A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments

Suraj Pandey[1], Linlin Wu[1], Siddeswara Mayura Guru[2], Rajkumar Buyya[1]

[1]Cloud Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{spandey, linwu, raj}@csse.unimelb.edu.au

[2]CSIRO Tasmanian ICT Centre
Hobart, Australia
siddeswara.guru@csiro.au

Abstract

Cloud computing environments facilitate applications by providing virtualized resources that can be provisioned dynamically. However, users are charged on a pay-per-use basis. User applications may incur large data retrieval and execution costs when they are scheduled taking into account only the 'execution time'. In addition to optimizing execution time, the cost arising from data transfers between resources as well as execution costs must also be taken into account. In this paper, we present a particle swarm optimization (PSO) based heuristic to schedule applications to cloud resources that takes into account both computation cost and data transmission cost. We experiment with a workflow application by varying its computation and communication costs. We compare the cost savings when using PSO and existing 'Best Resource Selection' (BRS) algorithm. Our results show that PSO can achieve: a) as much as 3 times cost savings as compared to BRS, and b) good distribution of workload onto resources.

1 Introduction

Modern collaborative scientific experiments in domains such as structural biology, high-energy physics and neuroscience involve the use of distributed data sources. As a result, analysis of their datasets is represented and structured as scientific workflows [7]. These scientific workflows usually need to process huge amount of data and computationally intensive activities. A scientific workflow management system [14] is used for managing these scientific experiments by hiding the orchestration and integration details inherent while executing workflows on distributed resources provided by cloud service providers.

Cloud computing is a new paradigm for distributed computing that delivers infrastructure, platform, and software (application) as services. These services are made available as subscription-based services in a pay-as-you-go model to consumers [3, 2]. Cloud computing helps user applications dynamically provision as many compute resources at specified locations (currently US east1a-d for Amazon[1]) as and when required. Also, applications can choose the storage locations to host their data (Amazon S3[2]) at global locations. In order to efficiently and cost effectively schedule the tasks and data of applications onto these cloud computing environments, application schedulers have different policies that vary according to the objective function: minimize total execution time, minimize total cost to execute, balance the load on resources used while meeting the deadline constraints of the application, and so forth. In this paper, we focus on minimizing the total execution cost of applications on these resources provided by Cloud service providers, such as Amazon and GoGrid[3]. We achieve this by using a meta-heuristics method called Particle Swarm Optimization (PSO).

Particle Swarm Optimization (PSO) is a self-adaptive global search based optimization technique introduced by Kennedy and Eberhart [8]. The algorithm is similar to other population-based algorithms like Genetic algorithms but, there is no direct re-combination of individuals of the population. Instead, it relies on the social behavior of the particles. In every generation, each particle adjusts its trajectory based on its best position (local best) and the position of the best particle (global best) of the entire population. This concept increases the stochastic nature of the particle and converge quickly to a global minima with a reasonable good solution.

PSO has become popular due to its simplicity and its effectiveness in wide range of application with low computational cost. Some of the applications that have used PSO are: the reactive voltage control problem [23], data mining [16], chemical engineering [13], pattern recognition [9] and environmental engineering [10]. The PSO has also been applied to solve *NP-Hard* problems like Scheduling [24, 21] and task allocation [22, 26].

[1]http://aws.amazon.com
[2]http://aws.amazon.com/s3/
[3]http://www.gogrid.com

Our main contributions in this paper are as follows:

- We formulate a model for task-resource mapping to minimize the overall cost of execution
- We design a heuristic that uses PSO to solve task-resource mappings based on the proposed model

The rest of the paper is organized as follows: Section 2 presents related work. In Section 3, we describe the task-resource scheduling problem and its formulation with the help of an example workflow. In Section 4, we present our scheduling heuristic that uses PSO and introduce the PSO algorithm. Section 5 presents an experimental evaluation of the performance our heuristic. Section 6 concludes the paper and discusses some future work.

2 Related Work

Workflow applications are commonly represented as a directed acyclic graph. The mapping of jobs to the compute-resources is an *NP-complete* problem in the general form [19]. The problem is *NP-complete* even in two simple cases: (1) scheduling jobs with uniform weights to an arbitrary number of processors and (2) scheduling jobs with weights equal to one or two units to two processors [19]. So, past work have proposed many heuristics based approach to scheduling workflow applications. Data intensive workflow applications are a special class of workflows, where the size and/or quantity of data is large. As a result, the transfer of data from one compute node to another takes longer time. This incurs higher transmission and storage cost than computing cost running on these data.

Deelman et al. [5] have done considerable work on planning, mapping and data-reuse in the area of workflow scheduling. They have proposed Pegasus [5], which is a framework that maps complex scientific workflows onto distributed resources such as the Grid. DAGMan, together with Pegasus, schedules tasks to Condor system. The Taverna project [12] has developed a tool for the composition and enactment of bioinformatics workflows for the life science community. Other well-known projects on workflow systems include GridFlow [4], ICENI [6], GridAnt [1], Triana [18] and Kepler [11]. Most of the past work schedule tasks on resources based on earliest finish time, earliest starting time or the high processing capabilities. We term these as "best resource selection" (BRS) approach, where a resource is selected based on its performance.

Since task scheduling is a *NP-Complete* problem, Genetic Algorithm (GA) has been used for scheduling workflows [25]. However, GA may not be the best approach. Salman et al. [15] have shown that the performance of PSO algorithm is faster than GA in solving static task assignment problem for homogeneous distributed computing systems based on their test cases. Lei et al. [27] have shown that the PSO algorithm is able to get better schedule than GA based on their simulated experiments for Grid computing. In addition, the results presented by Tasgetiren et al. [17] have provided evidence that PSO algorithm was able to improve 57 out of 90 best known solutions provided by other well known algorithms to solve the sequencing problems.

We use PSO as it has a faster convergence rate than GA. Also, it has fewer primitive mathematical operators than in GA (e.g. reproduction, crossover, mutation), making applications less dependent on parameter fine-tuning. Moreover, using discrete numbers, we can easily correlate particle's position to task-resource mappings.

3 Task-Resource Scheduling Problem Formulation

The mapping of tasks of an application workflow to distributed resources can have several objectives. We focus on minimizing the total cost of computation of an application workflow.

We denote an application workflow as a Directed Acyclic Graph (DAG) represented by G=(V, E), where V=$\{T_1, ..., T_n\}$ is the set of tasks, and E represents the data dependencies between these tasks, that is, $f_{j,k} = (T_j, T_k) \in E$ is the data produced by T_j and consumed by T_k. We have a set of storage sites $S = \{1, ..., i\}$, a set of compute sites $PC = \{1, ..., j\}$, and a set of tasks $T = \{1, ..., k\}$. We assume the 'average' computation time of a task T_k on a compute resource PC_j for a certain size of input is known. Then, the cost of computation of a task on a compute host is inversely proportional to the time it takes for computation on that resource. We also assume the cost of unit data access $d_{i,j}$ from a resource i to a resource j is known. The access cost is fixed by the service provider (e.g. Amazon CloudFront). The transfer cost can be calculated according to the bandwidth between the sites. However, we have used the cost for transferring unit data between sites, per second. We assume that these costs are non-negative, symmetric, and satisfy the triangle inequality: that is, $d_{i,j} = d_{j,i}$ for all $i, j \in N$, and $d_{i,j} + d_{j,k} \geq d_{i,k}$ for all $i, j, k \in N$.

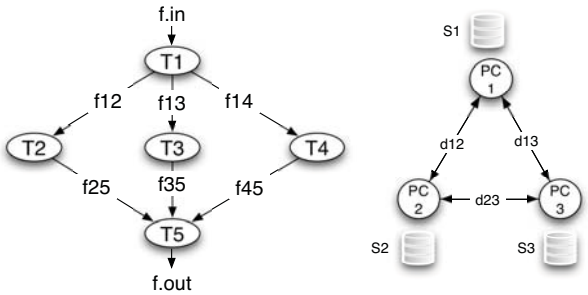

Figure 1: An example workflow, compute nodes (PC) & storage (S).

Figure 1 depicts a workflow structure with five tasks, which are represented as nodes. The dependencies between

tasks are represented as arrows. This workflow is similar in structure to our version of the Evolutionary Multi-objective Optimization (EMO) application [20]. The root task may have an input file (e.g. $f.in$) and the last task produces the output file (e.g. $f.out$). Each task generates output data after it has completed ($f_{12}, f_{13}, ..., f_{45}$). These data are used by the task's children, if any. The numeric values for these data is the edge-weight ($e_{k1,k2}$) between two tasks $k1 \in T$ and $k2 \in T$. The figure also depicts three compute resources ($PC1, PC2, PC3$) interconnected with varying bandwidth and having its own storage unit ($S1, S2, S3$). The goal is to assign the workflow tasks to the compute resources such that the total cost of computation is minimized.

The problem can be stated as: *"Find a task-resource mapping instance M, such that when estimating the total cost incurred using each compute resource PC_j, the highest cost among all the compute resources is minimized."*

Let $C_{exe}(M)_j$ be the total cost of all the tasks assigned to a compute resource PC_j (Eq. 1). This value is computed by adding all the node weights (the cost of execution of a task k on compute resource j) of all tasks assigned to each resource in the mapping M. Let $C_{tx}(M)_j$ be the total access cost (including transfer cost) between tasks assigned to a compute resource PC_j and those that are not assigned to that resource in the mapping M (Eq. 2). This value is the product of the output file size (given by the edge weight $e_{k1,k2}$) from a task $k1 \in k$ to task $k2 \in k$ and the cost of communication from the resource where $k1$ is mapped ($M(k1)$) to another resource where $k2$ is mapped ($M(k2)$). The average cost of communication of unit data between two resources is given by $d_{M(k1),M(k2)}$. The cost of communication is applicable only when two tasks have file dependency between them, that is when $e_{k1,k2} > 0$. For two or more tasks executing on the same resource, the communication cost is zero.

$$C_{exe}(M)_j = \sum_k w_{kj} \qquad \forall M(k) = j \tag{1}$$

$$C_{tx}(M)_j = \sum_{k1 \in T} \sum_{k2 \in T} d_{M(k1),M(k2)} e_{k1,k2} \qquad \forall M(k1) = j \; and \; M(k2) \neq j \tag{2}$$

$$C_{total}(M)_j = C_{exe}(M)_j + C_{tx}(M)_j \tag{3}$$

$$Cost(M) = max(C_{total}(M)_j) \quad \forall j \in P \tag{4}$$

$$Minimize(Cost(M) \quad \forall M) \tag{5}$$

Equation 4 ensures that all the tasks are **not** mapped to a single compute resource. Initial cost maximization will distribute tasks to all resources. Subsequent minimization of the overall cost (Equation 5) ensures that the total cost is minimal even after initial distribution. For a given assignment M, the total cost $C_{total}(M)_j$ for a compute resource PC_j is the sum of execution cost and access cost (Eq. 3). When estimating the total cost for all the resources, the largest cost for all the resources is minimized (Eq. 5). This indirectly ensures that the tasks are not mapped to a single resources and there will be a distribution of cost among the resources.

4 Scheduling based on Particle Swarm Optimization

In this section, we present a scheduling heuristic for dynamically scheduling workflow applications. The heuristic optimizes the cost of task-resource mapping based on the solution given by particle swarm optimization technique. The optimization process uses two components: a) the scheduling heuristic as listed in Algorithm 1, and b) the PSO steps for task-resource mapping optimization as listed in Algorithm 2. First, we will give a brief description of PSO algorithm.

$$v_i^{k+1} = \omega v_i^k + c_1 rand_1 \times (pbest_i - x_i^k) + c_2 rand_2 \times (gbest - x_i^k), \tag{6}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1}, \tag{7}$$

where:

v_i^k	velocity of particle i at iteration k
v_i^{k+1}	velocity of particle i at iteration $k+1$
ω	inertia weight
c_j	acceleration coefficients; $j = 1, 2$
$rand_i$	random number between 0 and 1; $i = 1, 2$
x_i^k	current position of particle i at iteration k
$pbest_i$	best position of particle i
$gbest$	position of best particle in a population
x_i^{k+1}	position of the particle i at iteration $k+1$.

4.1 Particle Swarm Optimization

Particle Swarm Optimisation (PSO) is a swarm-based intelligence algorithm [8] influenced by the social behavior of animals such as a flock of birds finding a food source or a school of fish protecting themselves from a predator. A particle in PSO is analogous to a bird or fish flying through a search (problem) space. The movement of each particle is co-ordinated by a velocity which has both magnitude and direction. Each particle position at any instance of time is influenced by its best position and the position of the best particle in a problem space. The performance of a particle is measured by a fitness value, which is problem specific.

The PSO algorithm is similar to other evolutionary algorithms. In PSO, the population is the number of particles in a problem space. Particles are initialized randomly. Each particle will have a fitness value, which will be evaluated by a fitness function to be optimized in each generation. Each particle knows its best position $pbest$ and the best position so far among the entire group of particles $gbest$. The $pbest$ of a particle is the best result (fitness value) so far reached

by the particle, whereas $gbest$ is the best particle in terms of fitness in an entire population. In each generation the velocity and the position of particles will be updated as in Eq 6 and 7, respectively.

PSO algorithm provide a mapping of all the tasks to a set of given resources based on the model described in Section 3.

Algorithm 1 Scheduling heuristic.

1: Calculate average computation cost of all tasks in all compute resources
2: Calculate average cost of (communication/size of data) between resources
3: Set task node weight w_{kj} as average computation cost
4: Set edge weight $e_{k1,k2}$ as size of file transferred between tasks
5: Compute PSO($\{t_i\}$) /* a set of all tasks $i \in k$*/
6: **repeat**
7: **for** all "ready" tasks $\{t_i\} \in T$ **do**
8: Assign tasks $\{t_i\}$ to resources $\{p_j\}$ according to the solution provided by PSO
9: **end for**
10: Dispatch all the mapped tasks
11: Wait for $polling\_time$
12: Update the ready task list
13: Update the average cost of communication between resources according to the current network load
14: Compute PSO($\{t_i\}$)
15: **until** there are unscheduled tasks

Scheduling Heuristic: We calculate the average computation cost (assigned as node weight in Figure 1) of all tasks on all the compute resources. This cost can be calculated for any application by executing each task of an application on a series of known resources. It is represented as TP matrix in Table 1. As the computation cost is inversely proportional to the computation time, the cost is higher for those resources that complete the task quicker. Similarly, we store the average value of communication cost between resources per unit data, represented by PP matrix in Table 1, described later in the paper. The cost of communication is inversely proportional to the time taken. We also assume we know the size of input and output data of each task (assigned as edge weight $e_{k1,k2}$ in Figure 1). In addition, we consider this cost is for the transfer per second (unlike Amazon CloudFront which does not specify time for transferring).

The initial step is to compute the mapping of all tasks in the workflow, irrespective of their dependencies (Compute PSO(t_i)). This mapping optimizes the overall cost of computing the workflow application. To validate the dependencies between the tasks, the algorithm assigns the "ready" tasks to resources according to the mapping given by PSO. By "ready" tasks, we mean those tasks whose parents have completed execution and have provided the files necessary for the tasks' execution. After dispatching the tasks to resources for execution, the scheduler waits for $polling\_time$. This time is for acquiring the status of tasks, which is middleware dependent. Depending on the number of tasks completed, the ready list is updated, which will now contain the tasks whose parents have completed execution. We then update the average values for communication between resources according to the current network load. As the communication costs would have changed, we recompute the PSO mappings. Also, when remote resource management systems are not able to assign task to resources according to our mappings due to resource unavailability, the recomputation of PSO makes the heuristic dynamically balances other tasks' mappings (online scheduling). Based on the recomputed PSO mappings, we assign the ready tasks to the compute resources. These steps are repeated until all the tasks in the workflow are scheduled.

Algorithm 2 PSO algorithm.

1: Set particle dimension as equal to the size of ready tasks in $\{t_i\} \in T$
2: Initialize particles position randomly from $PC = 1, ..., j$ and velocity v_i randomly.
3: For each particle, calculate its fitness value as in Equation 4.
4: If the fitness value is better than the previous best $pbest$, set the current fitness value as the new $pbest$.
5: After Steps 3 and 4 for all particles, select the best particle as $gbest$.
6: For all particles, calculate velocity using Equation 6 and update their positions using Equation 7.
7: If the stopping criteria or maximum iteration is not satisfied, repeat from Step 3.

The algorithm is dynamic (online) as it updates the communication costs (based on average communication time between resources) in every scheduling loop. It also recomputes the task-resource mapping so that it optimizes the cost of computation, based on the current network and resource conditions.

PSO: The steps in the PSO algorithm are listed in Algorithm 2. The algorithm starts with random initialization of particle's position and velocity. In this problem, the particles are the task to be assigned and the dimension of the particles are the number of tasks in a workflow.

The value assigned to a each dimension of a particles are the computing resources indices. Thus the particle represent a mapping of resource to a task. In our workflow (depicted in Figure 1) each particle is 5-D because of 5 tasks and the content of each dimension of the particles is the compute resource assigned to that task. For example a sample particle could be represented as depicted in Figure 2.

The evaluation of each particle is perform by the fitness

function given in Eq. 5. The particles calculate their velocity using Eq. 6 and update their position according to Eq. 7. The evaluation is carried out until the specified number of iterations (user-specified stopping criteria).

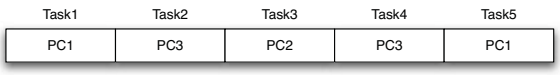

Figure 2: A sample particle for the workflow shown in Figure 1 .

5 Experimental Evaluation

In this section, we present the metric of comparison, the experiment setup and the results.

5.1 Performance metric

As a measure of performance, we used cost for complete execution of application as a metric. We computed the total cost of execution of a workflow using two heuristics: PSO based cost optimization (Algorithm 1), and best resource selection (based on minimum completion time by selecting a resource with maximum cost).

5.2 Data and Implementation

Table 1: The TP-matrix, PP-matrix and DS-matrix. The values shown are an example of 1 instance of the experiment run.

$$TP[5 \times 3] = \begin{bmatrix} & PC1 & PC2 & PC3 \\ T_1 & 1.23 & 1.12 & 1.15 \\ T_2 & 1.17 & 1.17 & 1.28 \\ T_3 & 1.13 & 1.11 & 1.11 \\ T_4 & 1.26 & 1.12 & 1.14 \\ T_5 & 1.19 & 1.14 & 1.22 \end{bmatrix}$$

$TP[i,j] = Cost\ of\ execution\ of\ T_i\ at\ PC_j$
$(EC2\ price\ of\ resources\ for\ High\ CPU\ instance)$
$(Example\ matrix\ values\ are\ in\ the\ range\ \$1.1 - \$1.28/hr)$

$$PP[3 \times 3] = \begin{bmatrix} & PC1 & PC2 & PC3 \\ PC1 & 0 & 0.17 & 0.21 \\ PC2 & 0.17 & 0 & 0.22 \\ PC3 & 0.21 & 0.22 & 0 \end{bmatrix}$$

$PP[i,j] = Cost\ of\ communication\ between\ PC_i\ \&PC_j$
$(Values\ in\ \$/MB/second)$

$$DS_{T2,T3,T4}[2 \times 2] = \begin{bmatrix} & totaldata \\ i/p & 10 \\ o/p & 10 \end{bmatrix}$$

$$DS_{T5}[2 \times 2] = \begin{bmatrix} & totaldata \\ i/p & 30 \\ o/p & 60 \end{bmatrix}$$

$row_1 = i/p\ data\ size(MB), \quad row_2 = o/p\ data\ size(MB)$

We have used three matrices that store the values for: a) average computation cost of each task on each resource (TP-matrix), b) average communication cost per unit data between compute resources (PP-matrix), and c) input/output Data Size of each task (DS-matrix), as depicted in Table 1.

The values for PP-matrix resemble the cost of unit data transfer between resources given by Amazon CloudFront[4]. We assume PC1 to be in US, PC2 in Hong Kong (HK) and PC3 in Japan (JP), respectively. We randomly choose the values in the matrix for every repeated experiment, but keep these values constant during the PSO iterations.

The values for TP-matrix varies for two classes of experiments. While varying the size of data, we choose the TP-matrix values to resemble the Evolutionary Multi-objective Optimization (EMO) [20] application. While varying the processing cost, we use the Amazon EC2's[5] pricing policy for different classes of virtual machine instances. E.g. if we were to use small+medium instances of Linux machines in both US and Europe, the TP-matrix would have values between \$0.1-\$0.3/hr, assuming all the tasks complete within 1 hour.

As each task has its own DS-matrix, the sum of all the values in the matrix varies according to the size of data we experiment (64-1024 MB). The total data is divided among tasks such that if x is the output data size of $T1$, then tasks $T2, T3, \& T4$ each receive x data as input and produce x data as output. Finally, task $T5$ consumes $3x$ data and produces $6x$ data.

We used the JSwarm[6] package to conduct our simulation experiments in PSO. Table 2 gives the experimental setup of the PSO algorithm. The number of executions represent the number of independent experiments done in order to calculate the Confidence Interval (CI) of the results.

Table 2: The values for PSO optimization

Number of particles = 25
Number of iterations = 45
Number of executions = 30

5.3 Experiments and Results

We evaluated the scheduling heuristic using the workflow depicted in Figure 1. Each task in the workflow has input and output files of varying sizes. Also, the execution cost of each task varies among all the compute resources used (in our case $PC1 - PC3$). We analyze the performance of our heuristic by varying each of these in turn.

We plot the graphs by averaging the results obtained after 30 independent executions. In every execution, the x-axis parameters such as total data size (e.g. 1024MB), range of computation cost (e.g. 1.1-1.3 \$/hour) remain unchanged, while the particle's velocity and position change. The graphs also depict the value of the plotted points together with the CI (represented as "+/-" *value*).

[4] http://aws.amazon.com/cloudfront/
[5] http://aws.amazon.com/ec2/
[6] http://jswarm-pso.sourceforge.net/

5.3.1 Variation in Total Data Size of a Workflow

We varied the size of total data processed by the workflow in the range 64-1024 MB. By varying the data size, we compared the variance in total cost of execution and the distribution of workload on resources, for the two algorithms as depicted in Figure 3 and Figure 4, respectively. We fixed the compute resource cost in the range $1.1 - 1.3\$/hr$ for the experiments in the sub-section 5.3.1 and sub-section 5.3.3.

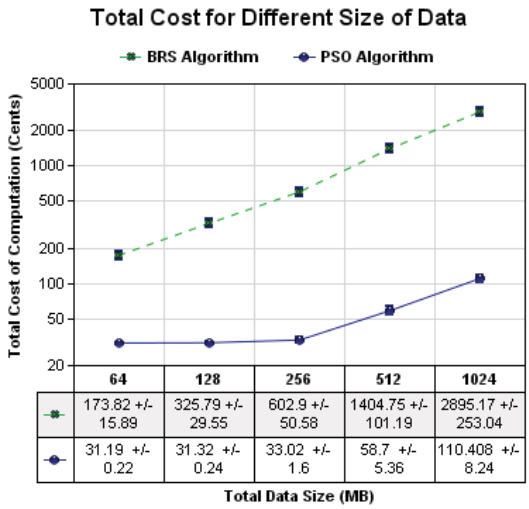

Figure 3: Comparison of total cost between PSO based resource selection and best resource selection algorithms when varying total data size of a workflow.

Total Cost of Execution: Figure 3 plots the total cost of computation of the workflow (in the **log** scale) with the increase in the total data processed by the workflow. The graph also plots 95% Confidence Interval (CI) for each data point.

The cost obtained by PSO based task-resource mapping increases much slower than the BRS algorithm. PSO achieves at least three times lower cost for 1024MB of total data processed than the BRS algorithm. Also, the value of CI in cost given by PSO algorithm is +/- 8.24, which is much lower as compared to the BRS algorithm (+/- 253.04), for 1024 MB of data processed by the workflow.

The main reason for PSO to perform better than the 'best resource' selection is the way it takes into account communication costs of all the tasks, including dependencies between them. When calculating the cost of execution of a child task on a resource, it adds the data transfer cost for transferring the output from its parent tasks' execution node to that node. This calculation is done for all the tasks in the workflow to find the near optimal scheduling of task to resources. However, the BRS algorithm calculates the cost for a single task at a time, which does not take into account the mapping of other tasks in the workflow. This results in PSO based algorithm giving lower cost of execution as compared to BRS based algorithm.

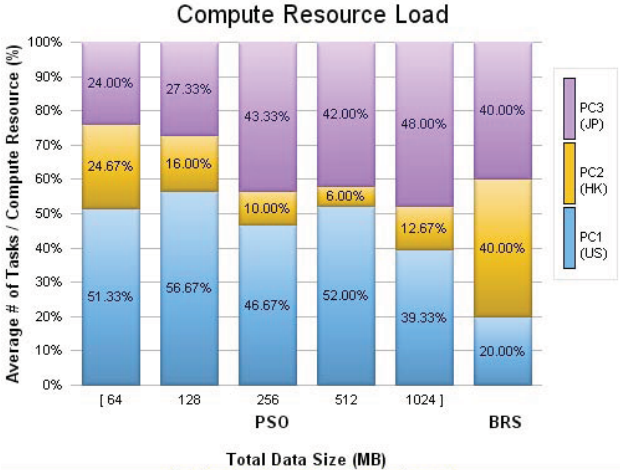

Figure 4: Distribution of workflow tasks on available processors.

Distribution of Load: We calculated the distribution of workflow tasks onto available resources for various size of total data processed, depicted in Figure 4. This evaluation is necessary as algorithms may choose to submit all the tasks to few resources to avoid communication between resources as the size of data increases, thus minimizing communication cost to zero. In our formulation, equation 4 restricts all tasks being mapped to the same resource, so that tasks can execute in parallel for increased time-efficiency. In Figure 4, The X-axis represents the total size of data processed by the workflow and the Y-axis the average number of tasks (expressed as percentage) executed by a compute resource for various size of data.

The figure shows that PSO distributes tasks to resources according to the size of data. When the total size of data is small (for 64-126 MB), PSO distributed tasks proportionally to all the resources ($PC1 - PC3$). However, when the size of data increased to (and over) 256MB, more tasks were allocated to $PC1$ and $PC3$.

As the cost of compute resources was fixed for this part of experiment, the BRS algorithm does not vary task-resource mapping. Also, it is indifferent to the size of data. Hence, BRS's load distribution is a straight line as depicted in Figure 4, with $PC1$, $PC2$ and $PC3$ receiving 20%, 40% and 40% of the total tasks, respectively.

The distribution of tasks to all the available resources in proportion to their usage costs, ensured that hotspots (resource overloading) were avoided. Our heuristic could minimize the total cost of execution and balance the load on available resources.

5.3.2 Variation in Compute Resource Cost

We experimented the performance of PSO by varying the cost of computation of all compute resources. This variation

is practically justifiable as different Cloud service providers (e.g. Amazon, GOGRID) can have varying pricing policies depending on the type and capabilities of their resources (virtual machines).

Figure 5 depicts the change in total cost of computation of applications for different range of compute resource prices (price range are similar to Amazon EC2 instances in US and Europe combined). The plotted values are an average of 30 executions. We use curve fitting to plot the lines along the points to show the trend: rise in cost in comparison to rise in compute resource cost for the two algorithms. The workflow processed a total of 128MB of data.

Clearly, PSO based mapping has much lower cost (at least 10 times) and CI values (lower than 0.3) as compared to that given BRS based mapping. In addition, the slope of the trend line shows that PSO based mapping increases the cost linearly, whereas BRS increases exponentially.

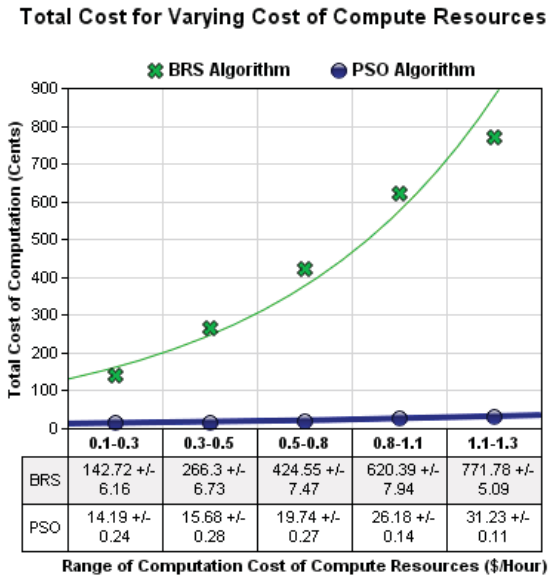

Figure 5: Comparison of total cost between PSO based resource selection and best resource selection algorithms when varying computation cost of all the resources (for 128MB of data).

The reason for PSO's improvement over BRS is due to PSO's ability to find a near optimal solutions for mapping all tasks in the workflow to the given set of compute resources. The linear increase in PSO's cost also suggest that it takes both computation and communication cost into account. However, BRS simply maps a task to the resource that has minimum completion time (a resource with higher frequency, lower load and thus having higher cost). As the resource costs increase, the use of BRS leads to more costs due to the affinity towards better resource, irrespective to the size of data. Whereas, PSO minimizes the maximum total cost of assigning all tasks to resources.

5.3.3 Convergence of PSO

Figure 6 plots the convergence of total cost computed by PSO over the number of iterations for different sizes of total data processed by the workflow in Figure 1. Initially, the particles are randomly initialized. Therefore, the inital total cost is always high. This initial cost corresponds to the 0^{th} iteration. As the algorithm progresses, the convergence is drastic and it finds a global minima very quickly. The number of iterations needed for the convergence is seen to be 20-30, for our application environment.

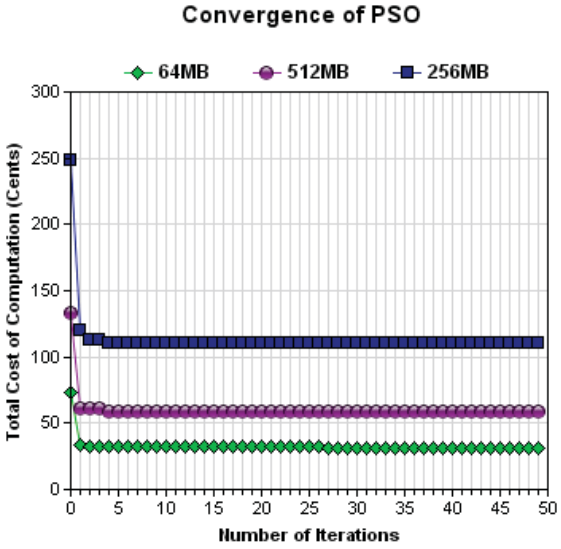

Figure 6: The trend of convergence of PSO with the number of iterations for different size of data.

6 Conclusions and Future Work

In this work, we presented a scheduling heuristic based on Particle Swarm Optimization (PSO). We used the heuristic to minimize the total cost of execution of application workflows on Cloud computing environments. We obtained total cost of execution by varying the communication cost between resources and the execution cost of compute resources. We compared the results obtained by our heuristic against "Best Resource Selection" (BRS) heuristic. We found that PSO based task-resource mapping can achieve at least three times cost savings as compared to BRS based mapping for our application workflow. In addition, PSO balances the load on compute resources by distributing tasks to available resources. The heuristic we proposed is generic as it can be used for any number of tasks and resources by simply increasing the dimension of the particles and the number of resources, respectively.

As part of our future work, we would like to integrate PSO based heuristic into our workflow management system to schedule workflows of real applications such as brain imaging analysis [14], EMO [20], and others.

Acknowledgments

This work is partially supported through Australian Research Council (ARC) Discovery Project grant. Some values in our experiments were obtained from Amazon Cloud Services (EC2, CloudFront and S3) pricing policies. We would like to thank William Voorsluys for his comments on the experiments and results.

Siddeswara would like to acknowledge the CSIRO ICT Centre Post-Doctoral Fellow capability development fund for the support.

References

[1] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. Gridant: A client-controllable grid work.ow system. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 7*, 2004.

[2] M. Armbrust, A. Fox, R. Grifth, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley, February 2009.

[3] R. Buyya, S. Pandey, and C. Vecchiola. Cloudbus toolkit for market-oriented cloud computing. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, volume 5931 of *LNCS*, pages 24–44. Springer, Germany, December 2009.

[4] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. Gridflow: Workflow management for grid computing. In *CCGRID '03: Proceedings of the 3rd International Symposium on Cluster Computing and the Grid*, pages 198–205, Washington, DC, USA, 2003.

[5] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237, 2005.

[6] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington. Iceni: an open grid service architecture implemented with jini. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, 2002.

[7] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12), 2007.

[8] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[9] J. Louchet, M. Guyon, M. J. Lesot, and A. Boumaza. Dynamic flies: a new pattern recognition tool applied to stereo sequence processing. *Pattern Recognition Letters*, 23(1-3):335–345, 2002.

[10] W. Z. Lu, H.-Y. Fan, A. Y. T. Leung, and J. C. K. Wong. Analysis of pollutant levels in central hong kong applying neural network method with particle swarm optimization. *Environmental Monitoring and Assessment*, 79(3):217–230, Nov 2002.

[11] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system: Research articles. *Concurrency and Computation: Practice & Experience*, 18(10):1039–1065, 2006.

[12] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, November 2004.

[13] C. u. O. Ourique, E. C. J. Biscaia, and J. C. Pinto. The use of particle swarm optimization for dynamical analysis in chemical processes. *Computers and Chemical Engineering*, 26(12):1783–1793, 2002.

[14] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. Dobson, and K. Chiu. A grid workflow environment for brain imaging analysis on distributed systems. *Concurrency and Computation: Practice & Experience*, 21(16):2118–2139, November 2009.

[15] A. Salman. Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 26(8):363–371, November 2002.

[16] T. Sousa, A. Silva, and A. Neves. Particle swarm based data mining algorithms for classification tasks. *Parallel Computing*, 30(5-6):767–783, 2004.

[17] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3):1930–1947, March 2007.

[18] I. Taylor, I. Wang, M. Shields, and S. Majithia. Distributed computing with Triana on the grid: Research articles. *Concurrency and Computation: Practice & Experience*, 17(9):1197–1214, 2005.

[19] J. D. Ullman. Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10(3), 1975.

[20] C. Vecchiola, M. Kirley, and R. Buyya. Multi-objective problem solving with offspring on enterprise clouds. *Proceedings of the 10th International Conference on High-Performance Computing in Asia-Pacific Region (HPC Asia 2009)*, pages 132–139, March 2009.

[21] K. Veeramachaneni and L. A. Osadciw. Optimal scheduling in sensor networks using swarm intelligence. 2004.

[22] P.-Y. Yin, S.-S. Yu, and Y.-T. Wang. A hybrid particle swarm optimisation algorithm for optimal task assignment in distributed systems. *Computer Standards and Interfaces*, 28(4):441–450, 2006.

[23] H. Yoshida, K. Kawata, Y. Fukuyama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage stability. In *the International Conference on Intelligent System Application to Power System*, pages 117–121, 1999.

[24] B. Yu, X. Yuan, and J. Wang. Short-term hydro-thermal scheduling using particle swarm optimisation method. *Energy Conversion and Management*, 48(7):1902–1908, 2007.

[25] J. Yu, R. Buyya, and K. Ramamohanarao. *Workflow Scheduling Algorithms for Grid Computing*, volume 146, pages 173–214. Springer Heidelberg, 2008.

[26] A. E. M. Zavala, A. H. Aguirre, E. R. Villa Diharce, and S. B. Rionda. Constrained optimisation with an improved particle swarm optimisation algorithm. *Intl. Journal of Intelligent Computing and Cyvernetics*, 1(3):425–453, 2008.

[27] L. Zhang, Y. Chen, R. Sun, S. Jing, and B. Yang. A task scheduling algorithm based on pso for grid computing. *International Journal of Computational Intelligence Research*, 4(1), 2008.