An Iterative Optimization Framework for Adaptive Workflow Management in Computational Clouds

LONG WANG[1], RUBING DUAN[1], XIAORONG LI[1], SIFEI LU[1], TERENCE HUNG[1], RODRIGO CALHEIROS[2], AND RAJKUMAR BUYYA[2]

[1]Institute of High Performance Computing, A*STAR, Singapore
[2]The University of Melbourne, Australia

***Abstract*—As more and more data can be generated at a faster-than-ever rate nowadays, it becomes a challenge to processing large volumes of data for complex data analysis. In order to address performance and cost issues of big data processing on clouds, we present a novel design of adaptive workflow management system which includes an SVM (Support Vector Machine) based prediction model, workflow scheduler, and iteration controls to optimize the data processing via iterative workflow tasks. We proposed a new heuristic algorithm, called Upgrade Fit, which dynamically and continuously reallocates multiple types of cloud resources to fulfill the performance and cost requirements. The iterative workflow tasks can be bursty bags of tasks to be executed repetitively for data processing. A real application of weather forecast workflow has been used to evaluate the capability of our system for large volume image data processing. Experimental system has been set up and the results indicate that the system can effectively handle multiple types of cloud resources and optimize the performance iteratively.**

I. INTRODUCTION

With the advent of parallel computing and service technologies, data analysis applications become more and more complex. It requires sophisticated controls of the parallel tasks and the data flows to enable big data to be processed on a large pool of distributed resources. Cloud computing is a scalable, secure and cost-effective utility that is widely accepted due to its convenience and pay-as-you-go model. It is characterized by rapid yet elastic demand pools, which are suitable for data-and compute-intensive workflow management systems (WfMS) [1]. Different from heterogeneous grid computing environments that are generally owned by different communities or organization with varied administration policies and capabilities, cloud computing can provide a scalable and flexible platform to satisfy high performance computing requirements.

Workflow is concerned with the automation of procedures whereby files and data are passed between participants according to a defined set of rules to achieve an overall goal. A workflow management system defines, manages, and executes workflows on computing resources, which should have the ability to build dynamic applications that orchestrate multiple tasks to be processed on distributed resources. A workflow is composed by connecting multiple tasks according to their dependencies. Workflow structure can be represented as a Direct Acyclic Graph (DAG) and non-DAG. Non-DAG has the same entities as DAG, but expands to one further structure – iteration, which is also known as loop or cycle. The iteration structure is quite frequently used in scientific applications, where one or more tasks need to be executed repeatedly. Iteration is a nature property that is highly practical in data analysis, and it gives system an opportunity to gather some related information and optimize resource pool and scheduling algorithm in term of cost and performance.

In order to execute iterative workflow tasks for data analysis, we proposed a novel workflow management system which handles diverse computing resources with dynamic resource provisioning strategy to process large volume data and optimize the performance periodically. We designed a SVM (Support Vector Machine) based prediction model to model the data-intensive tasks. Most importantly, we formulate the scheduling problem as a variation of multi-type bin-packing problem, and proposed a new approach, called *Upgrade Fit* algorithm, to extend the existing capability of workflow management systems to allow iterative optimization of workflow execution. Experimental results show that our new algorithm outperforms most existing algorithms in terms of economic cost and resource efficiency.

To illustrate the ability of our system, we used a weather forecast application that processes a large amount of radar cloud image data to analyze rainfall distribution. The workflow is composed of many relational tasks, which repeat over time to extract information from different data sets. Our proposed framework can improve the performance of running such data- and compute-intensive analytic workflow in clouds with less execution time and lower economic cost.

The rest of this paper is organized as follows: Section II presents an overview of the architectural design of the workflow management system. Section III presents the proposed approach for cloud resource allocation and workflow scheduling. Section IV evaluates our methods for a real-world weather forecasting application executed in a real computational cloud, and a formal approach to predict workflow performance. We compare our approach with state-of-the-art and other traditional approaches in Section V, and finally we conclude in Section VI.

II. SYSTEM AND APPLICATION OVERVIEW

This section provides a comprehensive view of the iterative workflow management system and an typical iterative application running on it.

Corresponding author. *Tel*: +6564191551; *Fax*: +6564631452;
E-mail: duanr@ihpc.a-star.edu.sg (Rubing Duan).

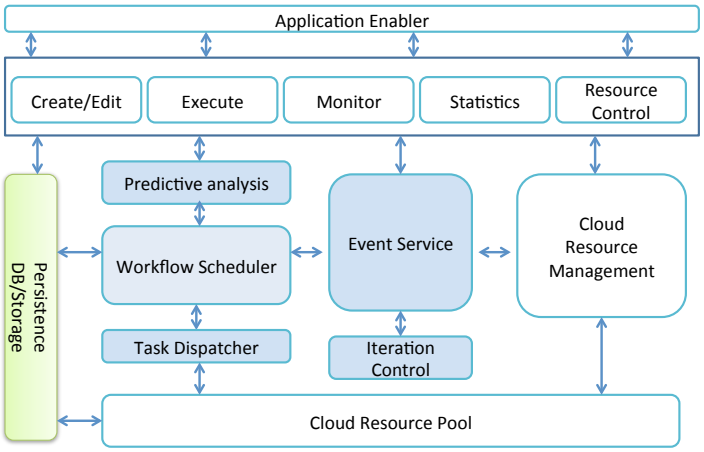


Fig. 1. System Architecture.

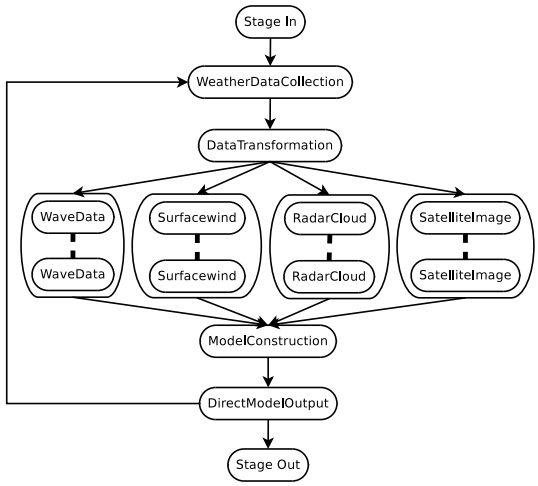


Fig. 2. Weather forecast workflow.

A. System Architecture

The architecture of our workflow management system is depicted in Figure 1. The top layer of the architecture acts as the application tier that enables workflows to be executed by the middleware of the system. It performs application-specific activities such as application monitoring and execution, creation of workflows, resource control, etc. The workflow scheduler connects the application layer and interacts with the clouds (e.g., private/public/hybrid cloud) infrastructure to enable application execution. This layer includes actual scheduling, iteration control, predictive analysis, and dispatching of workflow tasks and management of Cloud resources. Communication among all parts happens via events, which are managed by the Event Service. The Workflow Scheduler component extends the Cloudbus Workflow Engine [1] by adding adaptive computing methods to process complex analytic workflows.Task Dispatcher submits workflow tasks to the resources selected by the scheduler. The Iteration Controller controls each iteration of the workflows. In each iteration, the information about the task execution such as execution time, execution status, VM (Virtual Machine) type, etc. is sent back to the system for further analysis. Predictive analysis module trains the data sets after each iteration and estimates the execution time of data processing tasks. Hence, our system can learn the performance model from each iteration.

The *Cloud Resource Management* layer interacts with the physical cloud infrastructure and has features such as adaptive resource allocation and selection of the best resource allocation to meet the user time and cost requirements. With the design of iterative optimization, our system can adjust the resource allocation by knowing the characteristics and performance of tasks on actual execution environments, and improve the workflow performance gradually at runtime.

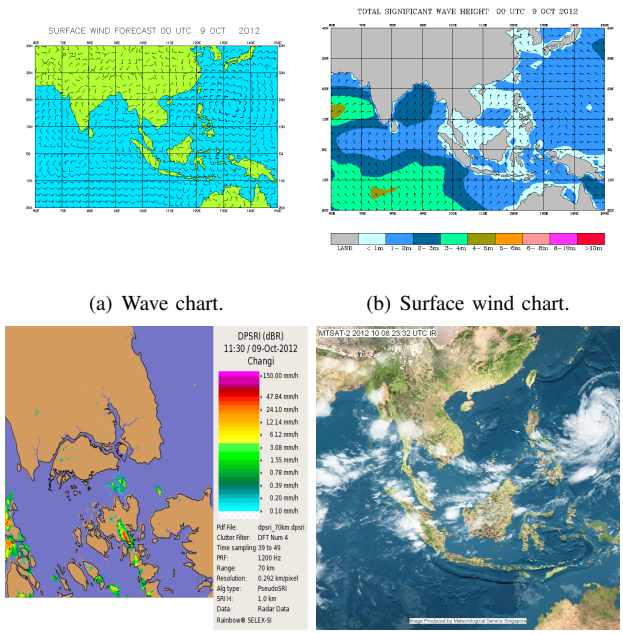
A schedule of a workflow is an assignment that allocates all the workflow tasks to cloud resources. With this architecture, we aim to derive an optimal schedule to achieve the minimum completion time and economic cost. As the above problem of mapping tasks onto distributed heterogeneous resources is NP-complete, we proposed heuristic iterative optimization methods to solve it, as described in the following section.

B. Weather Forecast Application

As shown in Figure 2, the weather forecast workflow application analyzes satellite images, wave data, radar cloud data, and surface wind data to understand the rainfall distribution over the time and at different locations. This application is particularly interesting because it requires several features that must be supported by a WfMS, e.g., large amount of data in workflows, integration of sensor data, long-running workflow, etc.

Over a specified period of time (e.g., 1 hour) several different variables are measured and observed. This data needs to be collected from the different sources and stored for later access, which is collected by the task `WeatherDataCollection` in Figure 2. The collected data is analyzed and transformed into a common format (e.g. Fahrenheit to Celsius scale) by the task `DataTransformation`. The normalized values are used to create the current state of the atmosphere. Then, a numerical weather forecast is made based on mathematical-physical models by four groups of tasks: `WaveData`, `SurfaceWind`, `RadarCloud`, and `SatelliteImage`. These four groups of tasks process the collected image data as shown in Figure 3. For instance, rainfall weather 70km radar image in Figure 3(c) is collected from National Environment Agency (NEA) Singapore, and shows the rainfall information on 2012 October 9 at 07:10 am. Data is captured every 5 minutes, and there are 16 GB for 3 years data. `ModelConstruction` complements the results of the numerical models with a statistical interpretation. The numerical post-processing is done with the task `DirectModelOutput`: the numerical results are interpolated for specific geological locations.

The sizes of these data and image files for each small region vary from 1397KB to 93KB, and the data generates tens of thousands of records that represent neighborhood data dependent on the information included in radar image data. The execution time in small VM instances with 2 CPU core and 2 GB memory for single task ranges from 622 seconds to 8 seconds. Since the execution time of tasks depend on the complexity of collected data, it is very challenging to schedule large-scale workflows and manage multiple types of cloud resources.



(a) Wave chart. (b) Surface wind chart.

(c) Radar cloud chart. (d) Satellite image.

Fig. 3. Weather forecast application.

III. Iterative Optimization

We proposed an iterative optimization method that extends the existing capability of workflow management based on DAG processing to allow iterative structures in a workflow application. It is designed for workflow analytical applications in which analytic tasks/functions are periodically repeated especially those with bursty bags of tasks. In order to optimize the performance and economic cost for such applications, the workflow engine analyzes historical data, generates performance prediction for each task, and optimizes accordingly. Hence, the performance of running the analytic programs can be continuously improved by adjusting system configurations to meet users requirements.

A. Formulation

The mapping of workflow tasks to cloud resources is a variation of typical "bin-packing" combinatorial optimization problem that can be formulated as follows:

Definition III.1 *Given a set of computational resources* $V\{v_1, \ldots, v_K\}$ *and a list of* N *tasks with execution time* $t_1, \ldots, t_N$. *Each type of resources* v_k *has different computing capability and price –* p_k. *The objective of the* scheduling *problem is to find a solution that assigns all tasks to a group of* R *resources such that within a fixed makespan* m, *economic cost of all tasks* $F(s)$ *is minimized:*

$$\begin{aligned} \underset{s}{Minimize}\ F(s) &= \sum_{i=1}^{R}\sum_{j=1}^{N} t_j \cdot p_k \cdot x_{ij}, \\ &\text{subject to} \\ \sum_{i=1}^{N} t_j \cdot p_k \cdot x_{ij} &\le m, i \in \{1, \ldots, R\}, \\ \sum_{j=1}^{r} x_{ij} &\le 1, j \in \{1, \ldots, N\}, \\ x_{ij} \in \{0,1\}, i \in \{1, \ldots, R\}, &j \in \{1, \ldots, N\}, \\ s &\in S, \end{aligned} \tag{1}$$

where s is a solution, S is the set of feasible solutions, and $F(s)$ is the image of s in the multiobjective space and represents the economic cost objective.

There are many variations of this problem, such as 2D packing, linear packing, packing by weight, packing by cost, and so on. In our case, the difference from other variations is that the "bins" or computational resources are upgradeable. Therefore, the most important thing to our system is to determine both the number and the types of computational resources. Since the bin packing problem is a combinatorial NP-hard problem, it is impossible to obtain optimal solutions especially for large-scale workflows with thousands or millions of tasks. In order to solve this problem, we proposed a new heuristic algorithm called "Upgrade Fit".

To run the analytics in clouds, we assume there is on demand resource provisioning and various types of VMs have different performance and prices. The workflow scheduler determines the appropriate amount of resources and assigns tasks to suitable resources to fulfill user requirements. Clouds add extra complexity to the workflow scheduler because the execution time of tasks noticeably varies when executed on different resources. Therefore, the cloud workflow scheduler needs to assign tasks and to find an acceptable compromise between budget expenditure and execution speedup. Our workflow management system provides a new iterative mechanism and a new scheduling algorithm to help users allocate appropriate resources and complete their applications with less completion time and economic cost.

In what follows, we describe the *Upgrade Fit* algorithm, as depicted in Algorithm 1. It consists of three steps:

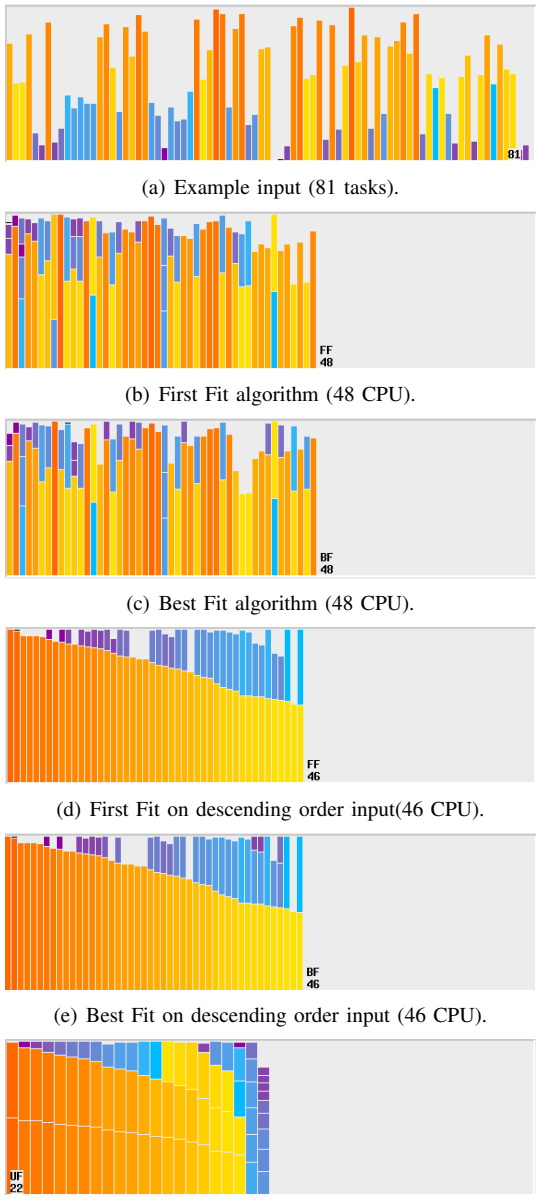
Step 1. Identify the makespan for some applications.

Suppose there are N tasks, the execution time on type k VM are $t_i^k, i \in 1 \ldots N, k \in 1 \ldots K$. For the weather forecast application, the makespan m is a fixed value because the image data is collected over a specified period of time. However, for other applications, the longest task dominates the execution time of the current iteration, which ultimately leads to worse performance/cost. Therefore, it is neccessary to identify and minimize the makespan in Step 1.

If a group of random generated tasks in Figure 4(a) is assigned to the VMs with best price/performance ratio by using First Fit or Best Fit algorithm [16], as shown in Figure 4(b) and 4(c), the scheduling time is short but the economic cost is bad. First Fit and Best Fit are very straightforward greedy approximation algorithms. For example, First Fit algorithm processes the tasks in arbitrary order (Figure 4(b) and 4(c)) or in descending order

Algorithm 1 Schedule iterative workflows by upgrading VMs and combining tasks (*Upgrade Fit* algorithm).

Input: Task $t_i, i \in 1..N$ in workflow G
Output: Task distribution and VM allocation
Schedule each task t_i on a VM, and identify t_{max}
Step 1: identify the makespan m for this iteration, and for some applications m is a constant
if m is not set **then**
  while t_{max} changes to a new task **do**
    for all v_i in available VMs sorted by type **do**
      compute marginal utility U_{k+1} by applying Eq. 2
      if $U_{k+1} \leq 0$ **then**
        upgrade the VM for t_{max} to new type v_i
      end if
    end for
  end while
  $m = t_{max}$
end if
while No more optimization can been done **do**
  Step 2: Upgrade VMs by combining tasks (vertical optimization)
  $mct = \lceil p_{k+1}/p_k \rceil + 1$ // #Minimum Combined Tasks
  while at least mct tasks can be combined **do**
    if $\sum_1^x t_i^{k+1} < m$ **then**
      upgrade one VM from v_k to v_{k+1} and remove other $mct - 1$ VMs
      combine these mct tasks
    end if
  end while
  Step 3: Minimize #VMs by using *Best Fit* (horizontal optimization)
  while More tasks can be combined **do**
    places a task in the fullest VM that still has room
  end while
end while



(a) Example input (81 tasks).
(b) First Fit algorithm (48 CPU).
(c) Best Fit algorithm (48 CPU).
(d) First Fit on descending order input(46 CPU).
(e) Best Fit on descending order input (46 CPU).
(f) Upgrade Fit on descending order input (2 × 22 CPU).

Fig. 4. Scheduling example (X axis – CPUs (bins), Y axis – Execution time). A simple example that illustrates the situation where Upgrade Fit outperforms other algorithms. The bars in (f) are two times wider because these bars are execution time on dual-core VMs, but the bars in other sub-figures are the execution time on single-core VMs. In this simple case, we only have 2 types of VMs and can upgrade once. In a real case, Upgrade Fit can upgrade VMs multiple times.

(Figure 4(d) and 4(e)). For each task, it attempts to place the task in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item within the new bin.

We iteratively decrease the execution time of the longest task by upgrading VM until the total cost for the whole iteration cannot be reduced. Our approach calculates marginal utility by the following equation:

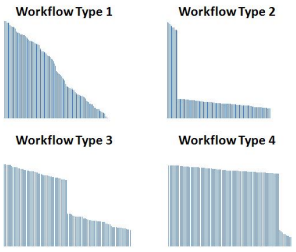
$$U_{k+1} = (N-1) \cdot m' \cdot p_k + m' \cdot p_{k+1} - N \cdot p_k \cdot m, \quad (2)$$

where p_k is the price of VM v_k, v_{i+1} is the upgraded type of VM v_i, m is the execution time of the longest task before upgrading the VM type v_i to v_{i+1}, m' is the execution time of the longest tasks after upgrading. In our case, marginal utility means the additional benefit or cost that a consumer derives from buying an upgraded unit of cloud service. $U \leq 0$ means there is no cost increment, and we can safely upgrade the VM for the longest task continuously until $U > 0$.

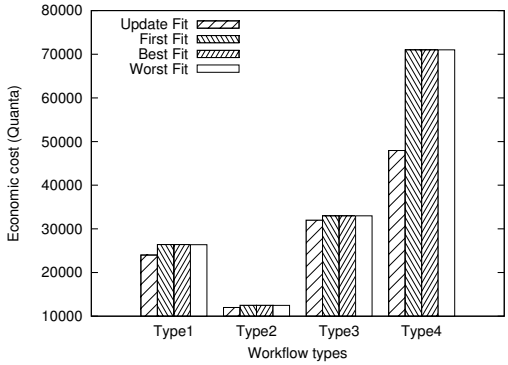
Step 2. Vertical optimization: upgrade VMs and combine tasks. After upgrading the VM types for the longest tasks, the allocated resources become heterogeneous and the execution time also varies from task to task. If the selected mct tasks are combined and one VM that is used to run the mct tasks is upgraded from v_k to $v_k + 1$, then other lower level $(mct - 1)$ VMs can be removed to decease the cost without increasing the current makespan. The minimum number of combined task mct is determined by the following equation:

$$mct = \lceil p_{k+1}/p_k \rceil + 1 \quad (3)$$

In commercial clouds like Amazon EC2, the price of the upper level resources is usually twice that for lower level resources, as shown in Table I. Our proposed algorithm combines the mct tasks, removes $mct - 1$ VMs with smaller type v_k, and upgrades



(a) 4 types of workflows (X axis – Tasks, Y axis – Execution time).



(b) Schedule comparison of 4 types of workflows.

Fig. 5. Four different types of workflows illustrate that Upgrade Fit outperforms other algorithms in most cases.

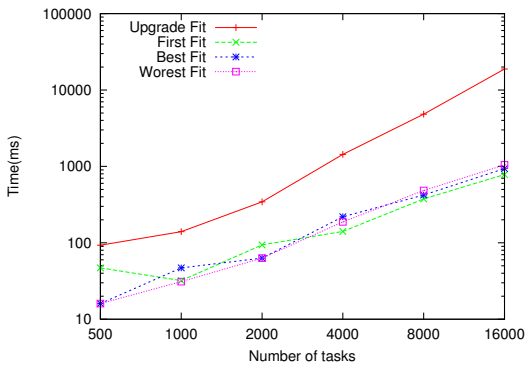


Fig. 6. Algorithm execution time.

Name	*Virtual Cores*	*Memory (GB)*	*Comp. Unit*	*HDD (GB)*	*$/hour*	*$/Unit /hour*
m1.micro	1	0.5	0.5	100	0.160	0.020
m1.mini	1	1	1	200	0.160	0.040
m1.medium	1	3.75	2	410	0.160	0.080
m1.large	2	7.5	4	850	0.320	0.080
m1.xlarge	4	15	8	1690	0.640	0.080
m2.xlarge	4	34.2	13	850	0.900	0.069
c1.xlarge	8	7	20	1690	0.660	0.033
cc1.4xlarge	8	23	33.5	1690	1.300	0.039
cc1.8xlarge	2×8	60.5	88	3370	2.400	0.027

TABLE I
PRICING OF EXPERIMENTAL CLOUD.

one VM from v_k to v_{k+1}. This step is repeated until there is no VM upgrade for any mct tasks.

Step 3. Horizontal optimization: minimize the number of VMs by using the Best Fit algorithm. As a sequel to the vertical optimization in Step 2, most of the larger tasks are merged and some VMs are upgraded to upper level. There are only some smaller tasks left for further optimization. Hence, in this step the algorithm simply uses the Best Fit algorithm to minimize the number of VMs. Step 2 and Step 3 are repeated until there is no optimization to be done. Figure 4(f) shows the scheduling result of our Upgrade Fit algorithm, which improves the existing algorithms in Figure 4(d) and 4(e) from 46 compute units to 44 compute units.

The time complexity of the Upgrade Fit algorithm is $O(N \cdot (\log N)^2)$ and the space complexity is $O(N)$, where N is the number of tasks. Other bin-packing algorithms such as Best Fit and First Fit has the same space complexity, but their time complexity is $O(N \cdot \log N)$. The algorithm execution time for different algorithm is shown in Figure 6. Although the algorithm execution time of Upgrade Fit is longer than others, it is still acceptable and practical compared with the execution time of all workflow tasks.

For the completeness of the algorithm evaluation and the universality of the experimental results, we tested these algorithms for four different types of workflow iterations that have different completion time distributions for tasks, as shown in Figure 5(a). For instance, the workflow type 4 is the most difficult one to be scheduled, since most tasks of this type have long completion time that is close to the makespan of the whole iteration. Upgrade Fit can effectively schedule this type of workflow iteration by upgrading VMs. In this case, Upgrade Fit achieve less cost than other algorithms by at least 30%. In all other cases, Upgrade Fit outperforms the other three algorithms in terms of economic cost with fixed makespan.

IV. Experimental Results

A. Experiment Setup

We conducted extensive experiments to examine the performance of our proposed system in terms of processing time, economic cost, and resource efficiency in clouds. The experimental testbed is composed of heterogeneous resource in a private Cloud. It includes m1.micro (0.5 compute unit), m1.mini (1 compute unit), m1.medium (2 compute units), m1.large (4 compute units) and some m1.xlarge VM instances with Linux CentOS 5.8, as shown in Table I. One compute unit provides the equivalent CPU capacity of a 1.5 GHz 2007 Opteron or 2007 Xeon processor.

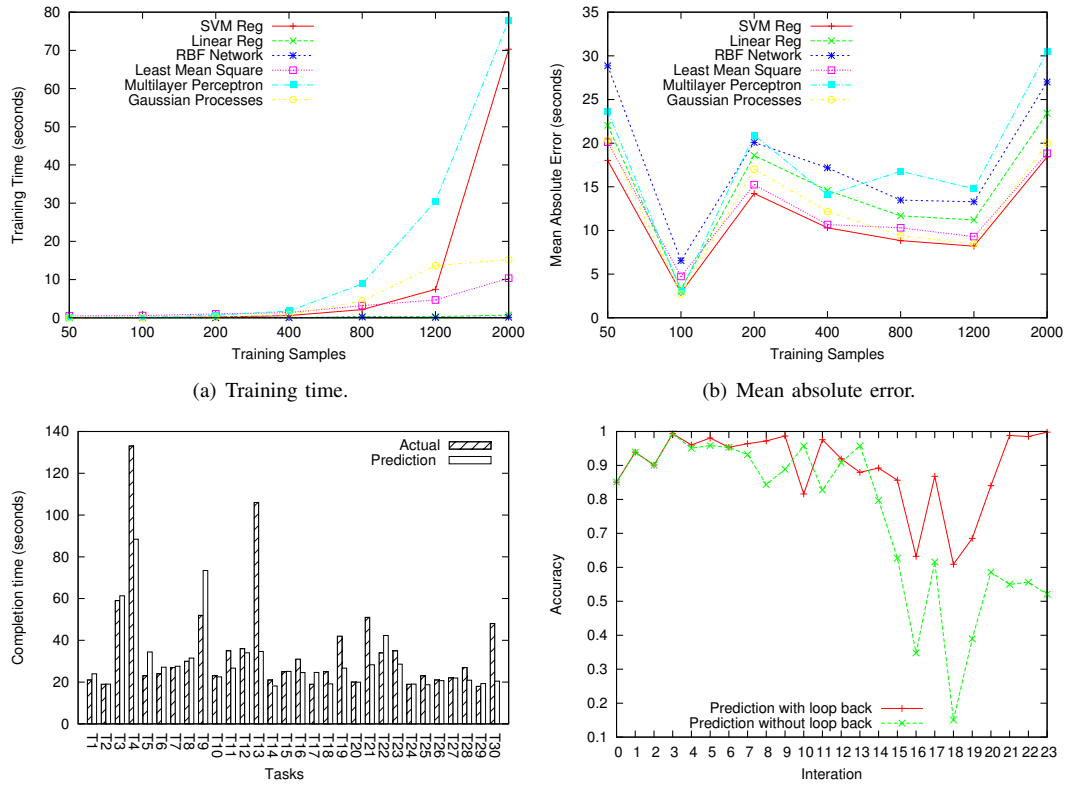
The metrics we used to evaluate our system are makespan, cost, and efficiency. We adopt the price policy of EC2 to calculate the cost. For example, the price of a medium instance was twice of a small instance, as shown in Table I. Makespan/completion time is the duration to complete the workflow task, and cost is the expense for running the tasks using computer resources. We define efficiency as ratio of the speed to the cost of running the application:

$$Efficiency = \frac{1}{Makespan \cdot Cost} \quad (4)$$

In Table II, we provide the runtime from an execution of a weather forecast workflow on the cloud. We provide the total sizes (i.e. the sum of the sizes of all files) of input and output consumed and generated by each job. Note that the same input data item may be consumed by multiple jobs.

Executed on Virtual Machine with 2 CPU at 2.4 Ghz processor								
Job	*Meaning*	*Count*	*Runtime*		*Inputs*		*Outputs*	
			Mean(m)	Variance	Mean(MB)	Variance	Mean(MB)	Variance
WeatherDataCollection	Data prepare	24	5.00	2.00	575.34	30.75	0.00	0.00
DataTransformation	Data transformation	24	5.00	2.00	0.00	0.00	575.34	30.75
RadarCloud	Data Analysis	720	33.07	39.64	575.34	30.75	0.00	0.00
ModelConstruction	Model Construction	24	2.00	2.00	0.00	0.00	0.00	0.00
DirectModelOutput	Model Output	24	2.00	2.00	0.00	0.00	1.80	1.98

TABLE II
WEATHER FORECAST WORKFLOW EXECUTION PROFILE.



(a) Training time.

(b) Mean absolute error.

(c) Prediction Accuracy.

(d) Prediction with and without loopback.

Fig. 7. Experiments on performance prediction.

B. Predictive Analysis

Predicting the execution time of workflow tasks always is a big challenge. We compared some existing prediction methods such as linear regression, RBF Network, Least mean square, multilayer perceptron, and Gaussian processes. Support Vector Machines (SVM) [4] is a supervised learning algorithm that analyze data and recognize patterns used for regression analysis. SVM performs well on data sets that have many attributes, even if there are just a few samples on which to train the model. More importantly, SVM has the advantage of learning to ignore irrelevant factors and requires fewer parameters to be tuned to achieve similar accuracy [8]. Figure 7(a) shows that SVM is reasonably fast for practical size of iterative workflow. In Figure 7(b), we can find that the mean absolute error of SVM is the smallest compared with other predictive methods. Based on above facts, we choose SVM as our prediction model to estimate execution time of task.

We collected the weather forecast data from April to May 2011 to train the model and predict the rainfall distribution in June 2011. In order to evaluate the accuracy, we define the prediction accuracy as

$$Accuracy = \left(1 - \frac{|t_a - t_p|}{t_a}\right) \times 100\%, \tag{5}$$

where t_a and t_p are the actual completion time and predicted completion time, respectively.

Figure 7(c) shows the predicted values of the completion time of 30 tasks in the workflow and the actual completion time. We obtained good prediction in this use case with average prediction accuracy of 80.65%.

In order to further improve the prediction accuracy, we implemented the SVM with loop back mechanism that uses the new generated data to train the predictive model. Loop back is one benefit of our iterative workflow structure, where in every loop, it is possible to re-train our prediction model, adjust model parameter, make it more dynamic to deal with exceptional data. Figure 7(d) shows that the prediction with loop back is able to improve the accuracy substantially compared to the prediction without loop back. For example, from iteration 17 to 24, the improvement is up to 50%. In the real working scenario, the

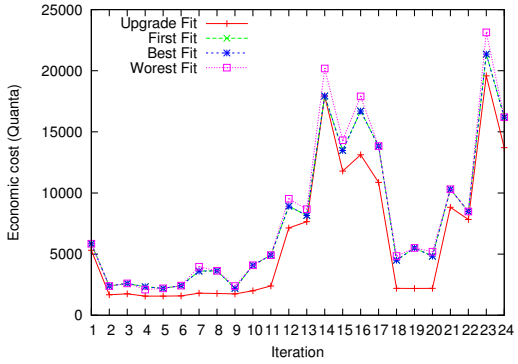


Fig. 8. Economic cost of different algorithms.

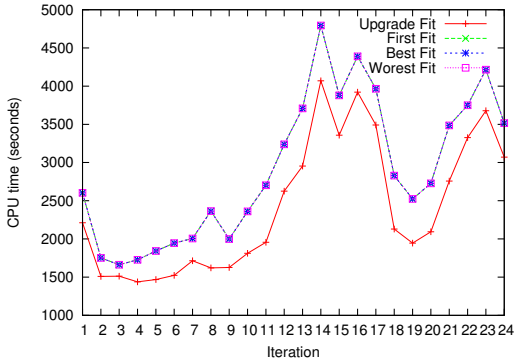


Fig. 9. CPU time of different algorithms.

prediction needs to be fast and efficient. We implemented our SVM model using the Weka [17] library on a machine with Intel Core Duo processor, 2.93GHz, and 2GB of RAM. The training time takes quite small amount of time with 0.11, 0.11 and 0.14 seconds for 60*24, 120*24, 200*24 records, respectively. It means the size of dataset will not affect the training time significantly, which is important for us to re-train the predictive model by adding new generated data iteratively.

C. Real Application

Figure 8, 9 and 10 compare the cost, CPU time and efficiency of different algorithms under heterogeneous computing environments, respectively. In all cases, the cost of Upgrade Fit is better than others because it can effectively utilize multiple types of resources. Other three algorithms simply schedule all tasks onto the resources with the best price/performance ratio. Intuitively, we intend to think that the resources with the best price/performance ratio should be able to generate good results. However, in Figure 8, we can see that our proposed Upgrade Fit can reduce the economic cost by more than 50% for iteration 7, 8, 10, 11, 18, 19, and 20. Hence, we can safely say that scheduling all tasks onto the resources with the best price/performance ratio does not result in the minimum cost. The proposed iterative workflow scheduling method, Upgrade Fit, also has less CPU time and higher efficiency than Best Fit, First Fit and Worst Fit. Upgrade Fit can adjust the resource allocation according to various conditions, and it determines the suitable types as well as the number of the VMs to run the tasks iteratively. However, Best Fit, First Fit, and Worst Fit are not designed to schedule iterative workflows to upgradeable resources.

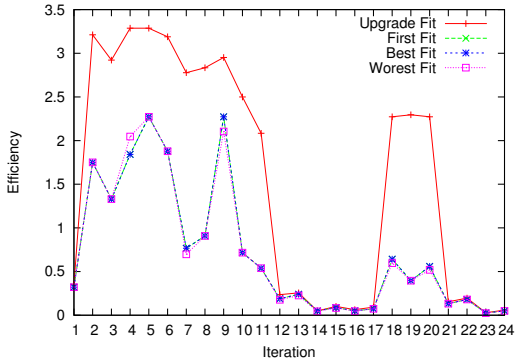


Fig. 10. Efficiency of different algorithms.

V. Related Work

Workflow System is concerned with the automation of processes in which tasks are structured based on their control and data dependency. There are some existing workflow systems, such as DAGMan [14], GridFlow [3], Cloudbus [1], Triana [15], ASKALON [9], Pegasus [7], etc. Most workflow management systems support only DAG-based applications.

DAGMan [14] developed by the Condor project allows scheduling of Grid workflow applications using opportunistic techniques such as matchmaking based on resource offers, resource requests, and cycle stealing with no support for advanced optimization heuristics. Our system proposed a new heuristics with dynamical prediction and optimization at runtime.

The Grid Application Development Software (GrADS) project [10] developed techniques for scheduling MPI, iterative, master-slave, and workflow applications. Workflow scheduling is approached by adapting Max-Min, Min-Min, and Suffrage heuristics originally developed for throughput scheduling of independent tasks. We proposed in this paper an approach that proves to be more effective for the class of workflows with iterative activities.

The scheduler in Cloudbus [1] provides just-in-time mappings using grid/cloud economy mechanisms. It makes scheduling decisions on where to place jobs in grids or clouds depending on the computational resources characteristics and users' Quality of Service (QoS) requirements. Our system supports multi-type adaptive resource allocation.

Pegasus [7], [11] is a workflow manger developed by the University of Southern California. Pegasus performs a mapping from an abstract workflow to the set of available Grid resources, and generates an executable workflow. Our system is integrated with dynamic prediction mechanism that can collect and make prediction at runtime with aggregated data.

Triana [15] provides a visual programming interface with functionality represented by units. Triana clients such as Triana GUI can log into a Triana Controlling Service (TCS), remotely build and run a workflow and then visualize the result on their device. Our system supports the public cloud services like Amazon EC2 and is extensible to other services.

ASKALON [9] is a cloud/grid application development and computing environment developed by the University of Innsbruck, Austria. ASKALON provides a new hybrid approach for scheduling workflow applications on the Grid through dynamic monitoring and steering combined with a static optimization [5].

The performance estimation of the workflow is conducted based on a combination of historical data obtained from a training phase and analytical modeling [8]. The main difference between ASKALON and our system is that they are using different prediction model and algorithms.

Other related works such as by M. Rahman et al. [13] proposed a workflow management system for deploying workflow application on virtualized environments which is able to utilize resources from public clouds. However, it does not try to optimize the resource. Our proposed method leverages the flexibility of cloud to reduce the VMs to lowest level and to maximize the performance at the same time.

There are some works try to find technology to determine the right amount of resources required for execution of workflow. Eun-Kyu Byun et al. [2] introduced a Balanced Time Scheduling method to estimate the minimum resources required to execute a workflow within a user-specified finish time. From the users' point of view, before they submit a workflow to a system, the user is not always aware of the completion time, especially when the workflow is executed for the first time. Our system provides a solution to use less resources to finish a workflow at reasonable time line.

Ashish Nagavaram et al. [6] present another dynamic resource allocation mechanism by using the elasticity of cloud computing. They monitor the entire workflow execution, and the system automatically decides to add or release resources through calculating two parameters. However, they only consider homogeneous resources. Our system can leverage heterogeneous resources to adaptively execute a workflow.

Oprescu et al. [12] proposed an algorithm to minimize completion time while respecting an upper bound for the budget. Their problem formulation is based on the classic Bounded Knapsack Problem (BKP), and they solve the problem using a classic dynamic programming. In contrast, our Upgrade Fit algorithm minimizes the economic cost after minimizes the upper bound for the completion time, i.e. our algorithm optimizes both performance and economic cost. We assume a more realistic economic model for resource utilization, and successfully utilize the characteristic of this model to minimize cost by upgrading resources.

All in all, our system has a more realistic cloud model and can effectively schedule bags of tasks onto multi-type cloud resources based on iteratively improved prediction.

VI. Conclusions and Future Work

In this paper, we addressed the interesting and important new problem of dynamically resource provisioning for compute- and data-intensive workflows in clouds. The goal of this work is to minimize the economic cost and makespan of periodic bursty bags of tasks for data processing.

A new system and a solution algorithm are developed to support dynamic cloud resource allocation and task assignment in public clouds. Specifically, our model considers selection of a proper mix of multi-type computing resources such that performance and cost requirements for a given number of tasks are satisfied. We proposed an iterative workflow scheduling method that extends the existing capability of workflow management from DAG to non-DAG processing so as to handle periodic bursty bags of tasks in the workflow, which have become increasing important in modern workflow management systems.

Experimental results indicate that our proposed algorithm is effective and efficient in solving problems of a practical size. In order to optimize the performance for such applications, the workflow engine analyzes the record of each iterative run, generates the predictive model of performance of each task, and optimizes performance and economic cost accordingly. Hence, the performance of iterative workflow applications can be continuously improved by adjusting the resource allocation to minimize the execution time and cost. In the future, we plan to investigate the reliability issues in the iterative workflow management systems and explore new fault tolerance methods which can further improve the system by adapting the resource allocation accordingly.

References

[1] Rajkumar Buyya, Suraj Pandey, and Christian Vecchiola. Cloudbus toolkit for market-oriented cloud computing. In *CloudCom*, 2009.

[2] Eun-Kyu Byun, Yang-Suk Kee, Jin-Soo Kim, and Seungryoul Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Comp. Syst.*, 27(8), 2011.

[3] Junwei Cao, Stephen A. Jarvis, Subhash Saini!, and Graham R. Nudd. GridFlow: Workflow Management for Grid Computing. In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid 2003))*, Tokyo, Japan, May 2003. IEEE Computer Sociery Press.

[4] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[5] Rubing Duan, Radu Prodan, and Thomas Fahringer. Run-time optimization for grid workflow applications. In *7th IEEE/ACM International Conference on Grid Computing (GRID 2006)*, Barcelona, Spain, September 28th-29th 2006.

[6] Ashish Nagavaram et al. A cloud-based dynamic workflow for mass spectrometry data analysis. In *eScience*, 2011.

[7] Ewa Deelman et. al. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing, LNCSD9, ISSN 1570-7873*, 1:25–39, 2003.

[8] Rubing Duan et al. A hybrid intelligent method for performance modeling and prediction of workflow activities in grids. In *IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2009.

[9] Thomas. Fahringer. ASKALON - A Programming Environment and Tool Set for Cluster and Grid Computing. http://dps.uibk.ac.at/askalon, Institute for Computer Science, University of Innsbruck.

[10] Chuang Liu, Lingyun Yang, Ian Foster, and Dave Angulo. Design and Evaluation of a Resource Selection Framework for Grid Applications. In *HPDC-11, the Symposium on High Performance Distributed Computing*, Scotland, 2002.

[11] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, 2012.

[12] Ana-Maria Oprescu and Thilo Kielmann. Bag-of-tasks scheduling under budget constraints. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, 2010.

[13] Mustafizur Rahman, Xiaorong Li, and Henry Novianus Palit. Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environment. In *IPDPS Workshops*, 2011.

[14] The Condor Team. Dagman (directed acyclic graph manager). http://www.cs.wisc.edu/condor/dagman/.

[15] The triana project. http://www.trianacode.org/about.

[16] V.V. Vazirani. *Approximation Algorithms*. Springer, 2004.

[17] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann, 2005.