Title	A learning architecture for scheduling workflow applications in the cloud
Author(s)	Barrett, Enda; Howley, Enda; Duggan, Jim
Publication Date	2011-09-15
Publication Information	Barrett, E., Howley, E., & Duggan, J. (14-16 Sept. 2011). A learning architecture for scheduling workflow applications in the cloud. Paper presented at the Web Services (ECOWS), 2011 Ninth IEEE European Conference on.
Publisher	IEEE
Link to publisher's version	http://dx.doi.org/10.1109/ecows.2011.27
Item record	http://hdl.handle.net/10379/3935
DOI	http://dx.doi.org/10.1109/ecows.2011.27

A Learning Architecture for Scheduling Workflow Applications in the Cloud

Enda Barrett
IT Department
National University of Ireland, Galway
Galway, Ireland
Email: Enda.Barrett@nuigalway.ie

Enda Howley
IT Department
National University of Ireland, Galway
Galway, Ireland
Email: Enda.Howley@nuigalway.ie

Jim Duggan
IT Deparment
National University of Ireland, Galway
Galway, Ireland
Email: James.Duggan@nuigalway.ie

Abstract—The scheduling of workflow applications involves the mapping of individual workflow tasks to computational resources, based on a range of functional and non-functional quality of service requirements. Workflow applications require extensive computational requirements, and often involve the processing of significant amounts of data. Furthermore, dependencies that exist amongst tasks require that schedules must be generated strictly in accordance with defined precedence constraints. The emergence of cloud computing has introduced a utility-type market model, where computational resources of varying capacities can be procured on demand, in a pay-per-use fashion. In general the two most important objectives of workflow schedulers are the minimisation of both cost and makespan. As well as computational costs incurred from processing individual tasks, workflow schedulers must also plan for data transmission costs where potentially large amounts of data must be transferred between compute and storage sites. This paper proposes a novel cloud workflow scheduling approach which employs a Markov Decision Process to optimally guide the workflow execution process depending on environmental state. In addition the system employs a genetic algorithm to evolve workflow schedules. The overall architecture is presented, and initial results indicate the potential of this approach for developing viable workflow schedules on the Cloud.

I. Introduction

The recent advancement of Cloud computing and its pay per use model enable the procurement of large amounts of computational resources on demand. Cloud providers leveraging on large economies of scale are capable of delivering increasing amounts of computational resources at lower costs and with greater reliability. Cloud computing delivers computational resources by means of virtualisation technologies. Through the instantiation of virtual machines users can deploy their applications on resources with varying performance and cost levels. Most recently numerous scientific experiments in fields such as astronomy [1], epigenomics [2] and neuroscience involve complex computational analyses on large data sets. These communities rely greatly on insights gained through computational analysis in order to advance the state of the art in their respective communities. Many of these computational experiments can be characterised neatly as workflows, with large segments capable of being executed in parallel. Initially the processing of many large scientific models and application workflows involved a signification investment in high performance computing infrastructures. The initial monetary outlay, coupled with setup and on-going maintenance costs, rendered it prohibitive to most. Apart from a small number of well funded research projects most didn't have the resources. In an answer to this computational grids were established, where through Virtual Organisations the scientific community could share resources among each other in a large distributed grid network. For sharing the computational resources at their disposal, individuals were allocated pre-reserved slots to their own experiments. However these platforms possessed a number of limitations both technical and bureaucratic [3]. Often minor research groups could not gain access to the global grids. Furthermore, many of these grids use pre-defined platforms with specific operating systems, application suits and API's. Platform limitations present a clear barrier to enabling workflows to execute without a significant technical overhead. Recently there has been significant high profile attention regarding the possibilities of executing scientific workflows on the cloud [4]. A recent study highlighted the cost effectiveness of executing scientific workflows on Amazon EC2[1] instances was cost effective [5]. However with the market oriented model of clouds, existing workflow management tools need to be adapted to better utilise cloud platforms. Previous workflow scheduling approaches for grids focussed on resources which were reserved in advance. However through virtual machines greater amounts of resources can be procured in real time and on demand. Scheduling approaches for clouds should be dynamic (online) and adaptive to system behaviours. They should also be capable of adjusting to fluctuating costs for communication and computation. To address these issues we propose a novel cloud workflow scheduler capable of scheduling application workflows in a cloud computing environment. We schedule workflow applications based on user specified QoS constraints, namely cost and makespan. We adopt a multifaceted approach to scheduling where a genetic algorithm is used to generate optimal schedules and a Markov Decision Process (MDP) chooses from among them based on the observed state of the environment. The virtualised nature of cloud computing, with multiple virtual machine deployed on the same hosts, competing for CPU cycles and memory, means that task completion times can vary due to temporal

---

[1]http://aws.amazon.com/ec2/

fluctuations. In addition to this, data transfer times among tasks in a workflow can also be affected as a result of network congestion. By employing a MDP our approach is capable of observing temporal dynamics and adjusting schedule selection accordingly.

The rest of this paper is structured as follows: Background Research provides an overview of relevant and related work in this field. A number of aspects of workflow scheduling and cloud computing are discussed. Workflow architecture details the specific components of our cloud workflow scheduling approach. Cloud workflow scheduling details the problem of scheduling workflows onto cloud resources. Also included is a detailed description of both MDPs and Genetic Algorithms. Initial Results details our preliminary findings, leading to Conclusions & Future Work.

II. BACKGROUND RESEARCH

Extensive research has taken place on the scheduling of workflow applications onto distributed resources in grid computing. Yu et al [6] proposed a workflow scheduling based on MDPs to schedule tasks on utility grids. Their approach partitions the workflow into branches, where they allocate sub-deadlines to each branch based on the overall deadline. A branch containing sequential tasks are then mapped to services using a MDP. The main objective is to meet the overall deadline, concurrently minimising cost. In addition to that a scheduling approach based on genetic algorithms was also developed [7]. Depending on user preference the approach was constrained to optimising either execution cost or the overall makespan. This work was extended further [8] with the inclusion of multiobjective evolutionary algorithms. Our scheduling approach has its roots based in here where a genetic algorithm is used to evolve solutions to the workflow scheduling problem. However our approach differs to these in two regards, firstly we schedule on clouds where virtual machines of varying capacity can be created on demand and without advanced reservation. Secondly we employ a Markov Decision Process (MDP) to optimally choose from amongst the evolved schedules.

Over the years many grid and cluster workflow management tools have been proposed. Many of these have been devised to facilitate the execution of scientific workflows. Tools such as Pegasus [9], Kepler [10] and Taverna workbench [11] have all successfully executed scientific workflows on computational grids and clusters. These generally schedule tasks based on earliest finish time, earliest starting time or high processing capabilities. These can be considered as "best resource selection" (BRS) [12] approach, where a resource is selected solely based on its performance.

Pandey et al. recently developed cloud scheduling approach based on particle swarm optimisation [12]. Experiments have shown that PSO's can provide a good distribution of tasks across the available services, reducing resource overloading and performed better than the BRS approach. This improved performance is due to the PSO scheduler factoring in communication costs between all tasks was the fact that the PSO

scheduler took in account communication costs between all tasks. Wu et al. [13] recently proposed a market oriented hierarchical scheduling approach to cloud workflow scheduling. They adopt a two stage scheduling approach, aiming to optimising scheduling at the overall service level and task virtual machine level. They employ a number of meta-heuristic algorithms to evolve different schedules allowing the user to choose a schedule from amongst those returned.

In general much of the existing work scheduling workflows on computational grids or clouds focus on developing different heuristic and meta-heuristic algorithms capable of finding optimal or near optimal workflow schedules. A workflow management system then monitors the execution progress, rescheduling as necessary depending on task failures, delays or exceptions. Rescheduling in this manner will most likely incur additional costs, as faster/greater numbers of resources will need to be acquired, should problems arise.

A number of studies have looked at applying reinforcement learning to resource allocation problems [14]. The author presented a framework using reinforcement learning, capable of dynamically allocating resources in a distributed system. Reinforcement learning methods are methods to solve an MDP where a complete model of the environment is not available. They allow the learning agent to learn through observed rewards as they explore the state space. While resource allocation is more concerned with low level scheduling of tasks at the virtual machine level, the parallels between them still merit their inclusion. Tesauro investigated the use of a hybrid reinforcement learning technique for autonomic resource allocation [15]. He applied this research to optimizing server allocation in data centers. Germain-Renaud et al. [16] looked at similar resource allocation issues. Here a workload demand prediction technique was used to predict the resource allocation required each time. Reinforcement learning has also been successfully applied to grid computing as a job scheduler. Here the scheduler can seamlessly adapt its decisions to changes in the distributions of inter-arrival time, QoS requirements, and resource availability [17]. The objective of applying reinforcement learning to resource allocation for load balancing purposes. Our work differs from these in that we schedule tasks with dependencies and our objective is optimise both costs and makespan.

III. WORKFLOW ARCHITECTURE

Figure 1 details the architecture of the cloud workflow management system. Users submit their application workflows through the user interface, along with their non-functional QoS constraints. The next stage of the process involves performance estimation on the submitted workflow. Performance estimation calculates the average execution time that a given task will take to run on a specified resource. In cloud environments resources can be virtualised based on CPU, memory and storage. As the cost of cloud services is inversely proportional to their processing capabilities, once you know the average execution time for a specific resource configuration you can infer across all resources. Techniques such as analytical modelling [18]
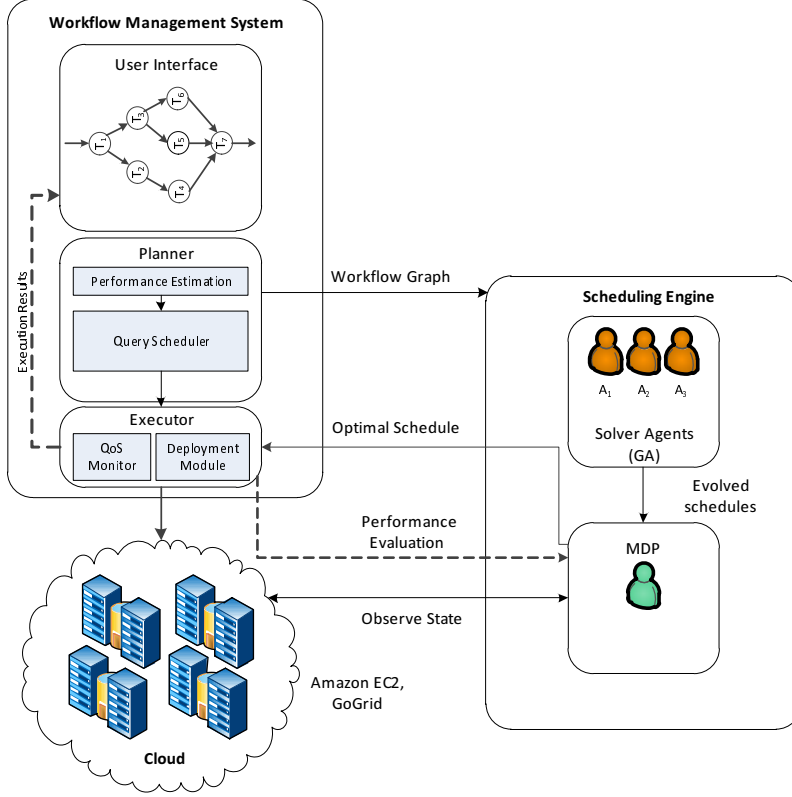
Fig. 1. Workflow Management System Architecture

or historical data [19], can be used to determine average execution times.

These values are used by the solvers to estimate the makespan of the workflow. A number of solvers with ranging configurations are instantiated to produce schedules of varying cost and makespan. From these schedules an agent utilising a MDP computes the optimal schedule based on the current state of the cloud environment. The scheduling plan is executed on the cloud via the Executor module. This module queues the tasks from the schedule with the associated resources. It also monitors the tasks and returns the results to the user interface. Once a processing schedule has completed on the cloud, the QoS monitor returns the actual cost and makespan incurred by the schedule. This is then returned to the MDP agent who updates the transition probabilities accordingly.

IV. Cloud Workflow Scheduling

Workflow applications can generally be modelled as a Directed Acyclic Graph (DAG) $G = \{V, E\}$. The set of vertices $V = \{T_1, ..., T_n\}$ denotes each individual task in the workflow. E the set of directed edges represents precedence constraints between task nodes. A directed edge $E_{i,j}$ states that T_i is the parent task of T_j. Child tasks can only be executed once all the parent tasks have completed. $F_{i,j}$ denotes a data dependency between tasks T_i and T_j.

We have a finite set of compute services $C = \{C_1, ..., C_n\}$ which are capable of executing T_i. The execution of task T_i on a compute service C_j incurs a cost. This cost is inversely

proportional to time taken to process it, where the greater the expense, the faster the resource. We also have a set of storage sites $S = \{S_1, ..., S_n\}$. The cost of data transfer per unit of data between C_i and S_j is fixed and known in advance. The total data costs incurred by tasks executing on a given compute service C_i is $D_{total}(C)_i$. This includes all data costs between tasks executing on service C_i and those that are not. The overall makespan M_{total} of the workflow application is defined as the latest finished time on all the virtual machine. Let M_{C_i} be equal to the total makespan of compute service C_i. Then the total makespan for the entire workflow is

$$M_{total} = max(C_i) \forall i \in C \tag{1}$$

Data transfer costs between compute services can be calculated by the file size of the output of the parent task. Generally for two tasks executing on the same resource there is no data transfer cost. The total processing costs P for a given compute service is P_{total}. Total task execution costs are

$$Ex_{total} = D_{total}(C)_i + P_{total}(C)_i \forall i \in C \tag{2}$$

Figure 2a depicts a sample workflow DAG containing 7 task nodes. The edges between tasks indicate the input files and output files between task nodes. Figure 2b depicts a valid schedule for the adjacent workflow. Each individual task is assigned to a specific resource and can only execute once its parent tasks have completed. The objective of the cloud workflow scheduler is to find a mapping from tasks to compute

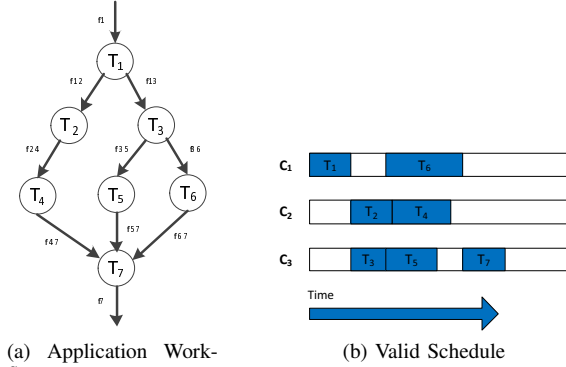services that minimises the cost of execution and the overall makespan of the workflow.

(a) Application Workflow

(b) Valid Schedule

Fig. 2. Sample Workflow application with feasible schedule

$$Min(Ex_{total} + M_{total}) \tag{3}$$

In general workflow scheduling onto distributed resources is an Np-Complete problem [20]. To generate near optimal solutions we apply a metaheuristic algorithm to solve the workflow scheduling problem.

A. Genetic Algorithms

Genetic algorithms are stochastic search and optimization techniques based on evolution. In their simplest form, a set of possible solutions to a particular problem are evaluated in an iterative manner. From the fittest of these solutions, the next generation is created and the evaluation process begins once more. A solutions suitability to its environment is determined using a fitness function. By iterating through successive generations good approximate solutions can be found for the given environment.

A task to resource mapping $\{T_i, C_j\}$ represents a single gene in the chromosome. A valid chromosome contains a sequence of genes, mapping every task in the workflow to corresponding resource. The order of the genes represents the schedule execution order on the chosen resources. A feasible solution to the scheduling problem must maintain the precedence constraints between the tasks specified in the directed acyclic graph.

The genetic algorithm used is outlined by Algorithm 1. Firstly an initial population of feasible schedules is created. Next a given solution's fitness is evaluated according to Equation 3. Individuals are selected for reproduction using roulette wheel selection, based on their fitness. Roulette wheel selection involves ranking chromosomes in terms of their fitness and probabilistically selecting them. The selection process is weighted in favour of chromosomes possessing a higher fitness. To ensure that agents, already optimal for their environment are not lost in the evolutionary process elitism is applied. Elitism involves the selection of a certain percentage of the fittest chromosomes and moving them straight into the next generation, avoiding the normal selection process. In creating offspring for the next generation, the selection of two

---

Algorithm 1 Genetic Algorithm

Initialise population of feasible schedules

repeat
    Evaluate chromosome fitness Equation 3
    Rank chromosomes according to overall population fitness
    Select parents using roulette wheel selection
    if (random > crossoverRate) **then**
        Apply single point crossover
    end if
    if (random > mutationRate) **then**
        Apply mutation
    end if
    Convert to feasible schedules
    Create next generation
until end

---

parents is required. Each pairing results in the reproduction of two offspring. Figure 4 shows the crossover of two parents and subsequent production of two offspring. Crossover involves taking certain aspects/traits of both parents' chromosomes and creating a new chromosome. There are a number of ways to achieve this including, single point crossover, two point crossover and uniform crossover. Our crossover function employs single point crossover, where a point in the bit string is randomly selected, at which crossover is applied. Crossover generally has a high probability of occurrence. Figure 4 shows two valid schedules combining to create two valid offspring. However crossover can also result in the creation of schedules which are invalid. To convert an invalid schedule into a feasible solution we apply an adjustment after crossover has completed [21]. Mutation involves randomly altering the bit string altering aspects of a chromosome. Mutation occurs on the assigned service in a given task-service mapping as seen in Figure 3. Once the required number of offspring have
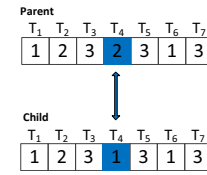

Fig. 3. Mutation

been created they form the next generation and the process begins once more. The algorithm terminates when the stopping condition is met.
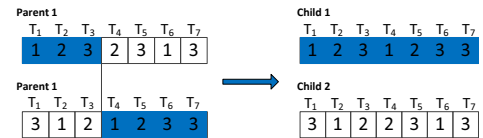

Fig. 4. Single point crossover

1) Solver Agents: Each agent employs a GA to evolve solutions to the scheduling problem. For our experiments evolution occurs over the entire population, with offspring from the fittest solutions replacing only the weakest schedules in the population. Fitness is evaluated according to the Equation 3. After each generation a solutions total cost and execution time infers its fitness in the environment. For selection purposes we normalise fitness according to the following equation.

$$F(i) = \frac{f_{max} - f_i}{f_{max} - f_{min}} + \mu \qquad (4)$$

The fitness $F(i)$ is the normalised fitness of individual i. f_{max} and f_{min} are the maximum and minimum fitness values in the population, with $\mu = 0.3$. The normalised fitness ensures that the highest probability is apportioned to the fittest solutions during the selection process. The average values for population size, elitism, cross-over and mutation are 50, 5%, 85% and 3%. Each solver is given its own unique configuration, to ensure a diversity of schedule solutions. The solvers also apply a weighting to the average values associated with task processing times. This value is in the range of $[0, 1]$. A value close 1 associates high degree of confidence in the predicted task execution times, whereas a value close to 0 indicates low confidence. These parameters ensure a diverse range of solutions with respect to cost and makespan are returned.

V. MARKOV DECISION PROCESSES

Markov Decision Processes are a particular mathematical framework suited to modelling decision making under uncertainty. It can be represented as a tuple (S, A, P, R). In general the learning agent interacts with its environment through a sequence of discretized time steps. At the end of each time period t the agent occupies state $s_t \in S$, where S represents the set of all possible states. The agent then chooses an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of all possible actions within state s_t. The transition probability P defines the probabilities of moving from one state to the next.

$$P_{s,s'}^a = Pr\{s_{t+1} = s'|s_t = s, a_t = a\} \qquad (5)$$

R is the expected value of the next reward [22]. The objective of the learning agent is to optimize its value function. The agent makes decisions on its value estimates of states and actions. $V^\pi(s)$ is called the state-value function. It is the expected value of being in a particular state under policy π.

$$V^\pi(s, a) = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} \qquad (6)$$

The goal of the learning agent is to maximize its returns in the long run often forgoing short term gains in place of long term benefits. By introducing a discount factor γ, $(0 < \gamma < 1)$, an agents degree of myopia can be controlled. A value close to 1 for γ assigns a greater weight to future rewards, while a value close to 0 considers only the most recent rewards. To solve an MDP value iteration or policy iteration algorithms from dynamic programming can be used. In this work we use value iteration in order to compute optimal policies for the MDP.

Approximations of $V^\pi(s, a)$ which are indicative as to the benefit of taking action a while in state s are calculated after each time interval. Actions are chosen based on π the policy being followed. The policy denotes the optimal mapping from states to actions.

Algorithm 2 outlines the value iteration algorithm used to solve the MDP. Firstly all states are initialised arbitrarily. Our state signal is comprised of three variables, which are **Time**, **Resource Load** and **Execution Success**. The first variable *Time* is necessary in order to analyse the effects of temporal fluctuations on the executing workflows. At peak times (e.g between 7-9pm) in any given region there could be significant effect on the workflow execution performance. By including time into the state space the MDP can reason about the possible effects of this. The second variable is the overall *Resource Load* of the virtual machines which the workflow system has instantiated. It coarsely evaluates its own resources by summing up all workflow tasks currently executing and divides this by the number of virtual machines. Resource Load is split into distinctions, *light*, *moderate* and *heavy*. This coarse interpretation is surprisingly effective at enabling a good estimate of the system load. The final variable *Execution Success* defines whether or not a given selection was successful in executing the workflow. This information is returned via the QoS monitor. There are many other possible metrics that could, added to the state signal, give even greater insights (CPU utilisation, disk reads, data transfers). However for the sake of simplicity and to enable a manageable state set we focus on these three. If we assume time to transition on an hourly basis (0-23), then the total state space consists of a total of 144 states. The set of all possible actions in the MDP are defined by the number of solver agents. The MDP may choose a schedule from amongst the set of solvers. The rewards are defined as the cost incurred as a result of the execution. If a schedule resulted in the violation of the specified deadline, then additional penalties are added to the reward. The goal of the MDP is to ensure the successful completion of the workflow execution within the budget and deadline constraints.

Algorithm 2 Value Iteration

Initialize V arbitrarily
> **repeat**
>> $\Delta \leftarrow 0$
>> For each s $\in$ S
>> $v \leftarrow V(s)$
>> $V(s) \leftarrow min_a \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V(s')]$
>> $\Delta \leftarrow max(\Delta, |v - V(s)|)$
> **until** $\Delta < \Theta$
> Output policy π
> For all s $\in$ S
> $\pi(s) = argmin_a \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V(s')]$

---

2) Bayesian Model Learning: One important consideration is that approximating the true values of $V(s)$ is largely dependent on the number of times state s is visited. In

fact the true values will only converge on the limit. Value iteration generally only guarantees asymptotic convergence. We adopt a Bayesian model learning approach similar to what was proposed by Doshi et al. [23]. The MDP updates its estimates of the transition probabilities based on experience. When submitting an evolved schedule for the execution, the MDP agent needs to have an estimate of the certainty with which the executed schedule will result in the desired outcome. These estimates are a measure of the uncertainty of the underlying processes and are manifested in the transition function P. The optimal policy $\pi*$ is naturally dependent on these estimates. In order to achieve better policies one needs to update approximate transition probabilities based on experience. In a MDP, choosing an action a in state s results in a state transition from s to s'. The experience we have of this transition is a measure of how certain our actions will result in the predicted outcome. In our model we initialise an experience counter $Expc$ to 1 and each time we observe a particular state transition the experience associated with it is incremented by one. The updated transition probability is given by Equation

$$P'(s = s'|a, s = s) = \frac{P(s = s'|a, s = s) \times Expc + 1}{Expc'} \quad (7)$$

where Expc' is the incremented counter. We interleave workflow execution with model learning to continuously update the MDP's estimates of the true transition probabilities. This allows the MDP increase its approximations of the true transition probabilities.

VI. INITIAL RESULTS

This section presents our initial findings with our cloud workflow system.

A. Experimental Setup

To evaluate our MDP approach we used Cloudsim a simulator developed by the CLOUDS group at the University of Melbourne [24]. Cloudsim is cloud simulation tool which allows you to model typical cloud scenarios on a single processor. It facilitates the creation of data centres and the deployment of multiple hosts and Virtual machines of varying capacities. You can also specify detailed pricing policies with regard to virtual machines and data transmission costs. In our experiments we simulate four separate data centers in four geographic locations. We place one in the US, one in Europe, one in Hong Kong/Singapore and one in Japan. Data transfers between compute sites are serviced by a content delivery network (CDN) such as Amazon CloudFront[2]. Each data centre supports a single host capable of instantiating multiple virtual machines. Table I depicts Amazons current costs per unit data transfer between the four geographic regions at the time of writing.

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

TABLE I
COMMUNICATION COSTS BETWEEN REGIONS

	C_1	C_2	C_3	C_4
C_1	0	0.15	0.19	0.20
C_2	0.15	0	0.19	0.20
C_3	0.15	0.15	0	0.20
C_3	0.15	0.15	0.19	0

B. Increasing Data Size

Our first experiment analyses the cost savings of the MDP approach as the amount of data increases across the system. The BRS approach performs badly in this regard as it does not take the costs of data transfer between resources into account. It myopically assigns a task to a resource based on the most expensive resource. As the size of data increases BRS incurs additional costs as tasks are executed on different resources in different regions. The costs for executing the MDP schedule over BRS increase much more slower rate. The performs much better as the schedule it deploys, makes considerations over the entire workflow.
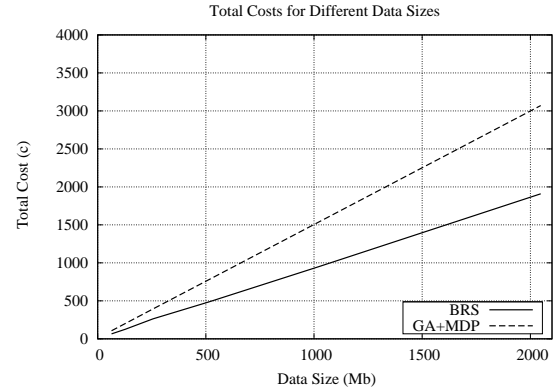

Fig. 5. Overall cost as data increases

C. Load Variance

To simulate a variable work load on the system we fix the arrival rate of workflows to be processed. We vary the number of tasks in each workflow application from 20 to 50. The number of compute services is constrained from $5 - 15$. By controlling the arrival rate of workflow applications and the number of tasks at any time we can vary execution load across the virtual machines over time. Figure 6 shows the performance of the MDP as the load varies. We compare the performance of the MDP approach against the BRS approach and a GA. The genetic algorithm parameter configuration was determined experimentally through identifying the best settings to encourage the best solutions across the widest range of system loads. Initially as the MDP assigns equal probabilities to all actions and doesn't perform as well as the GA. It spends time exploring suboptimal actions and get penalised as a result. As the MDP explores the state space, it outperforms both the GA and BRS. Since the data size used for

the experiment was fixed, BRS does not incur any significant loss of costs due to load variances.
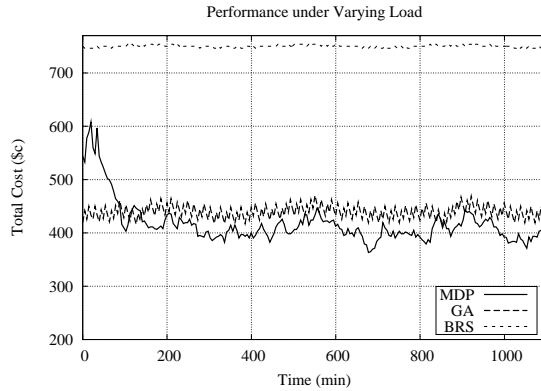

Fig. 6. Performance of MDP as system load varies

VII. Conclusions & Future work

We presented an initial attempt at developing a cloud workflow scheduler and some preliminary results. We have presented a continuous state action space formalism to schedule workflows on a cloud computing environment. Our results show that our MDP agent can optimally choose from a set of evolved workflow schedules. This is achieved in spite of an environment that has varying computational loads and data sizes. We presented a Bayesian model learning approach capable of quickly learning approximate transition probabilities with no prior knowledge. However, it is more challenging to learn optimal value functions for states that are less frequently visited. These states are most likely those where system load approaches its maximum or minimum values. Therefore, MDP performance is heavily dependant on it gaining greater experience of all states. This means the performance of the MDP will suffer unless it can gain greater experience. In future work we hope to investigate a solution based on developing policies offline using sample data from previous executions. This will facilitate the development of better policies online and produce more optimal solutions. Furthermore, we hope to introduce a range of optimisation algorithms that can provide a broader range of schedules and thereby improve overall system performance. Finally we hope to perform a qualitative evaluation of each workflow application and incorporate this into the state space.

References

[1] Montage: An astronomical image engine, http://montage.ipac.caltech.edu.
[2] Illumina: , http://www.illumina.com.
[3] C. Vecchiola, S. P, and R. Buyya, "High-performance cloud computing: A view of scientific applications."
[4] G. Juve and E. Deelman, "Scientific workflows and clouds," *ACM Crossroads*, vol. 16, no. 3, pp. 14–18, 2010.
[5] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 50:1–50:12. [Online]. Available: http://portal.acm.org/citation.cfm?id=1413370.1413421
[6] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *Proceedings of the First International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 140–147. [Online]. Available: http://portal.acm.org/citation.cfm?id=1107836.1107867
[7] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," in *Scientific Programming*. IOS Press, 2006, pp. 217–230.
[8] J. Yu, M. Kirley, and R. Buyya, "Multi-objective planning for workflow execution on grids," in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, ser. GRID '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 10–17. [Online]. Available: http://dx.doi.org/10.1109/GRID.2007.4354110
[9] E. Deelman, G. Singh, M. hui Su, J. Blythe, A. 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," *Scientific Programming Journal*, vol. 13, pp. 219–237, 2005.
[10] B. Ludscher, 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," in *Concurr. Comput. : Pract. Exper.*, 2005, p. 2006.
[11] T. Oinn, M. Addis, J. Ferris, D. Marvin, T. Carver, M. R. Pocock, and A. Wipat, "Taverna: A tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, p. 2004, 2004.
[12] S. P, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments."
[13] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *The Journal of Supercomputing*, pp. 1–38, 2011, 10.1007/s11227-011-0578-4. [Online]. Available: http://dx.doi.org/10.1007/s11227-011-0578-4
[14] D. Vengerov, "A reinforcement learning approach to dynamic resource allocation," *Eng. Appl. Artif. Intell.*, vol. 20, no. 3, pp. 383–390, 2007.
[15] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "On the use of hybrid reinforcement learning for autonomic resource allocation," *Cluster Computing*, vol. 10, no. 3, pp. 287–299, 2007.
[16] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload analysis and demand prediction of enterprise data center applications," in *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, Sept. 2007, pp. 171–180.
[17] J. Perez, C. Germain-Renaud, B. Kégl, and C. Loomis, "Grid differentiated services: A reinforcement learning approach," in *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 287–294.
[18] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, "Pace - a toolset for the performance prediction of parallel and distributed systems," 1999.
[19] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," 1997.
[20] J. D. Ullman, "Np-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, pp. 384–393, June 1975. [Online]. Available: http://dx.doi.org/10.1016/S0022-0000(75)80008-0
[21] J. Oh and C. Wu, "Genetic-algorithm-based real-time task scheduling with multiple goals," *J. Syst. Softw.*, vol. 71, pp. 245–258, May 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=993017.993022
[22] R. S. Sutton and A. G. Barto, "Reinforcement learning :an introduction," 1998.
[23] P. Doshi, "Dynamic workflow composition using markov decision processes," *International Journal of Web Services Research*, vol. 2, pp. 1–17, 2005.
[24] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya, "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services," *CoRR*, vol. abs/0903.2525, 2009.