Usage Patterns to Provision for Scientific Experimentation in Clouds

Eran Chinthaka Withana and Beth Plale
School of Informatics and Computing, Indiana University
Bloomington, Indiana, USA.
{echintha, plale}@cs.indiana.edu

***Abstract*—Driven by the need to provision resources on demand, scientists are turning to commercial and research test-bed Cloud computing resources to run their scientific experiments. Job scheduling on cloud computing resources, unlike earlier platforms, is a balance between throughput and cost of executions. Within this context, we posit that usage patterns can improve the job execution, because these patterns allow a system to plan, stage and optimize scheduling decisions. This paper introduces a novel approach to utilization of user patterns drawn from knowledge-based techniques, to improve execution across a series of active workflows and jobs in cloud computing environments. Using empirical analysis we establish the accuracy of our prediction approach for two different workloads and demonstrate how this knowledge can be used to improve job executions.**

***Keywords*-User Patterns; Knowledge-based Computing; Cloud Computing; Predictions; Scientific Experimentation;**

I. Introduction

Grid computing has addressed large-volume, large-scale science through provisioning of infrastructure that utilizes geographically distributed compute resources. While it has advanced science in significant ways, its strengths, which have supported growth, also serve to limit its use. Grid computing, which often utilizes traditional HPC resources, is largely batch oriented and under even moderate loads can result in long queue times. This is a challenge to scientists working in research areas that require responsiveness to current conditions, such as severe weather. Grid computing resource providers offer best-effort guarantees on the quality of the service of these resources. Too, grid computing makes no claims to access transparency. That is, failures in the underlying infrastructure generally propagate upward to the application for handling. The quota system on CPU usage requires that maximum resource needs be known in advance and approved through a request process. Finally job submission requires a high level of computer science expertise to utilize optimally. Cloud computing addresses some of the limitations, though it introduces others, so has shown promise as a complementary solution to grid computing. Cloud computing resources are rented to users on an on-demand basis and users are billed according to their usage. These resources offer several key advantages for scientific computing [1] that include providing a programmer with a sense of infinite computing resources available the instant they are needed, eliminating the need for advance reservations. The pay-as-you-go model used by cloud providers eliminates an up-front commitment by users. Experiments can be started small and scaled as need grows. The pay-as-you-go model also encourages scientists to budget for and use resources they can and are willing to pay for. On the flip side, cloud computing presents new issues that can be resolved either through additional distributed infrastructure or passed on to the scientist and these include slow interconnects, virtualization startup times, and consumption based billing. Grid computing infrastructures and local departmental clusters employ fast interconnects between nodes that has had the effect of encouraging distributed application design that takes advantage of it. Cloud computing, however, is built on the notion of commodity hardware, high rates of failure, and slow interconnects - a different underlying model entirely. On-demand provisioning of compute resources is enabled by virtualization technologies within cloud computing resources, but these virtualized resources have overheads of their own.

This fundamentally different distributed compute environment could be addressed by redesigning our distributed and parallel applications, and programming models such as MapReduce [2] attempts to do this, but mapreduce is limited in the applications it serves. Services are needed that aid deployment and minimize hidden latencies in order to reduce the total monetary cost of doing research and education with cloud computing. One can also attempt to reduce the level of computer science expertise required in the process. Both of these issues we address through services that deploy an application on a cloud resource. This middleware is commonly called scheduling. Research in meta-scheduling and scheduling algorithms is predominantly focused on improving batch queue times, reducing job execution times and securing resources as and only as needed. With grid computing such as with the TeraGrid [3], queue times can be significant and be dependent on factors such as total system load, wall clock limits on a job, etc. A batch queue by nature implies mediating access to a finite resource. Cloud providers claim to provide access to unlimited resources, or at least support that illusion, so queue times are effectively non-existent. Jobs do however suffer seconds to minutes start up overheads within virtualized environments [4] [5] [6] [7] [8], and thus are a major source of latency that we propose to reduce. Scheduling algorithms also focus on optimal utilization of relatively homogeneous grid or cluster

resources through resource allocation algorithms. Maximal utilization of a resource does not make sense either in a domain of unlimited resources, or at least not a problem that a user needs to deal with. But clouds, particularly when taken across available cloud providers [9] [10] present a heterogeneous set of resources with different hardware and software configurations that can be maximally utilized to a user's advantage.

Prediction algorithms using historical data are employed in scheduling to predict the execution times of an application. On a cluster, this can be done by looking at the history of the application alone. But cloud environments with their different hardware configurations, the execution time predictions must factor in the non-uniformity of hardware resources. Grid or time-shared resources are designed to allocate a large fraction of computational resources to be consumed to a single job or user [11] whereas in cloud computing systems only a tiny fraction of total resource availability is being used by a given user or a job at any given time. Cloud schedulers must in response scale to support large number of simultaneous users. Too, the emergence of the semi-structured workflow as a way of bringing the human into the loop of an investigative process results in more smaller-scale and data dependent workflows that are seen in large-scale computations. These workflows present a challenge to the body of work on workflow planning and execution that we discuss in the related work section. Finally, grid computing or time-shared resources provide an abstraction of federated resources [11] in the form of heterogeneous hardware, network and software resources across different administrative domains bound together in support of virtual organizations where the administrative domain supports authentication but membership and authorization are at the virtual organization level. But federated resources or virtual organizations for that matter, are not provided for in a cloud setting.

The broader vision of our research is on giving eScience applications options to achieving low cost execution that minimizes turnaround time and maximizes the successful completion rate over set workflows where the workflow set is drawn from multiple users submitting workflows over a fixed period of time. The solution has two primary pieces. The first is a meta-scheduler that draws from AppleS [12] and GRADS [13] in its use of historical information to anticipate future activity. The second is a resource abstraction service, which shares aspects in common with Nimrod/G [14].

In this paper our focus is on the former and propose an algorithm to predict future jobs by extracting user patterns from historical information. These predictions will, we posit, reduce the impact of high startup overheads for the class of applications that are time-critical. We use knowledge-based techniques to extract patterns from a knowledge-base, which can start either from zero knowledge or pre-populated job information consisting of connections between jobs. The similar cases retrieved from the knowledge-base will be used to predict the future jobs, helping to reduce or hide the impact of startup overhead. We assess our algorithm using two different workloads representing individual scientific jobs executed in at a Los Alamos National Lab supercomputer and the other from set of experiments submitted by three users using five workloads. For these latter workloads we establish the accuracy of our predictions and the amount of time that can be saved in these job executions.

The remainder of this paper is organized as follows. Section II discusses related work and Section III focuses on the formal description of our algorithm. Section IV describes the system implementation and Section V lays out the performance evaluation. Section VI concludes and discusses future work.

II. Related Work

Scientists have in the past typically submitted jobs to batch queues and then waited for their eventual execution on a shared computational resource. But since cloud computing shifts this trend to acquire computational resources on demand, scientists have evaluated cloud computing resources for scientific job executions [15] [16] [17] [18] [19]. Prediction algorithms have been used to improve job execution in large-scale computing resources. These algorithms concentrate on prediction of execution times [20], job start times [21], queue-wait times [22] and resource requirements [23]. Prediction algorithms used to improve job executions in time-shared resources and cloud computing resources can be widely classified into two categories, namely statistical model based and artificial intelligence (AI) based algorithms. The statistical methods mainly focus on formulating a model and then deriving resulting statistical properties of a prediction parameter(s) through simulations. Previous work on predictions based on statistical analysis and modelling methods help to improve the job executions on time-shared resources. Work on this area concentrated on predicting execution times [24], job start times [21], queue wait times [25] [26] and resource requirements [21]. AppleS [12] in particular argues that a good scheduler must involve some prediction of application and system performance. The predictive models in AppleS have been used to evaluate and rank the schedule candidates. Ganapathi et al, [27] propose a statistical model to predict the resource requirements and execution times for Cloud computing applications. Inspired by the statistical techniques to predict query performance in parallel databases [28], Ganapathi et al, predict the performance of MapReduce [2] jobs by correlating pre-execution features and post-execution performance metrics. Artificial intelligence methods focus on learning from past experiences or using multiple exhaustive search algorithms together with heuristics to build a system for predictions. There is previous work related to prediction of execution times in time-shared resources [20] [29] [30], and prediction of job start times, queue wait times [22] [30] and resource requirements [31], using historical information. These methods use techniques including instance-based learning [32], genetic algorithms

[33], case-based learning [34], etc., together with heuristics for search and optimization. Hui et al, [29] proposes an instance-based learning [32] technique to predict application run times and queue wait times of large-scale grids from mining historical workloads. Most of the grid or time-shared scheduling techniques will not work in cloud computing environments without re-examination because they optimize metrics that are not the primary causes of latency in a cloud environment. Our work is different from the above AI approaches in that we use knowledge-based techniques to predict the next jobs to be executed in a cloud computing environment with the goal of improving the total execution time of a suite of workflows active at any one instant. To our understanding there has been no work on predicting next jobs to execute for cloud computing environments using knowledge-based or artificial intelligence methods. Matsuoka et al, [35] proposes a similar approach to improve execution times of grid applications by overlapping the execution of data staging with compute bound tasks. This work complements ours.

III. Conceptual Model for Prediction

A. Time Critical Scientific Experimentation

The scientific experimentation process often involves a large number of workflows. For example, in the LEAD project [36], meteorology researchers might approach a day's investigations by running one or two short-term workflows to identify interesting events in the atmosphere. They then will launch a suite of workflows, for instance, a parameter sweep, to understand the model behaviour under a particularly mesoscale phenomena. In response the system examines this as a collection of workflows that can be optimized. Also the usage of the same workflow or experiment, as well as a suite of workflows, can differ from domain to domain. For example, a meteorologist running a forecast workflow might run another forecast workflow over a small region and shorter time frame. But an agriculture researcher attempting to make predictions of growth or the harvest yield of an area, might run a single weather forecast run followed by a different set of jobs/workflows after the forecasting workflow. These examples are time-critical in that they require access to the latest environmental observational data at time of execution, and must begin immediately.

B. Formal Analysis

Suppose a sequence of jobs form the critical path of a scientific experiment. As shown in Figure 1, there are $N+1$ jobs to be executed and the total experiment takes T_∞ time to complete. Each job has to be set up before being executed. For example, if a job has to be setup to run inside a virtualized environment, the virtual machine (VM) must be started and proper services have to be deployed and started before being executed. Defining this startup overhead and the execution time of the ith job as s_i and t_i, respectively, one gets T_∞ as the following:

$$T_\infty = \sum_{i=0}^{N}(s_i + t_i) = \sum_{i=0}^{N} s_i + \sum_{i=0}^{N} t_i$$

Fig. 1: Serialized execution of n jobs

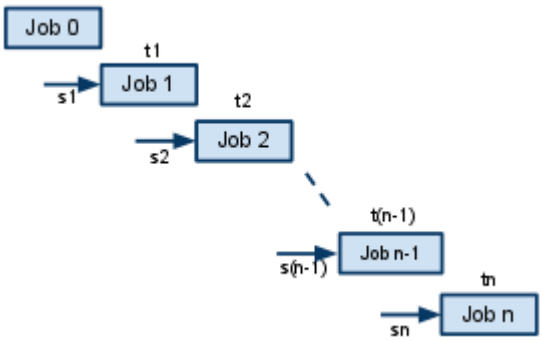

Fig. 2: Serialized execution of jobs, with advanced acquisition and preparation

Suppose further that the probability of successfully predicting the next job is r, where $(0 < r < 1)$ and unsuccessful prediction is $(1-r)$. Total experiment time (T_∞) has $(1-r)$ chance of having $\sum_{i=0}^{n} s_i$ component in it thus giving the T_∞ as:

$$T_\infty = (1-r)\sum_{i=0}^{N} s_i + \sum_{i=0}^{N} t_i$$

$$T_\infty = \sum_{i=0}^{N} s_i + \sum_{i=0}^{N} t_i - r\sum_{i=0}^{N} s_i \quad (1)$$

The insight from equation (1) is that the upper bound of the execution time for the DAG is reduced by $r\sum_{i=0}^{N} s_i$. The time savings depends upon the critical path (N), the set-up time s_i and more importantly, the accuracy of the prediction r. If predictions can be made accurately, then the serialized execution of jobs in the critical path (Figure 1) can be changed to overlapped job execution and set-up of the environments (Figure 2).

We define percentage time reduction as,

$$\text{Percentage time reduction} = \frac{r\sum_{i=0}^{N} s_i}{\sum_{i=0}^{N}(s_i + t_i)}$$

For simplicity, assuming equal job execution times and set-up times for all the jobs,

$$\text{Percentage time reduction} = \frac{r*(s*N)}{(t+s)*N} = \frac{r*s}{(t+s)} = \frac{r}{\frac{t}{s}+1}$$

Figure 3 demonstrates the variation of percentage time reduction against work-to-overhead ratio (t/s), for different levels of prediction accuracies. This figure helps to determine the prediction accuracy needed to gain the required time reduction at a given work-to-overhead ratio. For example, when work-to-overhead ratio is 2.5, the system needs to have

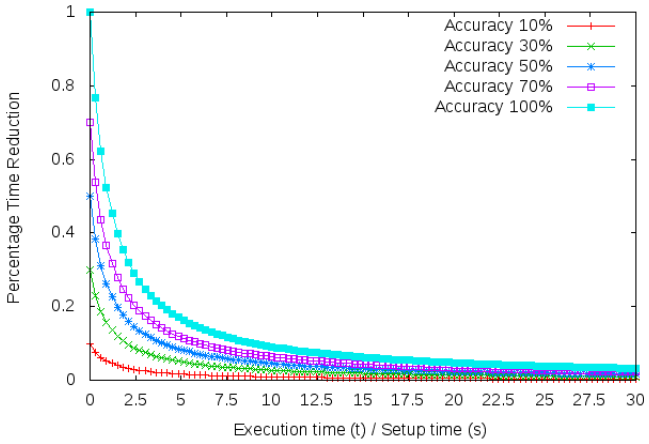

Fig. 3: Variation of percentage time reduction with respect to execution time to set-up time ratio for different prediction accuracy levels

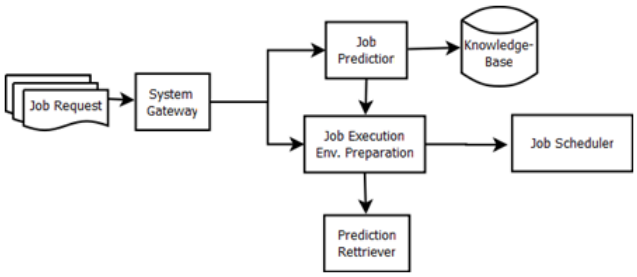

Fig. 4: System Architecture

at least 70% accuracy in predictions to achieve 20% reduction in total time. But, the amount of time reduction that can be gained is reduced exponentially with the increase of work-to-overhead ratio. Also with a given work-to-overhead ratio, the percentage of time reduction increases with the prediction accuracy levels.

IV. System

A. Architecture

Our prediction engine is middleware that sits between a user and a cloud scheduler. The prediction engine we propose can also be used as a gateway for a community of users interacting with cloud-based systems, because it is scientific domain independent. Figure 4 shows the architecture of our system. Following is a listing of the steps taken to handle a job request coming in to the system.

1) When a job request enters the system, it will start two independent tasks, one to check whether the job already has prediction information (Job Execution Environment Preparation) and the second task to predict the next job that will be coming after this job request (Job Predictor).
   a) If this job is already predicted, then the relevant details are retrieved and handed over to the cloud scheduler service.
   b) If the job is not predicted, the system will start two independent parallel tasks.
      i) task 1 : hands over the job to the scheduler to prepare the environment and execute the job
      ii) task 2 : will try to find out the reason for system not being able to predict the job
2) The job prediction component predicts the next expected job by extracting user patterns from historical information, using a Case-based reasoning approach.
3) The prediction component, after receiving a prediction, notifies the scheduler to prepare the environment for the predicted job and saves enough data in the system so that the prepared environment can be retrieved when the actual job arrives.

1) Case-based Reasoning System: Our system uses Case-based reasoning (CBR) [34] to store and retrieve similar cases from the system. The logical reasoning model inside CBR is closer to how humans use past experiences to face new tasks. It uses a problem resolutions and a learning process. It first retrieves similar cases from the knowledge-base and then reuses the retrieved older cases to find a solution. Adaptation of a retrieved case to the current problem is the next step. Once it is adapted, the new case is stored in the system to retrieve later. Architecture of the CBR system inside our system is shown in 5.During the retrieval process, mentioned above, CBR system employs a similarity score to give a value to the similarity between two cases (discussed in Section IV-A2). Depending on the score and a given threshold, the system will retrieve 0 or more cases. This system will minimize the wrong predictions because if the system cannot find a similar case with the given similarity threshold, system will refrain predicting from next jobs. Once a case is selected, out of the provided solutions, the system will look at the next jobs associated with the selected case. Based on a probability model, the highest probable next application of the retrieved solution is predicted as the next application of the current job. When the actual job arrives, system saves the actual job as the next job of the previous job updating the probability values. For example, if application B and C are executed 3 times and 7 times respectively after application A, then $p(B|A)$ = 0.3 and $p(C|A)$ = 0.7. Then the system will predict application C as the highest probable application that will be following application A. If this prediction turns out to be true, compared to the actual case coming in to the system next, $P(C|A)$ will increase to $\frac{8}{11}$ = 0.73 and $p(B|A)$ will be reduced to 0.27. If the prediction turns out to be wrong, the $p(C|A)$ and $p(B|A)$ will change to 0.64 and 0.36 respectively, increasing the probability of application B. Because of the probability model being used, previous errors will improve the accuracy of our predictions. In our system, meta-data from a previous job execution is considered a case. CBR is ideal for our job prediction algorithm, because it enables to identify prominent set of parameters to be used and uses the similarity between these parameters to select the similar cases, yielding best matches. This section explains the case organization and how cases are retrieved based on the similarity. This approach is similar to the approach defined in here [30]. Two memory organizations of cases are used in our system. In flat memory organization of cases, all cases are at the same level and

nearest neighbour algorithm will be used to find similar cases. In structured memory organization, decision trees or B-Trees are used for case organization. Even though flat memory organization is always guaranteed to provide the best case match, it requires more computation time. On the other hand in complex tree structures, even though less computation time is required, the best matching case is not guaranteed to be retrieved. Our CBR system design, shown in Figure5, is based on the PredCase flow diagram defined in Nassif2010 [30]. A new job coming in to the system is encapsulated within a Problem object. These problem objects contain the parameters to be used during the similarity evaluation phase. Then the CBR system evaluates the similarity of this case with the ActualCase objects existing inside the knowledge-base. These ActualCase objects contain meta-data about the previous job executions. Since this evaluation happens in the run time, when a new job is received, the system can suffer from performance issues if a flat memory hierarchy is used. So it is important to switch to the structural memory hierarchy, when the case base grows beyond a certain level. After the similarity evaluations (explained in section 5.2), CBR system returns a list of ActualCase objects to the prediction system. Prediction system, then selects a best case out of the returned cases. The naive way to select a single case from the return solutions is to select the case with the highest similarity score. Then the system looks at the next cases of the most similar case. Prediction system selects the highest occurring case, from the next cases, and predicts that as the next case that is coming from the user, and hands over details about the next case to Scheduler to prepare itself for the next case. Since we used username and the parameters that user used previously, as features during the similarity calculation, we deduce that the selected case is based on user patterns.

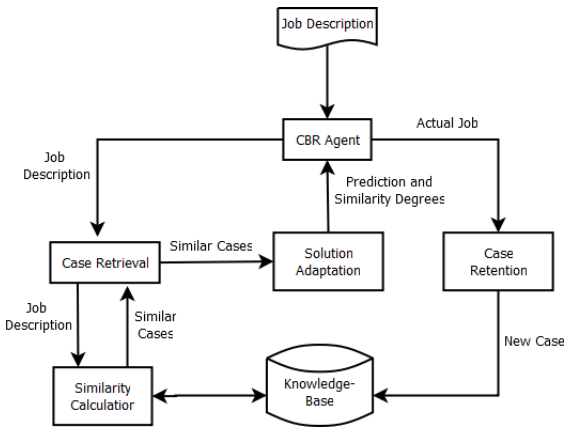

Fig. 5: CBR system (adapted from PredCase flow diagram in Nassif2010 [30])

2) Similarity Measurement: Each job contains a set of attributes (user id, application name, data sizes, memory usage, number of processors, etc.,) to uniquely identify itself. Determining the similarity between two cases requires comparing each and every attribute in two given cases. During the case retrieval phase, similarity between attributes of two cases is compared to obtain local similarity, which in turn is weighted to obtain the global similarity measure. The weighting ensures the emphasis on correct attributes during the similarity measurement and heuristics are used to come up with correct weights. Similarity between two cases A and B is defined as

$$Sim(A,B) = \sum_{i=0}^{n} \omega_i(a_i b_i)$$

where,
a_i and b_i are corresponding attributes from case A and B
$sim(a_i b_i)$ is the local similarity between attributed a_i and b_i
ω_i is the associated weight for the attribute.

The selection of the case depends on this calculated similarity being compared against a given threshold level (T). For a case to be selected it should have the highest similarity score with respect to the given Case G, among all the other cases and this score should be higher than the given threshold. i.e. a case S is selected iff

$$\forall X \neq S, Sim(X,G) < Sim(S,G)$$

and

$$Sim(S,G) > Threshold$$

Commonly used similarity calculation methods can be found here [37]. Local similarity calculation is not the same for all the attributes. During the local similarity score calculations, values of the attributes, inside cases, can range from string literals to numeric to decimals. Also the outcome of a comparison can be discrete or continuous values. For example, matching of user name is always a true/false answer. But matching data sizes is not. Out of the job attributes defined in one of our workloads, we carried out an initial evaluation to find the most significant attributes that contribute to the similarity measure. Results of this evaluation helped us to identify 6 most significant attributes to describe a job in the system. Starting with random set of weights for these attributes, the weights were tuned manually to obtain the best similarity scores. But when we increase the number of attributes in the job to uniquely represent and distinguish jobs, we are planning to use machine learning techniques to learn the weights.

V. Evaluation

A. Assumptions

We make the following assumptions about the job workload coming in to the system and the cloud scheduler used to schedule jobs.

1) The execution of a task completely uses the virtual machine(s) assigned to the job and no other jobs can be executed on the same node concurrently.
2) Our framework has unlimited access to the cloud computing provider resources and our system can use any number of virtual machines at a given point of time.
3) Compared to the task execution time, the resource preparation time is significant.

User	Workflows in the experiment
User 1	Workflow 1, Workflow 2, Workflow 5
User 2	Workflow 2, Workflow 4
User 3	Workflow 2, Workflow 3, Workflow 4

TABLE I: Workflows included inside user experiments

4) The number of unique applications inside the workload is low compared to the number of total jobs in the workload
5) The job execution times are known before hand (this helps to approximate the work-to-overhead ratio).

In our evaluation of the prediction algorithm, based on user patterns, we focus on the following use cases.

B. Use Cases

We use two workloads to represent the two methods of job submissions. **Individual Jobs Workload:** We obtained a job log [38] containing two years (October '94 to September '96) of accounting records produced by the DJM software running on the 1024-node CM-5 at Lost Alamos National Lab, listed in Parallel Workload Archive [38], and submitted those jobs to our prediction system. We used the first 40000 job submissions which contained 762 unique applications being invoked. **Workflow Use Case:** We defined three experiments, each containing 2-3 workflows, executed by three different users. Figure 6 shows the workflows used and Table I shows the workflows contained in the experiments. There were 25 unique applications inside these workflows.

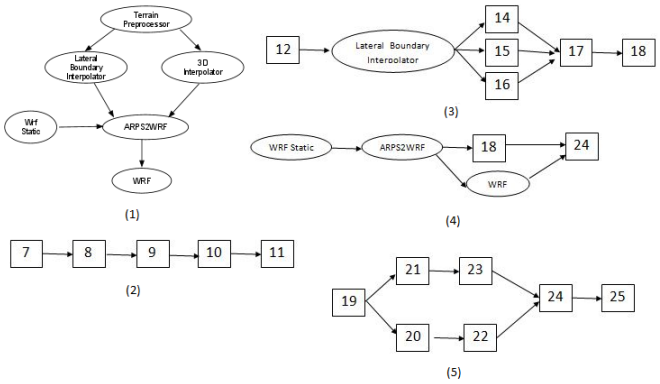

Fig. 6: Workflows Used in Evaluation

We ran these experiments 100 times, for each user. The workflow simulator we wrote, to facilitate the evaluation, takes each workflow from the experiment and submits the jobs inside the workflow to our prediction system. Given the current job, if the system has predicted this job, then the prediction is marked as accurate. A wrapper written for the job prediction system records the accuracy of predictions for each and every job submitted to the system. It counts the number correct predictions, up to that point and records that as a percentage, relative to the total number of jobs, with the job number.

To compare the predictions our system, we also implemented a predictions system based on the service name, without looking at the user. Services coming after each and every service are recorded, and when a new service arrives to the system, the service with maximum occurrence to the new services, retrieved from earlier cases, is predicted as the next case.

All the evaluations were run inside a windows box with a 2.0GHz dual-core processor, 4GB memory and on a 64-bit operating system.

Average Accuracy of Predictions

Figure 7 shows the results of our experiment, when ran against the individual jobs retrieved from the parallel workloads [38]. For this workload, our system has saturated around 78% accurate predictions, while service name based predictions could only saturate around 30% accuracy, making the predictions of our system close to 2.5 times better than base case (Figure 11).

Fig. 7: Individual jobs workload

Fig. 8: Workflow workload

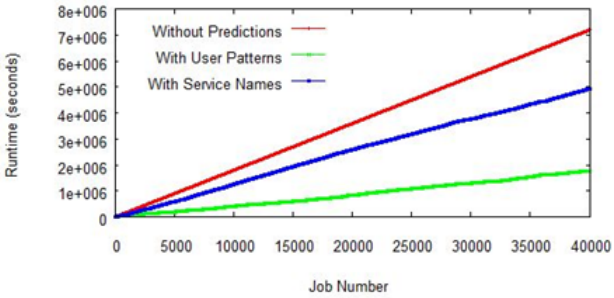

Fig. 9: Time saved for single job use case

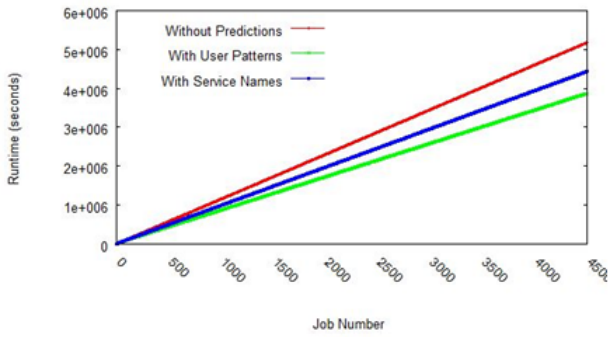

Fig. 10: Time saved for workflows use case

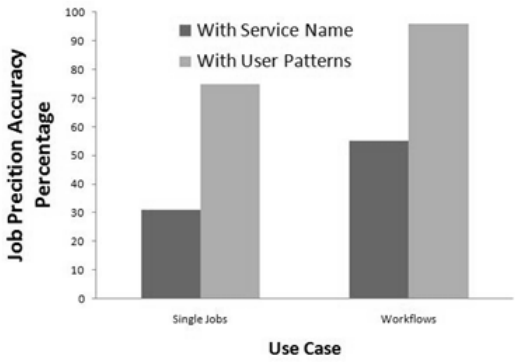

Fig. 11: Prediction accuracies for use cases

Figure 8 shows the results of our experiment, when ran against the workflow workload. For this workload, our system has saturated around 96% accurate predictions, while service name based predictions could only saturate around 54% making the predictions of our system close to 2 times better than the base case (Figure 11). As can be seen in both the graphs, it takes certain number of cases to get in to an acceptable level of prediction accuracy. The upward steep curve shows the learning period of the system, and once the graph get in to a horizontal line, the system has learned enough information to predict the jobs. The number of jobs required to get in to acceptable level of accuracy is different from one job mix to the other.

Time Saved Figure 9 and Figure 10 show the total run time of jobs executed with predictions, with predictions using service name based predictions and with predictions using user pattern based predictions. For this experiment it is assumed that the set-up time for a job is 3 minutes (this is a close approximation to the average time it takes to reserve computational resources in Amazon EC2).

Single jobs use case has saved more time because of the predictions, because the set-up time is significant compared to the average job execution times. With a prediction accuracy of 78%, our prediction system could reduce the total execution time by a factor of 2.5. The maximum time for a job is 4870s and 78 seconds for the smallest job with 984 seconds being the average for jobs inside our workflow use case. When the system starts learning, it slowly improves the prediction accuracy saving more time in the scientific experiment. At the end of the simulation of 100 job runs, our prediction framework had saved 206 hours in total job execution time. Figure 11 shows the summary results of this experiment. For single jobs, which are coming in to a system, user patterns based system could perform 2.5 times better than the service name based predictions. For the workflow use case our proposed system is almost 2 times better.

VI. Discussion and Future Work

This paper presents a new framework for efficiently scheduling scientific jobs in to Cloud computing systems, exploiting the usage patterns embedded inside historical information. It uses CBR techniques to retrieve user patterns from a knowledge base.

Initial evaluation results show that the system is efficient for both individual jobs, with around 78% accuracy, and for workflow based systems, with around 96% accuracy. The number of jobs required to get in to the stable level of accuracy depends the uniqueness of the applications in the workload. If there are more unique applications in the workload, it will take more number of cases to get in to the highest level of accuracy, because of the learning rate. From Figure 3, one can find out the level of accuracy needed to achieve the expected amount of time reduction which in turn can be used to find the number of cases required to get the expected level of accuracy. Also as shown in Figure 3, the effectiveness of the system depends on the characteristics of the job mix and the amount of time reduction needed by a user. For example, we also used HPC2N and OSC data sets from parallel workloads archive [38] and we could achieve 90% prediction accuracy within first 100 jobs and this number is improved to 98% with more cases.

This work focuses on efficiently utilizing cloud resources to minimize scientific job execution. But we plan to improve our system on following directions to improve the effectiveness of job executions in clouds. Current similarity measurement calculations required manual turning of weight assignments. But we will also evaluate the use of machine learning techniques to classify the cases in the knowledge-base. The trained classifier will improve the similarity calculation and will in turn will improve the accuracy of predictions. Second, we want to use CBR techniques to optimize on the cost, time and also reliability of executions - we believe that a similar approach can be used to retrieve previous cases and predict this quality of service parameters of current cases, using CBR techniques. Third, we want to extend our work on resource abstraction layer [39], the job management system for time-shared systems, to provide a common level of abstraction to interact with multiple cloud providers - different cloud providers have their own interfaces to provision computational resources, move and store data and carry out job executions. But if a middleware can hide the complexities between these cloud providers and time-shared resources, then the scheduler can pick different providers, depending on a given set of optimization parameters and run jobs on multiple providers, without worrying about the complexities.

ACKNOWLEDGEMENT

This work is funded in part by NSF CNS 0720580. The authors would like to thank Dennis Gannon (Microsoft Research), David Leake and Chathura Herath (Indiana University) for their invaluable inputs to this research and Dror Feitelson for archiving workload logs on parallel machines [38].

REFERENCES

[1] M. Armbrust *et al.*, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] C. Catlett, "The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility," in *ACM International Symposium on Cluster Computing and the Grid*. Published by the IEEE Computer Society, 2002.

[4] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. Sprenkle, "Dynamic virtual clusters in a grid site manager." in *HPDC*. IEEE Computer Society, 2003, pp. 90–103.

[5] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, "A case for grid computing on virtual machines," in *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2003, p. 550.

[6] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual clusters for grid communities," in *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 513–520.

[7] K. Keahey, T. Freeman, J. Lauret, and D. Olson, "Virtual workspaces for scientific applications," *Journal of Physics: Conference Series*, vol. 78, p. 012038 (5pp), 2007.

[8] B. Sotomayor, K. Keahey, and I. Foster, "Overhead matters: A model for virtual resource management," in *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2006, p. 5.

[9] "Amazon elastic computing cloud," http://aws.amazon.com/ec2/.

[10] "Windows azure platform," http://www.microsoft.com/windowsazure/.

[11] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid-Volume 00*. IEEE Computer Society, 2009, pp. 124–131.

[12] F. Berman *et al.*, "Adaptive computing on the grid using apples," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 369–382, 2003.

[13] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crumme *et al.*, "The GrADS project: Software support for high-level grid application development," *International Journal of High Performance Computing Applications*, vol. 15, no. 4, p. 327, 2001.

[14] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid," in *hpc*. Published by the IEEE Computer Society, 2000, p. 283.

[15] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," *eScience, IEEE International Conference on*, vol. 0, pp. 640–645, 2008.

[16] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," *Cloud Computing and Applications*, vol. 2008, 2008.

[17] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 140–147.

[18] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[19] D. J. Abadi, "Data management in the cloud: Limitations and opportunities," *IEEE Data Eng. Bull.*, vol. 32, pp. 3–12, 2009.

[20] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," in *Job Scheduling Strategies for Parallel Processing*. Springer, p. 122.

[21] H. Li, D. Groep, J. Templon, and L. Wolters, "Predicting job start times on clusters," in *ccgrid*. IEEE, 2004, pp. 301–308.

[22] D. Nurmi, J. Brevik, and R. Wolski, "QBETS: Queue bounds estimation from time series," in *Job Scheduling Strategies for Parallel Processing*. Springer, pp. 76–101.

[23] A. A. Julian, J. Bunn, R. Cavanaugh, F. V. Lingen, M. A. Mehmood, H. Newman, C. Steenberg, and I. Willers, "Predicting the resource requirements of a job submission arshadali," in *In Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP 2004*, 2004, p. 273.

[24] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-level scheduling on distributed heterogeneous networks," in *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*. IEEE Computer Society, 1996, p. 39.

[25] A. B. Downey, "Predicting queue times on space-sharing parallel computers," in *IPPS '97: Proceedings of the 11th International Symposium on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 209–218.

[26] A. Downey, "Using queue time predictions for processor allocation," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1997, vol. 1291, pp. 35–57.

[27] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-Driven Workload Modeling for the Cloud," Technical Report UCB/EECS-2009-160, EECS Department, University of California, Berkeley, Tech. Rep., 2009.

[28] A. Ganapathi, H. Kuno, U. Dayal, J. Wiener, A. Fox, M. Jordan, and D. Patterson, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2009.

[29] H. Li, D. Groep, and L. Wolters, "Efficient response time predictions by exploiting application and resource state similarities," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, 2005, pp. 234–241.

[30] L. N. Nassif, J. M. Nogueira, A. Karmouch, M. Ahmed, and F. V. de Andrade, "Job completion prediction using case-based reasoning for grid computing environments: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 19, no. 9, pp. 1253–1269, 2007.

[31] A. Ali, A. Anjum, J. Bunn, R. Cavanaugh, F. Van Lingen, R. McClatchey, M. Mehmood, H. Newman, C. Steenberg, M. Thomas *et al.*, "Predicting the resource requirements of a job submission," *Computing in High Energy Physics, Interlaken, Switzerland*, 2004.

[32] D. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[33] D. Goldberg, *Genetic Algorithms in Search and Optimization*. Addisonwesley, 1989.

[34] D. B. Leake, *Case-Based Reasoning: Experiences, Lessons and Future Directions*. Cambridge, MA, USA: MIT Press, 1996.

[35] S. Matsuoka, N. Sarai, S. Kuratomi, K. Ono, and A. Noma, "Role of individual ionic current systems in ventricular cells hypothesized by a model study," *The Japanese journal of physiology*, vol. 53, no. 2, pp. 105–123, 2003.

[36] K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves *et al.*, "Service-oriented environments for dynamically interacting with mesoscale weather," *Computing in Science & Engineering*, vol. 7, no. 6, pp. 12–29, 2005.

[37] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI communications*, vol. 7, no. 1, pp. 39–59, 1994.

[38] D. Feitelson, "Parallel workload archive," http://www.cs.huji.ac.il/labs/parallel/workload/.

[39] E. Chinthaka, S. Marru, and B. Plale, "Sigiri: Towards a lightweight job management system for large scale systems," School of Informatics and Computing, Indiana University, Bloomington, Indiana, Tech. Rep. TR681, 2009, http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR681.