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

Suraj Pandey[1], Linlin Wu[1], Siddeswara Guru[2], Rajkumar Buyya[1]
[1]Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{spandey, linwu, raj}@csse.unimelb.edu.au

[2]Tasmanian ICT Centre, CSIRO
Tasmania, 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 scheduling heuristic for data intensive applications 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 analyze the cost savings when using PSO as compared to using existing 'Best Resource Selection' (BRS) algorithm. Our results show that we can achieve: a) as much as 3 times cost savings as compared to BRS, b) good distribution of workload onto resources, when using PSO based scheduling heuristic.

1 Introduction

Nowadays, many scientific experiments such as structural biology and chemistry, neuroscience data analysis and disaster recovery are conducted through complex and distributed scientific computations that are represented and structured as scientific workflows [8]. 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, which are made available as subscription-based services in a pay-as-you-go model to consumers [2]. It 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 an iterative method called Particle Swarm Optimization (PSO).

Particle Swarm Optimisation (PSO) is a self-adaptive global search optimisation technique introduced by Kennedy and Eberhart [9]. The algorithm is similar to other population-based algorithms like Genetic algorithms but, there is no direct combination of individuals of the population. Instead, it relies on the social behaviour of the par-

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

ticles. 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, 6], data mining [16], chemical engineering [4, 13], pattern recognition [10] and environmental engineering [11]. The PSO has also been applied to solve *NP-Hard* problems like Scheduling [24, 21] and task allocation [22, 26].

Our main contributions in this paper are as follows:

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

The rest of the paper is organized as follows: Section 2 presents related work, Section 3 describes the PSO model. In Section 4 we describe the cost minimization problem with a help of an example workflow. In Section 5, we present our scheduling heuristic that uses PSO. Section 6 presents an experimental evaluation of the performance our heuristic. Section 7 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. 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, incurs higher transmission cost and storage cost, than compute processes 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 [3], ICENI [7], GridAnt [1] and Triana [18]. 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.

Previous work have used Genetic Algorithm (GA) for scheduling workflows [25]. However, GA is not 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. Also, 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. It allows us to use the fitness function directly for the optimization problem. Moreover, using discrete numbers, we can easily correlate particle's position to task-resource mappings.

3 Particle Swarm Optimization

Particle Swarm Optimisation (PSO) is a swarm-based intelligence algorithm [9] influenced by the social behaviour 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 initialised randomly. Each particle will have a fitness value, which will be evaluated by a fitness function to be optimised 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. The particle will have velocity, which directs the flying of the particle. In each generation the velocity and the position of particles will be updated as follows:

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

$$x_i^{k+1} = x_i^k + v_i^{k+1}, \quad (2)$$

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 Cost Minimization Problem

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 minimize the cost such that it completes within the time (deadline) a user specifies.

We denote an application workflow using a Directed Acyclic Graph (DAG) 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, $tdata^k = (T_j, T_k) \in E$ 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 $txcost_{i,j}$ (datacenter access cost and data transfer cost) from a storage resource S_i to a compute resource PC_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. We assume that these costs are non-negative, symmetric, and satisfy the triangle inequality: that is, $txcost_{i,j} = txcost_{j,i}$ for all $i, j \in N$, and $txcost_{i,j} + txcost_{j,k} \geq txcost_{i,k}$ for all $i, j, k \in N$.

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 to the Evolutionary Multi-objective Optimization (EMO) application [20]. Each task may have an input file (e.g. f_1) and generates output files after the task has been completed

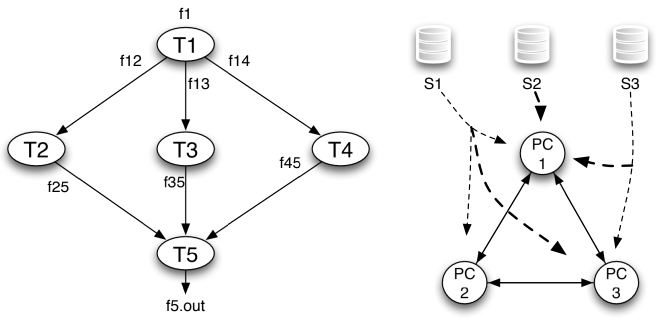


Figure 1: An example workflow, storage (S) and compute (PC) nodes.

$(f_{12}, f_{13}, ..., f_{45})$. The figure also depicts three compute resources $(PC1, PC2, PC3)$ connected with storage units $(S1, S2, S3)$ with varying network bandwidth (dark lines represent faster links). The goal is to assign the workflow tasks to the compute resources such that the 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. 3). 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. 4). 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.

Equation 6 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 7) ensures that the total cost is minimal even after initial distribution.

$$C_{exe}(M)_j = \sum_k w_{kj} \qquad \forall M(k) = j \quad (3)$$

$$C_{tx}(M)_j = \sum_{k1 \in k} \sum_{k2 \in k} d_{M(k1),M(k2)} e_{k1,k2} \qquad \forall M(k1) = j \; and \; M(k2) \neq j \quad (4)$$

$$C_{total}(M)_j = C_{exe}(M)_j + C_{tx}(M)_j \quad (5)$$
$$Cost(M) = max(C_{total}(M)_j) \quad \forall j \in P \quad (6)$$
$$Minimize(Cost(M) \quad \forall M) \quad (7)$$

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 transfer cost (Eq. 5). Then, the total cost for all the assignments will be dominated by the highest cost of a compute resource (Eq. 6). Hence, the goal of the assignment is to minimize this cost (Eq. 7).

5 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 2, and b) the PSO steps as listed in Algorithm 1.

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. 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 time between resources per unit data. This time can be obtained from logs, or through predictions. 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).

Algorithm 1 PSO algorithm.

```
1: Set particle dimension as equal to the size of ready tasks in $\{t_i\} \in T$
2: Initialise particles position randomly from $PC = 1, ..., j$ and velocity $v_i$ randomly.
3: For each particle, calculate its fitness value Equation 6.
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 1 and update their positions using Equation 2.
7: If the stopping criteria or maximum iteration is not satisfied, repeat from Step 3.
```

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 tasks that are "ready" 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 will have changed, we recompute the PSO mappings. 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.

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 conditions.

Algorithm 2 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 optimized particle position given 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
```

PSO: The steps in the PSO algorithm are listed in Algorithm 1. The algorithm starts with random initialization of particle's position and velocity. Based on these values, it evaluates the fitness function. On each evaluation, the $pbest$ and $gbest$ values are updated according to Equations

1-2. The evaluation is carried out in a loop until the results converge or until the specified number of iterations (user-specified stopping criteria).

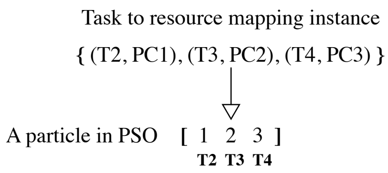

Task to resource mapping instance
{ (T2, PC1), (T3, PC2), (T4, PC3) }
A particle in PSO [1 2 3]
T2 T3 T4

Figure 2: A representation of task to resource mapping as a PSO particle.

PSO Particle: Figure 2 depicts the representation of a $3 - dimentional$ particle. Each position in the particle represents a task and the value at that position represents the mapping of the task to a resource.

6 Experimental Evaluation

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

6.1 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 2), and best resource selection (based on minimum completion time by selecting a resource with maximum cost). The heuristic that achieves minimum cost of execution but still meets the users' deadline constraint is considered superior to others.

6.2 Data and Implementation

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

[4]http://aws.amazon.com/cloudfront/

Table 1: The TP-matrix, PP-matrix and DS-matrix

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

$$TP[i,j] = Costofexecutionof T_i \ at PC_j$$

$$PP[3 \times 3] = \left[\begin{array}{c|ccc} & PC1 & PC2 & PC3 \\ \hline PC1 & 0 & 0.17 & 0.21 \\ PC2 & 017 & 0 & 0.22 \\ PC3 & 0.21 & 0.22 & 0 \end{array}\right]$$

$$PP[i,j] = Costofcommunicationbetween PC_i \ \& PC_j$$
$$(EC2priceofresourcesintherange 1.1 - 1.28/hr)$$

$$DS_{T2,T3,T4}[2 \times 2] = \left[\begin{array}{c|cc} & f1 & f2 \\ \hline i/p & 5 & 5 \\ o/p & 5 & 5 \end{array}\right]$$

$$DS_{T5}[2 \times 2] = \left[\begin{array}{c|cc} & f1 & f2 \\ \hline i/p & 15 & 15 \\ o/p & 30 & 30 \end{array}\right]$$

$$row_1 = i/p \ file \ sizes, \quad row_2 = o/p \ file \ sizes$$

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/hour, 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).

For solving our PSO formulation, we modified and used the JSwarm[6] package available from sourceforge. Table 2 lists the values we used for the number of particles, iterations and the number of experiments (for calculating the confidence intervals) to obtain the results.

Table 2: The values for PSO optimization

Default # of particles = 25
Number of iterations = 20
Number of experiments = 30

6.3 Experiments and Results

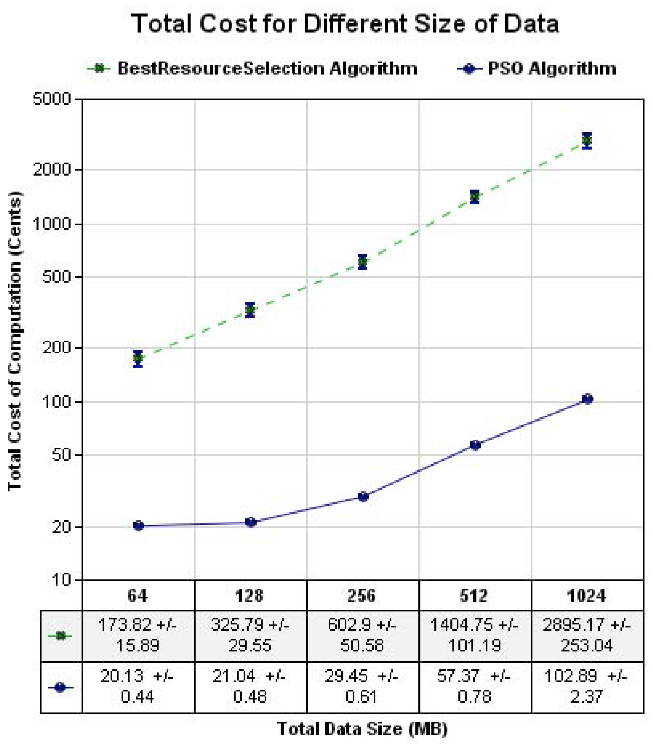
We evaluated the scheduling heuristic using the workflow depicted in Figure 1. Each task in the workflow has

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

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.

6.3.1 Variation in Total Data Size of a Workflow

We varied the size of total data processed by the workflow uniformly in the range 64-1024 MB. By varying data size, we compared the variance in total cost of execution for the two algorithms as depicted in Figure 3. We also measured the distribution of workload on resources, as depicted in Figure 4.



	64	128	256	512	1024
BestResourceSelection Algorithm	173.82 +/- 15.89	325.79 +/- 29.55	602.9 +/- 50.58	1404.75 +/- 101.19	2895.17 +/- 253.04
PSO Algorithm	20.13 +/- 0.44	21.04 +/- 0.48	29.45 +/- 0.61	57.37 +/- 0.78	102.89 +/- 2.37

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) when the size of data increases. The graph is plotted by averaging the result of 30 executions. The graph also plots 95% Confidence Interval (CI) for each data point. The cost obtained by PSO based task-resource mapping increases much slowly 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 +/- 2.37, which is much lower as compared to the BRS algorithm (+/- 253.04), for 1020 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, hence a global minima is found. 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.

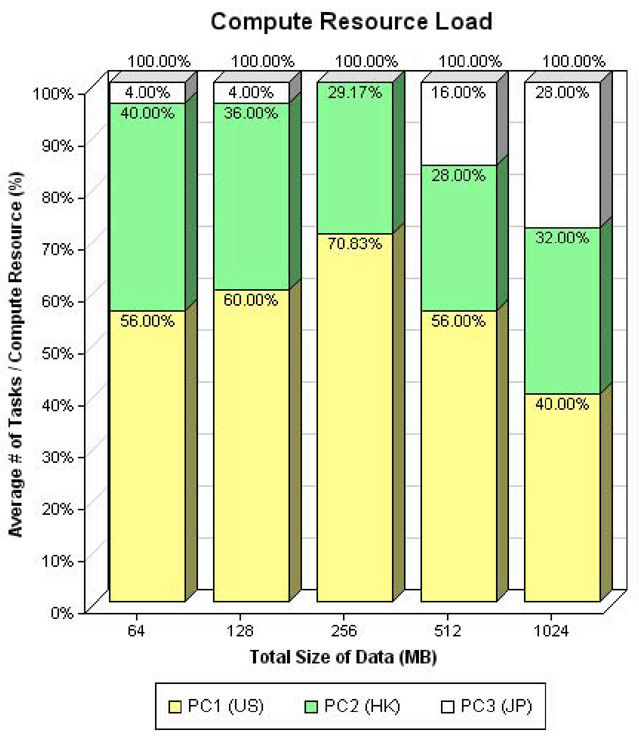


Figure 4: Distribution of workflow tasks on available processors.

Distribution of Load: For various sizes of total data processed, we calculated the distribution of workflow tasks onto available resources, 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, thus minimizing communication cost to zero (Equation 6 tries to avoid this as much as possible). The X-axis has the size of data and the Y-axis the average number of tasks a compute resource received for every size of data processed, as a percentage. The figure clearly shows that tasks are well distributed for all the sizes of data. When the size of data is smaller (for 64-256 MB) the tasks were distributed to the cheaper resources ($PC1 - PC2$). However, when the size of data increased over 256MB, the tasks were distributed fairly equally among all the compute resources. 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, as well as faily balance the load on available resources.

6.3.2 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).

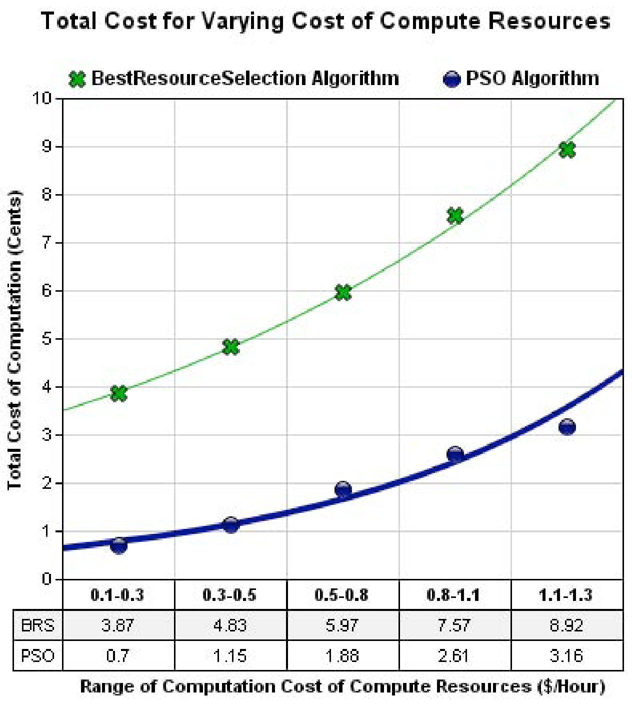
Figure 5 depicts the change in total cost of computation of applications for different range of compute resource prices (price range are similar to Amazon EC2 instances in US and Europe combined). The plotted values are an average of 30 executions. We use curve fitting to plot the lines along the points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

Clearly, PSO based mapping has much lower cost as compared to BRS based mapping. In addition, the slope of the trend line shows that PSO maps tasks to resources such that the percentage rise in total cost for increasing resource costs is lower than that computed by BRS.

The reason for PSO's improvement over BRS is due to PSO's global optimization strategy when mapping tasks in the workflow. 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 user of BRS leads to more costs due to the affinity towards better resource. Whereas, PSO analyzes the local best and the global best values before changing the task-resource mappings.

7 Conclusions & 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 scientific application workflows on Cloud computing environments. We varied the communication cost between resources, the execution cost of compute resources and compared the results 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. In addition, PSO balances the load on compute resources by distributing the tasks the available resources.



	0.1-0.3	0.3-0.5	0.5-0.8	0.8-1.1	1.1-1.3
BRS	3.87	4.83	5.97	7.57	8.92
PSO	0.7	1.15	1.88	2.61	3.16

Figure 5: Comparison of total cost between PSO based resource selection and best resource selection algorithms when varying computation cost of resources.

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 of the University of Melbourne for his comments on the experiments and results.

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] R. Buyya, S. Pandey, and C. Vecchiola. Cloudbus toolkit for market-oriented cloud computing. In *Proceeding of the 1st*

International Conference on Cloud Computing (CloudCom 2009), Beijing, China, December 2009. Springer, Germany.

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

[4] A. R. Cockshott and B. E. Hartman. Improving the fermentation medium for echinocandin b production part ii: Particle swarm optimization. *Process Biochemistry*, 36:661–669, 2001.

[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] Y. Fukuyama, S. Takayama, Y. Nakanishi, and H. Yoshida. A particle swarm optimization for reactive power and voltage control in electric power systems. In *the Genetic and Evolutionary Computation Conference 1999 (GECCO 1999)*, pages 1523–1528, 1999.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[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 and Experience*, In Press, 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, M. Shields, I. Wang, and R. Philp. Distributed p2p computing within triana: A galaxy visualization test case. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, 2003.

[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 146/2008, chapter 7, pages 173–214. Springer Berlin / 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. *International 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.