URegM: a unified prediction model of resource consumption for refactoring software smells in open source cloud

Asif Imran
aimran@csusm.edu
California State University San Marcos
San Marcos, California, USA

Tevfik Kosar
tkosar@buffalo.edu
University at Buffalo
Buffalo, New York, USA

Abstract

The low cost and rapid provisioning capabilities have made the cloud a desirable platform to launch complex scientific applications. However, resource utilization optimization is a significant challenge for cloud service providers, since the earlier focus is provided on optimizing resources for the applications that run on the cloud, with a low emphasis being provided on optimizing resource utilization of the cloud computing internal processes. Code refactoring has been associated with improving the maintenance and understanding of software code. However, analyzing the impact of the refactoring source code of the cloud and studying its impact on cloud resource usage require further analysis. In this paper, we propose a framework called *Unified Regression Modelling (URegM)* which predicts the impact of code smell refactoring on cloud resource usage. We test our experiments in a real-life cloud environment using a complex scientific application as a workload. Results show that *URegM* is capable of accurately predicting resource consumption due to code smell refactoring. This will permit cloud service providers with advanced knowledge about the impact of refactoring code smells on resource consumption, thus allowing them to plan their resource provisioning and code refactoring more effectively.

CCS Concepts: • **Software and its engineering** → **Maintaining software**.

Keywords: resource usage prediction, scientific application in cloud, unified regression modelling

ACM Reference Format:
Asif Imran and Tevfik Kosar. 2022. URegM: a unified prediction model of resource consumption for refactoring software smells in open source cloud. In *2022 The 3rd European Symposium on Software Engineering (ESSE 2022), October 27–29, 2022, Rome, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3571697.3571705

ESSE 2022, October 27–29, 2022, Italy
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9730-8/22/10…$15.00
https://doi.org/10.1145/3571697.3571705

1 Introduction

Cloud computing is the dynamic provisioning of resources from a shared resource pool, which can be provisioned by a pay-as-you go mechanism [1]. The dynamic nature and rapid provisioning of resources have made the cloud a desirable platform to run distributed scientific applications which are resource intensive. However, cloud service providers are constantly aiming to optimize resource consumption at the system end to make more resources available to its customers. Recent studies have shown that positive relationship exists between code smells and resource usage [6]. However, predicting this change of resource usage due to refactoring of code smells present in source code of the cloud have not been conducted. In this paper, we propose a model called *Unified Regression Model (URegM)* that predicts the impact of code smell refactoring on resource consumption of cloud computing processes. To the best of our knowledge, relationship between code smell refactoring of cloud source code and establishing its relationship to resource consumption at the data center level have been conducted to a limited extent.

In the cloud environment, optimizing resource usage of the cloud's own processes will provide business edge to the cloud service providers as they can provision the excess resource to perform the tasks of the cloud customers. However, the impact needs to be tested by running real life applications in the cloud. In this paper, we use a real-life scientific application as a workload to test the proposed framework. The selected scientific application is designed to perform in distributed environment and it can harness the power of the cloud to complete the jobs. The tasks in this application is divided in groups and distributed across multiple virtual machine (vm) instances of the cloud for parallel and collaborative computing. As a result, using the scientific application as a workload to test the resource usage of the cloud will provide an ideal and real life scenario to analyze performance.

Earlier techniques primarily focus on predicting resource provisioning for running scientific applications in cloud. However, effective framework is required that will provide advanced knowledge on the impact of refactoring code smells on resource consumption as this will enable cloud service providers to utilize the extra resource saved by refactoring. Alternatively, if certain refactoring increases resource consumption, this prediction framework will update the cloud service provider about it. Code smells have been traditionally refactored to improve maintainability and understandability of source codes. Exiting studies focus on extensively testing software code to identify bugs that may result in code smells [12]. However, code smells can be refactored in cloud to optimize resource usage by the cloud computing processes. This extra resource can be provisioned to support the resource requirement of resource intensive scientific applications. To the best of our knowledge, existing methods cannot accurately predict the impact of refactoring code smells on resource consumption in cloud environment.

To address the above problem, a novel prediction approach to analyze impact of code smell refactoring on resource usage is proposed which focuses on selection of features using Genetic Algorithm (GA) and unifies 4 regression algorithms to improve performance. The framework is tested using a real-life workload called *WildfireDB* which is a scientific application designed to simulate wildfire situations using large volume of historical and vegetation data. This tool has simulation components to analyze wildfire situations to be studied by climate change ecologists. The main contributions of this paper are provided below:

- *URegM* is a framework to predict the change in resource consumption due to code smell refactoring of cloud source code. The performance of the framework is tested for running real life scientific applications in cloud.
- A dataset is used with selection of correct features of the cloud platform such as CPU, memory, weighted methods per class, lookahead which is used for training *URegM* model.
- Comparative analysis of *URegM* to existing models like REAP [13] show that the proposed model outperforms the current models in terms of *mse, rmse, accuracy,* and *execution time.*

Rest of the paper proceeds as follows. Section 2 identifies the related research conducted in the field of predicting resource usage and highlights the importance of optimizing cloud resource consumption. Section 3 provides details of the proposed *URegM* framework and the prediction approach. Section 4 highlights the setup of experimental environment and analyzes obtained results. Section 5 concludes the paper and identifies scope of future research.

2 Related Work

Park et al. investigated whether existing refactoring techniques support resource-efficient software creation or not [10]. Since resource efficient software was critical in mobile environments, they focused their study on mobile applications. Results showed that specific refactoring techniques like *Extract Class* and *Extract Method* worsened energy consumption because they did not consider power consumption in their refactoring process. The goal was to analyze the resource efficiency of the refactoring techniques themselves, and they stated the need for resource-efficient refactoring mechanisms for code smells. Imran et al. conducted a study on the impact of code smell refactoring on resource usage [5]. They used automatic refactoring tool to detect and remove code smells, followed by calculating resource consumption change for a specific workload. However, they did not apply machine learning to predict the impact.

Platform-specific code smells in High-Performance Computing (HPC) applications were determined by Wang et al. [18]. AST-based matching was used to determine smells present in HPC software. The authors claimed that the removal of such smells would increase the speedup of the software. The assumption was that specific code blocks performed well in terms of speedup in a given platform. Perez-Castillo et al. stated that excessive message traffic derived from refactoring god class increased a system's power consumption [11]. It was observed that power consumption increased by 1.91% (message traffic = 5.26%) and 1.64% (message traffic = 22.27%), respectively, for the two applications they analyzed. The heavy message-passing traffic increased the processor usage, which proved to be in line with the increase in the power consumption during the execution of those two applications.

An automatic refactoring tool that applied the *Extract Class* module to divide *god class* smell into smaller cohesive classes was proposed in [3]. The tool aimed to improve code design by ensuring no classes were large enough, which was challenging to maintain and contained a lot of responsibilities. The tool refactored code by suggesting *Extract Class* modifications to the users through a User Interface. The tool was incorporated into the Eclipse IDE via a plugin. The authors consulted an expert in the software quality assessment field who gave his expert opinion to identify the effectiveness of the tool. Despite the existing effort in refactoring code smells, however, impact prediction of code smell refactoring on cloud resource consumption had been analyzed to a limited extent.

Liu et al. [9] enabled automatic resource maintenance in a cloud environment via forecasting the workload in advance. The cluster trace data from Google was used as a dataset in this regard. Ghobaei-Arani et al. [4] used reinforcement learning to predict future resource requirements based on the current workload on the cloud. Chen et al. [2] proposed a *Fuzzy Neural Network (FNN)* to analyze current resource demand in the cloud and predicted future resource requirements. Shaw et al. [14] proposed a novel predictive anti-correlated placement algorithm which made the cloud energy efficient by managing resources effectively. We saw that most of the prediction frameworks for the cloud were focused on effective resource provisioning for tasks that were executed on the cloud, however, research needed to be done to analyze and predict resource usage requirements for the cloud environment itself.

3 Resource prediction: *URegM* method

In this section, we present a case study where we amalgamate the four regression models to improve performance in terms of *mse, rmse, accuracy,* and *execution time*. We base our case study on one scientific application discussed later in this paper. The cloud environment has been specified to be *OpenStack*. Since cloud computing has become a significant paradigm where many real-life applications are executed, we select the open source cloud computing platform for our study. Cloud also offers a ubiquitous and parallel infrastructure to run critical scientific applications which are resource-intensive. Predicting the impact of cloud resource requirements after refactoring code smells of the cloud platform itself will be a significant contribution. This will enable cloud service providers to improve the resource scheduling scheme of the cloud.

Also, software engineers are placing time and effort to remove code smells in cloud platforms. As a result, predicting the change in resource requirement of the cloud after refactoring code smells is important to automatically and correctly provision future resource requirements and refactoring activities [8]. It will be beneficial from both the technical and financial perspectives of the cloud service provider. The ML approach used to predict CPU and memory usage in cloud for different workloads is called *Unified Regression Model* (*URegM*) based prediction. The various methods, classes, and data in cloud are dependent on each other. The resource consumption of a particular component before and after elimination of code smells may be caused by the other connected components. This potential dependency is modeled using regression techniques and the flow of the proposed model is shown in Figure 1. We compare our obtained results to the existing individual models and *REAP* [7] approach to analyze performance.

3.1 Feature Selection

The important components of our Machine Learning (ML) prediction for resource usage are the population, feature selection, and ML algorithm. Feature selection is an important task of ML activity in various research. We use genetic algorithm (GA) for feature selection, which is a greedy approach that selects the best features based on produced output. Feature selection using GA provides the constraints and inputs of software which has significant impact on the resource consumption of the software. We provide the procedure of GA in Algorithm 1.

Initially, random list of features is selected to form the population. Next the correctness of those features is determined by their fitness values. Acceptable fitness score range from *76.0 to 89.0* [17]. The fitness values are scaled in an acceptable range which is called expectation scores. These scores are used to select the features for prediction of resource consumption. The features which get low scores will be removed and can be used together with the next set of population. New set of features can be obtained by conducting mutation and crossover on the correct population. Mutation and crossover can be done individually or together. The process can be repeated and if the current set of population of features outperforms the earlier set, then the current set of features will replace the previous set of features.

3.2 Workload description

WildfireDB is a tool for modeling fire occurrence and spread by analyzing data about locations, sizes, vegetation, field, and other topographic features [16]. The architecture of the tool is shown in Figure 2. The dataset includes *2,367,209* data points of California wildfires. Each data point corresponds to a specific *375m X 375m* polygon area at a given time, and it also records the condition of the neighboring cells at that time frame. The relevant covariates are also present for analysis. Similar statistics are provided regarding topographic information of vegetation, fuel, etc which includes the maximum, minimum, median, sum, mode, count, and mean values. *Firesim* is incorporated with the database to simulate wildfire situations for climate change ecological analysis. Total simulation run was conducted for *5001.68* seconds.

Climate change ecologists face significant challenges in analyzing wildfire data because fire occurrence and covariates are available in various frameworks. Amalgamating the different frameworks and analyzing them require very high computation power. Secondly, the raster data in this dataset have more that two million data points for the state of California alone. Mining large scale feature data is a significant bottleneck in terms of computational resources. Third, the large size of data further complicates the resources required to combine them for analysis. Combining data sources from various frameworks require conversion of the data into a uniform representation. This conversion is computationally expensive and due to being two-dimensional data, the size will increase by 4 times when combined to a single uniform format. The computational complexity of combining the dataset of raster and vector data in*WildfireDB* is

$$O(np^2.c.r)$$

Here np is the number of polygons, and c and r are the number of rows and columns in the dataset.

WildfireDB is designed to conduct this data fusion activity followed by analysis in a fully decentralized approach. To ensure that computer scientists can define systems which can

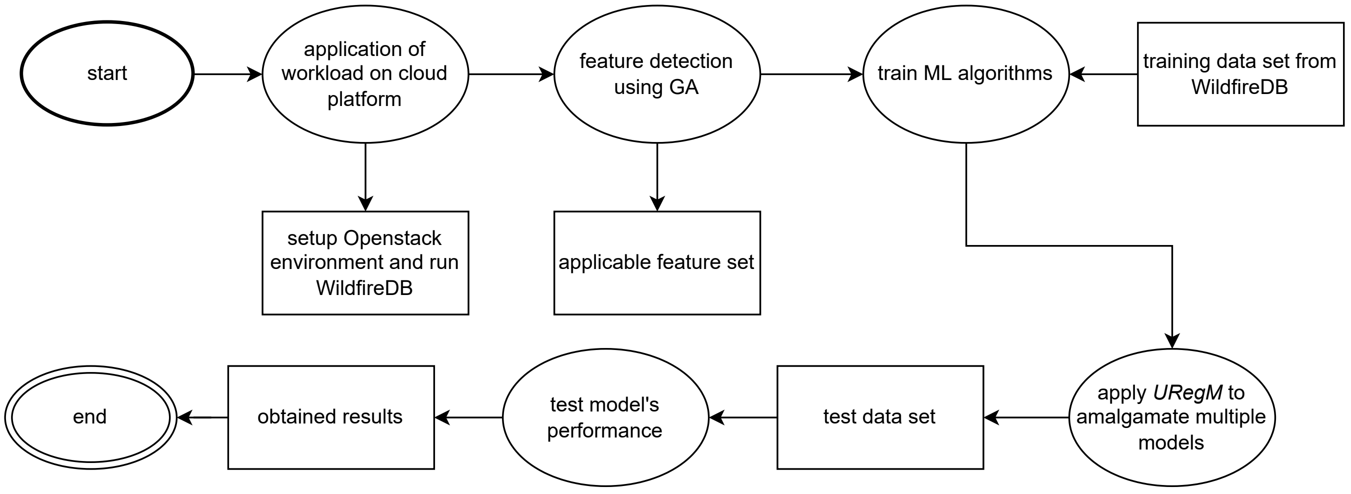
start
application of workload on cloud platform
feature detection using GA
train ML algorithms
training data set from WildfireDB
setup Openstack environment and run WildfireDB
applicable feature set
apply *URegM* to amalgamate multiple models
test data set
test model's performance
obtained results
end

Figure 1. Workflow of the machine learning procedure

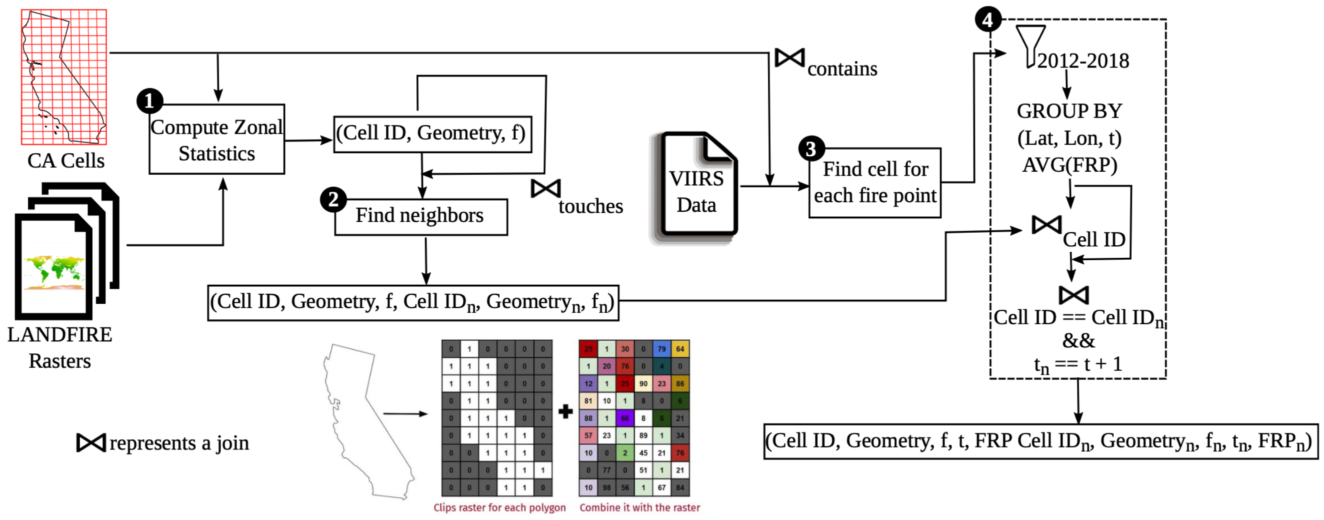
CA Cells
LANDFIRE Rasters
1 Compute Zonal Statistics
(Cell ID, Geometry, f)
2 Find neighbors
⋈ touches
(Cell ID, Geometry, f, Cell ID_n, Geometry_n, f_n)
⋈ contains
VIIRS Data
3 Find cell for each fire point
4 2012-2018
GROUP BY (Lat, Lon, t) AVG(FRP)
⋈ Cell ID
⋈ Cell ID == Cell ID_n && t_n == t + 1
(Cell ID, Geometry, f, t, FRP Cell ID_n, Geometry_n, f_n, t_n, FRP_n)
⋈ represents a join
Clips raster for each polygon
Combine it with the raster

Figure 2. WildfireDB workload description [16].

make this process more resource friendly, *WildfireDB* provides the distributed algorithms and data structures which can be run to stress the resources of cloud computing environment to analyze resource bottlenecks on the cloud processes. The simulation models which can be run to amalgamate the vector and raster data for generating the computationally expensive 4-dimensional dataset has been provided. At the same time it provides an algorithm which forms two parallel data sets to be used at run time. Those operations stress the exascale supercomputers like *Frontera* of *Texas Advanced Computing Center (TACC)* and *Blue Waters* of *National Science Foundation (NSF)*. Hence, the workload is substantial to test the impact on cloud resources.

3.3 URegM framework

The ML approach used to predict CPU and memory usage in cloud for different workloads is called *URegM*. The various methods, classes, and data in cloud are dependent on each other. The resource consumption of a particular component before and after elimination of code smells may be caused by the other connected components. This potential dependency is modeled using regression techniques. The regression analysis helps us to detect change in CPU and memory usage for each clock cycle on the same workload for varying data sizes. Initially the workload is applied on vm instances launched in *Openstack* and after a number of experimentation with the workload data, a regression based analysis of CPU and memory usage is obtained. Also, the null values in the dataset are removed as part of cleaning the dataset. Next, based on the obtained values of resource usage, GA is applied to select the features in the software code which affect the resource consumption.

At the same time, only the components of the software where code smells are found and refactored are considered in

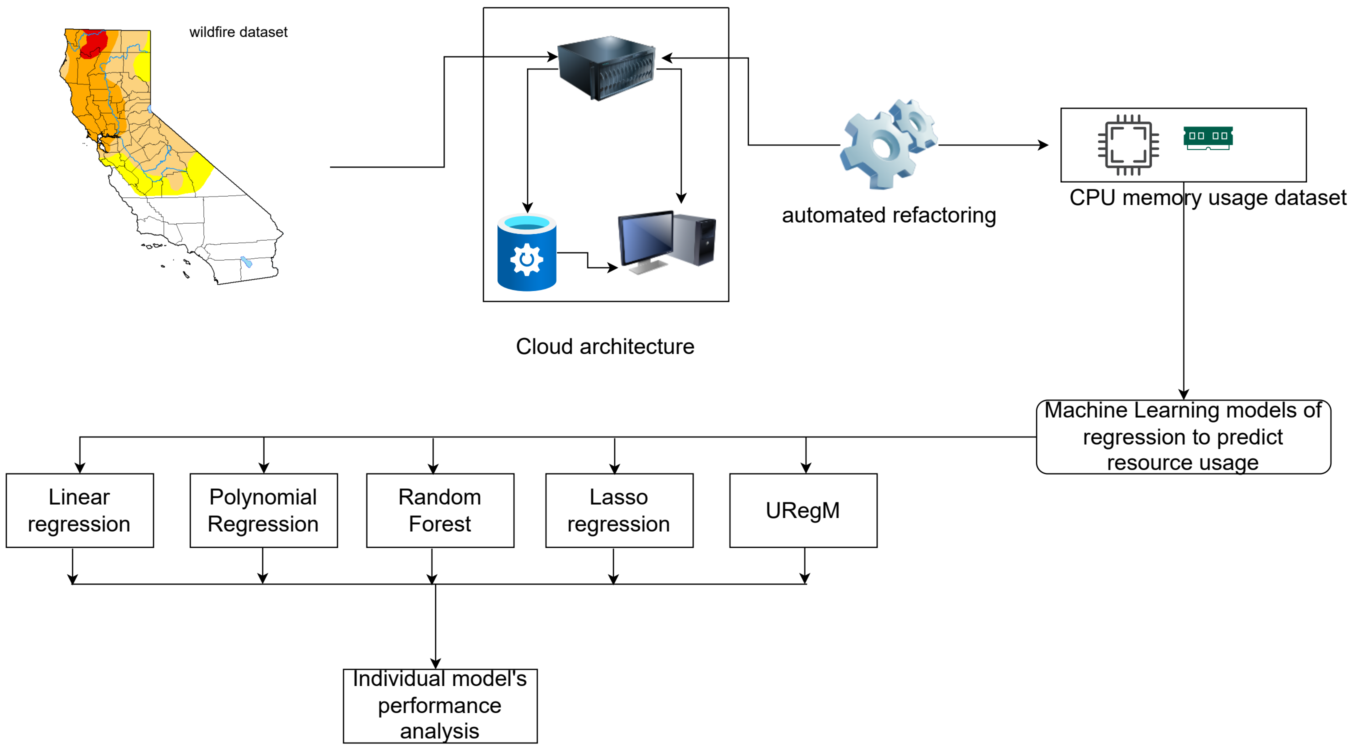

Figure 3. Application of workload and *URegM* approach in Openstack.

the dataset, thereby significantly reducing the overall size of the dataset and removing redundant code blocks. Next, the ML algorithms are applied on the final dataset and regression models are trained. We divide the dataset into two halves. One half which contains 80% of the data is used for training and 20% of the data is kept in other half which is used for testing. Algorithm 1 is provided which explains the ML approach. The model is trained on the training dataset and tested on the test dataset, and the metrics such as *mse, rmse, accuracy* and *execution time* are used to test its performance. The training dataset is obtained from the benchmarking study where code smells in 31 open source software are refactored using automated tools and the change in resource consumption is monitored before and after refactoring for each method of the software where refactoring was conducted [6]. Pre-specified workloads are used for the benchmark study. The researchers establish a benchmark where individual types of code smells are detected and refactored in each software, followed by an analysis of CPU and memory consumption impact. Afterward, they conducted a batch refactoring of smells and analyze their collective impact on resource usage. We use the dataset of 31 open source software to train *URegM*. Finally, the performance of the four ML models on the dataset together with the proposed *URegM* approach are recorded on cloud environment.

In Algorithm 1, *BSC* is a variable which represent the accuracy of the various combination of the four individual models, *BMSc* represent the combination of models which has highest accuracy, *models* is an array which lists all the models used for making the combination. *DS* contains the results of the prediction dataset which is applied to the various models for the training operation, datasets from *0 to n-1* is considered, which is used and $Train$ contains the combination of regression models which is produced from the four individual regression approaches. Next, the $Test$ process assembles the models and compares the accuracy. In case the calculated accuracy is better than the *BSC*, then the value of *BSC* is overwritten and the combination of models which gave the new accuracy is used to replace the existing set of models in *BMSc*.

URegM combines prediction results from various algorithms to obtain overall better results and get stronger predictions. It is a supervised learning approach which can be trained and improved. Unifying the various models through *URegM* will yield better results if the individual models are different from each other in terms of algorithmic design and data analysis. The ML procedure is shown in Figure 3. The input workload is the *WildFireDB* which consists of 268 GB of data and multiple files with various operations which are executed in cloud. Overall the CPU usage, memory usage, and file sizes are shown in Table 1. The experimentation is conducted with various number of tasks executed by the workload on the cloud and the CPU and memory usages are predicted.

Algorithm 1 URegM algorithm

```
Require: $BSc \geq 0$
  $BMSc = NULL$
  $models[i] \longleftarrow [LiR, PR, LR, RF, URegM]$
  $DS \longleftarrow predictionSet[]$
  $Main \longleftarrow predictionSet[1]$
  while $N \neq 0$ do
    while x in 1...n do
      $Train \longleftarrow models[0..n-1] \times X$
      $Test \longleftarrow \models [n]$
      URegM[Test]
      if $MSc$ is URegM == predictionSet[0,1]*0.1 then
        if $BMSc < MSc$ then
          $BMSc \longleftarrow MSc$
          $BSc \longleftarrow DS$
        end if
      end if
    end while
  end while
  return $BMSc\ and\ URegM[results]$
```

VMid	vCPU	RAM (GB)	Disk (GB)	OS
1	1	0.5	50	Ubuntu 18.04
2	1	1	256	CentOS 7
3	2	2	500	Ubuntu 18.04
4	2	4	500	CentOS 7
5	4	4	500	Ubuntu 18.04
6	4	6	500	CentOS 7

Table 1. Description of virtual machines launched in OpenStack

A positive correlation is determined between the number of tasks and the CPU usage, on the other hand, positive correlation is determined between the size of data and memory usage. For example, CPU usage for 500 tasks came out to be 3.6% whereas for 1500 tasks the usage incremented to 4.2%. It increased to 5.0% for 2000 tasks, then again jumped to 7.8% for 4000 tasks as observed. It must be mentioned that CPU and memory usage is monitored for the methods and classes which are impacted due to batch refactoring, the resource usage by the other parts of the code are ignored within the scope of this research. It can be determined that CPU resource usage is dependent on the number of tasks. After refactoring code smell, memory usage for 500, 1500, 2000, and 4000 tasks came out to be 2.9%, 3.8%, 4.3%, and 7.4% respectively.

4 Experimental setup

In this section, the experimental process of *URegM* approach is discussed. Initially, the experimental environment is highlighted, followed by discussion on the obtained results.

Tasks	Predicted % of CPU	Actual % of CPU
500	3.6	3.8
1000	4.0	4.1
1500	4.2	4.6
2000	5	5.3
2500	5.9	5.9
3000	6.6	6.7
3500	7.3	7.6
4000	7.8	8.2

Table 2. Prediction of CPU usage

Tasks	Predicted % of memory	Actual % of memory
500	2.9	3.4
1000	3.4	3.7
1500	3.8	3.10
2000	4.3	4.5
2500	4.9	4.16
3000	6.2	6.9
3500	6.12	6.28
4000	7.4	7.9

Table 3. Prediction of memory usage

4.0.1 Experimental environment.

We run multiple tasks of *WildFireDB* in cloud virtual machines to generate the workload and test the resource consumption before and after refactoring. *OpenStack* cloud platform is used to run the workload of *WildFireDB. OpenStack* is setup in *HP Proliant DL380P* server which had 8 cores and 32 GB RAM. We launched 6 virtual machine instances using kernel virtual machine (kvm) based virtualisation. The 6 virtual machine instances were setup and the workload was executed in those. The virtual machine instances were heterogeneous. We justify the obtained results of resource consumption prediction by running the workload in cloud environment.

The 6 virtual machines are described in Table 1. It is seen from the properties of the virtual machines that they have different configurations. This simulates the distributed environment in which the applications run in cloud. This also helps us to generate a real life scenario and so we can proceed to obtain the resource usage of cloud which will reflect real life consumption of CPU and memory. We aim to predict the CPU and memory usage change due to refactoring code smells for low, medium, and high workload. This is because the number of tasks performed by the workload may impact the resource usage. Table 2 highlights the predicted CPU usage change of the different VMs with various workloads before and after eliminating code smells. The various workloads applied are *firesim_500, firesim_1000, firesim_1500, firesim_2000, firesim_2500, firesim_3000, firesim_ 3500,* and *firesim_4000* which are deployed in the 6 virtual machine

Metrics	LiR	PR	LR	RF	REAP	URegM
mse	1.47	0.72	0.56	0.40	0.27	0.21
rmse	1.66	0.94	0.74	0.60	0.37	0.29
accuracy (%)	86.70	90.60	88.91	93.31	95.41	96.22
time (s)	3.60	1.54	1.67	1.89	0.48	0.33

Table 4. Performance of the ML models. (LiR:=Linear Regression; PR:= Polynomial Regression; LR:= Lasso Regression; RF:= Random Forest; REAP:= Regressive Ensemble Approach for Prediction); URegM (Unified Regressive Model)

instances starting *VMid_1* to *VMid_6*. This covers all possible situations in which the resource usage prediction models may face in real life cloud environment. Each workload was executed 6 times and the average CPU and memory usage was calculated. In Table 3 we identify the memory usage difference before and after refactoring code smells of the class files where the code smells were refactored. We proceeded on analyzing the impact of refactoring 5 code smell types namely *god class, god method, cyclic dependency, long parameter,* and *spaghetti code.*

4.0.2 Results Analysis. The outcome of extensive experimental analysis of the proposed approach is described in this section. The unified model is applied so that it can combine the strengths of the individual models and obtain effective performance of predicting resource usage changes due to code smell refactoring. Initially, we analyze the performance of the individual regression models. Next, we assess the performance of the ensemble model. The experimental results are shown in Table 4. Among the various regression models, we selected these 4 models to form the unified model based on their effectiveness and ease of implementation using the Algorithm 1 shown in this chapter. The *REAP* model is an ensembled model for prediction which is also executed on the dataset to compare the performance of our proposed framework.

It is seen that individual models do not perform to the extent of the unified approach. The performance of the models depend on the training dataset which is used in the *WildFireDB* workload. With the change in dataset, the selected evaluation metrics may show different results. It is seen that individual model may have better error rate than unified models, however, they have worse accuracy and execution time. To obtain the best performance, the individual algorithms are combined together to form the unified model. The accuracy of the proposed algorithm is seen to be highest at 96.22% and it has an average execution time of 0.33 seconds. The overall performance in terms of accuracy is increased by 1.8% and the execution time is decreased by 14.2%. Hence it can be concluded that for this task, *URegM* outperforms the individual regression models. The results of the prediction technique is promising and it shows that estimation of resource consumption due to refactoring code smells even before those smells are refactored will help the software engineers decide on the refactoring. The approach can be applied for other workloads besides the *WildFireDB*. As a result, this study helps the correct resource provisioning and decisions for large scientific applications in cloud computing environment.

5 Conclusion

In this paper, we proposed a method to predict change in computing resource requirements due to the refactoring of cloud source code. High accuracy of CPU and memory usage change prediction due to refactoring code smells was obtained by *URegM* which used *GA* to remove noise and irrelevant information from the dataset. A cloud computing environment was setup with a real-life scientific application to analyze the performance of the intelligent *URegM* algorithm. It was seen that although the individual algorithms were effective, *URegM* outperformed those by improved error rate and faster execution rate. Also, accuracy was improved by 2.3% using *URegM*, whereas the execution time was reduced by 10.9%. The results showed that *URegM* had better accuracy and execution time compared to both the individual learning automata-based approaches and *REAP* model.

In the future, an incremental learning paradigm may be added to enable *URegM* to accommodate the different types of code smells that are refactored by different auto-refactoring tools. This will make the prediction approach more adaptive. Also, we will study the relationship between refactoring code smells and its impact on software complexity [15]. We will also aim to test the performance of the proposed model on other cloud platforms.

Acknowledgments

This project is in part sponsored by the National Science Foundation (NSF) under award numbers OAC-1724898, OAC-1842054 and CCF-2007829.

References

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. 2009. *Above the clouds: A berkeley view of cloud computing.* Technical Report. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

[2] Zhijia Chen, Yuanchang Zhu, Yanqiang Di, and Shaochong Feng. 2015. Self-adaptive prediction of cloud resource demands using ensemble

model and subtractive-fuzzy clustering based fuzzy neural network. *Computational intelligence and neuroscience* 2015 (2015).

[3] Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia, and Alexander Chatzigeorgiou. 2011. JDeodorant: identification and application of extract class refactorings. In *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 1037–1039.

[4] Mostafa Ghobaei-Arani, Sam Jabbehdari, and Mohammad Ali Pourmina. 2018. An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. *Future Generation Computer Systems* 78 (2018), 191–210.

[5] Asif Imran and Tevfik Kosar. 2020. The Impact of Auto-Refactoring Code Smells on the Resource Utilization of Cloud Software. In *The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*, Raúl García-Castro (Ed.). KSI Research Inc., 299–304. https://doi.org/10.18293/SEKE2020-138

[6] Asif Imran and Tevfik Kosar. 2021. The Impact of Human Factors on Software Sustainability. In *Software Sustainability*. Springer, 287–300.

[7] Gurleen Kaur, Anju Bala, and Inderveer Chana. 2019. An intelligent regressive ensemble approach for predicting resource usage in cloud computing. *J. Parallel and Distrib. Comput.* 123 (2019), 1–12.

[8] Maria Kretsou, Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Ignatios Deligiannis, and Vassilis C. Gerogiannis. 2021. Change impact analysis: A systematic mapping study. *Journal of Systems and Software* 174 (2021), 110892. https://doi.org/10.1016/j.jss.2020.110892

[9] Chunhong Liu, Chuanchang Liu, Yanlei Shang, Shiping Chen, Bo Cheng, and Junliang Chen. 2017. An adaptive prediction approach based on workload pattern discrimination in the cloud. *Journal of Network and Computer Applications* 80 (2017), 35–44.

[10] Jae Jin Park, Jang-Eui Hong, and Sang-Ho Lee. 2014. Investigation for Software Power Consumption of Code Refactoring Techniques.. In *SEKE*. 717–722.

[11] Ricardo Pérez-Castillo and Mario Piattini. 2014. Analyzing the harmful effect of god class refactoring on power consumption. *IEEE software* 31, 3 (2014), 48–54.

[12] Chadatarn Raksawat and Pattama Charoenporn. 2021. Software testing system development based on ISO 29119. *Journal of Advances in Information Technology* 12, 2 (2021).

[13] Izzet F Senturk, Ponnuraman Balakrishnan, Anas Abu-Doleh, Kamer Kaya, Qutaibah Malluhi, and Ümit V Çatalyürek. 2018. A resource provisioning framework for bioinformatics applications in multi-cloud environments. *Future Generation Computer Systems* 78 (2018), 379–391.

[14] Rachael Shaw, Enda Howley, and Enda Barrett. 2019. An energy efficient anti-correlated virtual machine placement algorithm using resource usage predictions. *Simulation Modelling Practice and Theory* 93 (2019), 322–342.

[15] Mohammed A Shehab, Yahya M Tashtoush, Wegdan A Hussien, Mohammed N Alandoli, and Yaser Jararweh. 2015. An accumulated cognitive approach to measure software complexity. *Journal of Advances in Information Technology* 6, 1 (2015), 27–34.

[16] Samriddhi Singla, Tina Diao, Ayan Mukhopadhyay, Ahmed Eldawy, Ross Shachter, and Mykel Kochenderfer. 2020. WildfireDB: A Spatio-Temporal Dataset Combining Wildfire Occurrence with Relevant Covariates. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*.

[17] Roberto Verdecchia, René Aparicio Saez, Giuseppe Procaccianti, and Patricia Lago. 2018. Empirical Evaluation of the Energy Impact of Refactoring Code Smells.. In *ICT4S*. 365–383.

[18] Chunyan Wang, Shoichi Hirasawa, Hiroyuki Takizawa, and Hiroaki Kobayashi. 2014. A platform-specific code smell alert system for high performance computing applications. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 652–661.