feedback Computing 2017 In Conjunction with the 14th IEEE Int. Conference on Autonomic Computing (ICAC 2017)*.

[40] Lisa Palathingal, Rui Wu, Sergiu M. Dascalu, and Frederick C. Harris Jr. Data processing toolset for the virtual watershed. In *2016 International Conference on Collaboration Technologies and Systems (CTS 2016)*. IEEE, 2016.

[41] Russ Rew and Glenn Davis. Netcdf: an interface for scientific data access. *IEEE computer graphics and applications*, 10(4):76–82, 1990.

[42] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

[43] Amazon Web Services. Amazon ec2 pricing. URL: https://aws.amazon.com/ec2/pricing/. Accessed on 18 May 2017.

[44] Amazon Web Services. Amazon ec2 pricing. URL: https://aws.amazon.com/ec2/pricing/. Accessed on 18 May 2017.

[45] Kathy Sierra and Bert Bates. *Head first java*. O'Reilly Media, Inc., 2005.

[46] Michael Stonebraker and Lawrence A Rowe. *The design of Postgres*, volume 15 of number 2. ACM, 1986.

[47] Richard S. Sutton. *Machine Learning*, 3(1):9–44, 1988. DOI: 10.1023/a:1022633531479. URL: https://doi.org/10.1023/a:1022633531479.

[48] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[49] Celery Team. Celery-the distributed task queue, 2011.

[50] USGS. Summary of prms. URL: https://water.usgs.gov/cgi-bin/man_wrdapp?prms. Accessed on 1 June 2017.

[51] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[52] virtualwatershed.org. Virtual watershed. URL: https://hub.docker.com/u/virtualwatershed/. Accessed on 22 June 2017.

[53] Virtual Watershed. Virtual watershed. URL: https://virtualwatershed.org/. Accessed on 18 May 2017.

[54] Jonathan Wheeler and Karl Benedict. Functional requirements specification for archival asset management: identification and integration of essential properties of services-oriented architecture products. *Journal of Map & Geography Libraries*, 11(2):155–179, 2015. DOI: 10.1080/15420353.2015.1035474. eprint: http://dx.doi.org/10.1080/15420353.2015.1035474. URL: http://dx.doi.org/10.1080/15420353.2015.1035474.

[55] Rui Wu, Chao Chen, Sajjad Ahmad, John Volk, Cristina Luca, Frederick C. Harris Jr., and Sergiu M. Dascalu. A real-time web-based wildfire simulation system. In *2016 IEEE Industrial Electronics Conference (IECON 2016)*. IEEE, 2016.

[56] Rui Wu, Sergiu M. Dascalu, Lee. Barford, and Frederick C. Harris Jr. Floating-point data compression using improved gfc algorithm. In *25th International Conference on Software Engineering and Data Engineering (SEDE 2016)*, 2016.

[57] Rui Wu, Sergiu M. Dascalu, and Frederick C. Harris Jr. Environment for datasets processing and visualization using scidb. In *24th International Conference on Software Engineering and Data Engineering (SEDE 2015)*, pages 223–229, 2015.

[58] Rui Wu, Muhanna Muhanna, Sergiu M. Dascalu, Lee Barford, and Frederick C. Harris Jr. Data lossless compression using improved gfc algorithm with multiple gpus. *International Journal of Computers and Their Applications.*, 2016.

[59] Rui Wu, Jose T. Painumkal, Nimrat Randhawa, Lisa Palathingal, Sage R. Hiibel, Sergiu M. Dascalu, and Frederick C. Harris Jr. A new workflow to interact with and visualize big data for web applications. In *2016 International Conference on Collaboration Technologies and Systems (CTS 2016)*. IEEE, 2016.

[60] Rui Wu, Jose T. Painumkal, John Volk, Siming Liu, Sushil Louis, Scott Tyler, Sergiu M. Dascalu, and Frederick C. Harris Jr. Parameter estimation of nonlinear nitrate prediction model using genetic algorithm. In *IEEE Congress on Evolutionary Computation 2017*.

[61] Meng Zhang, Tianyu Yang, and Rui Wu. Space-efficient multiple string matching automata. *International Journal of Wireless and Mobile Computing,*308–313, 2012.

[62] Qian Zhu and Gagan Agrawal. Resource provisioning with budget constraints for adaptive applications in cloud environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 304–307. ACM, 2010.

[63] Ying Zhu. Introducing google chart tools and google maps api in data visualization courses. *IEEE computer graphics and applications*, 32(6):6–9, 2012.