Budget and User Feedback Control Strategy-based PRMS Scenario Web Application

Rui Wu Jose Painumkal Sergiu M. Dascalu Frederick C. Harris, Jr
Department of Computer Science and Engineering
University of Nevada, Reno
Reno, NV, USA
{rui, dascalus, fred.harris}@cse.unr.edu and josepainumkal@nevada.unr.edu

Abstract—**The Precipitation-Runoff Modeling System (PRMS) is used to study and simulate hydrological environment systems. It is common for an environmental scientist to execute hundreds of PRMS model runs to learn different scenarios in a study field. If the study case is complex, this procedure can be very time-consuming. Also, it is very hard to create different scenarios without an efficient method. In this paper, we propose a PRMS scenario web application. It can execute multiple model runs in parallel and automatically rent extra servers based on needs. The control strategy introduced in the paper guarantees that the expense is within the planned budget and can remind a system manager if the quantified user feedback score crosses the predefined threshold. The application has user-friendly interfaces and any user can create and execute different PRMS model scenarios by simply clicking buttons. The application can support other environmental models besides PRMS by filling the blueprint file.**

Keywords*—*PRMS. Budget Control, User Feedback, Web Application.

I. Introduction

The U.S. Geological Survey developed the Precipitation-Runoff Modeling System (PRMS) in the 1980s [8, 9, 10]. The model is widely used in hydrological research. PRMS can consume a very long time to finish a model run with a regular personal computer, especially if a user chooses to generate a PRMS animation file. This problem becomes worse if a user builds a different scenario and start multiple PRMS model runs. There are some similar works already done. However, most of them do not consider how to change a server size based on the needs. Another problem is that PRMS is executed only in a terminal without friendly user interfaces and all the input data is stored with the text files. A new user can spend a very long time to study how to start a model run and modify the input parameters.

To solve these problems, we built a web-based PRMS scenario tool. It executes PRMS models in parallel, changes service size based on budget, takes into account user opinions, and provides user-friendly interfaces. The tool contains two parts: server and client. The server contains multiple Docker workers [4] to finish the PRMS model run requests. These workers contain execution files and are independent. Usually, a server has better hardware than a normal personal computer. Therefore, the performance is guaranteed. If there are many PRMS model run requests and more than the owned server capability, the server can automatically rent machines to increase the server power. It can also quantify the user feedback and warn the system manager if users are not satisfied with the system. The client provides user-friendly interfaces. A user can execute model runs, check modifications, and create different scenarios by clicking mouse buttons.

Our proposed method controls the system from the budget and user feedback. Budget to set up a server is crucial. Unlike industry companies, the academia always has limited funding and the budget should be controlled by the plan. User feedback can be used to test if the users are satisfied. However, if there is a huge amount of the user feedback it can be hard to process them. In our opinion, the user feedback should be quantified and this can simplify the analysis procedure. The prototype system is set up and running in [16].

In the rest of the paper, Section II introduces some related work; Section III shows the system design and how the components are connected; Section IV presents how the system changes its size automatically based on the budget and quantify the user opinion; Section V explains how to create a PRMS scenario and other services.

II. 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.* [1] 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, the 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 could dynamically provision resources to meet the predefined QoS targets by analyzing the variations in the workload intensity. However, the approach offers no control over the expenses as it does not consider budget constraints and user feedback while provisioning resources to ensure guaranteed QoS.

Qian Zhu *et al.* [18] 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 than the static scheduling methods and conserving strategies on 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. In this paper, we propose an innovative server-usage optimization approach to facilitate 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. The proposed approach uses a modified queuing model to provide estimations on waiting time and queue length based on the budget amount which is very relevant as it helps the admin in making budget decisions more easily. The user feedbacks are continuously monitored in the system and auto alert emails are generated to notify the admin of experiencing severe performance issues.

There have been many types of research going on in the field of environmental modeling by different interdisciplinary research groups. Consortium of Universities for the Advancement of Hydrologic Science, Inc (CUASHI) [2] is a research organization comprising the universities in America to develop services and infrastructure for understanding and exploring the mysteries related to water science. Hydroshare [14] is one of their projects aimed at providing cyberinfrastructure to facilitate an online collaborative environment for sharing hydrological models and data. Hydroshare offers various web apps to share, visualize, analyze and run hydrological models. The goal of Hydroshare is to enhance the collaboration in the research community by helping in the discovery and access of data models published by other researchers. Geographic Storage, Transformation and Retrieval Engine (GSToRE) [15] is a data management framework to support the storage, management, discovery and sharing of scientific and geographic data. It was developed at Earth Data Analysis Center, the University of New Mexico with a goal to offer a flexible and scalable data management platform for scientific research. GSToRE offers a REST API to facilitate the storage, retrieval, and removal of geospatial data and associated meta-data.

III. System Design

The system mainly has three services: PRMS scenario tool, Data conversion, and authentication. The system is designed as Figure 1 shows.

A. Queue Master

To arrange resources reasonably and in order, our system has a queue component. All the user requests are handled by a docker container named "Queue Master" (see Figure 1). This container classifies the requests into different groups based on the required service types, Then different requests go into different queues. For example, if a user wants to run a PRMS scenario, the request is handled by the PRMS scenario service. Therefore, it will enter the corresponding queue. The system can offer an estimated waiting time based on a proposed method introduced in this thesis and the user can choose which queue based on their needs.

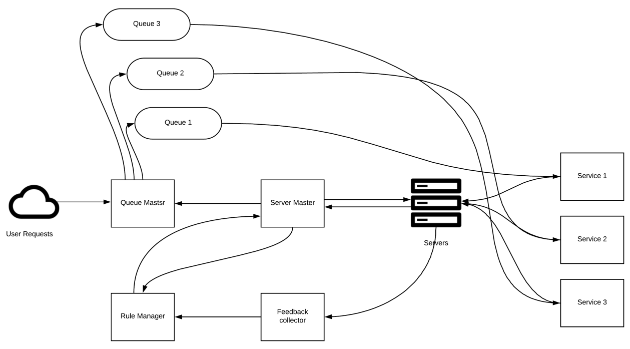


Figure 1. Architecture Design.The system contains queue master, queues, server master, servers, rule master, and feedback collector.

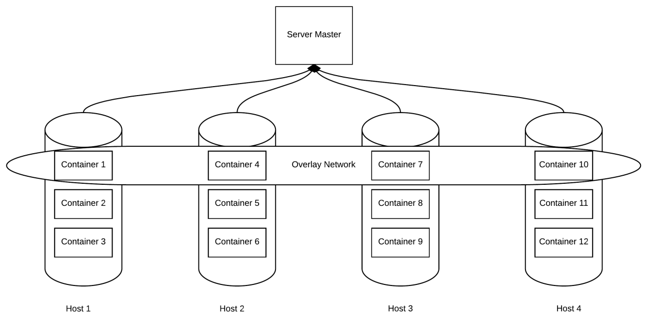


Figure 2. Server Master. The server master is in charge of all the host machines. It inspects all the host machine information, such as CPU and memory usage.

B. Server Master

The server master is used to inspect all the host machines health information, such as server failures and budget information. For example, the CPU, memory, and network usage percentages. It can automatically rent another host machine based on the control strategy. The rent host machine event can be triggered based on the rules stored in Rule Manager container.

In each host machine, there are worker containers [4]. These worker containers are arranged into different groups based on the service. For example, Container 1, Container 4, Container 7, and Container 10 are in the same group in Figure 2. All the workers in this group are created with the same Docker image and they know how to finish the job based on the input files. For example, the users can run PRMS scenarios in the system. They need to upload input files to start a model run. The workers in PRMS service group obtain the input files from the database based on the job ID and store the model output files into the database after the job is finished.

To connect different docker containers in different host machines, we set up a key-value store node. If docker containers are on the same host machine they can ping each other directly without any further operations. However, docker containers cannot ping each other if they are on different host machines. The key-value store node is used to store different host machine IPs, networks, and endpoints. After the node is setup, different host machines can view each other. Then, it is easy to create a docker overlay network across different host machines.

For each group, there should be a task manager node and a feedback collector. The task manager node is used to create worker containers in different host machines and delete the worker container after the job is finished. We did not use any orchestration tool, such as Docker swarm [5] and Apache Mesos [11], because it is not possible to stop a container in a certain machine with these tools based on our knowledge. These tools can change the server size based on some events, such as CPU and memory usage percentages. However, it is at a high level (server as a whole). Based on our experiences, only the worker container itself knows what happens inside. It is not reasonable to shrink the server size based on CPU usage or time. Sometimes, the CPU usage is low and the container is busy. For example, file transportation jobs mainly use I/O bandwidth instead of CPU or memory. In our opinion, we can stop the container, only when the container finishes the last line of the script. Therefore, each worker in our prototype system sends a termination request to the task manager after it finishes the job and then the task manager will stop the container. For a large cluster, it is possible that many containers report termination events to the server master container at the same time. It can cause problems if the network bandwidth is not big enough. There are two solutions for this problem: 1) more than one server master containers are set up in the system to process the reports. Each master container is only in charge a group of containers and this method reduces the burden. 2) Jittering APIs can be applied. Each container does not send the termination information to the server master through the API directly after the container finish the job. The worker container waits a random time and then send the information. This avoids too much information occupies the network at the same time.

The relationships between the server master node, host machines, and worker container are displayed in Figure 2.

C. Servers

Servers are physical machines in the system. These machines are set into different groups based on the needs of different services. For example, there are more PRMS scenario requests than other requests. Therefore, more machines are in PRMS scenario group than other service groups. The server master runs docker configuration files to download docker images and setup services in different machines. The server master may rent more machines automatically based on the rules stored in the rule manager.

D. Feedback Collector

The user feedback is very important for the project manager to set up reasonable rules. The survey is used to collect the user feedback in our prototype system. The system can turn the survey results into a feedback score and change the server size automatically or offer suggestions to the project manager based on the feedback score.

The feedback collector can send a survey invitation to the user after he/she uses the service. Each question has different weights and the options of different questions have different points. The project manager can setup a threshold for each question. If the point passes the threshold, it means the project manager needs to do something. Here is an example: the project manager wants to know the user's opinion about the service performance. Therefore, he puts two questions in the survey: 1) What's your opinion about the waiting time? (question weight 0.8) Options: A. Too long (2 points) B. Long (1 point) C. Not Sure (0 point) D. Short (-1 point) E. Very short (-2 points) 2) Do you want to pay more to have a faster service? (question weight 1.2) Options: A. Strongly Agree (2 points) B. Agree (1 point) C. Not Sure (0 point) D. Disagree (-1 point) E. Strongly disagree (-2 points). If a user chooses A for the first question and D for the second question, then it contributes 0.8*2+1.2*(-1)=0.4 to the global feedback score. If the global feedback score passes the threshold it means the users believe the server is slow and they want to pay more for a faster service. Therefore, the project manager may need to allow the server master to rent more machines from the third party companies.

Each service has a feedback collector. The feedback collector contains a survey predefined by the project manager. Based on the feedback score, the feedback collector can affect rules stored in the rule manager.

E. Rule Manager

Queue master and server master follow rules stored in the rule manager. The rules are applied to these two masters through RESTful APIs. When the project managers want to add a new rule, they may also need to modify RESTful APIs in the queue master and server master. This is because the RESTful APIs may not include the functions required in the rules. For example, there is a rule requiring average job waiting time in Service 1 queue should be 10 seconds. However, the queue master does not have a RESTful API to change the average job waiting time. Then, the project managers should work on the API first.

IV. Control Strategy

The control strategy is stored in the rule manager of the prototype system (introduced in Section III-E). The strategy includes how the service requests are handled in a queue and how to manage the server based on user feedback. This section introduces brief ideas. More details on the algorithm and validation can be found in our previous work [17].

The $M/M/1/1/\infty/\infty$ [7] queuing model was modified and used in the proposed self-managed elastic scale hybrid server to estimate the queue length and job waiting time in the modelling environment. The modified queuing model includes the allocated budget amount (B), budget period (T_b), cost of rented instances (\$P/hour), the average time for the job execution in rented instance (T_{rent}) and the average job execution time in owned instance (T_{own}). The maximum of number of jobs that can be processed with owned servers would be (N_0 * T_b)/T_{own}, where N_0 is the number of owned severs in the hybrid cluster.

B/ (P * T_{rent}) would be the total number of jobs that can be processed with rented servers for the allocated budget amount B. Hence, the total number of jobs processed by the hybrid cluster during the budget period T_b would be the sum of (N_0 * T_b)/T_{own} and B/ (P * T_{rent}). On incorporating the above details into the queuing model, 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:

$$L_{\mathrm{H}} = \frac{\lambda^2}{(\frac{N_0}{T_{\mathrm{own}}} + \frac{B}{P*T_{\mathrm{rent}}})^2 - \lambda(\frac{N_0}{T_{\mathrm{own}}} + \frac{B}{P*T_{\mathrm{rent}}})} \quad (1)$$

$$T_{\mathrm{H}} = \frac{\lambda}{(\frac{N_0}{T_{\mathrm{own}}} + \frac{B}{P*T_{\mathrm{rent}}})^2 - \lambda(\frac{N_0}{T_{\mathrm{own}}} + \frac{B}{P*T_{\mathrm{rent}}})} \quad (2)$$

Algorithm 1: Create Rent Worker

```
1  function rented_worker_creation ($RW, UR, N, T_{int}$);
   Input : RW denotes number of rented workers; UR
           denotes unused rentals; N denotes maximum
           number of models processed with rented
           workers for the input budget; $T_{int}$ denotes the
           time interval
2  if $RW < N$ then
3      if $Jobs\ in\ queue$ then
4          Create Worker;
5          $RW = RW + 1$;
6          while $UR > 0$ AND $RW < N$ AND Jobs in
             queue do
7              Create Worker;
8              $RW = RW + 1$;
9              $UR = UR - 1$;
10         end
11         Sleep $T_{int}$ and go to line 2
12     else
13         $UR = UR + 1$;
14     end
15 else
16     No more rented workers available;
17 end
```

Algorithm 1 illustrates 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_{rent}) is already available in the system from previous job execution details. Then the total number of jobs that could be processed with rented workers is N = T/ T_{rent}. The counter variable RW (Rented Workers) would keep track of the total number of jobs rented. To utilize the rented resources judiciously, the usage of rented workers 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. At 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. More details about the budget control strategy can be found in our previous work [12].

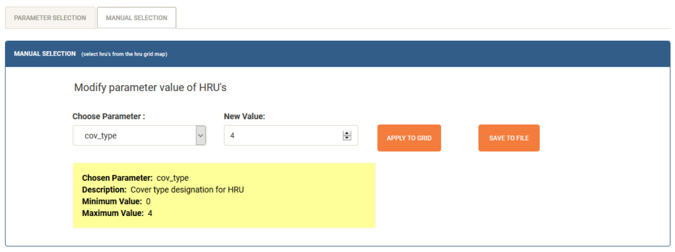
The system also includes a dashboard where the user can view the details of the finished jobs in real time. On finishing a model simulation job, the dashboard will display the details of the job such as the task id, the cost for the job execution, run-time of the job, waiting time in the queue, the name 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.

The prototype has a feedback survey form, where the user can provide feedbacks on the performance of the service. The manager can also view the results of the feedback survey from the users in Survey Results page. The page displays separate bar graphs for each question in the survey questionnaire. The bar graphs show 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 in taking decisions on increasing or decreasing the budget amount.

V. PRMS SCENARIO TOOL

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. It also provides data conversion service. This allows a user to modify model input and output files format. The user does not need to be required any programming skills to use our model modification component. The user interface is intuitive and user-friendly so that the user can perform the model modification activities through simple mouse clicks.

To create a user-defined simulation scenario, the user has to choose one of the existing model simulations or a default simulation for modification. Then, the user determines to modify which parameters to get the desired model scenario. For example, the user can change the vegetation types of the study area from "forest" into "bare ground" to study what can happen if people cut too many trees in the field. Once the parameters are decided, the user needs to specify Hydrologic Response Units (HRUs) of the study area that should be changed. Then, the system knows where and what to modify.



Modify parameter value of HRU's

Choose Parameter : cov_type

New Value: 4

Chosen Parameter: cov_type
Description: Cover type designation for HRU
Minimum Value: 0
Maximum Value: 4

Figure 3. Screenshot of the model modification component in PRMS Scenarios Tool

A. *HRU Selection Methods*

PRMS divides the model area into discrete HRUs, where each HRU is composed either of land, lake, swale or other types. The PRMS modification component offers two different ways to select the HRUs for parameter modification. They are parameter selection and manual selection. In parameter selection, the HRUs can be selected based on its parameter values and in the manual selection, the user can manually choose the desired HRUs from a 2D HRU grid map. After finishing the modification of HRUs, the user can re-run the model with the modified inputs. Figure 3 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.

1) Manual Selection: Using manual selection, the user could select the HRU cells directly on the 2D grid map through simple drag & drop mouse click operation. To select an HRU, place the mouse cursor over the desired HRU cell on the 2D grip map and then perform a left click. To select multiple HRUs, left click on the HRU cell, drag along the desired direction, and then release 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 get 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 shows a screenshot of model modification using manual selection, where the user is changing the vegetation type of selected HRUs to shrubs (Type 2) and grass (Type 1) and trees (Type 3). 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 a 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 and thereby saves the time and effort of the researchers while performing scenario-based studies.

2) 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 (greater than, less than or between), and the parameter

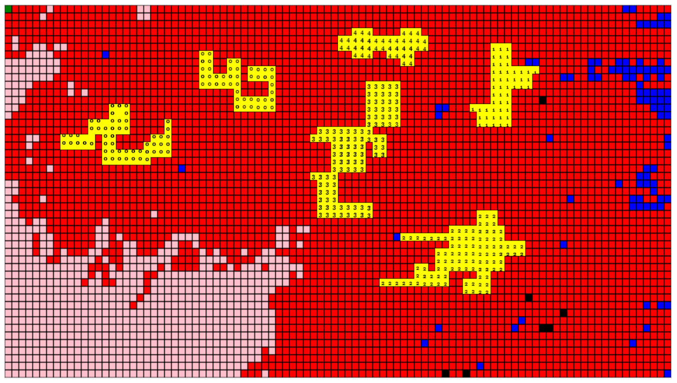


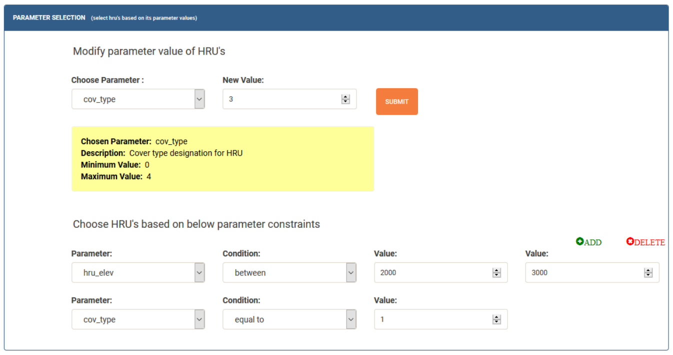
Figure 4. Model modification using manual selection. Modified the vegetation type of chosen HRUs to bare soil (0), shrubs(2), grasses(1), trees(3) & coniferous (4)

value. For example, Figure 5 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), 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 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 HRU's that satisfies 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 in 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. The final modified 2D grid map is overlapped 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.

B. *Other Services*

1) Data Convertor: In the PRMS Scenario tool, data is stored in NetCDF format. NetCDF is a self-describing and machine-independent data format [3]. It is widely used in climate data research. However, this file formats may not be supported by other tools used by a modeler. Therefore, the scenario tool contains a data convertor. It can convert NetCDF file into text and text file into NetCDF file. More details are introduced in this paper [13].

2) Authentication: The system offers the one-time authentication service. This means a user only needs to login once and the user can use all the services in the system. This is


PARAMETER SELECTION (select hru's based on its parameter values)

Modify parameter value of HRU's

Choose Parameter : cov_type

New Value: 3

Chosen Parameter: cov_type
Description: Cover type designation for HRU
Minimum Value: 0
Maximum Value: 4

Choose HRU's based on below parameter constraints

Parameter:	Condition:	Value:	Value:
hru_elev	between	2000	3000
cov_type	equal to	1	

Figure 5. Model modification using parameter selection of the HRUs

done by using JWT (JSON Web Token). It is safer Because the system uses JWT instead of passing the user's username and password. More details can be found in our previous work [6].

VI. Conclusion and Future Work

In this paper, we have proposed a PRMS scenario web application and a budget and user feedback control framework. It allows a modeler to create different scenarios and execute them in parallel. The application's server can rent extra machines to increase its computing power automatically based on needs and warn the system manger based on the quantified user feedback. The budget control strategy can also make sure the renting cost is within the plan. It is easy to extend the system with other models and control rules because of the proposed design.

In the future, we want to extend the tools with more environmental models and also add payment component. We also would like to improve the proposed queuing model by considering a server starting time. This can be a challenge because usually the server starting time is not fixed and can be very long. Last but not least, the proposed tools should be validated by different programs besides environmental models.

Acknowledgment

This material 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.

References

[1] 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. Washington, DC, USA: IEEE Computer Society, 2011, pp. 295–304.

[2] In Consortium of Universities for the Advancement of Hydrologic Science. *CUASHI*. Accessed on 18 July 2017. URL: https://www.cuahsi.org/.

[3] Open Geospatial Consortium. "OGC network Common Data Form (netCDF) standards suite." In: (2014).

[4] Docker. *Docker SDK for Python*. Accessed on 18 July 2017. URL: https://docker-py.readthedocs.io/.

[5] Docker. *Docker Swarm | Docker*. Accessed on 18 July 2017. URL: https://www.docker.com/products/docker-swarm.

[6] M. M. Hossain et al. "Web-Service Framework For Environmental Models." In: *Seventh International Conference on Internet Technologies Applications (ITA)*. IEEE. 2017.

[7] Samuel Karlin and James McGregor. "Many server queueing processes with Poisson input and exponential service times". In: *Pacific J. Math* 8.1 (1958), pp. 87–118.

[8] G. H. Leavesley et al. "Precipitaion-Runoff Modeling System:User's manual." In: *Water-Resources Investigations Report.* (1983), pp. 83–4238.

[9] R. G. Markstrom S. L.and Niswonger et al. "GSFLOW — Coupled Ground-Water and Surface-Water Flow Model Based on the Integration of the Precipitation-Runoff Modeling System (PRMS) and the Modular Ground-Water Flow Model". In: *Water-Resources Investigations Report.* (2005).

[10] S. L. Markstrom et al. *the Precipitation-Runoff Modeling System*. Version 4. U.S. Geological Survey Techniques and Methods, Book 6, Chap. B7, http://doi.org/http://dx.doi.org/10.3133/tm6B7. Clarendon Press, 2015, p. 158.

[11] Apache Mesos. *Apache Mesos*. Accessed on 18 July 20177. URL: http://mesos.apache.org/.

[12] T. J. Painumkal et al. "Self-managed Elastic Scale Hybrid Server Using Budget Input and User Feedback." In: *12th International Workshop on Feedback Computing.* 2017.

[13] L. Palathingal et al. "Data Processing Toolset for the Virtual Watershed." In: *2016 International Conference on Collaboration Technologies and Systems (CTS).* IEEE. 2016, pp. 281–287.

[14] D. G. Tarboton et al. "HydroShare: An online, collaborative environment for the sharing of hydrologic data and models (Invited)". In: *AGU Fall Meeting Abstracts* (Dec. 2013).

[15] Jonathan Wheeler and Karl Benedict. "Functional Requirements Specification for Archival Asset Management: Identification and Integration of Essential Properties of Services-Oriented Architecture Products". In: *Journal of Map & Geography Libraries* 11.2 (2015), pp. 155–179. 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.

[16] R Wu. *Virtual Watershed Platform*. Accessed on 18 May 2017. URL: https://virtualwatershed.org/.

[17] R Wu et al. "Self-managed Elastic Scale Hybrid Server Using Budget Input and User Feedback". In: *12th FC: Workshop on Feedback Computing*. ICAC 2017. Columbus, Ohio, USA, 2017.

[18] 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*. ACM. 2010, pp. 304–307.