A Learning Architecture to Support Autonomous Resource Scheduling and Allocation in the Cloud

Ph. D. Dissertation

Enda Barrett B.Sc, M.Sc

Discipline of Information Technology
School of Engineering & Informatics
National University of Ireland, Galway, Ireland

Dr. Jim Duggan (Supervisor)
Prof. Gerard Lyons (Internal Examiner)
Dr. Ivona Brandic (External Examiner)

Abstract

The advent of on-demand computing facilitated by computational clouds, provides an almost unlimited resource supply to support the execution of applications and processes. Through a process known as *virtualisation* large server machines are divided up into smaller units known as virtual machines. These virtual machines can then be procured on demand to support application deployments, workflow executions and service delivery via the cloud. However, optimally allocating these virtual resources to support a given application deployment or workflow execution on a cloud platform presents a number of significant research challenges.

Virtualisation is enabled through a domain level hypervisor which controls access to the shared hardware amongst the competing virtual machines. Switching between domains and attempting to distribute access to these shared mediums is non-trivial and causes performance interference effects amongst the virtual machines. This presents a challenge when attempting to plan a resource allocation to support a given application or workflow running in these environments. Removing these interference effects entirely, is a very difficult problem and is one of the principle challenges facing virtualisation research in the coming years. However from a resource planning perspective it is possible to reason over these variabilities to achieve a near optimal resource allocation which satisfies the defined objective criteria.

Markov Decision Processes provide a decision theoretic framework which facilitates planning and scheduling under uncertainty. By modeling the allocation of resources under this framework and solving using techniques such as reinforcement learning and dynamic programming this thesis provides a learning architecture to allocate/schedule resources adhering to defined optimisation criteria. Using data from real cloud deployments we empirically evaluate our proposed solutions with respect to two different application types. The first is a workflow application deployment where the requirement is to schedule tasks to resources to ensure that both cost and makespan constraints are achieved. The second is an application scaling problem where the goal is to optimise application response time at a minimum cost for varying numbers of user requests. For both of these problems the underlying resource is variable and changes accordingly. We present a number of novel advancements from both a learning and optimisation perspective.

List of publications

The list of publications arising from the research conducted for this thesis is:

1. **Enda Barrett**, Jim Duggan & Enda Howley. A Parallel Framework for Bayesian Reinforcement Learning. *Connection Science, Taylor & Francis, 2013*

2. **Enda Barrett**, Enda Howley & Jim Duggan. Parallel Bayesian Model Learning. *Adaptive Learning Agents (ALA) Workshop, held at AAMAS '13, St. Paul, Minnesota, United States, 2013.*

3. **Enda Barrett**, Enda Howley & Jim Duggan. Applying Reinforcement Learning Towards Automated Resource Allocation and Application Scalability in the Cloud. *Concurrency and Computation: Practise and Experience 2012*

4. **Enda Barrett**, Enda Howley & Jim Duggan. A Learning Architecture for Scheduling Workflow Applications in the Cloud. *European Conference on Web Services (ECOWS 2011), Lugano, Switzerland.*

5. **Enda Barrett**, Enda Howley & Jim Duggan. A Reinforcement Learning Approach to Demand Estimation for Heterogeneous Services. *4th Workshop on Modelling, Design, and Analysis for the Service Cloud (MDA4ServiceCloud), Paris, France, 2010*

6. **Enda Barrett**, Enda Howley & Jim Duggan. A Comparison of Learning Approaches to Support the Adaptive Provision of Distributed Services. *Adaptive Learning Agents (ALA) Workshop, held at AAMAS '10, Toronto, Canada, 2010.*

7. **Enda Barrett**, Enda Howley & Jim Duggan. Evolving Group Coordination in an N-Player game. *10th European Conference on Artificial Life (ECAL 2009), Budapest, Hungary.*

Acknowledgement

Firstly I'd like to thank my supervisor Dr. Jim Duggan for his tireless work and consistent guidance throughout the course of this thesis document and for the previous four years. I'd like to thank my fellow PhD friends in the IT Department in NUI Galway. Without their countless hours of discussion and deliberation over arguably thousands of cups of coffee none of this would have been possible.

I'd also like to thank my parents, Bernadette and Alfred and my two siblings Damien and Maria, for their support, kindness and love, without which none of this would have been possible.

Finally I'd like to thank Laura, who in the words of Christopher Watkins "will be the happiest person in world now that this has finally come to an end."

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Infrastructure as a Service (IaaS) clouds rely on economies of scale to deliver computational resources to consumers in a cost effective way. Similar to traditional utilities such as electricity and gas [Buyya 09], consumers typically pay only for what they use, provisioning resources as needed in an on-demand fashion. This elasticity or ability to scale resources as required is one of the principle differences between current computational clouds and previous utility computing forms such as computational grids and clusters, which require advanced reservations. In delivering resources to consumers, IaaS providers utilise virtualisation technologies such as Xen [Barham *et al.* 03] and VmWare [VmWare 12] to partition a single physical server into multiple independent Virtual Machines (VMs). These VMs reside in a co-located manner and have no visibility or control over the host environmental configuration or neighbouring VMs. Figure 1.1 demonstrates a typical virtualisation scenario where multiple VMs are co-located on a single physical host server. Depending on its configuration each VM is allocated a portion of the physical host resource i.e. CPU cycles, RAM, Disk and Network bandwidth. A Virtual Machine Monitor (VMM) is installed on the physical host and is responsible for controlling VM access to the host's resources. The VMM attempts to isolate individual VMs with respect to security, failure and their respective environment, but not in respect of performance [Koh *et al.* 07, Pu *et al.* 10]. Consequently performance unpredictability has been identified as one of the key obstacles facing growth and greater adoption of cloud computing [Armbrust *et al.* 09]. Much uncertainty [Zhang *et al.* 01a] exists in relation to how ported applications will perform, and indeed scale, once they are deployed in the cloud.

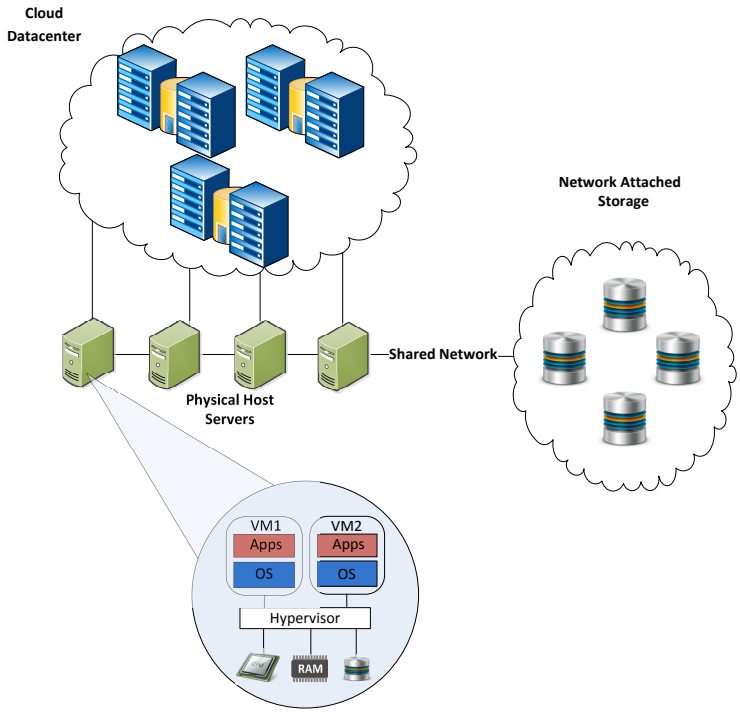


Figure 1.1: Physical host with three instantiated virtual machines.

One approach to dealing with this uncertainty from a planning perspective is to extend classical planning techniques to provide mechanisms to handle domain uncertainty. One of the earliest conditional planning techniques originating from classical planning was the partial-order planning extension C-BURIDAN [Draper *et al.* 94]. This approach extended partial-order planning via conditional probability mechanisms and was demonstrated as a viable approach for handling problems in uncertain domains. The performance however was only suitable on the most simplest of these problems [Karlsson 01]. The MAHINUR [Onder & Pollack 97] contingency planner addressed this by focusing on contingencies which have the greatest estimated impact on goal achievement, demonstrating much improved performance over the C-BURIDAN approach. Numerous other extensions have also been devised, the WEAVER system interleaves classical planning with probabilistic evaluation [Blythe & Veloso 97], PTLplan utilises strategic knowledge such as context dependency and possibility theory to handle uncertainty [Karlsson 01] and C-MAXPLAN and ZANDAR [Majercik & Littman 99] transform propositional probabilistic planning problems to stochastic satisfiability problems. Relevant techniques have also been proposed for extending Graphplan [Weld *et al.* 98, Smith *et al.* 98] to effectively deal with uncertainty by including sensor information to develop contingencies or the ability

to develop conformant plans for the initial conditions and the outcome of actions.

The extension of classical techniques to handle uncertainty is an acceptable solution and can produce effective results for a wide variety of domains. An alternative mathematical framework capable of planning under uncertainty and which does not require any additional extensions for the purposes of planning is the Markov Decision Process (MDP) [Bertsekas & Tsitsiklis 96]. In a planning context MDPs are generally described as probabilistic planning techniques which are capable of expressing the environmental transitions through probabilistic mechanisms. MDPs facilitate the modeling of a set of states, actions, the probability of state transition as a result of action execution and a set of rewards achievable each time. Under this framework, the planning process can reason effectively over non-deterministic events occurring within the environment. The goals of the planning process can be framed within the reward distribution, where actions leading to a goal state result in a large reward. Thus the optimal policy mapping states to actions is the plan which achieves the greatest amount of reward over the entirety of the process.

MDPs infer some limiting assumptions around the transitions between states as a result of action execution and the accurate modeling of the reward distribution [Powell 07]. In particular, solution schemes proposed by dynamic programming methods require accurate and complete models of this representation in order to determine the optimal plan for the MDP [Sutton & Barto 98]. In the absence of a complete model, reinforcement learning [Kaelbling *et al.* 96] provides an effective mechanism which can generate solutions to MDPs for real world tasks, with little computational overhead. The basic principle involves repeatedly interacting directly with the environment, executing actions, observing the responses and learning in a trial and error manner. Over time the learner can build up an approximation of how to act in order to achieve the greatest amounts of reward possible.

To contextualise the problem of planning under uncertainty, we investigate the mapping of applications such as workflows or web applications onto distributed computing resources. These types of problems are regularly examined in the scientific workflow scheduling [Deelman *et al.* 05, Juve & Deelman 10] and resource allocation [Tesauro *et al.* 05, Vengerov 07] communities. These problems are particularly challenging from a planning perspective as they involve mapping tasks onto resources which exhibit regular fluctuations in performance. In the specific case of mapping workflow tasks on distributed resources such as in computational grids or

clouds, the goal is to determine an optimal resource allocation, which maps resources to tasks satisfying certain predefined constraints such as the makespan of the workflow or the cost of execution. In the case of allocating resources to support an application or process, the goal would be to optimise some defined application response time threshold or availability metric.

This thesis examines novel solutions and approaches with regard to the scheduling and allocation of computational resources from public Infrastructure as a Service clouds (such as Amazon EC2 [Amazon 13]) to support these activities. We examine the application of probabilistic planning approaches, detailing their efficacy when applied to the problem and propose a number of advancements over existing research. In contrast to previous approaches for workflow scheduling in computational grids and clouds, which employ heuristic based approaches to the allocation of tasks to resources, our solution is capable of planning and acting given the inherent uncertainty present within these environments. We demonstrate through a number of different learning techniques improvements over existing research when optimising multiple Quality of Service (QoS) constraints. In addition we provide advancements with respect to the learning and optimisation techniques themselves by extending them to operate in a parallel context with regard to the specific problem area.

1.1 Research Opportunities

The majority of research to date in the scheduling and allocation of distributed computational resources has focused on the development of techniques suitable for deployment on computational grids in the case of workflow scheduling [Yu & Buyya 06] or by adding/removing virtual servers in the case of application scaling [Tesauro *et al.* 06]. However the recent growth of cloud computing resources has opened up new and promising opportunities with regard to allocating cloud virtual machines to support workflow applications or indeed web applications deployed in the cloud. Listed below are some of the main differences introduced by computational clouds over previous distributed computing forms such as clusters, grids and shared hosting models.

- In the past large collaborative science grids relied on members providing the computational resources to support the grid services. By providing resources a member was allowed to access time on the entire grid for their own experiments. With the advent

of computational clouds and their hourly billing structures users can access an abundance of compute resource without requiring advanced reservation mechanisms or without having to join a large global grid infrastructure and establish virtual organisation within it.

- These previous grid infrastructures did not support the notion of on-demand instances, i.e. the adding and removing of instances to and from the grid as deemed necessary based on changes in user demand or looming workflow deadline constraint.

- When compared to previous shared or dedicated hosting solutions for deploying web applications the cloud adds an elasticity dimension which wasn't present prior to this. Hosting providers required leasing contracts usually on an annual basis. Thus it wasn't possible procure additional resources for a week or a day or even an hour to support your application scaling needs in a temporal context. If you required additional hosting, the lowest level of granualarity with regard to resource leasing was generally a yearly leasing plan. Thus in order to cater for periodical or seasonal fluctuations in demand ones only option was to over provision the necessary resources.

Infrastructure clouds leverage upon advancements made from virtualisation technologies to slice up large physical server machines into smaller virtual servers which reside in a co-located manner on the same physical host. This process increases server utilisation by allowing different operating systems, applications and services to run side by side in apparent isolation. A domain level monitor known as a hypervisor controls access to the underlying hardware such as disk, network, CPU and memory infrastructures, sharing them as equitably as possible between the machines. With the many benefits of cloud computing one may be forgiven for thinking that the cloud is relatively problem free and the long awaited answer to our computing needs, however there are quite a number of unresolved issues which are considered barriers to the greater adoption and growth of cloud computing.

Michael Armbrust et.al. details a number of these factors which continue to affect the growth and greater adoption of cloud computing [Armbrust *et al.* 10]. These are listed in table 1.1 and include *Availability of Service, Data Lock-In, Data Confidentiality and Auditability, Data Transfer Bottlenecks, Performance Unpredictability, Scalable Storage, Bugs in Large-Scale Distributed Systems, Scaling Quickly, Reputation Fate and Software Licensing*. For each of

Table 1.1: Top 10 problems preventing the growth of cloud computing [Armbrust *et al.* 10]

Problem	**Opportunity**
Availability of Service	Use multiple cloud providers
Data Lock-In	Standardise APIs; Compatible software for Surge or Hybrid Cloud Computing
Data Confidentiality and Auditability	Deploy encryption, VLANs, firewalls
Data Transfer Bottlenecks	FedExing Disks; Higher Bandwidth Switches
Performance Unpredictability	Improved VM support; Flash Memory; Gang Schedule VMs
Scalable Storage	Invent Scalable Store
Bugs in Large-Scale Distributed Systems	Invent Debugger that relies on Distributed VMs
Scaling Quickly	Invent Auto-scaler that relies on Machine Learning; Snapshots for Conservation
Reputation Fate Sharing	Offer reputation guarding services like those for email
Software Licensing	Pay-per-use licenses

these problems Armbrust proposes a number of possible solutions and open research challenges aimed at mitigating the impact of each problem. These solutions range from the application of machine learning techniques and the introduction of large scale flash memory storage to the development of higher bandwidth network switches and infrastructure.

In this thesis our focus is on researching solutions capable of addressing two of these problems, *Performance Unpredictability and Scaling Quickly*. By applying our proposed solutions to the problems of workflow scheduling and application scaling, our goal is to advance upon existing work by orientating our solutions specifically to deal with the vagaries of the cloud. *Performance unpredictability* refers to the dynamic nature of infrastructure clouds where competition between virtual machines for the underlying shared resources causes contention issues. Technically referred to as virtual machine interference but sometimes more colloquially as the "noisy neighbour" effect, it is particulary problematic for disk and network I/O operations [Pu *et al.* 10]. Disk and network I/O is difficult to isolate in a virtual context due to the physical constraints placed on the hardware, i.e. the platters on the disk have to physically spin to the relevant location, the actuating arm has to move into position over the correct sector/track.

These physical motor constraints, coupled with the problem that different applications residing on different virtual machines will request different files from various parts of the disk simultaneously can result in poor I/O performance.

Any attempt to optimise virtual machine allocations to support applications or execute tasks as part of workflow execution, without due cognisance to this inherent uncertainty will likely result in a suboptimal allocation. This becomes especially prevalent at peak usage times in the data center, when shared bandwidth can become saturated as greater quantities of virtual machines increase their activity [Pandey *et al.* 10]. In fact much of the previous work within this context has largely ignored this effect with many assuming that the processing time per task is known or can be easily estimated. Techniques such as analytical modelling [Nudd *et al.* 00] or empirical [Jang *et al.* 04] and historical data analysis [Smith *et al.* 98] have regularly been cited by researchers as viable approaches for modeling task execution times prior to deployment onto distributed resources. Whilst these techniques can provide some estimate as to how the task may perform, they don't have access to the real-time state of the environment. Thus where a learning technique could possibly make associations given the current state of the environment and choose actions accordingly, the scheduling techniques which base their decisions on these model estimation techniques will not be able to improvise as readily. Recent advancements with regard to monitoring techniques within clouds can improve upon this [Emeakaroha *et al.* 11], ensuring certain goals are met. The research proposes a management framework capable of monitoring the status of workflow executions in the cloud and dynamically allocating resources in response. From a performance unpredictability perspective within clouds this solution can allocate additional resources in response to real time observations in a cloud environment. However whilst being a valuable extension to existing workflow management techniques, the approach is still a reactive one, which adjusts the execution plan in response to violations of certain conditions. A question worth exploring is whether further improvements can be made through more predictive approaches which focus on longer term strategic objectives?

Applications experience variations in demand from their users on a regular basis, *Scaling Quickly* refers to the need to respond quickly to changes in demand for the deployed applications and services. Some of the research opportunities within this context involve the building of more accurate predictive models of cloud dynamics, training artificial learning agents

through techniques such as machine learning to make strategic decisions and defining adaptive threshold mechanisms to scale the resources faster. Scaling of resources in cloud environments from a physical perspective can be achieved in two ways. Vertical scaling refers to increasing/decreasing the volume of resources within the bounds of the instance itself, such as doubling the amount of RAM, or allocating more virtual CPUs to the virtual machine instance. Horizontal scaling refers to the spawning of additional virtual machines and load balancing requests across them. The virtual machines can be homogenously configured or heterogeneously configured with respect to type (OS, CPU, Disk, RAM). In addition they can also be deployed to balance load across different application tiers such as the data layer or business logic layer. One of the challenges with vertical scaling solutions is that vertically allocating resources in real-time is not supported natively by all operating systems. Many systems require a reboot to account for the newly added system resources whereas horizontal scaling can be achieved without any rebooting but requires active load balancing solutions to distribute requests among a dynamic pool of changing resources.

Within the literature two approaches to horizontal scaling are widely discussed, threshold based approaches and decision theoretic solutions. Threshold models are currently the methods favoured by commercial cloud scaling solutions such as Amazon's AutoScale [AutoScale 13] and RightScale's [RightScale 13] solutions. One of the principle challenges with regard to threshold solutions is that they tend towards myopic actions. If a threshold is breached then the corresponding pre-defined heuristic is executed, such as the addition of a virtual machine to support the application or service. Potentially in the longer term this may not be optimal, such as in scenarios where thresholds are breached frequently. The ensuing yo-yo effect where additional resources are continuously added and removed could prove expensive to maintain. The smallest billing granularity for infrastructure cloud providers is hourly billing, therefore if you spawn up additional resources to cope with a ten minute spike in demand, you will pay for the remaining 50 minutes even though it is no longer required. What if the existing configuration had remained unchanged, would the performance degradation have been acceptable for such a short time period? Perhaps if you had the benefit of prior experience you would be able to make better decisions. Unfortunately defining strict thresholds can only solve allocation from a reactive perspective and cannot observe the long term effects of the actions.

The solutions proposed in this dissertation involve training reinforcement learning agents to adaptively and dynamically scale resources in response to performance and demand fluctuations for services deployed in cloud computing environments. Reinforcement learning is a trial and error learning framework which can learn how to act optimally within an environment without any prior knowledge. This solution offers flexibility over other machine learning mechanisms such as supervised learning which requires explicit environmental models describing the corrective actions to be taken each time for each possible world state. In contrast reinforcement learning methods can learn through observing responses to their actions, attempting to discover sequences of actions which yield the greatest reward in the long run. These types of artificially intelligent techniques are capable of associating past observations with present ones to allow them to predict the likelyhood of certain events occurring again given the current state. A good example which illustrates the potential advantage of applying artificial intelligence to application scaling via resource allocation, is to view how decisions are made from the perspective of both the threshold based models and the more intelligent decision theoretic (learning) models. The threshold based model will simply allocate a resource to support the application as soon as it detects a performance violation, irregardless of what this may mean in the long run. In contrast the intelligent agent through past experience and observations may infer from the learning system..."*previously when I encountered the current state, the performance drop only lasted for five minutes. The last time my decision was to allocate an additional virtual machine. However once the ten minutes had elapsed the surge in demand for the application subsided and I began to incur penalties as a consequence of resource over provisioning until finally* 50 *minutes later I was able to release the resource. This time round I will choose not to allocate any additional resources for the moment and wait for ten minutes to see what happens*". Whilst this is a somewhat simplified example, in reality the learner does reason via logical inference over the past observations to discover better options going forward.

Within these decision theoretic solutions a number of challenges still remain and have not yet been satisfactorily answered by existing research. In particular, solution techniques involving methods from dynamic programming take a very long time to determine optimal or even near optimal scaling configurations. With dynamic programming methods the guiding principle of optimality is only applicable on the limit as the number of trials tends towards infinity, in accordance with the law of large numbers [Bellman 57]. Whilst for most scenarios the

methods from dynamic programming will discover the optimal policy well short of this point, it can still take quite a while to determine a good solution. This may be unacceptable for certain applications where sub-optimal performance results in either expensive allocations or poor application performance whilst learning in the initial stages. Other decision theoretic techniques, particularly those involving temporal difference reinforcement learning, can be much faster, but often at the expense of greater accuracy. There is still a significant challenge remaining to improve these even further to generate better policies in a shorter time frame.

One problem faced with all learning techniques is in regard to changes within the underlying environment. Techniques have always struggled in regard to effectively handling changes to the underlying model. For instance if a learning agent has built up a knowledge model of the environmental dynamics and suddenly the real world shifts, the present knowledge model is no longer accurate for this new world and any learner which continues to plan and act with the old model will be suboptimal. This is because it's still making decisions about an environment which has now changed. In a cloud context it's likely that model shifts could occur frequently, caused by a number of different factors such as, rolling upgrades within the data center, improvements in the performance of the underlying technologies such as new approaches to disk and network I/O virtualisation or increased congestion due to higher bandwidth consumption within the cloud. In addition, changes within the applications themselves such the issuing a new patch to fix a performance issue or the optimising of the data layer with a new data model could alter how the application performs in the cloud. All of these will require some re-learning to ensure optimal decisions can be made in a scaling context going forward. Techniques which can demonstrate robustness to model changes will be vital to the success of the decision theoretic solutions.

1.2 Goals, Objectives and Hypotheses

Through the employment of learning and optimisation techniques this dissertation proposes a number of solutions which advance the current research with regard to resource scheduling and allocations within cloud infrastructures. Cognisant of this fact we examine the following research questions:

- Can workflow scheduling performance be improved in cloud environments by employing a learning approach to choose an appropriate schedule from amongst a number of competing schedules?
- Will a parallel learning framework, utilising Q-learning, achieve improved performance for dynamically scaling applications under a variety of conditions, such as variable workload models and resource performance?
- Can Bayesian reinforcement learning be successfully extended to a parallel method and will this improve its performance with regard to improved convergence times?

Given these questions a number of hypotheses can be proposed:

- **H1** : Through analysis of the environmental state and in particular observing temporal changes in performance, one can improve the schedule selection process to achieve a more optimal solution.
- **H2** : Employing multiple learners which share their experiences and are capable of observing specific environmental conditions, will improve performance over previous solutions for dynamically scaling applications in the cloud.
- **H3** : Using the Kullback-leibler divergence will render the sharing of probability density estimates amongst agent learners possible, and will result in a reduction in learning convergence times overall.

To evaluate these objectives and hypotheses we present a series of simulations which empirically analyses our approaches using real data taken from virtual machine instances deployed in the cloud. We hope to further these hypotheses to gain a greater understanding of the dynamics of cloud environments and how best to automate control actions within them. Adopting novel learning approaches to resource scheduling, allocation and scaling, we hope to prove that the learning techniques proposed are viable solutions in a cloud computing context.

1.3 Thesis Structure

This section provides an overview of each chapter within this thesis. The initial chapters describe the domains of cloud computing, workflow scheduling, reinforcement learning and optimisation, describing the current and ongoing work relevant to our research. The ensuing results chapters present our contributions and findings within these domains. Through empirical analysis we demonstrate our advancements over existing work. Finally we summarise the contributions and discuss possible extensions to this work in the concluding chapter.

The rest of this thesis is organised as follows:

- Chapter 2 provides an overview of cloud computing, resource allocation and workflow scheduling. Firstly we analyse the history of the cloud, tracking it through a number of paradigm changes from cluster, grid to cloud computing. We discuss the "as a service" movement describing Infrastructure, Platform and Software as a Service and also the causes of virtual machine interference within computational clouds. Finally we discuss resource allocation and workflow scheduling, describing the related works in the field, the various techniques currently employed and possible directions going forward.

- Chapter 3 details agent learning techniques, including reinforcement learning, statistical learning and parallel learning. We describe the Markov Decision Process, the solution of which, via learning, forms the basis of one of the core contributions of this dissertation. Finally we discuss evolutionary computation and in particular the genetic algorithm, which we employ as an optimisation technique within this research.

- Chapter 4 presents the data traces and performance models generated from live cloud deployments. Firstly we detail performance variability from a data intensive perspective, focusing on the disk I/O rate fluctuations resulting from competing virtual machines in the cloud. We also present data detailing application response times provided to us by Cedexis, the multi cloud deployment strategists. Through sampling these data sets we generate realistic simulations for the experiments in the results chapters.

- Chapter 5 describes the first of our three main results chapters, detailing a Bayesian model learning algorithm combined with dynamic programming to choose from amongst a set of evolved schedules in an application workflow scheduling problem. We present

a series of simulations comparing our learning framework to previous approaches and demonstrating its efficacy for handling cloud deployments.

- Chapter 6 presents a novel parallel temporal difference reinforcement learning framework for dynamically scaling applications in the cloud. By simulating variable user requests and virtual machine performance we demonstrate the effectiveness of our proposed solution under a variety of workload conditions.

- Chapter 7 examines a parallel Bayesian reinforcement learning framework on two distinct problems, a stochastic GridWorld and a virtual machine allocation problem. We demonstrate convergence speedups as result of the parallel learning mechanism, reinforcing its viability as an alternative to other speedup mechanisms such as function approximation.

- Chapter 8 summarises the research presented in this dissertation, offering conclusions and possible future directions within the domain.

Chapter 2

Cloud Computing & Resource Scheduling

2.1 Introduction

"As of now, computer networks are still in their infancy, but as they grow up and become more sophisticated, we will probably see the spread of '*computer utilities*' which, like present electric and telephone utilities, will service individual homes and offices throughout the country."
Prof. Leonard Kleinrock, UCLA press release, July 3rd, 1969

Ideas about the possibilities of computing utilities have been touted since the early 1960s when John McCarthy of MIT, was one of the first to publicly state that "computing may one day be organised as a public utility". Analogous to the operation of existing utilities such as electricity, telecommunications, water and gas, a fifth utility, the computing resource, could one day service our homes and businesses throughout the land [Buyya 09]. Traditional utilities providers operate a pay-per use model offering a seemingly unlimited resource supply to the end consumer. If one takes the example of the electricity market, the consumer does not need to worry about transmission, storage, quality or availability, instead it assumes that it will always have an on-demand supply to the said resource whenever it may need it. In addition the consumer's usage of the resource is metered on a per unit basis and billed in accordance with the level of consumption.

In a computing context, recent developments in compute clouds such as Amazon EC2, GoGrid and Rackspace facilitate the provision of large amounts of computational resources in a pay-per use on-demand fashion. With on-demand computing, the responsibility for hosting applications and services is delegated to the cloud provider. Similar to traditional utilities the consumer only pays for what it consumes and no longer requires the expensive purchasing of equipment or the on-going staffing and maintenance costs. In fact these days, an individual in possession of a credit card and an internet connection, can procure the equivalent of what once required a serviced office space, a skilled system admin staff, an abundance of hardware/software components and all from the confines of their office chair.

However outsourcing ones entire computing requirements to an external provider is a daunting prospect for many, with consumers raising concerns ranging from data security/ownership to performance unpredictability [Armbrust *et al.* 10]. Consumers need to ensure that their applications and services can still maintain the same levels of quality and service similar to their previous in-house hosted scenario. The solutions to these challenges as well as numerous others will require significant research effort and advancement in the upcoming years if we are to realise the full vision of the utility computing paradigm.

Our literature review with respect to cloud and infrastructure computing details cloud, grid and cluster computing, virtualisation technology and the "as a service" computing movement. We review the relevant research in these areas, discussing the merits of each and providing a focussed discussion of the topics. We then follow up this examination with a discussion on resource allocation and the scheduling of workflows to resources. Using virtual machines from computational clouds as the resources to be distributed, we detail the relevant works within these areas and provide a general overview of the topics.

2.2 Cloud Computing

In the past, network engineers when designing a network, spent considerable amounts of time mapping out all the network devices which would form part of that network. This involved careful examination of the domain and how the devices were connected/managed. When the engineer needed to represent an external network, one in which their network needed to connect to but was outside of their control, they used the symbol of the cloud to denote it. This

external network such as the Internet was considered an unknown quantity i.e. the details about which they couldn't be certain. Thus the cloud was adopted to represent this new form of computing as it signified the consumption and provision of resources outside of the local network accessed via the web.

It is challenging to provide a universal definition which is broad enough to encompass the entirety of the cloud computing spectrum, with many efforts focussing on defining a particular aspect of the cloud [Travostino 08, Hwang 08]. A good concise definition which encapsulates cloud computing, is the definition provided by the National Institute of Standards and Technology (NIST) [Mell & Grance 11] :

"*Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*"

This definition is broad enough to encompass the majority of what is considered cloud computing by modern standards, covering infrastructure clouds, software as a service and platform as a service. Additionally another useful definition is proposed by proposed by Luis M Vaquero et. al. [Vaquero *et al.* 08] stating

"*Clouds are a large pool of easily usable and accessible virtualised resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs*"

Vaquero's definition alludes to one of the principle differences between cloud resources and previous shared hosting solutions, which is that cloud resources are *elastic*. Elasticity in a cloud context refers to the ability to scale the resources at your disposal at a moments notice, either by increasing or decreasing the number of resources (virtual servers, storage, applications) supporting your application/service. Within the cloud infrastructure the speed at which this is executed is significant. Within minutes an individual can ramp up the number of virtual servers from a just few, to hundreds and drop them as quickly when they are no longer needed. Rapid scaling removes the need for an individual to over-provision resources to meet peaks in

demand. In real-time a elastic load balancer can adjust the resources at their disposal and easily mitigate against an unexpected surge. Elasticity is one of the key differentiable characteristics which separates modern clouds from other previous utility computing forms.

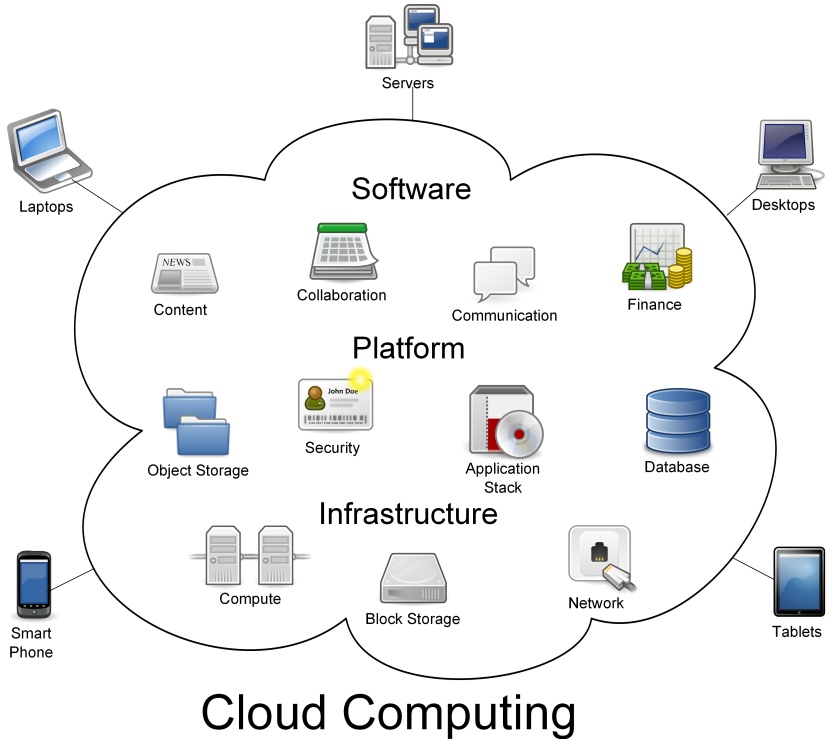


Figure 2.1: Overview of cloud computing

Figure 2.1[1] details an overview of cloud computing from the perspective of the three principle layers: the infrastructure layer; the platform layer; and the software layer. The infrastructure layer refers to the "nuts and bolts" layer of the cloud, consisting of the virtual servers, the network infrastructure and the block storage mechanisms. Within the framework of the infrastructure cloud one can replicate their in-house server scenario, building an array of virtual servers and storage solutions to support their daily operations. The consumer can then access their data, applications and resources anywhere in the globe across a range of devices (laptops, tablets, smart phones).

The infrastructure layer requires greater technical support and management from the end user itself. Whilst one does not need the knowledge to virtualise the hardware or establish a network configuration connecting the hardware, one does still need the technical expertise to configure their servers from a operating system level, install their required applications/business logic

[1] Courtesy of www.wikipedia.org

and manage their data. They are required to effectively build their own platform layer on top of the barebones infrastructure in the cloud.

Procuring platform level cloud infrastructure removes the need to build your own supporting platform and allows the end user to deploy their solutions on a pre-configured stack. The infrastructure and the platform is managed by the service provider offering the user the freedom to focus on their core operations and competencies. An example of this managed platform is the Windows Azure cloud. Windows Azure offers a Microsoft Windows environment suitable for the deployment and scaling of applications designed specifically for running on Windows. The solution stack consists of components such as the .NET framework and a scalable MS Sql Server solution. This assists the speedy deployment of compatible applications to the cloud. Applications and software compiled and built for other operating systems and platforms are not deployable.

The software layer in a cloud environment refers to a subscription based software model which is hosted on the cloud and accessed via the web. Salesforce's Customer Relationship Management (CRM) is a prime example, where you pay a monthly subscription to use the service which is hosted remotely by them. The service is usually accessible via a controlled web interface or thin client mechanism. In the forthcoming section on Computing as a Service (Section 2.2.2) , each of these layers is discussed in greater detail.

2.2.1 Origins of Cloud Computing

Many years prior to the development of virtualisation technology, time sharing operating systems were deployed on main frame computers which allowed for CPU time to be shared amongst multiple users. Access to the machine was facilitated through dumb terminals and different users were able to simultaneously submit jobs for processing [Akhter & Roberts 06]. Subsequently this approach became less popular as misuse of the system by a single user could result in halting the entire system for all other users [Rose 04]. To solve this problem IBM invented the virtual machine to isolate user instances from each other. With the virtual machine a single large mainframe computer could be split up via virtualisation into smaller components which were isolated from each other. This resulted in greater system utilisation of the expensive computing technology.

The cloud computing paradigm currently available to us today was driven largely by a requirement to handle user requests on the massive web-scale. This forced companies such as Amazon and Google to develop large scale data centers to support these requests. Cloud computing as we know it is effectively a commercialisation of their knowledge in building data centers over the years [Jaeger *et al.* 08]. In 2006 Amazon decided to open their computing infrastructures with their official launch of Amazon Web Services (AWS). It subsequently became known as cloud computing with the key benefits being, low cost on-demand computing available always over the web.

A number of computing movements have preceded the cloud paradigm with each iteration bringing significant advancements and research breakthroughs. It is clear that current cloud infrastructures would not have been developed if it wasn't for the advancements made in the areas of Grid and Cluster computing in the years leading up to its development.

2.2.1.1 Grid Computing

The Computational Grid was borne out of the necessity to process large volumes of data for scientific, engineering and medical domains. Data processing of this magnitude requires high end computational resources which incur a significant upfront investment to establish. This initial substantial monetary outlay, coupled with setup and on-going maintenance costs, rendered it prohibitive to most. With the exception of governments, very well funded research groups or large corporations, most individuals did not have the resources to develop these types of computational infrastructures.

Grid Computing emerged in the 1990s as an alternative approach which allows multiple institutions, academic or otherwise to network lower specification machines into a grid structure to process large computational tasks. To date a number of successful computational grids have been established including the Open Science Grid [Pordes *et al.* 07] the Earth System Grid [Bernholdt *et al.* 05] and TeraGrid [Catlett & *et al.* 02]. Grids are facilitated through the establishment of Virtual Organisations (VO) between interested parties who agree to share a small amount of their resources with the Grid in order to get access to the larger more powerful collective resource. Once a VO is established the community can share resources amongst each other through defined interfaces in a large distributed network. In response to sharing

resources, each member of the VO is allocated pre-reserved slots to execute their own tasks on the Grid.

The term Grid was chosen as an analogy with the electric power grid, which provides pervasive access to electricity [Foster & Kesselman 03] with seemingly unlimited supply. It was envisioned that computing power could be accessed on-demand as a utility, similar to what was suggested by John McCarthy back in 1961, when he said "computation may someday be organised as a public utility". In many ways the vision behind the grid computing movement is similar to what modern day clouds offer with respect to on-demand computing [Foster *et al.* 08].

However these platforms possessed a number of limitations in both technical terms and bureaucratic [Vecchiola *et al.* 09]. Often it was more difficult for smaller research groups to gain access to the global grids as they lacked the required infrastructure to partake in the virtual organisation. Furthermore, many of these grids use pre-defined platforms with specific operating systems, application suites and APIs. Thus specific platform requirements across the Grid could not be facilitated easily as it would require re-imaging of all the devices and was generally not possible.

2.2.1.2 Computational Clusters

Cluster Computing like Grid Computing involves networking large volumes of machines to present a unified computing viewpoint to the end user. A single operating system is installed across all nodes in the cluster and special software is used to deploy and schedule tasks across the cluster. In contrast to the Grid, Clusters are generally networked over a high speed Local Area Network (LAN) connection with a single owner.

A popular cluster framework is the Beowulf cluster. The Beowulf cluster is a particular type which exploits mass market personal computers known as simply commodity machines [Sterling 02]. By networking the machines into an n-node cluster powerful parallel programs can be executed on the cluster which utilise powerful message passing interfaces such as MPI.

From a research perspective, the community has largely focussed on developing scheduling algorithms for optimal distribution of computational tasks across the nodes of the cluster [Fallenbeck *et al.* 06, Zhang *et al.* 01b, Shin *et al.* 08]. Zaharia et. al. [Zaharia *et al.* 10] analyse

the conflict between scheduling fairness amongst users and data locality. To address this they propose a simple algorithm called *delay scheduling*, which involves introducing a small execution delay for jobs which should execute according to fairness but cannot achieve significant locality constraints. By waiting for a short time period, it allows other tasks to be executed. Testing the approach on a 600-node cluster at Facebook, the authors demonstrated that delay scheduling can achieve near optimality for a wide variety of workloads and yields a 2x improvement in throughput whilst preserving fairness. Large clusters consisting of thousands of nodes are often susceptible to failures within the cluster, adversely affecting job completion times and the overall performance. Zhang et. al. develop a general failure modeling framework using data extracted from large cluster computations [Zhang *et al.* 05]. Using this modeling framework the authors are able to model the impact of failures on a wide range of scheduling algorithms. The authors demonstrated results which indicate that current scheduling approaches which ignore failures, experience significant degradation with regard to mean job response times and mean job slowdown. Information about the temporal and spatial correlation of failure occurrences is very useful when defining a scheduling algorithm to enhance system performance.

Frey et. al. investigate aggregation of resources spanning cluster and grid environments [Frey *et al.* 02] . They successfully utilise Condor and Globus to create Condor-G which is able to aggregate resources and present them as a single unified domain. Their work describes approaches on how to handle job management, resource selection, security and fault tolerance, utilising resources in an inter-domain fashion.

2.2.2 Computing as a Service

Modern day cloud infrastructures are built on the principles of offering packaged up computing components as a service. Consumers lease these services according to hourly, monthly and even yearly plans. It is the responsibility of the service provider to maintain the service integrity according to agreements defined between the consumer and provider. These service level agreements often consist of metrics such as service availability, uptime and even response time levels.

Cloud service offerings predominately come in three varieties, Infrastructure as a Service

(IaaS), Software as a Service (SaaS) and Platform as a Service (PaaS). More recently a further service model has emerged called Business Process [Zhang & Zhou 09] as a service, however further analysis of this paradigm is outside the scope of this discussion. This growth of the "as a service" movement in Cloud Computing has touted some commentators to announce the arrival of the Everything as a Service [Banerjee *et al.* 11] paradigm. Our discussion henceforth is limited to IaaS, PaaS and SaaS.

2.2.2.1 Infrastructure as a Service

Infrastructure as a Service (IaaS) is the delivery of hardware (server, storage and network), and associated software (operating systems, virtualization technology, file system), as a service [Bhardwaj *et al.* 10]. In simple terms, it's effectively the leasing of virtual servers and storage as a service over the web. Delivering services via the internet allows users to access their resources from anywhere in the world through a standard internet connection. This makes cloud resources and infrastructures incredibly flexible and mobile, and allows users to access their data from virtually anywhere in the planet. The IaaS provider is responsible for ensuring that the service meets certain quality constraints regarding service uptime and availability. These agreements ensure that the user can access their resources and data when they need them most.

IaaS offerings typically consist of leasing structures which are billable by the hour or on a yearly basis. Hourly leasing is one of the principle benefits of IaaS clouds over previous hosting models and allows users to cost effectively cater for temporary spikes in demand through resource scaling. Service consumers can include additional resources as required, paying per hour and releasing them when the demand has abated. This model eradicates the previously wasteful practise of resource over provisioning where individuals had to procure resources over their average need to cater for temporary increases.

Currently the dominant market player for providing IaaS services is Amazon Elastic Compute Cloud (EC2), although there are a number of other providers such as Microsoft Azure, GoGrid and Rackspace. The current resource delivery mechanism favoured by IaaS clouds has been largely based on virtualisation technologies. Virtualisation allows for multiple virtual machines containing disparate or similar guest operating systems, to be deployed in apparent

isolation on the same physical host. A more extensive discussion on Virtualisation technologies is covered in the upcoming section 2.2.3.

2.2.2.2 Platform as a Service

PaaS systems are generally hosted, Web-based application-development platforms, providing end-to-end or, in some cases, partial environments for developing full programs online [Lawton 08]. With Platform as a Service (PaaS) the consumer has access to a solution stack of software and hardware components which they can utilise to develop their own solutions. Again the responsibility of hosting, maintaining and updating the platform lies with service provider, allowing the consumer to focus on developing solutions applicable to them. Common examples of PaaS include Google Apps, the Microsoft Azure platform and AWS Elastic Beanstalk. As usual consumers typically only pay for what they consume in accordance with the "as a Service" model.

With PaaS, there are some distinct variations with respect to platform flexibility. Certain platforms provide scalability solutions and security but do not include development or debugging components with their service. Amazon's Elastic Beanstalk is one such example which offers non-programmable HTTP stacks for deploying .NET, PHP and Java applications, allowing one to scale their applications on a secure environment. Some platforms such as the AppScale PaaS, allows an individual to deploy their own PaaS in a private cloud capacity or in a public virtualised environment. AppScale supports the deployment of Google Apps, reducing the reliance on the Google App Engine hosting environment. Movements such as the Openstack initiative, which includes partners as diverse as NASA, Intel, Red Hat and Cisco aims to build an open source PaaS model to provide a common cloud infrastructure which can be used for public and private purposes. In addition Eucalyptus cloud, Nimbus and OpenNebula are also competing open cloud platforms.

From a research perspective, Benson et. al. aim to address the limitations of the current cloud networking solutions [Benson *et al.* 11] . They cite a number of problems from a networking perspective and propose a solution platform called CloudNaaS. The goal of their platform is allow users to replicate their in-house configuration and performance levels in a public cloud environment. According to the authors, the limitations with current cloud infrastructures from

a networking perspective are threefold. 1) The ability to specify bandwidth requirements. Public clouds currently do not provide bandwidth provisioning solutions which ensure dedicated bandwidth between cloud instances or end users, meaning that during peak usage times in the datacenter bandwidth will be depleted. 2) The ability to add in so called "middlebox" items, such as front-facing server caches, application accelerators, firewalls and load balancers. 3) The lack of a broadcast domain and cloud-assigned ip addresses for virtual servers.

The authors present a network service platform specially designed for the cloud, capable of addressing these limitations. Lee and Hughes [Lee & Hughes 10] adopt principles of PaaS clouds for sensor network research. They propose a standardised platform for building wireless sensor networks using the principles of Cloud Computing.

2.2.2.3 Software as a Service

The Software as a Service (SaaS) model enables individuals to procure software on a subscription basis, e.g monthly or yearly or based on the number of users for the application. This reduces the Total Cost of Ownership (TCO) and introduces novel pricing strategies, opening up new payment models and revenue streams [Greschler & Mangan 02]. Often the software is accessible via a web interface or thin client. The application service provider is responsible for the hosting of the application and is bound by service level agreements with respect to performance, uptime and availability. The service level will often depend on the level of support you subscribe to. Salesforce.com's Customer Relationship Management (CRM) software is one of the leading examples of current SaaS applications. Procuring software in this manner allows companies to reduce their own setup, licensing, infrastructure and maintenance costs associated with hosting applications in house. Additionally they do not need to configure their network infrastructure to support access outside the office, the SaaS provider is accessible $24/7$ via a web connection.

The scientific communities focus on SaaS has revolved around the calculation of optimal pricing strategies and the pay-per-use model [Choudhary 07]. A correct pricing strategy by the SaaS convinces the consumer that significant value can be achieved via the leasing structure as opposed to buying the software, procuring the infrastructure and maintaining it [Huang & Sundararajan 05]. Additionally researchers have analysed contingent auctions, showing how

commitment levels affect prices and revenues [Bhargava & Sundaresan 04]. They have also focused on the firm's decision to buy the software outright and host in-house versus buying the service on a per user basis [Seidmann & Ma 04] where the authors found that the key deciding factor relates to the number of users per application, with the greater the number of users the more likely a company is to host in-house. Another significant advantage of the SaaS model is it reduces the problem of software piracy significantly. Since the software is not distributed widely and installed on each user's personal computer or server, the ability to pirate the software is diminished enormously.

2.2.2.4 Cloud Elasticity

One of the notable characteristics which distinguishes clouds from previous computational resource models in the past, is that the resources are considered elastic. In response to changes in demand, cloud infrastructures can be scaled up in minutes and scaled down again when the demand for the deployed service or application has subsided. Anomalies such as the slash dot effect, where a huge surge in traffic is experienced on a smaller website often causing it to crash as result of a link to it being placed on a larger site, can be catered for under the cloud model.

Newly created applications often experience an initial surge in demand, where users are keen to try it out as its popularity spreads across the social networks and micro blogging sites (Facebook and Twitter). In a traditional in house server setting, the computing infrastructure would have to be scaled up to meet the surge in user demand. Expanding in-house infrastructure in a timely fashion is difficult to achieve as there usually is a time delay of a number of weeks between when the equipment is ordered, shipped and operational. In addition, purchasing the extra servers, storage, memory and perhaps even additional office space incurs a substantial upfront monetary outlay. A common occurrence with new applications, is that soon after the initial surge in demand, the usage pattern reduces to a more moderate level, some distance from the lofty peaks of its earlier days. Unfortunately the application owner now has a substantial computing infrastructure which was purchased at great expense, incurs ongoing maintenance costs, with a good portion of it highly under utilised.

Armbrust et. al. [Armbrust *et al.* 10] describe the case of Animoto.com which illustrates

this point. When launching in 2011 Animoto experienced a doubling of load every 12 hours for nearly three days. Instead of hosting their application in house they chose to deploy it on Amazon EC2. Utilising cloud elasticity they were able to increase the number of virtual servers dramatically to account for the increased traffic. Once the traffic volumes subsided to a lower threshold the company released the excess VMs. The benefit was obvious, Animoto incurred no upfront equipment purchasing expense. It did not need to hire sys-admins or maintenance staff to service the hardware. It was also able to expand its services rapidly, a task most difficult in a traditional sense, where it requires hardware procurement, setup and configuration, all within the hectic time-frame of when the app is growing at exponential rates.

Elasticity has economic and user experience benefits as it ensures that outages as a result of server overloading are minimised and even in some cases eradicated. Consumers can be assured that they will not experience any loss of earnings associated with down times due to increased user demand for their applications and services. They can also increase user satisfaction/experience by distributing their application load across a greater number of virtual servers, reducing response times and improving application performance.

2.2.3 Virtualisation Technology

Much of the recent achievements in cloud computing has been made possible through advancements in virtualisation technology. Virtualisation is the faithful reproduction of an entire architecture in software which provides the illusion of a real machine to all software running above it [Kiyanclar 05]. It facilitates cloud computing as it allows larger physical servers to be partitioned into multiple smaller virtual machines. These machines are insulated from each other, with respect to fault tolerance, security and operating system heterogeneity. Xen [Barham *et al.* 03] the popular open source virtualisation framework, supports a wide range of guest operating systems and is used by a large number of cloud providers including Amazon Web Services.

Virtualisation began with IBM back in the 1960s where it was first applied to mainframe computers. In 1967 and 1968 the first hypervisor, the $CP-67$ was built by IBM which allowed memory sharing between virtual machines. IBM's Control Program-Console Monitor System (CP-CMS) introduced a two pronged virtualisation solution. The CP component was run on

the mainframe computer where its main responsibility was to spawn virtual machines which run the CMS system. This provided each user with his/her own machine from which they could interact with. This approach greatly improved the utilisation of these large mainframe computers as this had been a significant early problem.

Prior to the development of modern virtualisation technologies, many high-end servers lay wholly under utilised as each server was responsible for the management of a specific piece of software or application component. It was not uncommon for companies to have separate servers for different applications and functions, such as a print server, mail server and domain server. This led to enormous under-utilisation of the available resources even though it was considered best practice at the time. The reason why this was done was to isolate the possibility of system wide failures due to a single fault or bug proliferating throughout the server. Thus if the print server was to fail, you would not lose the mail server as well!

With the advancement of virtualisation technology multiple applications could reside in apparent isolation on the same physical resource. Secure from each other, a known vulnerability in one virtual machine or a malware infection in another could not impact or spread to all the other virtual machines. Now a single large physical server could host all of your applications and business functions securely on the same host machine, reducing the server overhead significantly.

Virtualisation itself encompasses a number of sub-domains, including full virtualisation, partial virtualisation and para-virtualisation. Full virtualisation involves an almost complete simulation of the actual hardware, allowing software which usually includes the guest operating system to run completely unmodified as if it were running on the physical machine directly. With partial virtualisation some of software requires modification to run on the virtual machine as not all of the target environment is virtualised. Para virtualisation requires software modifications to successfully run in the environment. Special operating system images have to be used in order to deploy software successfully under this framework. However most new operating systems contain updated kernels and special drivers to allow the operating system to be deployed in a para-virtualised environment unmodified.

2.2.3.1 Virtual Machine Monitor - Hypervisor

The sharing of resources amongst the respective virtual machines is handled by the Virtual Machine Monitor (VMM) [Barham *et al.* 03], an independent domain level monitor which has direct access to the underlying hardware. A single machine instance created by the hypervisor is known as a virtual machine. The virtual machine is a software controlled emulation of a real physical system upon which the operating system and subsequent applications reside. The hypervisor is inserted between the server's hardware and the operating system, allowing for the deployment of multiple virtual machines in isolation, decoupling the operating system from the physical host. From the operating systems perspective, the virtual machine represents a completely isolated environment that looks and feels just like a physical machine. Access to CPU cycles, memory, disk storage and network infrastructure is handled by the VMM hypervisor, meaning that no special software is required and the operating system can be deployed directly on the virtual machine.

The virtual machine is the enabling technology of the cloud and in particular for IaaS clouds where private users can purchase virtual machines with variable cost, size(cpu, memory, disk) and leasing structure. Numerous hypervisor technologies have been developed such as the open source Xen [Barham *et al.* 03] and Linux Kernel Virtual Machine (KVM) [Kivity *et al.* 07], in addition to commercial offerings such as VmWare ESXi and Microsoft's Hyper-V.

The Xen VMM was borne out of the Xenoserver research project conducted by Keir Fraiser and Ian Pratt in the late 1990s at Cambridge University. The Xen hypervisor is a software controlled solution which is inserted between the server's hardware and the operating system. Operating at a higher privilege CPU state than any other software component, the hypervisor is responsible for memory management and CPU scheduling of its guest virtual machines. The Xen project has been supported by industry and academia alike, with partners such Citrix, IBM, Intel, Hewlett-Packard and more, contributing to its development and production. Xen is the virtualisation technology presently used to power Amazon's EC2 computing infrastructure.

The Kernel Virtual Machine (KVM) is a linux based domain level hypervisor which supports para virtualisation of the physical host. It leverages upon the x86 virtualisation extensions to add VMM hypervisor capability to Linux [Kivity *et al.* 07]. KVM supports a number of guest operating systems, including Microsoft Windows, Mac OS X, and numerous flavours of Linux.

One of the main performance challeges encountered by VMM hypervisors relates to I/O virtualisation and the sharing of the host devices amongst the VMs. When it comes to virtualising memory and the scheduling CPU time between guests there has been great success but access to shared I/O such as the underlying disk subsystem is still a problem.

2.2.3.2 Co-location & Virtual Machine Interference

Co-location refers to the scenario where multiple independent virtual machines reside side by side (co-located) on the same physical host machine.

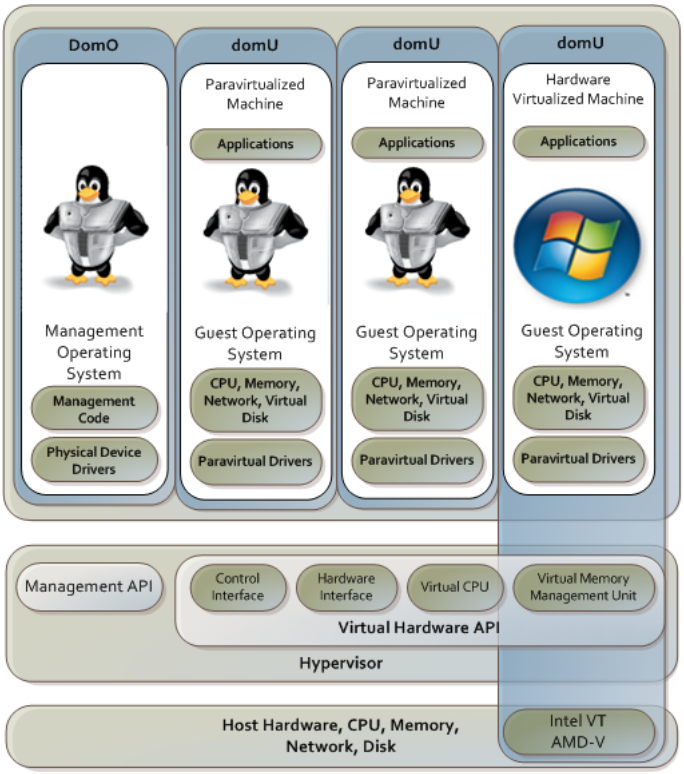


Figure 2.2: Typical virtulasation configuration, including co-located VMs and hypervisor

Figure 2.2 [2] describes a typical virtualisation configuration consisting of a Dom0 hypervisor which controls access to the underlying hardware and a number of DomU paravirtualised machines with different operating systems and applications. Each virtual machine is isolated from each other and not aware of the neighbouring virtual machines with which it shares its physical host with. Virtualisation technologies such as Xen or Oracle VM server provide excellent security, fault and environmental isolation but they do not ensure performance isolation. Thus

[2] Courtesy of docs.oracle.com

you cannot guarantee a particular performance level as a function of the configuration of the virtual machine. This problem becomes more apparent in public cloud environments when disparate applications, resources and interests are co-located, causing them to interfere with each other. In particular when neighbouring virtual machines are executing large read and write volumes to the shared disk. The shared I/O bandwidth amongst all virtual machines becomes saturated, slowing down your application's read and write operations resulting in a performance degradation.

Koh et. al. state that there are three principle causes of disk I/O interference [Koh *et al.* 07]. The first cause is due to the fact that each independent virtual machine on the hypervisor has its own resource scheduler which is attempting to manage shared resources without the visibility of others. Secondly guest operating systems and applications inside a virtual machine have no knowledge about ongoing work in co-located virtual machines and are unable to adapt in response. Thirdly some hypervisors such as the Xen hypervisor offload operations such as I/O operations to service virtual machines. This particularly affects I/O-intensive applications as the Xen hypervisor forces all I/O operations to pass through a special device driver domain, this forces the context to switch into and out of the device driver domain and then ensuing drop in performance.

Interference caused by other co-located virtual machines in clouds is often colloquially referred to as the "noisy neighbour" problem. Mitgation against these problems is incredibly difficult in virtualised environments as while CPU and memory maybe relatively easy to virtualise, other system devices such as the disk subsystem are not [Armbrust *et al.* 10]. VMM hypervisors such as Xen also allow virtual CPUs to go over their allocation if there are idle cycles available on the host machine. This allows your applications to request and use extra cycles if they are available. This is known as "*CPU steal time*" and it can be observed in Linux OSs by executing the sysstat command. The formal definition of steal time is "the time the virtual CPU waits for a real CPU while the hypervisor is servicing another virtual processor". Observing high steal times on your instance may mean that your application requires extra cycles over its allocation but cannot be given them as result of other host operations. The problem with this is the variability at which it occurs, making it difficult to plan in advance and can result in abnormal application behaviours. At a cloud performance summit [Cockcroft

11] in 2011, Adrian Cockcroft of Netflix gave a presentation on how Netflix manages performance on EC2 and in particular their focus on steal time. Their solution is to define a threshold concerning the CPU steal time metric, that once breached, initiates the winding down of that particular virtual machine and the spawning of it elsewhere (hopefully) in a different part of the data center. Their hope is that the new virtual machine will now reside on a more under utilised physical host where steal time is normal. Whilst large organisations such as Netflix possess the required resources in conjunction with the necessary in-built systems to handle these eventualities, smaller firms may not.

2.3 Resource Allocation & Workflow Scheduling

2.3.1 Resource Allocation

Resource allocation refers to the process of distributing a limited resource amongst a set of activities which require the resource to achieve a specified goal or function. Given the limited availability, resources need to be allocated to activities efficiently in order to maximise the overall utility. The optimal resource allocation approach is the one which yields the greatest throughput/utility over the problem, often considering multiple constraints and challenging conditions.

The problem of allocating resources according to specific constraints has been the subject of extensive research activity for many decades. It is encountered across numerous problem domains, such as allocating resources to products in light of competing product champions for SmithKlein Beecham in operations research [Sharpe & Keelin 98], analysing monopolistic resource allocation and its effects [Harberger 54], the efficacy of auction strategies on the fair allocation of resources [Bernheim & Whinston 86], a networking centric resource allocation approach to ensure a level of data rate fairness among users [Shen *et al.* 05] and in computer science Jain et. al. propose a quantitative index of fairness for ranking allocation strategies [Jain *et al.* 84].

This thesis discusses resource allocation in the context of allocating computational resources to support applications or services executing on infrastructures such as grids or clouds. In the literature, dynamically allocating resources to support applications is referred to as application

scaling. Thus subsequent references to resource allocation and the scaling of resources to support applications/services reference the same concept, in the context of this work.

Our work is particularly focussed on horizontal scaling techniques which involve scaling applications by increasing or decreasing the number of virtual machines supporting the given application or service. Our discussion of horizontal scaling solutions examines the state of the art from two perspectives, threshold based approaches and artificially intelligent solutions. Defining thresholds provides a set of heuristics by which resources are allocated and deallocated in response to violations of the thresholds. Threshold models do not possess any long term strategic planning capabilities, however they are easy to define for relatively simple applications. The second approach involves utilising techniques from Artificial Intelligence (AI) to learn optimal allocation strategies. These approaches can successfully learn in the absence of prior information and complete knowledge, but require expertise in machine intelligence and may require a significant learning period during which the approach operates at a sub-optimal level.

2.3.1.1 Resource Allocation & Uncertainty

Allocating resources in the absence of uncertainty reduces to straightforward planning and can be solved with techniques such as the STRIPS [Fikes & Nilsson 72] approach, Graphplan [Blum & Furst 97] or partial-order planning [Barrett & Weld 94] techniques. However mapping tasks to distributed resources such as computational grids and clouds creates problems with deterministic approaches as the resources themselves exhibit non deterministic behaviours. In order to optimally plan and act within these domains non-deterministic solutions are required.

As discussed previously we specifically examine resource allocation problems, focussing on mapping to computational resources. The resources support the execution of applications and services which experience variable numbers of user requests throughout each day. The goal is to allocate resources to these applications to maintain quality guarantees, such as response time levels given by service providers. However as there is an associated cost with each additional unit resource, the service provider needs to ensure that it is optimising its resources as efficiently as possible.

Throughout this thesis, we define a resource instance as an independent virtual machine. The instances are allocated and deallocated to support processes and applications in accordance with the defined resource allocation strategies. In computational clouds the virtual machines can be composed to different configurations with respect to CPU, memory, disk, which governs their performance. In addition virtual machines can be spawned in different geographical regions allowing for lower latency links between provider and consumer within these regions.

Virtual machines experience daily fluctuations in their performance due to the nature of virtualised computing models (detailed in section 2.2.3.2). Co-located virtual machines through the so called "noisy neighbour" phenomenon can adversely affect each others performance due to the mechanisms by which shared devices are handled on the physical hosts. The noisy neighbour effect and entities such as internal network latencies and congestion lead to performance unpredictability in infrastructure computing environments.

We classify three observable patterns of occurrence with respect to performance uncertainty within infrastructure computing environments. We have previously discussed the causes of these interferences in section 2.2.3.2, we now wish to identify them.

1. Instance specific: Certain instance types are more vulnerable to interference than others. A common example would be shared hosted instances like the Amazon EC2's micro instances. In addition different virtual machine types(e.g. m1.small, c1.medium on EC2) have access to a different share of Disk and Network I/O within the host.

2. Temporal variations: Temporal fluctuations in performance generally follow a cyclical or periodical pattern. An example of this is peak usage times in the data center, which causes a widespread slow down across instances due to greater volumes of people accessing the applications.

3. Regional variations: Virtual machines spawned in different regions may suffer performance issues specific to that region. Problems such as unreliable or congested network infrastructures in these countries cause significant slow downs for requests between the cloud and the end user.

The approaches we consider in the context of AI are capable of reasoning over this uncertainty

using experience from operating directly within the environment. Observability over the necessary state information allows the learning agent to associate the effects of their actions with changes in state. Through repeated interactions they can over time build up a representation of the environmental dynamics.

2.3.1.2 Threshold Based Models

One approach to allocating resources to applications or services is to quantitatively monitor resource consumption levels in real time and allocate/deallocate quantities of the resource in response. These approaches are known as threshold based solutions as one defines usage thresholds over the specified resources. Once a threshold has been violated i.e. where either a resource or a number of resources lay under or over utilised, an allocation decision is executed. The configuration of both the thresholds and the corresponding actions require a domain expert with knowledge of both the current environment and the individual nuances of the application/service itself.

2.3.1.2.1 Commercial Solutions Threshold solutions are currently the standard approach to scaling on commercial public infrastructure cloud platforms. Public clouds such as Amazon Web Services [Amazon-WS 13] provide commercial auto scaling solutions to facilitate resource allocation based on predefined thresholds. Amazons Auto Scaling software in conjunction with their proprietary CloudWatch monitoring system can be used jointly to automate resource allocation decisions based on the pre-defined thresholds. These thresholds stipulate decision points at which an action is triggered i.e. to add or remove a particular instance type. RightScale [RightScale 13] similarly allows for the definition of process load metrics which trigger allocation responses once a given metric has been breached. Rightscale facilitates the allocation of resources to applications running across multiple clouds. With this approach a mapping exists denoting the representative action which must be executed once a specific threshold has been breached. These are rule based systems which are both myopic and reactive.

2.3.1.2.2 Holistic Threshold Models Threshold based policies such as those employed by RightScale and Amazons Auto Scaling, focus on scaling at the machine or virtual machine

level. They do not facilitate the definition of higher business functions and objectives when scaling a given service or application running on the cloud. Instead, the applications resources are altered through allocation and deallocation of virtual machines.

Rodero-Merino et. al. propose a dynamically scaling solution aimed at addressing this issue [Rodero-Merino *et al.* 10]. Their proposed solution operates at a higher level of abstraction than those offered currently by public infrastructure providers. It decomposes high level objectives specified in a Service Description File and also the scalability rules defined by the Service Provider. Rodero-Merino examines three different scalability mechanisms which consider a more holistic approach to service management on federated clouds. They propose an application layer called Claudia which employs a Virtual Infrastructure Management solution avoiding vendor lock-in and capable of interfacing between different clouds.

Extending from this, Moran et. al. argued that the mechanisms employed by Claudia were not expressive enough to enable a fine grained control of the service at runtime [Moran *et al.* 11], citing the fact that the scalability rules defined are specified in an ad-hoc manner and are not designed with sufficient generality. To interchange the abstract level languages used to specify application behaviour in clouds, they propose the usage of the Rule Interchange Format, in conjunction with the Production Rule Dialect. This generates an appropriate mapping to industry rule engines such as *JBoss Drools* or *Java Jess*. These approaches improve upon the industrial led approaches offered by Amazon and Rightscale in an attempt to auto scale applications in a more holistic manner but they still require a certain amount of domain knowledge, with rules and conditions to be defined to scale the resource.

Threshold based approaches have also been developed for elastic storage solutions. Lim et al. developed an automated controller for elastic storage in a cloud computing IaaS environment based on proportional thresholds [Lim *et al.* 10]. The controller was broken down into three components;a Horizontal Scale Controller for adding and removing nodes; a Data Rebalance Controller, for controlling data transfers; and a State Machine to coordinate the actions. This approach demonstrated speedy adaption to fast changing events, however this is not always optimal given a particular event may be very short lived.

Accurately planning the appropriate corrective action for each possible scenario can prove extremely difficult especially when one has to consider a large number of possible parameters

[Dutreilh *et al.* 10, Rolia *et al.* 06] such as for example the present CPU, Disk, memory utilisation or the current number of incoming requests. Achieving a high level of accuracy and precision requires a high degree of expertise with respect to both the application being scaled and the resources used to scale it.

2.3.1.3 Artificially Intelligent Solutions

An alternative solution to defining thresholds is to utilise methods from Artificial Intelligence (AI) to autonomously control allocation decisions to maintain certain service levels at minimum cost. AI based learning techniques such as machine or reinforcement learning can solve a number of problems in this regard. Firstly they can reason over the inherent uncertainty observed within these environments, devising strategic plans in order to optimally choose actions which maximise the user defined objectives in the long run. The second benefit is that techniques such are reinforcement learning can learn this plan without prior knowledge operating directly within the environment. This reduces the need for expertly defined thresholds and mappings for each state of the environment. In addition the learning techniques can cope with changes to the application or workload request models by adjusting their actions based solely on their observations.

2.3.1.3.1 Reinforcement Learning Gerald Tesauro from the IBM J Watson scientific research center proposed a solution to the massive over provisioning of server resources in order to cater for spikes in demand, through autonomic computing methods [Tesauro *et al.* 05]. Using a simulated "Trade 3" application (simulation of a real time trading platform application) and an open loop Poisson process for modelling the incoming user requests, the author compared a queueing theoretic model based solution, a static allocation, a random allocation and a model-free SARSA(0) reinforcement learning algorithm. The authors found that for the specific problem of allocating virtual servers to support the Trade 3 application, both the model based queuing algorithm and the model free SARSA(0) methods performed best, with virtually no performance difference between them. Each solution had different merits for the specific problem with the author suggesting that the SARSA(0) required less systems knowledge than the queuing approach. In addition the author suggested that certain conditions such

as changing user behaviours and service-level utility functions will affect each approach differently, citing that SARSA(0) would likely perform better in scenarios where user behaviours change but would require complete re-learning of value functions should the service-level utilities change. In contrast the queuing model would not require any changes coping easily with the service level utility changes but pointing out that changes in user behaviour could lead to "model drift" [Tesauro *et al.* 05] making the approach less accurate over time.

Extending from this work Tesauro investigated the efficacy of a hybrid reinforcement learning technique for autonomic resource allocation [Tesauro *et al.* 06] which combines model based approaches with online reinforcement learning. This approach involved training the reinforcement learning system offline using a dataset generated from a good model based policy. This provided the learning algorithm a solid offline training environment removing the likely poor performance which naturally occurs during the initial exploratory stages of learning when the the system chooses sub-optimal exploratory actions prior to learning more optimal configurations. Once the technique has built up a good approximation of how the environment responds to its actions offline, it can act more optimally once it goes live.

David Vengerov [Vengerov 07] combined reinforcement learning with a fuzzy rulebase to allocate CPU and memory from a common resource pool to multiple entities which share the resources. The author does not strictly rely entirely on reinforcement learning methods instead applying a fuzzy rule base. The reinforcement learning technique computes value function estimates using the fuzzy rules as a guiding policy. The approach has the capability to override the policy if observations provide better solutions. They implemented their DRA-FRL architecture in Solaris 10 demonstrating good performance at matching statistically changing workloads even in the presence of migration costs.

J. Perez et al. [Perez *et al.* 08] applied reinforcement learning to optimise resource allocation in Grid computing. This work focused on optimising the utility of both the users submitting jobs and the institutions providing the resources through Virtual Organisations. Virtual Organisations consist of a set of individuals or organisations that share resources and data in a conditional and controlled manner [Foster *et al.* 01]. Galstyan et al. [Galstyan *et al.* 04] implemented a decentralised multiagent reinforcement learning approach to Grid resource allocation. With incomplete global knowledge, and no agent communication, the authors showed that the reinforcement learning approach was capable of improving the performance of a large

scale grid. More recently a Q-learning approach was developed for allocating resources to applications in the cloud [Dutreilh *et al.* 11]. This work developed an automated controller capable of adding and removing virtual machines based on a variable workload model. The author presented an adaptive approach capable of keeping up with a variable workload model driven by a sinusoidal function. Using convergence speedups, Q function initialisation and model change detection mechanisms, the author was able to further optimise and fine tune the approach.

Finally Rao et al [Rao *et al.* 09] developed a reinforcement learning approach to virtual machine auto-configuration in clouds. In response to changes in demand for hosted applications the virtual machine itself is reconfigured. The approach was able to determine near optimal solutions in small scale systems.

2.3.2 Workflow Scheduling

Increasingly, modern day scientific experiments in fields such as computational epigenetics [Bock & Lengauer 08], astronomy [Berriman *et al.* 04] and earthquake analysis [Maechling *et al.* 05] require the processing and transforming of large scientific data sets, to advance research discoveries within their respective domains. Due to the scale of the experiments, large computational infrastructures are required in order to process the data within a suitable and useful time frame. Institutions lacking the appropriate hardware often rely on shared access to resources such as science grids to conduct their experiments.

The principle challenge of processing scientific experiments using shared resources, relates to both the scale of the individual experiments and the variability of the executing resources. A typical experiment can often comprise thousands if not hundreds of thousands of tasks [Deelman *et al.* 05], transforming terabytes of data in the process. In addition the resources within entities such as computational grids can vary with respect to the hardware, performance and geography.

Workflows have emerged as a paradigm for representing and managing complex distributed scientific computations [Gil *et al.* 07] in recent times. Modelling application processes in this manner allows for the speedy representation of a plethora of complex tasks, maintaining their dependencies in a valid hierarchical structure. Described as workflow applications, the

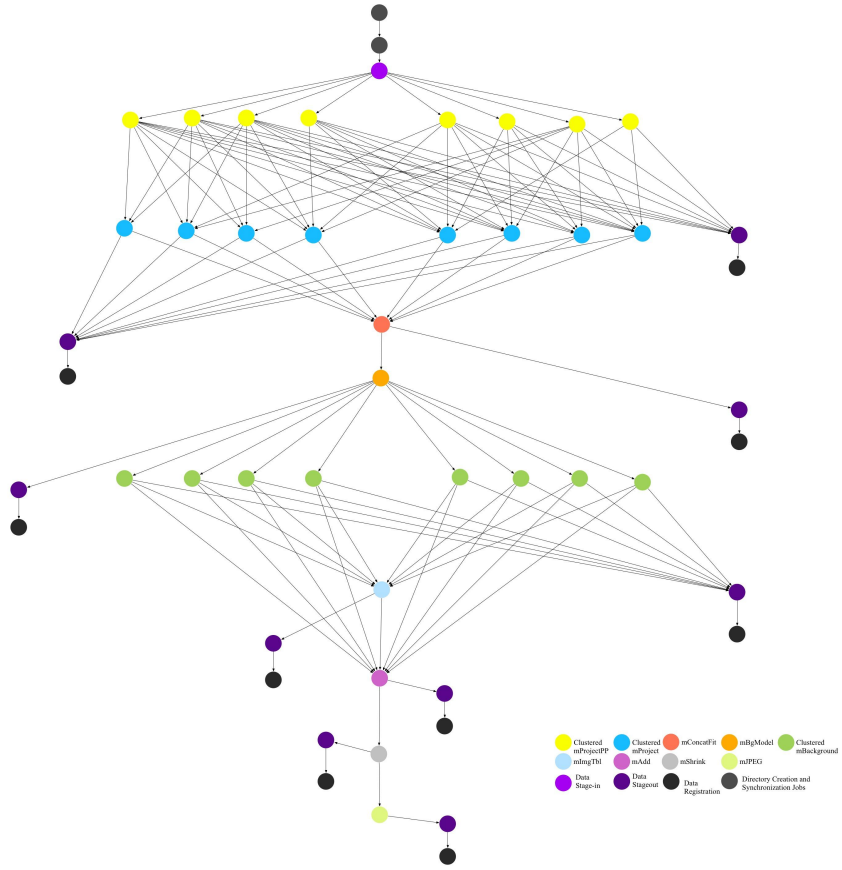


Figure 2.3: Montage : Image mosaic workflow application

workflow itself defines an order of precedence and dependence over a suite of computational tasks based on their control or data dependencies [Yu & Buyya 06]. The approach is also abstract enough to simplify the application development and execution process [Deelman *et al.* 05].

Once the workflow has been defined, any task for which its parent tasks have completed can then be executed. Thus a workflow scheduling solution can distribute tasks amongst the available resources as they become available, harnessing the potential of parallel hardware in a process intensive way. However scheduling algorithms must be cognisant of the fact that placing a task on the next available resource can be a naive strategy. This is particularly prevalent in data intensive workflows where scheduling child operations on the same resources as their parents reduces data transfer operations dramatically. In addition during the execution process a schedule that was valid in the beginning may become invalid due to resources going offline or increased network latencies/congestion.

Workflow applications are generally classified as either processor intensive, data intensive or memory intensive applications. A resource scheduler must consider the type of application it is mapping to resources when determining the appropriate execution schedule. An example of a workflow application designed to be run on computational grids is the NASA montage workflow [Berriman *et al.* 04], an application for computing detailed image mosaics of the sky collected from telescopic observations. Figure 2.3[3] depicts the structure of this workflow application. The NASA montage workflow is a data intensive workflow, with the size of data requirements dependant on the area of the sky one wishes to capture. An 8-degree mosaic of the night sky will result in a workflow consisting of $10,429$ tasks, read 4.2Gb of input data and produce 7.9Gb of output data [Berriman *et al.* 10]. Montage is data intensive as it spends 95% of its time waiting on I/O operations to complete. From figure2.3 it can clearly be seen that certain tasks which share the same level, can be executed on parallel hardware to reduce the compute time. However with such large data transfer requirements between tasks, careful consideration must be given to the resource locations when processing tasks.

2.3.2.1 Workflow Scheduling Algorithms

In this thesis, we examine the state of the art concerning workflow scheduling onto distributed resources such as computational grids and clouds. The problem of mapping tasks onto distributed disparate resources is generally considered an NP-complete problem [Ullman 75]. Computational complexity theory is an area of mathematics and computer science which defines classifications over the complexity or the inherent difficulty of problems. The NP classification stands for Non-deterministic Polynomial and a problem is considered NP-complete if there are no known algorithms capable of deriving the optimal solutions in polynomial time. In addition, specific to NP-complete problems, a candidate solution can be evaluated and verified easily, but finding the solution in the first place is challenging.

Cognisant of this fact we examine workflow scheduling from two perspectives, heuristic approaches and meta-heuristic approaches. In the field of search and optimisation numerous meta-heuristic algorithms have been proposed. These algorithms have been successfully deployed across a wide range of domains to generate good solutions to NP-complete and even

[3] Courtesy of confluence.pegasus.isi.edu

NP-hard problems. Christian Blum and Andrea Roli [Blum & Roli 03] characterise two separate meta-heuristic approaches. One strategy attempts to improve local search algorithms with algorithms such as simulated annealing, Greedy Randomised Adaptive Search Procedure (GRASP) and tabu search prevalent in the field. The second approach is to integrate a learning component within the search mechanism, with techniques such as Genetic Algorithms (GAs), Particle Swarm Optimisation (PSO) and Ant Colony Optimisation (ACO) prevalent in the field. In keeping with the theme of this work and to contextualise our research contributions, we concentrate our discussion on meta-heuristics which integrate learning and search.

2.3.2.1.1 Meta-heuristic Workflow Scheduling Meta-heuristic algorithms such as Genetic Algorithms (GAs) [Goldberg 89], Particle Swarm Optimisation (PSO) [Kennedy & Eberhart 95] and Ant Colony Optimisation (ACO), have been devised and inspired from studies into the processes of biological mechanisms. These algorithms have demonstrated efficacy and performance, across an extended array of disparate and cross-functional domains.

Jia Yu and Rajkumar Buyya [Yu & Buyya 06] propose a workflow scheduling solution which uses Genetic Algorithms (GAs) to evolve valid workflow schedules in an iterative manner. The approach involves a user specifying either a deadline or a budget constraint. Depending on the constraint the evolved schedules are biased towards minimising the execution cost or the makespan of the workflow. The authors compared the GA approach to two heuristic scheduling solutions, Greedy Cost-Time Distribution and Greedy Time-Cost Distribution [Yu & Buyya 06]. In addition, the performance on two different structural workflow applications, balanced-structure [Mandal *et al.* 05] and the more complex unbalanced-structure were evaluated. The authors varied the initial GA populations from randomly generated, to those informed by the simple heuristic approaches. The results showed that the GA found better solutions than the heuristic approaches for the more complex workflow scenarios. The greatest performance was observed when the initial population of the GA was generated by the output of the heuristic algorithms, effectively combining the heuristic solution with the GA approach.

Extending on this work the authors proposed a workflow scheduling approach utilising techniques from Multi Objective Evolutionary Algorithms (MOEAs). MOEAs can produce a set of solutions given a number of competing constraints. In this work the authors specified two constraints a cost and a deadline constraint. Their goal was to produce a number of non-dominated

scheduling solutions from which a user can choose from. Their simulation results demonstrated that MOEAs could produce a range of compromise solutions within a short time. They demonstrated this using three separate MOEA algorithms, Non-dominated Sorting Genetic Algorithm (NSGAII) [Deb *et al.* 00], Strength Pareto Evolutionary Algorithm (SPEA2) [Zitzler *et al.* 01] and Pareto Archived Evolution Strategy (PAES) [Knowles & Corne 99].

Gharooni et. al. [Gharooni-fard *et al.* 10] devised a workflow scheduling approach using a modified GA called a Chaos-GA. Adopting a similar formalism to Yu's [Yu & Buyya 06] work, the authors demonstrate improved scheduling solutions on both the balanced and unbalanced workflows as a result of using the chaos-GA approach. Chaos-GA's utilse a chaotic system approach as opposed to the random process approach commonly employed by GAs.

Pandey et al. [Pandey *et al.* 10] recently developed a cloud based workflow application scheduling approach using Particle Swarm Optimisation (PSO). The PSO is a stochastic search and optimisation algorithm inspired by the movement of organisms such as the flocking of birds and the schooling of fish. The authors specifically designed the approach to consider both the resource execution costs and the data transmission costs inherent in cloud infrastructures. Their experiments showed that the PSO can provide a good distribution of tasks across the available cloud resources, reducing resource overloading. They compared the PSO solution against a Best Resource Selection (BRS) approach similar to what is used to schedule tasks to resources in systems such as Condor DAGMan. The PSO performed better than the BRS approach, as the BRS approach did not factor in the data transmission costs between resources in the cloud.

Zhangjun Wu et. al. [Wu *et al.* 10] improved upon this work by implementing a modified PSO algorithm for scheduling workflow applications in the cloud. Called the Revised Discrete Particle Swarm Optimisation (RDPSO), the authors were able to demonstrate improved performance over the standard PSO approach proposed by Pandey et. al.

Ant Colony Optimisation(ACO), like the PSO is a swarm intelligence technique first proposed by Marco Dorigo in his thesis in 1992 [Dorigo 92]. The idea arose from observations of the behaviour of ants seeking an optimal path between their colony and a source of food. Chen and Zhang [Chen & Zhang 09] propose the scheduling of workflow tasks onto grid resources using an ACO approach. Optimising user defined Quality of Service (QoS) constraints the

authors propose seven new heuristics and provide an adaptive scheme allowing the ants to choose heuristics based on the pheromone values [Chen & Zhang 09]. The QoS constraints include, reliability, time, cost and are defined by the user.

2.3.2.1.2 Heuristic Scheduling Schemes The crux of heuristic scheduling systems revolves around the definition of ordered lists and the assignment of task priorities by which one allocates resources to tasks. For this reason, list scheduling is a term commonly applied to describe a particular set of these heuristic approaches. Heuristic scheduling strategies for workflow applications are designed not to find the optimal solution each time but a good solution in a realistic time frame. A variety of heuristic strategies have been proposed by the research community for scheduling tasks to grids in workflow applications.

Jia Yu and Rajkumar Buyya [Yu *et al.* 08] separate list scheduling algorithms into three categories, Batch mode, Dependency mode and Dependency-Batch mode. Batch mode scheduling is suited to the scheduling of a batch or collection of tasks in parallel onto a pool of distributed resources. The tasks are considered independent of each other, meaning the algorithm does not need to consider precedence constraints. The goal is to schedule tasks to resources in the order that will result in the earliest finish time of all the tasks. The Min-min, Max-min and Sufferage algorithms are all exponents of this and are utilised extensively in workflow management systems (covered in section 2.3.2.1.3). These approaches were first proposed by Maheswaran et. al [Maheswaran *et al.* 99] in 1999.

The Min-min task scheduling algorithm calculates the Estimated Completion Time (ECT) for all tasks on each available resource. It then schedules the task onto the resource which has the minimum completion time resulting in a high proportion of tasks executing on the most suitable resource. This approach prioritises the scheduling of tasks which have the shortest execution times. However one of the disadvantages of this simple heuristic is that it has been shown to leave a significant number of resources idle during execution under certain circumstances [Blythe *et al.* 05].

The Max-min heuristic is the opposite to the Min-min algorithm and instead schedules tasks to resources which have the longest completion time first. By calculating the Maximum Estimated Completion Time (MCT) for each task, the approach allocates the task to the resource which it expects will complete it in the quickest possible time. This approach prioritises tasks

which have the longest execution time.

The Sufferage algorithm defines a metric called the sufferage value, which is the difference between its first and second Earliest Completion Times (ECT) on the best available resources. It prioritises the scheduling of tasks to resources where the difference between the first and second ECTs are greatest.

Depending on the type of tasks being scheduled the performance of each approach varies. Results have shown that Min-min outperforms Max-min [Maheswaran *et al.* 99] [Casanova *et al.* 00] in most scenarios except for situations where there are a higher proportion of short tasks in the workflow [Braun *et al.* 01] as the longer running tasks are delayed until the shorter ones have completed. The Sufferage algorithm performs better where there is a greater performance difference between the underlying resources [Maheswaran *et al.* 99].

Dependency mode schedulers produce schedules which take into account the inter dependencies between the various tasks in the workflow. Unlike the previous Batch mode algorithms, Dependency mode algorithms rank all the tasks together and analyse the dependencies with respect to the entire task graph [Yu *et al.* 08]. A number of weight dependency schedulers have been developed which are designed for homogenous environments where the network uplinks and node types are identical througout [Kwok & Ahmad 99]. These approaches then weight the task graph accordingly, specifying individual weights for nodes and edges. In grid and cloud environments however, resources can be heterogenous from a physical host perspective and even at the virtual machine level. In addition bandwidth between resources can vary as a result of internal congestion in the data center for both network and disk I/O.

Within the Vienna Grid Environment Ivona Brandic [Brandic *et al.* 05] proposed a QoS aware Grid workflow engine to support the execution of time critical workflow applications. Through a proposed Grid workflow language, users could specify QoS constraints for the Grid workflow. Two planning approaches for scheduling workflows were proposed, a static and dynamic planning mechanism. The static method utilised integer programming to optimise the defined objective. The dynamic planning approach utilised a Multi Criteria Decision Making mechanism [Zeng *et al.* 04] to map tasks to services. Their initial results demonstrated promising performance with regard to the mapping of time critical workflows on the Vienna Grid Environment. Extending from this the Amadeus framework [Brandic *et al.* 08] was developed

to provide a holistic service oriented environment for QoS aware Grid workflows. Amadeus allowed users to specify QoS requirements which were considered during the planning, specification and execution of the workflow applications onto computational Grids. One of the contributions of this framework was to include not just performance and economic aspects but also legal and security aspects within the framework.

The Heterogenous Earliest Finish Time (HEFT) [Topcuoglu *et al.* 02] extends classical list scheduling schemes suited for homogenous systems, by defining an approach suitable for list scheduling onto heterogenous environments. It has been successfully deployed to schedule tasks for scientific workflows on the ASKALON [Fahringer *et al.* 05] project. In addition it has also been deployed in a quantum chemistry application, WIEN2K [Blaha *et al.* 08], a hydrological application, Invmod [Theiner & Rutschmann 05] and on the Austrian Grid. The algorithm computes not only the average execution time of each task but also the data transfer times between two successive tasks in the workflow, assigning each task a rank value. The tasks are then sorted in descending order according to the computed rank values, with the highest priorities in a scheduling context going to the tasks which have the highest rank values. Resources are then allocated to tasks according to an earliest finish time strategy. Zhao and Sakellariou [Zhao & Sakellariou 03] conducted an investigation into the performance of HEFT by experimenting with a number of different schemes for calculating the task weights. Their experiments showed that choosing the average execution time for tasks on resources is the not the most optimal approach [Zhao & Sakellariou 03].

2.3.2.1.3 Workflow Management Systems Workflow Management Systems (WMS) are software frameworks which coordinate workflow setup, planning, mapping, execution and run-time monitoring onto a collection of resources such as a computational grid, cluster or cloud platform. Workflow management platforms allow non-technical personal from various backgrounds to execute their workflows in a simplified user oriented manner. All aspects of the execution process are coordinated, including task failures and the subsequent rescheduling, data transfers between tasks, specifying QoS constraints and of course the initial scheduling of tasks to resources.

Deelman et. al. propose Pegasus [Deelman *et al.* 05] a WMS which facilitates the execution of complex scientific workflow applications across a variety of different platforms. One the

advantages of using Pegasus is that the user no longer needs to worry about the particulars of the underlying execution platform, i.e. grid/cluster or installed middleware components, Condor/Globus. Instead the user can focus on defining their workflow application in abstract form and allow the system to automate the data sourcing and task mappings. Pegasus has been used to schedule workflows in domains such as Astronomy [Deelman *et al.* 03b] with the NASA montage image mosaic engine, high energy physics [Deelman *et al.* 03a] and Gravitational wave analysis [Abramovici *et al.* 92]. More recently Gideon Juve and Ewa Deelman [Juve & Deelman 10] successfully deployed the pegasus WMS on the Amazon EC2 infrastructure cloud highlighting this approach as a cost effective alternative when compared to High Performance Computing Systems [Deelman *et al.* 08]. Systems such as Pegasus in complement with Condor DAGman [Frey *et al.* 02] focus on allocating tasks to resources in a myopic fashion, assigning the best resource available to the task, irrespective of any long term strategic planning.

The Taverna WMS [Oinn *et al.* 04] is a tool for the mapping of scientific workflows onto distributed resources. It has been used in Bioinformatics research for the life science community to conduct insilico experiments in biology. In addition projects such as Gridflow [Cao *et al.* 03], Kepler [Ludäscher *et al.* 06], Triana [Taylor *et al.* 07], Gridant [Amin *et al.* 04] and ICENI [Mayer *et al.* 03, McGough *et al.* 04]. The majority of these systems map tasks to resources based on simple heuristics, similar to what was described in the previous section. The approaches calculate suitable metrics such as Earliest Finish Time (EFT), Earliest Starting Time (EST) and choose the best resources available in this context.

Chapter 3

Learning and Optimisation

3.1 Introduction

"Mistakes are the portals of discovery" *James Joyce 1882-1941*

The ability to learn from one's experience offers distinct advantages over non experiential methods when dealing with dynamic environments [Kaelbling *et al.* 96]. One of the first mathematical frameworks dictating how to adjust ones subjective beliefs in light of new evidence was proposed by the mathematician and Presbyterian minister Thomas Bayes in the early 18th century. Bayes proposed that a single posterior probability conditioned upon experience for a given random variable could be computed provided one possessed knowledge of a single conditional probability (likelyhood) and two unconditional probabilities (prior and marginal likelyhood). This theorem became known as Bayes theorem and it provides a mechanism through which experience can be used to infer a probabilistic hypothesis.

The learning methods detailed in this chapter and throughout this thesis are all experiential methods. Some utilise Bayesian inference to update a specific hypothesis whilst others rely on approximating the optimal solutions based on their past experiences. In either case both methods learn autonomously without prior information and are powerful control techniques applicable to many problem domains.

The first agent learning method we examine in this chapter is reinforcement learning [Russell *et al.* 95]. Reinforcement learning is a subset of machine learning [Goldberg & Holland

88, Mitchell 97] and involves agents learning through direct interaction with their environment. By updating their experiences in a trial and error manner the agent iteratively builds up a model of how the environment responds to its various actions. A broad analogy which aids in the understanding of this technique is to consider how one would teach a dog a new trick. By framing the learning environment to reward good behaviour and punish bad behaviour, the dog through repeated interaction builds up a mental picture of what actions yield the greatest amount of reward each time. Thus from the agents perspective correct decisions yield a positive reward whilst incorrect decisions result in negative rewards. The second experiential learning method we examine is statistical learning. Statistical learning utilises techniques such as Bayesian inference to approximate missing or hidden aspects of the model through interactions. By assuming no prior information the agent observes how the environment responds to its decisions and adjusts its beliefs about the missing components. The third section examines parallel learning methods, detailing approaches for both reinforcement and statistical learning methods. Parallel learning methods facilitate learning over large state space representations and reduce convergence times in learning tasks.

The final section of this chapter examines approaches from evolutionary computation. Inspired by Darwinian evolution and natural selection, evolutionary methods have been successfully applied to solve a wide range of optimisation problems.

But firstly our investigation begins with the examination of the Markov Decision Process the solution of which forms the bulk of learning theory.

3.2 Markov Decision Processes

Markov Decision Processes (MDPs) [Bertsekas & Tsitsiklis 96, Puterman 94] are a mathematical framework particularly suited to modeling planning problems under uncertainty [Doshi *et al.* 05]. The process was named after the Russian mathematician Andrey Markov in honour of his extensive work and valuable contributions to the field of stochastic processes in the early 1900s. The discerning feature distinguishing Markov processes from other stochastic processes is that they satisfy a constraint known as the Markov property. The Markov property posits that only the current state of the environment is required to make predictions about

future states. The history of all the previous states leading up to the current state are irrelevant, making the process effectively 'memoryless'. A good example of this is the position of pieces on a checkers board. At any given moment in time the piece configuration on the board presents the player with all the information it needs to make his or her next move. The history of moves which has led to the current configuration on the board is lost but what remains is sufficient to allow the player to make a move [Sutton & Barto 98].

Formally a MDP can be represented as a 4-tuple (S, A, p, q), comprising states, actions, transition probabilities and rewards.

- S : The set of states defines all the possible states of the environment. It governs what the agent can observe when operating within the environment.
- A : This represents the action space, defining the set of actions which may be executed within each state.
- $p(.|s, a)$, known as the transition probability defines a probability distribution governing state transitions $s_{t+1} \sim p(.|s_t, a_t)$. In simple terms it is a measure of the likelyhood of a decision resulting in a transition to a particular state.
- $q(.|s, a)$, the reward signal defines a distribution of rewards received $R(s_t, a_t) \sim q(.|s_t, a_t)$ each time. It indicates the immediate value (positive or negative) of a chosen decision within a particular state.

The state space representation must be sufficiently informative to allow predictions be made with respect to the environment. However one must ensure it is also sufficiently concise to guarantee the problem remains within the bounds of computational tractability. Commonly referred to as the 'Curse of Dimensionality' it refers to the exponential increase in the size of the problem as additional states and actions are considered. Accurately determining an appropriate state space representation for a given domain is a challenging problem which has a direct impact on the performance of the learning system.

Given a particular state s and action a for a specific interval t, the transition probability to the next state s' is formally

$$P^a_{s,s'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{3.1}$$

The reward signal sometimes referred to as a reinforcement guides the decision making process. Often represented as single scalar value, it is indicative as to the immediate benefit of the executed decision. The expected reward achievable given the current action a, current state s and next state s' is

$$R^a_{s,s'} = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\} \tag{3.2}$$

The goal is to maximise overall return in the long run even at the expense of short term gains [Sutton & Barto 98].

Figure 3.1 depicts a diagrammatic representation of a finite[1] two state two action MDP. In this example a complete environmental model is available meaning that both the entire state space and transition probabilities are *fully observable*. A learning agent residing in *State 1* wishing

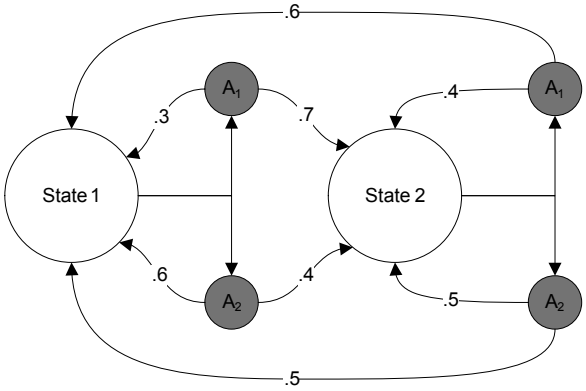


Figure 3.1: Simple two state two action MDP

to transition to *State 2* should choose action A_1 as it has a *70%* chance of proceeding to *State 2* and a *30%* chance of remaining in its current state *State 1*. Likewise an agent in *State 2* wishing to transition to *State 1* should choose action A_1, as it has a *60%* chance of transition to *State 1* and *40%* leaving its state unchanged.

The solution of a MDP involves the computation of state value $V(s)$ estimates for all states. State value estimates are a measure of "*how good*" it is to be in a particular state or the future return that can be expected from this state. Computing state value functions facilitates the discovery of optimal solutions to the MDP, the output of an optimal policy π. A policy π denotes a mapping from states to actions and governs the actions taken whilst in particular states. An optimal policy is the policy that yields the greatest amount of cumulative reward

[1] It is considered a finite-MDP because both the state and action spaces are finite.

over all states within the MDP. The value of a given state within the MDP is given by

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{3.3}$$

Equation 3.3 is known as the Bellman optimality equation for $V^{\pi}(s)$. Bellman demonstrated via his 'Principle of Optimality' that equation 3.3 is guaranteed to find the optimal policy in the MDP. Once we have calculated the value of each state, then the optimal policy greedily chooses the action each time which yields the highest value.

$$\pi^{*} = \arg\max_{a} \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V(s')] \tag{3.4}$$

γ represents the discount factor, where discounting the value of future states models the environmental uncertainty with achieving this state, i.e. you may never reach this state. It also acts to ensure convergence for MDPs which do not have a terminal state. By wrapping up the Bellman equation into an iterative technique we come to the first of the methods for solving the MDP known as *dynamic programming*.

3.3 Dynamic Programming

Dynamic Programming (DP) is a collective name for a suite of techniques which involve iteratively computing the solution to a problem in a recursive manner. The term was first used by Richard Bellman in the 1940s who chose the name dynamic programming because he thought it sounded impressive [Eddy 04]. The principle is based on splitting the problem up into smaller sub-problems which can then be solved individually. This is similar to the divide and conquer approach [Muniswamy 09], which is commonly used in domains such as sorting, large number multiplications and fast fourier transforms.The difference between DP methods and the divide and conquer approach is that DP problems need to have an optimal substructure and overlapping sub-problems [Denardo 03]. Optimal substructure refers to the fact that it must be possible to efficiently compute the optimal solution through the combination of solutions to the subproblems. If this is not possible then DP methods cannot be used to solve these problems. The subproblems must also overlap with each other otherwise the approach is considered divide and conquer. DP methods optimise the solution process by storing the outcomes to specific subproblems in a lookup table. Thus the next time the subproblem is encountered

its solution can be retrieved from the table negating the need to recompute each time. This section introduces two dynamic programming methods for computing optimal solutions to the MDP. The first method discussed is Policy Iteration.

3.3.1 Policy Iteration

The Policy Iteration approach begins with a specific policy and improves it iteratively [Poole & Mackworth 10]. The policy is a mapping between states and actions and it denotes what action is selected in each state. DP methods for solving MDPs are concerned with computing the optimal policy. Policy Iteration achieves this by manipulating the policy directly, as opposed to finding it indirectly by computing the optimal value function [Kaelbling *et al.* 96].

Algorithm 1 Policy Iteration

Initialize V arbitrarily for all s $\in$ S

Policy Evaluation
repeat
$\Delta \leftarrow 0$
For each s $\in$ S
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_{s'} P^{\pi(s)}_{s,s'}[R^{\pi(s)}_{s,s'} + \gamma V(s')]$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
until $\Delta < \Theta$

Policy Improvement
policy-stable $\leftarrow$ true
For all s $\in$ S
$b \leftarrow \pi(s)$
$\pi(s) = arg\max_a \sum_{s'} P^a_{s,s'}[R^a_{s,s'} + \gamma V(s')]$
if $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ false

Policy Iteration is a two stage process, consisting of a phase of policy evaluation followed by a policy improvement step. Algorithm 1 outlines the steps involved in the approach. Firstly all state values are initialised arbitrarily e.g. $S = 0$. An arbitrary policy is evaluated and the state values under that policy are determined via the equation $V(s) \leftarrow \sum_{s'} P^{\pi(s)}_{s,s'}[R^{\pi(s)}_{s,s'} + \gamma V(s')]$. This is the Bellman equation and it computes the value of each state under the current policy π. The approach recursively backs up the state value estimates and loops until the absolute difference between successive estimates $|v - V(s)|$ is less than θ (a very small number).

Once the state value function for a given policy π is known, we can then adjust this policy to see if a better one can be found, this is known as policy improvement. To maintain efficiency, the approach must adjust the policy so that subsequent policies are an improvement upon previously evaluated policies. This is achieved by generating a new policy which acts greedily[2] with respect to the state value function estimates computed from the previous policy. This new policy is guaranteed to be an improvement over the previous policy in accordance with the policy improvement theorem [Sutton & Barto 98]. The policy is evaluated again and state value function estimates are computed for all the states. The approach continues until no more improvements can be made and the policy is considered stable. This final policy must then be the optimal policy π^* as no more improvements are found and a finite MDP has only a finite number of policies.

Policy Iteration is guaranteed to find the optimal policy to a given finite MDP however this may involve many sweeps of policy evaluation and policy improvement. Another approach to computing optimal policies is to restrict the number of iterations of policy evaluation to a single iteration and then adjust the policy based on these values, this is called Value Iteration.

3.3.2 Value Iteration

Policy Iteration requires many sweeps of policy evaluation to approximate a given policy, with the value V^* convergence only guaranteed on the limit. Often it is not necessary to evaluate a given policy to such a degree. After a small number of iterations, approximations of its value can be roughly evaluated leading to a truncated more efficient process. This approach is known as Value Iteration and involves combining one sweep of policy evaluation and one sweep of policy improvement into a single sweep [Sutton & Barto 98]. Note: It is incorrect to think of Value Iteration as simply a truncated Policy Iteration algorithm, this is not the case because the update rule of the Bellman equation denotes the maximum valued action to be backed up each time as opposed to the action denoted by a given policy.

Algorithm 2 outlines the steps involved in the Value Iteration algorithm. Firstly all states are initialised arbitrarily $S = 0$. Value Iteration calculates the value of each state by computing the max valued action achievable from within that state $V(s) \leftarrow max_a \sum_{s'} P^a_{s,s'}[R^a_{s,s'} + \gamma V(s')]$.

[2] look ahead one step and choose the max valued action

Algorithm 2 Value Iteration

Initialize V arbitrarily
  repeat
    $\Delta \leftarrow 0$
    For each s $\in$ S
    $v \leftarrow V(s)$
    $V(s) \leftarrow max_a \sum_{s'} P^a_{s,s'}[R^a_{s,s'} + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  until $\Delta < \Theta$
  Output policy π
  For all s $\in$ S
  $\pi(s) = argmax_a \sum_{s'} P^a_{s,s'}[R^a_{s,s'} + \gamma V(s')]$

If the difference between successive value approximations $|v - V(s)|)$ is lower than θ we assume convergence. From a policy iteration perspective this is the equivalent of iterating the greedy policy each time instead of an arbitrary deterministic one. Thus the output policy is then computed by acting greedily with respect to the value function estimates which they themselves are a function of the greedy policy. The convergence guarantees of Value Iteration and proof of optimality regarding output policies for finite MDPs is treated extensively here [Bertsekas & Tsitsiklis 89].

The DP methods discussed here have been modified extensively and applied to domains ranging from operations research [Luss 82] to robotics [Bernstein *et al.* 05, Peters *et al.* 03]. The modifications generally focus on limiting the number of sweeps [Puterman & Shin 78] or prioritising sweeps [Moore & Atkeson 93] in which there were large changes in V or π. Additionally a number of enhancements have been made in deploying value iteration approaches to Partially Observable Markov Decision Processes [Pineau *et al.* 03, Cassandra *et al.* 97]. Partially Observable Markov Decision Processes (POMDPs) [Monahan 82] are a particular framework for modelling environments where one only has a partial or limited view of the state space. In this environment the agent's ability to directly observe the underlying state is restricted. The agent maintains a probability distribution over the set of possible states, based on actual observations and observation probability estimates. In addition to the state space limitations, transition probabilities and the distribution of rewards may also be unknown. This thesis deals strictly with MDPs where the states and actions are fully observable, discrete and stationary but the transition probabilities and distribution of rewards are potentially unknown.

We discuss two different reinforcement learning approaches to solving the MDP in the absence of this complete environmental model and demonstrate a new approach to speeding up convergence through parallel learning methods.

3.4 Reinforcement Learning

Dynamic Programming methods are efficient solvers of MDPs when one possesses a complete environmental model. When the set of states, actions, transition probabilities and distribution of rewards are fully observable and available then DP methods can effectively compute solutions of the MDP. However situations often arise particularly when dealing with real world problems where a complete environmental model is not available or cannot easily be approximated. In these scenarios an alternative approach is required to solve for optimal policies. The approach most commonly employed is known as reinforcement learning.

Stemming from analyses of behaviourist psychology [Schwartz *et al.* 02], reinforcement learning involves learning in response to corrective stimuli where positive rewards indicate relatively good action choices and negative rewards represent bad decisions. Typically the control problem is modeled as a MDP [Daw 03] however with the distinction that a complete model is unavailable rendering traditional DP approaches unsuitable for deriving solutions. Within the reinforcement learning framework the agent learner's goal is to maximise its reward in long run by choosing actions which have previously yielded the greatest amounts of reward.

Reinforcement learning can be broadly divided into two main categories, model based learning and model free learning. Model based learning methods utilise statistical techniques to approximate the missing model [Grzes & Kudenko 10] and then solve using traditional DP methods once they have approximated the missing model. Many established techniques exist with approaches such as Dyna [Sutton 91], Certainty Equivalent Methods [Kumar & Varaiya 86] and Prioritised Sweeping [Moore & Atkeson 93] prevalent in the area.

On the other hand, model free learners attempt to approximate a control policy through direct interactions with their environment. By observing the responses to their actions they maximise their reward by consistently choosing actions which yield the greatest amount of reward throughout the course of their interactions. Prevalent approaches in the area such as

Q-learning [Watkins 89], SARSA [Rummery & Niranjan 94] and TD(λ) [Sutton 88] have received significant attention from the research community in recent years.

3.4.1 Monte Carlo Methods

Monte Carlo (MC) reinforcement learning extends upon dynamic programming techniques such as policy iteration by computing value functions in domains where only partial models exist. MC methods offer an approach to solving finite MDPs through sampling the environment directly and observing how it responds. Surprisingly often situations arise where it is easier to obtain samples according to desired probability distributions but infeasible to obtain the distributions themselves in explicit form [Sutton & Barto 98].

One of the defining characteristics of MC methods is that they are episodic learners. This means that the environmental reward is only observable after the learning episode has completed. Thus the MDP formalism must model a problem which is guaranteed to terminate eventually with the reward allocated once the learning episode completes. A good example of this is the game of blackjack. If one were to model it as a finite MDP it would neatly break down into the various plays of the game. After the outcome of each game or episode the player would be rewarded depending on whether the game resulted in a win, lose or draw. The important property which defines MC domains is that each episode terminates and a return is observed at the end of the episode.

MC methods are policy iterators similar to the DP policy iteration approach outlined in the previous section. Given a state action space representation and a stationary deterministic policy MC methods can be used to evaluate that policy for a given MDP. This offers significant advantages over DP based policy iteration which requires the complete model dynamics. MC policy evaluation involves averaging the returns observed from successive visits to the same states under the current policy π. As greater amounts of rewards are observed the average value converges to the expected value, thus approximations of $V^{\pi}(s)$, the state value function under the current policy π can be determined.

MC methods represent the first real alternative to deriving solutions to a given MDP in the absence of complete environmental models but are not suited to many problem domains. The problem with MC methods is that they are not effective solvers for continuous tasks, thus

only problem domains which break easily into small finite MDPs with terminating episodes are suitable to be solved in this way. Additionally since they are in effect policy iterators, an initial policy must be evaluated first prior to being improved, this can result in very slow computational process with many episodes of experience required.

3.4.2 Temporal Difference Learning

One of the significant breakthroughs of reinforcement learning is the development of a suite of methods known as Temporal Difference (TD) learning [Sutton & Barto 98]. TD learning combines aspects of both dynamic programming and monte carlo methods and is applicable in scenarios where one only possesses a partial environmental model. Unlike monte carlo methods TD learners do not wait until the end of the episode to make predictions instead they incrementally update value function estimates $V^{\pi}(s)$ at the end of each timestep. This opens greater possibilities with regard to the problem domains which can modeled and solved using this approach. In particular for continuing tasks which do not easily break down into terminable episodes.

3.4.2.1 SARSA

State-Action-Reward-State-Action (SARSA) is an on-policy reinforcement learning algorithm and quite similar to the policy iterators discussed previously from DP and MC methods. The algorithm is depicted in procedural form (Algorithm 3) below which is taken from [Sutton & Barto 98].

Algorithm 3 SARSA

Initialise $Q(s, a)$ arbitrarily
Repeat (for each episode)
  Initialise s
  Choose a from s using policy derived from Q (e.g. $\epsilon - greedy$)
  Repeat (for each step of episode)
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g. $\epsilon - greedy$)
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'; a \leftarrow a';$
  until s is terminal

The principle item of note is that this approach approximates the optimal control policy directly instead of evaluating with a complete MDP model. This greatly enhances the domain of applicability yielding good policies across a wider variety of domains, as the complete environmental model is no longer needed. As before with dynamic programming (section 3.3.1) and Monte Carlo policy iteration methods (section 3.4.1), once a given policy has been evaluated, further improvements can be made to yield even better policies.

In addition the approach learns in an incremental online manner, where new information is immediately incorporated into the action value $Q(s, a)$ function estimates. This affords a marked improvement over previous methods particularly MC methods where episodic termination is the only way new information can be incorporated into the knowledge model. With SARSA, policies can be evaluated much faster which is extremely valuable when dealing with the real world or tasks which do not terminate i.e. there is no terminal state which when entered terminates the episode of learning.

The algorithm was first introduced by Rummery and Niranjan [Rummery & Niranjan 94] in a technical report in 1994. It has since been applied to numerous control problems such as the allocation of virtual servers to support an application with variable user requests [Tesauro *et al.* 06], virtual machine auto configuration [Rao *et al.* 09] and has even been applied in Robocup soccer keepaway [Stone *et al.* 05].

SARSA is considered an on-policy method because not only does the policy denote the actions chosen by the agent in the current state, it also governs the information backed up as part of the learning process. The policy being followed controls which actions are included in the value function approximations

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \tag{3.5}$$

for this reason SARSA is considered an *on-policy* approach. It is a logical next step then to consider the scenario where one backs up the max valued action $max_{a'}$ achievable from the next state s'. This is known as Q-learning.

3.4.2.2 Q-learning

If SARSA can be considered as the equivalent to policy iteration in dynamic programming then Q-learning is the equivalent of value iteration. Q-learning [Watkins 89] is a reinforcement

learning algorithm where the policy denotes only the actions chosen each time but does not govern which next action a' contributes to the value function estimates. This is an important distinction allowing Q-learners to directly update value functions as if they are following the greedy policy whilst actually following another. For this reason Q-learning is often referred to as an off-policy method.

Algorithm 4 below outlines the main steps involved in Q-learning. Many of the steps involved in the policy iteration algorithm (Algorithm 1) are repeated, one difference to note is in regards to the update rule $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$. The max next action $max_{a'}$ is chosen to be backed up each time as opposed to the action denoted by the policy ($\epsilon - greedy$).

Algorithm 4 Q-learning

Initialise $Q(s,a)$ arbitrarily
Repeat (for each episode)
  Initialise s
  Repeat (for each step of the episode)
    Choose a from s using policy derived from Q (e.g. $\epsilon - greedy$)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$
  until s is terminal

Q-learning was first introduced by Christopher Watkins in his seminal thesis in 1989, with a convergence proof [Watkins & Dayan 92] following in 1992. It has received much attention due to its low computational overhead for deriving policies in the absence of a complete model. Q-learning has been successfully deployed to model problems in a non-cooperative multi-agent scenario using general-sum stochastic games [Hu & Wellman 03], multi-agent systems where the agents have competing or interacting goals [Littman 94] and dynamic packet routing in computer networks [Boyan & Littman 94].

3.4.3 Reinforcement Learning Applications

In this thesis our focus is on the optimal allocation of computational resources in response fluctuations in user requests and performance. This section covers a number of application areas where reinforcement learning techniques have been successfully applied to automate control or find optimal solutions in complex domains.

In 1992 Gerald Tesauro working at the IBM Thomas J. Watson research centre developed an artificially intelligent agent capable of playing backgammon to international level. TD-Gammon [Tesauro 92] utilised a temporal difference learning algorithm known as TD(λ) [Sutton 88] to train a neural network via temporal difference learning. In contrast to the widely used supervised methods for training neural nets which require a complete set of inputs and the resultant output component each time, TD-gammon trains the neural network using only a reinforcement signal which is dependent upon the outcome of each game (win, lose, etc) and the complete sequence of board positions for the entire game. The TD agent discovers successful strategies through repeatedly playing games against itself and observing the outcome. With no prior knowledge the agent was capable of learning strategies equivalent to a human player at a strong intermediate level. However by including an additional set of handcrafted features, the agent dramatically improved to play backgammon at a strong masters level [Tesauro 94]. Tesauro's work demonstrated reinforcement learning's capability of operating over expansive state and action spaces learning solely from its experiences.

Robert Crites and Andrew Barto [Crites & Barto 96] researched the application of reinforcement learning theory to improve the problem of elevator dispatching in buildings. They simulated a 10 storey building with 4 elevator cars where passengers arrived at various floors according to a Poisson process. This problem represents a particularly complex domain as elevator systems operate in continuous state spaces and in continuous time as discrete event dynamic systems [Crites & Barto 96]. The approach involves utilising a number of Q-learning agents, each responsible for an individual car. They offer two possible reinforcement signals depending on whether one wishes to learn offline via a simulator or directly online. For online learning the waiting time for the first person in the queue (the wait time from when the call button is pressed) forms the basis of the reward signal. The second approach is to assume it is available via a simulator. Due to the intrinsic size of the state space the Q-values ($Q(s, a)$) are estimated using function approximation via a feed-forward neural network. The authors investigated training the neural network based on two distinct learning architectures. The first approach involves a parallel learning method where the agents share a single function approximator and learn from each others experience. The agents all approximate the same globally optimal policy. The second method is a decentralised learning approach where each agent maintains its own value function estimates via an individual neural network. Both architectures were compared against the best heuristic dispatching algorithms that were available at

the time. Their results showed that the reinforcement learning approaches surpassed the performance of the majority of the best known heuristic methods of the day across the range of problems tested.

Justin Boyan and Michael Littman of Carnegie Mellon University applied Q-learning to the problem of packet routing in computer networks [Boyan & Littman 94]. This problem is challenging as computer networks often comprise irregular topologies and present unpredictable usage patterns [Rudin 76, Tanenbaum 88]. The explicit goal of their learning approach is to learn a routing policy which balances minimising the number of hops each packet takes against the possibility of congestion along a particular route [Littman & Boyan 93]. Based on the experience with different routing policies a learning agent can assess the merits of separate individual routing policies given the current environmental state. Using a discrete event simulator to model the movement of packets in the network the authors compared the Q-learning approach to that of one based solely on shortest path schemes. The results demonstrated for all cases that the learning approach was able to sustain a greater amount of network traffic than that of the non-learning approach.

3.5 Statistical Learning

"*We are drowning in information, while starving for wisdom. The world henceforth will be run by synthesizers, people able to put together the right information at the right time, think critically about it, and make important choices wisely.* " *E.O. Wilson, Harvard University, 1998*

Complex problems such as robotic vision systems or navigation control, produce large quantities of data describing in detail many aspects of the agent environment. Often described as data rich but knowledge poor, the challenge involves determining how to accurately and concisely model the environment to a useful state given the limited internal resources available. Ideally the learned model needs to describe the underlying environment in sufficient detail to enable the learner to act appropriately and even optimally.

The mathematical field of statistics has for centuries been applying techniques to make logical inferences given the evidence or information available. Statistical learning provides mechanisms through which a learning agent can update their estimates over hypotheses given new and often changing experiences. Often these methods involve building a predictive system model through a simulator or training system and then applying the trained model on new or unseen objects [Hastie *et al.* 05]. This method of training a model when given a set of inputs and the correct set of outputs is known as supervised learning.

Supervised learning involves training an artificially intelligent agent by providing a sample set of data and the correct output value based on the input samples. A typical example would be training an agent to recognise an object such as a car in a collection of images. The training set would consist of multiple images some with cars and some without. For each image in the training set an external instructor informs the agent as to the content of the image. When presented with a new collection of unseen images the agent then classifies the images according to its trained model. Techniques such as neural networks [Pao 89], support vector machines [Joachims 98] and Bayesian networks [Heckerman 08] are all popular methods used widely in the domain.

Unsupervised learning methods do not strictly require a complete set of outputs for every input variable and instead attempt to cluster data based on its statistical similarity. Capable of inferring over large datasets, they have been extensively deployed to solve problems in areas as diverse as pattern recognition [Carpenter & Grossberg 88] and even the detection of road lanes from GPS data [Wagstaff *et al.* 01]. These techniques can also highlight non-intuitive relationships in the data and allow for the formation of new and interesting hypotheses. Methods such as k-means clustering [Hartigan & Wong 79] and hierarchical clustering [Murtagh 83] are prevalent in the area.

Guan et. al [Guan *et al.* 11] propose the use of supervised and unsupervised learning methods as a failure prediction mechanism for cloud environments. Their approach includes a supervised learning method which employs decision tree learning to make predictions about future failures. The unsupervised element uses an ensemble of Bayesian models, describing a variety of states including normal system execution and anomalous behaviour. Once the anomalies are labeled by system administrators the labeled examples are then available for classification purposes. Machine learning methods such as these can achieve high performance as they are

aided by human intervention, describing the correct outputs, allowing the classification process to correct its behaviour.

Techniques from machine learning and data mining often suffer inefficiencies when it comes to massive data sets and executing operations in parallel. Frameworks like MapReduce provide effective data processing capabilities when it comes to operating over large datasets but often losses in efficiency occur when similar mechanisms are applied to learning frameworks. Low et. al. [Low *et al.* 12] propose Distributed GraphLab, a framework which naturally expresses asynchronous, dynamic, graph-parallel computation while ensuring data consistency and achieving a high degree of performance in a parallel setting. Evaluations of GraphLab on Amazon EC2 demonstrated performance improvements of 1-2x over MapReduce approaches.

We approach the application areas of workflow scheduling and application scaling from a control and optimisation perspective, meaning that a learning system is capable of making decisions autonomously simply by observing the state of the environment. Whilst supervised methods would be applicable to these types of problems, there would be a strict requirement for extensive domain knowledge and expertise. Within these domains we feel that the inherent unpredictability means that it would be difficult to provide this expertise in a meaningful way and without this, the approach would have limited scope and application. Unsupervised methods commonly aim to detect patterns in large data sets and are frequently employed in data mining operations, where it is hoped valuable patterns and information can be discovered from the data, however for control tasks such as resource scaling or workflow scheduling it's applicability is limited given the precision required for decisions. For these reasons we have chosen a purely experiential learning method, which can learn without prior information allowing it learn control policies in a domain independent manner.

The goal of reinforcement learning methods is to approximate optimal control policies for stochastic processes such as MDPs. These are trial and error mechanisms which make decisions based on their expectations around the reward distributions. One important consideration is that approximating the true state values $V(s)$ is largely dependent on the number of times state s is visited. In fact the true values will only converge on the limit. In situations where the domain is huge and an approximate solution is considered satisfactory [Grzes & Kudenko 10] then model free methods prove extremely useful.

Statistical methods offer an alternative mechanism where approximations of the underlying process are determined using experiences of operating within the system. The goal of the learning process is to approximate the full posterior distribution over a given component of the missing model. By computing the posterior distribution conditioned upon the available experience an approximate model can be derived which is then used to solve the stochastic process via dynamic programming.

Prior to the development of model free learners, the Artificial Intelligence community focussed on devising model based solutions to solving these problems. The advantage of these model based techniques is that dynamic programming methods are guaranteed to find the optimal policy for any given model approximation. The only problem is that the learned model must be sufficiently accurate otherwise the ”optimal policy” produced by dynamic programming is not optimal with respect to the real system. Thus the focus of these methods is to try to estimate as closely as possible the underlying system dynamics to approximate the true system processes.

3.5.1 Bayesian Inference

Section 3.4.2 on temporal difference learning describes in detail the background theory behind model free learning methods with some of the prevalent approaches in the area such as Q-learning and SARSA discussed in detail in sections 3.4.2.1 and 3.4.2.2.

TD learning is useful for problem domains which contain large state spaces and where good approximations of the optimal policy are satisfactory [Strens 00]. The availability of a large number of learning trials is also a requirement to ensure the correct evaluation of discovered policies.

What if one were able to correctly estimate the missing underlying model of the stochastic process and then use this model in conjunction with dynamic programming to solve for optimal policies. Could this be a useful alternative to model free methods?

One approach to model learning involves implementing ideas from Bayesian inference and using as evidence the resultant outcome to various actions executed in the environment. Over time one can compute the likelyhood of a given event occurring based solely on past experiences.

Bayesian inference is a statistical method which provides a mathematical formula to determine how one should change one's existing beliefs in light of new evidence. In the general form for a given random variable X the posterior probability $P'(X = x|e)$, is the probability that X is equal to a value x given experience e. In the general case the posterior probability is computed using Bayes theorem

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{3.6}$$

which requires a single conditional probability $P(X|Y)$ and two unconditional probabilities $(P(Y), P(X))$ to compute the conditional posterior probability $P(Y|X)$ [Russell *et al.* 95].

Bayes rule can be better understood through a simple example. Suppose you have tested positive for a particular disease(the flu virus). What is the probability, having tested positive that you actually have the disease? This depends on a number of factors including the test procedure, i.e. its accuracy and sensitivity and on the background prior probability of an individual person in the population actually having the flu virus. When this information is available Bayes rule can powerfully predict the posterior probability governing the probability of having tested positive that you actually have the disease. The principle drawback with this approach is that certain information is required apriori in order to compute the posterior distribution. With many domains this is quite simply not available and must be built up iteratively in order to make successful approximations.

3.5.2 Current Research and Applications

Malcolm Strens [Strens 00] proposed an approach to represent the uncertainty about the model parameters when an explicit environmental model exists. The author demonstrated how to represent the full posterior density distribution over the model parameters by updating various metrics each time. For the complete list of metrics see [Strens 00]. A new hypothesis is generated at the beginning of each learning episode based on the currently computed posterior distribution over the model parameters. By greedily acting with respect to the full posterior over the process the approach was capable of finding optimal and near optimal policies very quickly. Simulating three separate learning challenges Strens compared his Bayesian DP against five other techniques including, Q-learning with a semi-uniform policy, Q-learning with a Boltzman policy, interval estimation Q-learning plus, Bayes value of perfect information with mixture updating and Dynamic programming with a heuristic policy. The results

showed that the Bayesian DP method outperformed all the other methods for the demonstrated problems.

Reinforcement learning architectures learn policies through trial and error and update value function estimates by incorporating the rewards observed after the actions are executed. One of the disadvantages of this approach is that these techniques do not learn the underlying dynamics of their environment i.e. what causes what [Sutton 92], instead they are concerned only with what to do (policy) whilst in a particular scenario. They do not have a knowledge model with which they can use to predict how the environment will respond to their actions. If one does not possess a knowledge model a simple change in a target goal can have drastic effects, requiring exploratory processes again to determine the new optimal policy. Dyna architectures solve this for reinforcement learners by including a real-world internal model of the environment [Sutton 90, Whitehead 89, Werbos 87]. This model is updated each time an observation is made, where given state s and action a it outputs the predicted reward r and next state s'. This simple extension allows the learner to gain valuable knowledge about the underlying dynamics of the system for which it is trying to optimally control. The internal model approximations are dependent on the observability of the system itself with solutions for partially observable domains still a open challenge [Sutton 90]. Methods from supervised learning have also been applied successfully to update a model using training examples from observations of the environment [Sutton 92].

Andrew Moore and Christopher Atkeson propose prioritsed sweeping [Moore & Atkeson 93] which combines temporal difference one step methods with classical methods. One of the drawbacks with the classical approaches such methods from dynamic programming is that they can't handle large state spaces and can't be applied in a real time manner, as they require a complete in order to solve for policies. They performed their on a number of different control problems and improved the speed of convergence as well.

3.6 Methods to Reduce Convergence Time

Previous sections have discussed in detail numerous methods aimed at discovering policies for stochastic processes, including both model based and model free learning mechanisms. One

of the greatest challenges facing the different learning techniques is the large amounts of experience required to learn optimal policies for all but the simplest of domains. This results in certain problems proving difficult to solve, even at times proving to be computationally intractable. Commonly known as the curse of dimensionality, it refers to the exponential increase in the problem size and complexity with each additional state-action variable added.

Solving for problems which contain large state & action spaces has received much attention by the research community in recent years [Sutton *et al.* 99, Melo *et al.* 08, Baird & *et al.* 95] and still remains an open challenge and active research area. Since so many problems of value contain such large dimensional space requirements, techniques limited to small state-action spaces have restricted value for the majority of real world problems. This dramatically limits reinforcement learning's usefulness for a wide variety of problems as it generally requires multiple sweeps of the state space to learn good policies.

This section details approaches capable of reducing the time it takes to converge to optimal or near optimal policies for a given learning task. We describe two approaches aimed at achieving this end, parallel learning methods and function approximation. For each approach we detail the relevant literature within the area.

3.6.1 Parallel Learning

Parallel learning involves multiple agents learning in parallel either on the same task or on a sub task of the problem, with the solution derived through aggregating the outcome of the individual subtasks. The process is facilitated through information sharing mechanisms which allow agents to update their knowledge by integrating the experiences of others into their knowledge model. With advances in parallel computing [Kasahara & Narita 84, Seiffert 04], multi-core processing [Urbanek 09] and cloud computing [Ekanayake & Fox 10], solving large computational problems in this manner has become increasingly popular and considerably cheaper. Additionally, improved software programming models such as MapReduce [Dean & Ghemawat 08], has rapidly expanded the popularity of distributed parallel processing in server clusters, due to the simplification and abstraction of the process away from the developers. Utilising the message passing interface, data and information can be transferred easily between computing nodes once the task processing has completed.

Stochastic processes by their nature ensure that no two agents will share the same experience even when learning on the same task in the same environment. Thus agents will explore different parts of the state space independently of each other. The benefit of knowledge sharing with respect to these types of problems is that each individual agent no longer needs to sample every aspect of its environment and can instead learn from other agent's experiences. This reduces its convergence time significantly even when deployed simultaneously with no prior knowledge. Parallel techniques have been successfully deployed in a number of areas such as artificial neural networks [Kramer & Sangiovanni-Vincentelli 89] and classification/regression tree learning in supervised learning [Panda *et al.* 09].

The technique involves agents sharing value or model estimates to reduce convergence times. One of the key challenges with knowledge sharing approaches is to ensure that only necessary information is shared amongst the respective agents. Reducing the amount of data transmitted between processes and agents is an important goal of parallel computational problems. Transmitting large amounts of data between agents and processes can result in costly bandwidth usage and sometimes result in delays in the learning process. A number of studies have recently examined scheduling tasks to maximise workflow completion times whilst concurrently minimising data transmission and cost [Yu *et al.* 05, Yu *et al.* 08]. Additionally it is intuitive to assume that only experience which deviates from the global knowledge view needs to be transmitted, as one usually doesn't need to broadcast something which everyone already knows. uThus approaches should be cognisant of this fact and aim to reduce the amount of information flow. The next section advances this discussion by describing both model based and model free learning approaches to parallel reinforcement learning.

3.6.1.1 Sharing Value Estimates

Model free learners attempt to directly approximate the control policy through trial and error, building up value estimations of the various actions and the expected reward achievable. Thus the goal of these approaches is to facilitate the sharing of value function estimates amongst the agents in an attempt to speedup the discovery of optimal or near optimal policies.

R. Matthew Kretchmar [Kretchmar 02] first proposed the sharing of value function estimates

in this manner and demonstrated empirically the convergence speedups made possible. Utilising multi-armed bandit tasks he deployed varying numbers of agents, learning in parallel on the same task. One of the principle contributions of Kretchmars work is the mechanism by which agents integrate knowledge from other agents. Rather than each agent integrating the global view of all other agents into their estimates, they instead maintain a separate value estimations based on their own experience $\widetilde{Q}_t(a)$. The global $\widehat{Q}_t(a)$ estimate represents the value estimates of all the other agents learning on the same task. By aggregating their experience with the respective global estimate, a weighted value function can be calculated $Q_t(a) = \frac{\widetilde{Q}_t(a)\times\widetilde{K}_a+\widehat{Q}_t(a)\times\widehat{K}_a}{\widetilde{K}_a\times\widehat{K}_a}$. The value of $K_t(a)$ is the number of trials where action a has been chosen and represents the experience associated with the estimation for $Q_t(a)$. Updating value estimates in this manner ensures that the agent's experience is based on actual action selections and not the subsequent integration of previous accumulations. Thus the possibility of agents sharing estimates based on the same learning trials and biasing this over actual experience is removed. Kretchmar demonstrated an almost linear speedup in performance, in conjunction with the increase in parallel agents for the given problem domain.

Continuing from this work, Matthew Grounds and Daniel Kudenko [Grounds & Kudenko 08] developed a model free parallel parallel learning approach where agents share value function representations described as neural weights of a linear function approximator. Each agent learns independently with a simulator on the same learning task transmitting information between each other periodically. The approach utilises the reinforcement learning technique SARSA(λ) to generate policies. One of their principle advancements over Kretchmar was to reduce the amount of unnecessary information being passed between the agents. Changes in weights are only communicated if the change in their value is significantly different from the previous episode. They authors deployed their approach on a Beowulf cluster of machines using the Message Passing Interface (MPI) to communicate information between agents. They empirically evaluated their approach on two benchmark reinforcement learning problems, the *mountain car task* and the *pole balancing task* detailed in the Sutton and Barto publication [Sutton 88]. Their findings concur with Kretchmar, with parallel learning achieving an almost linear speedup in convergence as the number of agents learning in parallel increased.

3.6.1.2 Sharing of Model Estimates

By Sharing model information amongst each other, agents can speed up the approximation of the underlying hidden model. In contrast with the previous section, where value function estimates are transmitted between learning agents, this section discusses sharing statistical values which directly approximate a hidden model.

It is important to note that this approach is only feasible for statistical learning methods such as those described in section 3.5. The approach involves the transmission of statistical observations between agents learning concurrently on the same learning task. The agents require a mechanism through which they can compute the relative distance between their estimates of 'ground truth' and of the global consensus (all other agents). In addition this mechanism should also reduce the information flow between agents as the approximations begin to converge.

Li and Schuurmans [Li & Schuurmans 12] demonstrate an approach to parallelising the classical dynamic programming methods of *policy evaluation*, *policy iteration* and *off-policy updates*. In addition the authors also provide algorithms for executing the model free methods of *temporal difference learning* and *gradient TD learning*. A number of algorithms are presented which successfully deploy reinforcement learning tasks using the MapReduce programming paradigm to distribute the learning tasks amongst a cluster of machines. MapReduce is a programming specification of which both Google [Dean & Ghemawat 08] and Hadoop have proposed specifications. It is designed for processing large scale computational tasks across large server clusters and computational clouds.

The key advantage of using parallel agent learning is that the convergence times in large problem domains can be significantly reduced, often by an order of magnitude equivalent to the number of agents deployed. Whilst parallel methods are viable approaches towards extending reinforcement learning algorithms to operate in domains with large state-action spaces, an important alternative approach is to use methods from supervised learning, applying regression analysis on the state-action space.

3.6.2 Function Approximation

Function approximation is a mechanism through which one can learn the explicit functional form of a given underlying function using a data set generated from that function. It relies on applying mathematical and machine learning techniques to approximate the underlying function. Once the functional form $f(x)$ is known then predictions can be made about future events.

From a learning perspective, function approximation is often deployed to calculate action value functions ($Q(s, a)$) for large reinforcement learning tasks [Tsitsiklis & Van Roy 97]. In these tasks, representing the value functions in tabular form becomes unwieldily as the number of states and actions grows to very large numbers. Once the values being approximated consist of a set of real numbers $\Re$ then regression, line fitting and interpolation techniques are applicable to derive the approximations.

One of benefits of applying function approximation such as regression models to approximate value function estimates, is their ability to generalise over states and actions not yet visited. In simple domains regression curves can be approximated using least squares [Geladi & Kowalski 86] or interpolation [Chow & Lin 71]. However for more complex domains where one attempts to approximate multivariate functions using large data sets, methods from supervised learning have become the de-facto standard [Friedman 94].

Function approximation does not employ a tabular storage mechanism with rows and columns for each respective state-action pair. Instead the value functions at time t, V_t are represented by parameter vectors $\vec{\theta_t}$, with the value V_t varying from time-step to time-step only as $\vec{\theta_t}$ varies [Sutton 88]. In the context of training a neural network, $\vec{\theta_t}$ would be the vector of connection weights [Sutton 88]. Thus when a single state observation is made, this change generalises to affect the values across a large number of states, speeding up the learning process without having to visit every state.

3.7 Evolutionary Computation

Mathematical optimisation refers to a process, where from amongst a set of possible alternatives a single element or solution is chosen which is considered to be the optimal or best solution. To differentiate amongst the available solutions, criteria must be established which places an ordering over the solution space governing their relative merit with regard to the specified criteria and each other. Thus choosing the optimal solution over the solution space is trivial once the ranking is in place. Optimisation problems span a broad range of domains including computer science, operations research and combinatorial mathematics [Aarts & Lenstra 97]. Within these domains, finding optimal solutions under varying constraints presents one of the hottest research topics of today.

An important constraint of any optimisation approach is its ability to discover and evaluate solutions under a realistic time setting. Algorithmic time complexity measures the quantity of time taken by an algorithm to run as a function of the length of the input to the algorithm [Sipser 06]. Time complexity analysis provides a means of classification for a given optimisation problem, grouping it into P, NP, NP-complete and NP-hard problems. P problems are considered solvable within polynomial time as their running time is a function of some polynomial and can be computed deterministically in a sequential manner. Non-deterministic Polynomials (NP) cannot be computed in sequential or step-by-step manner and instead require a non-deterministic method to yield a polynomial time solution. NP-complete and NP-hard describe a collection of optimisation problems for which there exists no known algorithms which can discover the solution within polynomial time even using non-deterministic methods. In the case of NP-complete problems a given solution can be verified quickly but there are no known efficient methods to find solutions in first place.

A suite of approaches which has gained prevalence as viable alternatives to classical search procedures when dealing with NP-complete and NP-hard problems are evolutionary algorithms. Evolutionary computation encompasses a suite of search and optimisation techniques inspired by the biological processes of evolution. These techniques are commonly classified as metaheuristic approaches, meaning they are designed to find good solutions to difficult optimisation problems [Bianchi *et al.* 09] in a relatively quick time. The techniques cover a

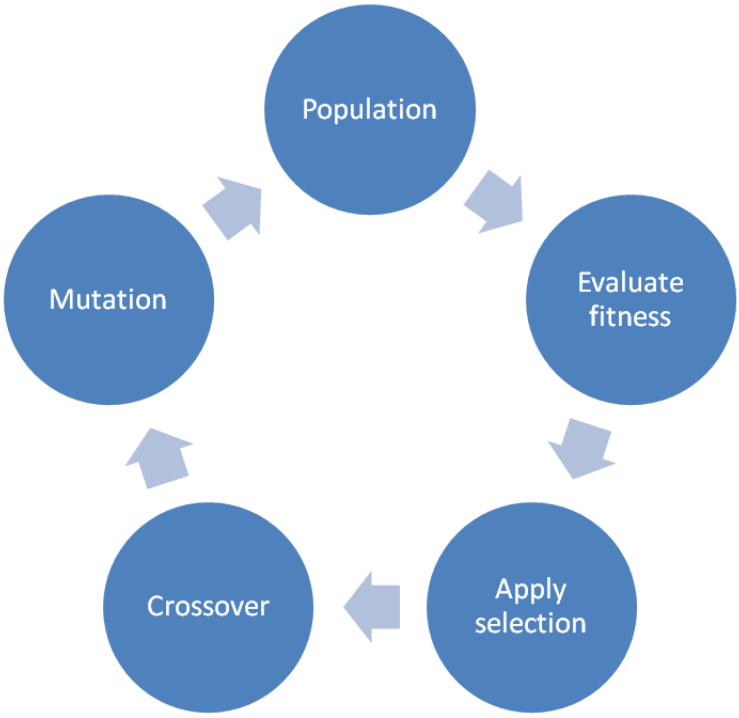


Figure 3.2: Genetic Algorithm

broad spectrum ranging from the evolutionary approaches such as Genetic Algorithms [Holland 92] and Genetic Programming [Koza *et al.* 99], to the swarm intelligence techniques of Ant Colony Optimisation [Colorni *et al.* 91] and Particle Swarm Optimisation [Kennedy & Eberhart 95, Liu *et al.* 11]. The efficacy of evolutionary approaches towards the determination of good solutions to NP problems (complete and hard) have been extensively documented in the literature [P. Jog & Gucht 89, Jiao & Wang 00]

3.7.1 Genetic Algorithms

Genetic Algorithms (GA) are stochastic search and optimization techniques based on evolution. The term Genetic Algorithm was first proposed by John Holland in 1975 [Reeves 03] in *Adaption in Natural and Artificial Systems* [Holland 92]. Inspired by the biological processes of natural selection and evolution, GAs attempt to evolve a solution to a given problem over successive generations rather than exhaustively searching the solution space. They are classified as guided random search mechanisms [Araújo *et al.* 01].

Figure 3.2 describes the different stages of a standard genetic algorithm. In its simplest form,

the genetic algorithm evaluates a set of possible solutions which are encoded as bit strings. The bit string representations are the equivalent of chromosomes in biological evolution. The solution set is called the population and its size is fixed. At the fitness evaluation stage of the process, each solution is given a fitness score and is ranked relative to the other solutions in the population. This places a hierarchical ordering over the solutions which facilitates the selection process to generate offspring. It is hoped that better solutions can be found by 'breeding' the fittest individuals with each other.

In an evolutionary context solutions are ranked according to their fitness. The concept of fitness is borrowed from Darwinian evolution where it is deemed that only the fittest individuals or those most suited for their respective environment survive to procreate. Genetic algorithms adopt the same principle, where reproduction procedures preference only the fittest solutions.

Algorithm 5 Genetic Algorithm

Initialise generation 0
$k = 0$
G_k - Population randomly generated
Evaluate G_k
Compute *fitness(i)* for each $i \in G_k$
repeat
  Create generation $k + 1$
    Elitism
      Choose top $\%$ of G_k and insert directly into G_{k+1}
    Selection
      Choose individuals i and j using the chosen selection strategy (*roulette/tournament*)
    Crossover
      Apply single point crossover to i and j; insert offspring into G_{k+1}
    Mutation
      Select n offspring randomly from G_{k+1}; invert a single bit of their bit string
    Evaluation G_{k+1}
      Compute $fitness(i)$ for each $i \in G_{k+1}$
    Increment
      $k = k + 1$
until individual solution i reaches required level of fitness
return the fittest individual i from G_k

Algorithm 5 depicts the process in procedural form outlining the various steps involved. In this depiction an initial population of random solutions is generated. Then the solutions are evaluated using the fitness function f. The procedure inside the loop describes the process of creating generation G_{k+1} using elitism, selection, crossover and mutation.

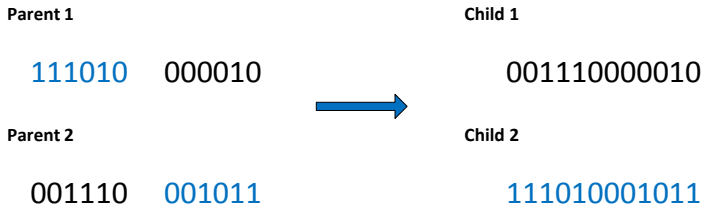


Figure 3.3: Crossover operation applied to two chromosomes

The first stage in the process is the application of elitism to the previous population G_k. Elitism involves taking the top n percentage of individuals from the generation hence and adding them straight into the next generation G_{k+1} without applying the genetic processes of crossover and mutation. Elitism ensures that the best solutions from the previous generation do not get lost in the evolutionary process. The value for elitism is usually restricted to a low integer value. This balances the need to exploit good solutions with the exploration of better ones, through the evolutionary process.

The next stage involves applying the crossover mechanism on the selected parents to produce offspring. Algorithm 5 depicts a single point crossover mechanism which is shown in figure 3.3. The procedure involves the selection of a single point in the bit string via a random process. This point is indicated by the space in figure 3.3. The first child (Child 1) is created by combining the left hand side of Parent 1 (bits before the point) with the right hand side of Parent 2. The second child (Child 2) is created by combining the remaining bits i.e. the right hand side of Parent 1 with the left hand side of Parent 2. It is hoped that these children will prove to be even better solutions than their parents. But due to the randomness inherent in the approach this of course is not guaranteed!

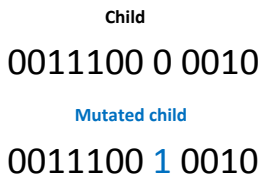


Figure 3.4: Single point crossover of two bit strings

Figure 3.4 details the mutation process where a single bit in the chromosome has its value inverted. The child solutions are selected for mutation based on a random sampling process. Once a child has been selected, a random location on their bit string is chosen and its value is changed by inverting it to either a 0 or 1. Figure 3.4 depicts the mutation of the 8th bit,

given that the current value at this position is 0, the application of mutation changes it to a 1. Mutation generally has a low probability of occurrence within the population. This parameter is also an experimental parameter and can be modified depending problem size and complexity.

The selection of the parents is an integral part of the process and can define the efficacy of the solution for the given problem domain. Quite a number of different solutions have been proposed over the years and a good comparison of these can be found here [Goldberg & Deb 91]. Two of the most frequently used are roulette wheel selection and tournament based selection [Goldberg & Deb 91]. Roulette wheel selection is a proportional fitness selection mechanism, where an individual's probability of being selected is proportional to its fitness in the population. The following equation determines the likelyhood of an individual being selected by roulette wheel selection.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3.7}$$

The equation (3.7) divides the fitness of a single individual by the fitness of all the individuals in the population. This normalises the fitness landscape into a range between 0 and 1 which allows for a probabilistically driven proportional selection process.

The reason it is called roulette wheel selection is due to its similarity with the game of roulette played in casinos. Analogous to the game, each of the individuals in the population occupies a slot on the wheel, the size of which is proportional to the individuals fitness. Thus there is a greater likelyhood that the ball will land on the fittest individuals.

Tournament based selection [Miller & Goldberg 95] differs slightly from roulette wheel and involves randomly selecting individuals from the population to take part in tournaments. Each tournament involves a finite number of individuals and the fittest individual from the tournament is considered the winner. This individual then produces offspring through the processes of crossover and mutation.

The final stage of the process involves evaluating the new population G_{k+1} of individuals and ranking them once more in terms of their fitness. The process continues for a predetermined number of generations or until a solution of the fitness value is found.

The application of the genetic operations of crossover and mutation between two individuals in the population is not deterministic. It is probabilistically driven and this factor can be adjusted

according to problems space or even the lack of diversity in the population [McGinley *et al.* 11].

3.7.2 Applications

The application of Genetic Algorithms to search and optimisation problems are as varied in type as they are numerous in quantity. In this section we discuss a selection of applications to some of the more interesting domains.

H.K. Ozturk and O.E. Canyurt extensively applied genetic algorithms variations to the prediction and estimation of a number of different quantities including electricity [Ozturk *et al.* 05] and oil demand [Canyurt & Öztürk 06] for Turkey. The goal was to make future predictions regarding the consumption of energy based on the figures for Gross National Product (GNP). They included a number of additional parameters in their evolutionary procedure and chose the approach which provided the best fit given their data observations. They were happy to conclude that the approach was suitable to be used to make predictions about the future energy consumption in the country.

Numerous authors have examined the application of genetic algorithms to job shop scheduling [Nakano & Yamada 91,Della Croce *et al.* 95,Fang *et al.* 93], an NP-hard problem [Sotskov & Shakhlevich 95]. The job shop scheduling problem is a commonly examined optimisation problem in computer science and operations research and the majority of approaches to solving them involve the branch and bound algorithm [Fisher & Thompson 63]. It involves allocating a collection of individual jobs of various sizes onto a finite number of identical machines. The optimsiation goal is to allocate jobs to machines in order to minimise the overall makespan, i.e. the total time taken to complete all the jobs. Goncalves et. al. [Gonçalves *et al.* 05] demonstrated a hybrid genetic algorithm approach where multiple valid schedules are generated using a genetic algorithm. A local heuristic search mechanism is then applied over these schedules to improve them even further. The authors compared the performance of their approach against 12 other algorithms tested on a set of 43 standard instances taken from the literature. Their results showed that their hybrid genetic algorithm found the optimal or near-optimal solution to all the standard instances tested. Their approach demonstrated an average relative deviation of just 0.39% from the best known solution to the instance. Yamada and

Nakano [Yamada & Nakano 92] demonstrated a genetic algorithm approach useful for large scale job shop scheduling problems. The approach involves the application of the genetic operators of crossover and mutation on the classic Giffler & Thompson algorithm for generating active schedules. Their experiments demonstrated the effectiveness of their approach on the difficult Muth & Thompsons 10×10 and 20×20 benchmark.

An interesting application of genetic algorithms is to find optimal solutions for the Vehicle Routing Problem (VRP) [Laporte 92]. Considered one of the most challenging in the broader spectrum of combinatorial optimisation problems [Toth & Vigo 87], the problem involves the distribution of goods from a central depot to customers who have placed orders for those goods. A finite number of vehicles are available to transport goods with each vehicle constrained by the distance they can travel and the weight of goods they can carry. Baker and Ayechew [Baker & Ayechew 03] demonstrated the performance of a classical genetic algorithm solution and a hybrid solution, comparing their results against the top performing approaches of simulated annealing [Osman 93] and tabu search [Gendreau *et al.* 94]. There results demonstrated that the classical genetic algorithm approach on its own could not find solutions as good as Tabu search. However when combined with a simple neighbourhood search mechanism the approach found solutions which contained distances that were only 0.5% above the best known results on average.

Chapter 4

Cloud Data Models for Empirical Evaluations

4.1 Research Methodology

Chapters 2 and 3 describe in detail both the application area of Cloud Computing and the optimisation techniques which we extend and apply to the specific problems of scheduling and allocating resources to tasks in an optimal manner. However, in order to perform extensive empirical analysis of our proposed techniques detailed in Chapters 5, 6 and 7, we need to define our evaluative methods and practices. Each of the hypotheses detailed in these chapters are evaluated through a simulation environment. To conduct our experiments we utilise existing simulation tools and develop techniques of our own. The following details the approaches used:

1. To evaluate our first hypothesis which posits that observations of environmental state can be used when scheduling workflow tasks to resources, we use Cloudsim, a simulator developed by the CLOUDS group at the University of Melbourne [Calheiros *et al.* 09]. Cloudsim is a discrete event simulation tool which allows you to model a typical IaaS cloud scenario on a single processor. The experiments are detailed in Chapter 5 of this dissertation.

2. Our second hypothesis maintains that employing multiple learners will reduce convergence times for application scaling in IaaS clouds. To simulate an application scaling example we build a custom simulation program in Matlab, utilising standard functions and capable of defining specific resources and applying performance data to them. This is detailed in Chapter 6.

3. The final hypothesis which puts forward the concept that the Kullback-Liebler divergence will render the sharing of probability density estimates possible amongst multiple agent learners employing a form of parallel Bayesian reinforcement learning. From an empirical viewpoint we simulate a benchmark problem common for evaluating algorithms in the domain of reinforcement learning, known as the GridWorld problem. Secondly we reuse the custom simulation tool developed for analysing the algorithms from the previous chapter.

With regard to the problems of application scaling and workflow scheduling we simulate real world executions by collecting data from two separate sources, microbenchmarking cloud instances using existing 3rd party tools and performance data provided by Cedexis the cloud deployment strategists. Simulation environments allow for experimental repeatability where the performance of the approaches can be consistently evaluated in a controlled manner. This ensures that anomalies which may be present in a real live environment cannot skew or interfere with the results, enabling greater comparisons between approaches.

This chapter details our methodology in this regard, how we collect the data which is used in our experiments, the configuration of these experiments and our findings.

- **Disk I/O**: **Chapters 5** and **6** analyse learning architectures suitable to optimising workflow scheduling and application scaling under variable performance conditions. To this end we carry out a microbenchmark analysis on a number of different instance types of EC2 virtual machines. In our experiments we employ a Gaussian sampling mechanism to ensure an equitable selection process.

- **Application response time**: **Chapter 7** focuses on optimising application performance to a response time measure. We sample real response time data provided by Cedexis from a global perspective.

4.2 Data Collection

In order to accurately evaluate the performance of our proposed learning architectures we generate a number of data traces focussing on two separate areas *Disk I/O* and *application response time*. The data we gather comes from a range of instance types, spread out geographically across Amazon's vast public cloud network. All of our simulations are based on performance analysis conducted on Amazon EC2 and the corresponding regions within the IaaS provider. Although a more comprehensive exploration of the performance variation of other cloud providers would be of research value and could possibly enable us to evaluate our hypotheses scaling applications and resources between clouds, it is currently outside the scope of this research. EC2 instance evaluation is conducted from both a geographical and configuration viewpoint, meaning we examine deployed instances of different sizes (w.r.t CPU, RAM, Disk) and in different locations (EC2 regions). With this we can generate the necessary data, to model a practical environment suitable for our experiments, without incurring the likely substantial costs involved in executing on live instances. It also provides a simulation environment within which we can repeat experiments in a controlled manner to analyse and evaluate our hypotheses from a number of perspectives.

Table 4.1: Instance types and costs for US-East Virginia

Instance Type	**Memory**	**ECUs**	**Disk**	**Cost** (per/hr)	**I/O Performance**
m1.small	1.7GB	1	160GB	$0.06	Moderate
c1.medium	1.7GB	5	350GB	$0.12	Moderate
m1.large	7.5GB	4	840GB	$0.24	Moderate
m1.xlarge	15 GB	8	1680GB	$0.48	High

Table 4.1 lists the current on demand instance prices for a selection of different VM instances (Linux in US East N. Virginia) and their associated properties currently supported by Amazon EC2. Full details of the range of instances offered including detailed specifications can be found on the Amazon EC2 website [Amazon 13]. As previously discussed in Chapters 2 & 3 the nature of shared virtualisation instances leads towards performance unpredictability as the underlying CPU, RAM and disk are shared amongst the VMs residing on the physical host. Whilst hypervisors are pretty good at sharing the CPU and memory resources amongst respective VMs, the same cannot be said for disk and network bandwidth. To address this Amazon support a number of type of instance types and classify them according to their projected I/O

performance. The classifications are *Very Low*, *Low*, *Moderate*, *High* and *10 Gigabit*. This is a qualitative measure defining a general classification over the various instances but for obvious reasons they cannot ensure quantitative guarantees over actual throughput and data rates.

Applications/services deployed on these instances will observe variable performance levels depending on the I/O classification of the instance, assuming they are I/O intensive applications or require certain levels of throughput for data operations. Thus any scaling or scheduling solution, trying to allocate resources to suppose said applications and service should be cognisant of the instance type and what the performance ought to be. The next section describes our data gathering process using micro-benchmarking tools to measure the performance of the VM instances.

4.3 Micro-benchmarking Amazon Elastic Compute Cloud

To evaluate disk I/O performance on the Amazon EC2 cloud, we conducted a series of micro-benchmarks on a number of EC2 instances to demonstrate I/O performance variability exhibited by storage volumes on IaaS clouds. We used the filebench[1] benchmarking utility to demonstrate sequential/random read/write performance on three separate instances (m1.small, c1.medium and m1.large). Filebench is a multi threaded, cross-platform benchmarking tool, capable of running an array of synthetic workloads designed to evaluate disk I/O analysis. The results displayed here highlight the variability deployed applications will observe in relation to I/O performance. Further analyses of performance [Iosup *et al.* 11] and performance interference [Pu *et al.* 10, Nadgowda & Sion 10] on I/O workloads have previously been published.

Each VM instance on Amazon EC2 can support two different types of storage volume. Instance based or ephemeral storage are storage volumes located within the physical host and shared amongst the resident VMs. This is a typical SATA hard drive installed in the physical machine which each VM has shared access to. Elastic Block Storage (EBS) volumes are network attached storage devices which are connected via a 1Gbps Ethernet connection. Since the VMs have no observability over each other, they cannot coordinate or schedule their data read and writes to optimise disk access. Striping disks using a RAID 0 configuration can improve throughputs and data rates as files are split across multiple machines. However we assume a

[1] http://sourceforge.net/projects/filebench/

standard configuration block and ephemeral storage mirroring the default setup on EC2.

In testing disk I/O performance on EC2, we selected four workload profiles to evaluate the random read/write and sequential read/write performance for an m1.small, c1.medium and m1.large instance on the Amazon EC2 cloud. For each experiment we created two instances of each type in a separate availability zone in the US-East Virginia data centre. The experiments began on a Tuesday evening at $19 : 00$ UTC and ran for a total of 24 hours. We chose a midweek starting point in order to emulate as closely as possible the variability which would occur during an average working week. The total number of experiments ran over the 24 hours was 1152. The following four workload profiles were designed to evaluate each instance type:

- **Sequential Read**. This workload profile evaluates large sequential file reads. For this experiment the file size was set to be larger than the allocated RAM, 6 Gb in the case of the m1.small/c1.medium instances and 10 Gb in the case of the m1.large instance. This eliminated possible interference due to memory caching. To get a true reflection of the underlying performance, caching was disabled, the iosize was set to 1 MB, and single threaded execution. The experiment was ran for 20 mins, for both EBS and Ephemeral storage each hour, allowing for a steady state evaluation of performance.

- **Sequential Write** This workload profile evaluates large sequential file writes. The individual file write sizes were set to 1 MB each. Caching was again disabled, with syncing enabled. This was executed single threaded. The file sizes generated through writing were 6 Gb for m1.small,c1.medium and 10 Gb for m1.large.

- **Random Read** This workload profile evaluates random file read performance. Caching was disabled, individual read sizes (iosize) was set to 2K, with single threading. Each run generated 128 MB of read data.

- **Random Write** This workload profile evaluates random write performance. Caching was disabled, synchronisation was enabled. The file sizes were set to be larger than the available RAM at 6 GB for m1.small, c1.medium instances and 10 GB for m1.large instances.

4.3.1 Disk I/O Performance

Chapter 5 describes an approach for the optimal scheduling of workflows on IaaS clouds. This research is primarily concerned with scheduling data intensive workflows. Sharing hard disk volumes between VMs in a virtual environment is volatile and it's difficult to guarantee performance at any given instance in time. Since data intensive workflows involve large data operations and transfers between tasks, understanding the variations in the performance of the virtual disk volumes will be critical to any perceived optimal scheduling solution.

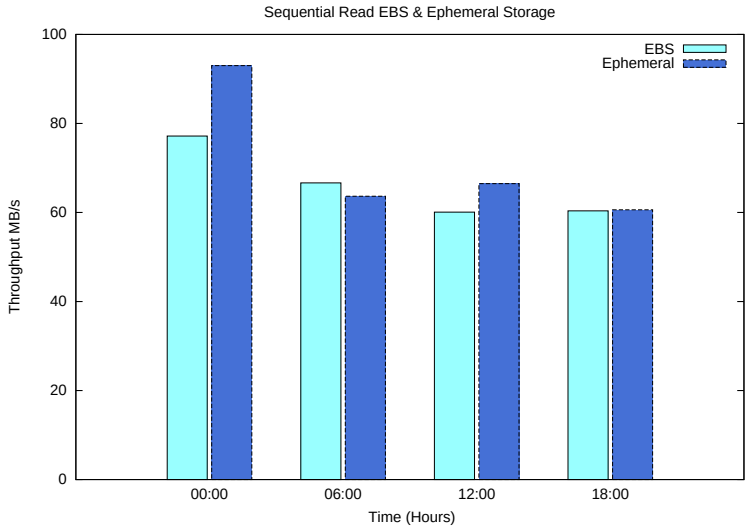


Figure 4.1: Sequential read EBS & Ephemeral storage volumes, c1.medium

We evaluate both Elastic Block Storage (EBS) volumes and Ephemeral (Eph) storage on Amazon's EC2. Figure 4.1 depicts the sequential read performance of both the EBS and Eph volumes over a 24hr period. Sequential reads involve reading a sequence of contiguous blocks of data residing on adjacent tracks of the disk. For both the EBS and Eph volumes significant variation in throughput is observed throughout the course of the day. For the local Eph storage, twelve midnight has an approximately 50% greater throughput than at 12 midday. It's clear that executing data intensive operations at 12 midnight will offer higher performance than at 12 midday.

Figure 4.2 details the sequential write performance for the $c1.medium$ instance over the 24 hour period. Sequential writes involve storing each consecutive block of data beside the previous in a sequential manner. This is how large files are usually written to disks. The hourly deviations are clearly seem here again particularly with respect to the local storage devices.

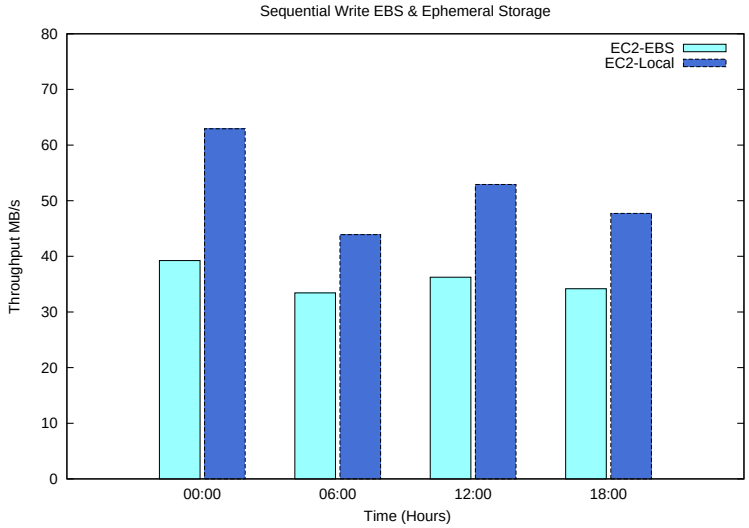


Figure 4.2: Sequential write EBS & Ephemeral storage volumes, c1.medium

The EBS volumes remain rather fixed throughout, which mirrors the observations in figure 4.1. We do not feel that multi-tenancy effects EBS volumes to a lesser extent but instead this may as a result of a network related bandwidth cap. The Eph volumes are not bound by network conditions as they reside within the physical host machine.

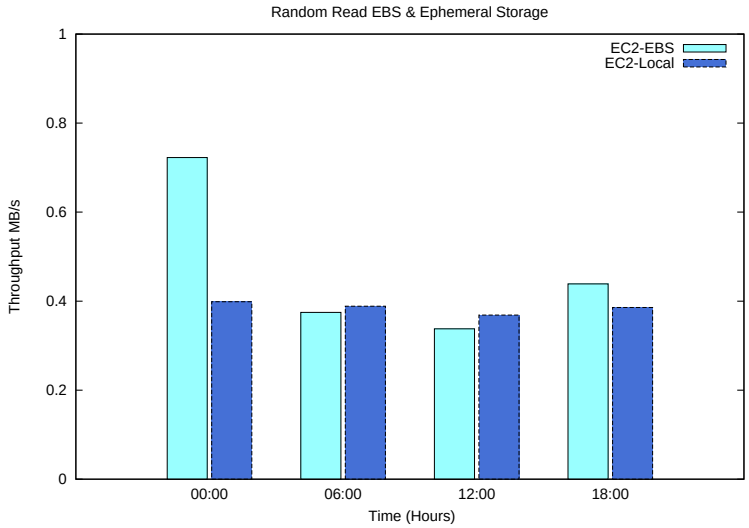


Figure 4.3: Random read EBS & Ephemeral storage volumes, c1.medium

Figure 4.3 depicts random read performance for the same instance $(c1.medium)$. Random reads involve reading blocks of data from different physical locations on the disk. It is particularly susceptible to latency as the arm has to physically position the head over the relevant track/sector on the disk in order to read the data. Multiple consecutive random reads will

usually have a low throughput for this reason. The local storage displays much less variation in throughput throughout the experiment, displaying relatively little difference with respect to different times of the day. It's clear that the EBS volumes are much more temporally effected, showing better performance at twelve midnight than during the middle of the day.

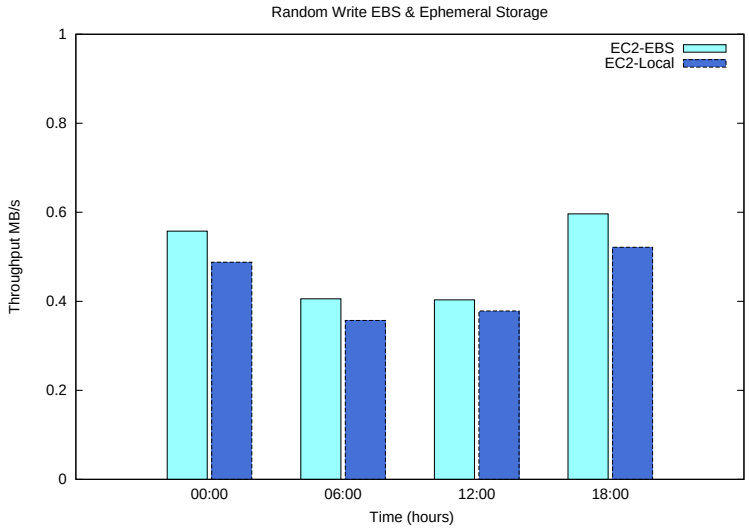
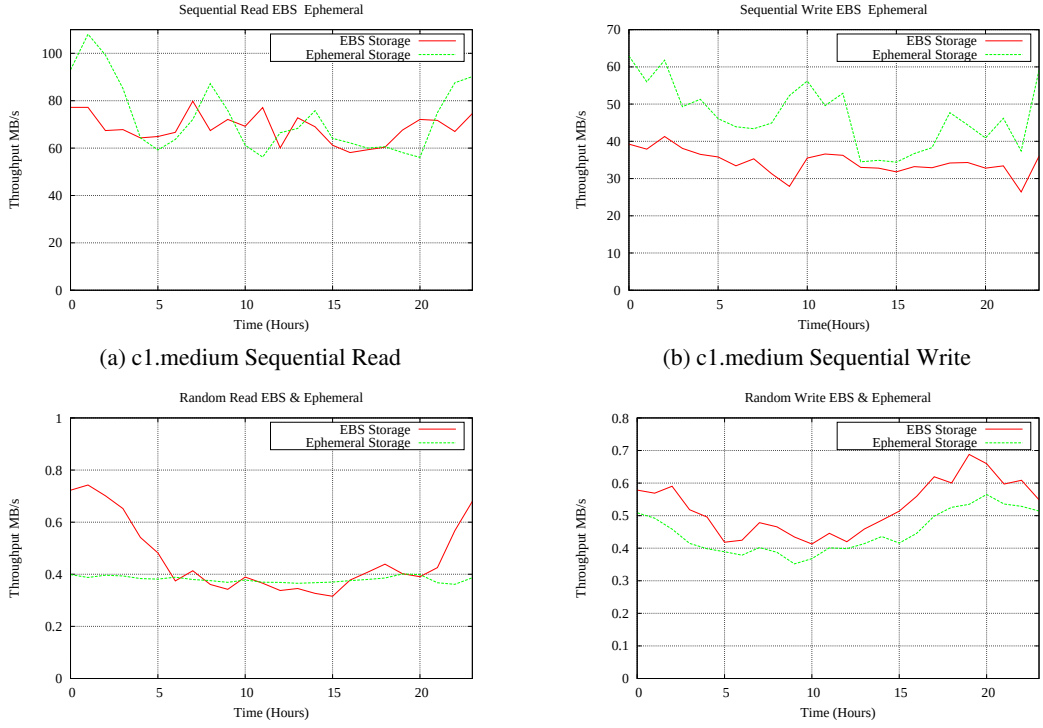


Figure 4.4: Random write EBS & Ephemeral storage volumes, c1.medium

Finally figure 4.4 details random write performance for both EBS and Eph volumes on EC2. Random write performance does show greater variations in throughput throughout the course of the day than random read, figure 4.3. However variations are still relatively small when compared to figures 4.1 and 4.2 for sequential read and write performance. If one is executing large random writes then one would observe only slight deviations throughout the course of the day.

Figure 4.5 depicts the entire breakdown sequential/random, read/write performance of the c1.medium instances over the entire 24 hour period. It clearly demonstrates hourly deviations in the total throughput exhibited in both read and write performance. Data intensive workflows and applications will likely involve the movement of large files between processing nodes, leaving sequential read and write performance more important than random read and writes. Thus variations in the sequential read and write throughput will have significant impact and any optimisation algorithm should be cognisant of this fact.

The complete breakdown of both EBS and Ephemeral storage for all the instances tested is listed in the Appendix. High variablilty is observed across all the instances tested. In order



(a) c1.medium Sequential Read

(b) c1.medium Sequential Write

(c) c1.medium Random Read

(d) c1.medium Random Write

Figure 4.5: Performance variations for c1.medium over 24 hours, EBS vs Eph

to handle this variability, an optimisation technique must be capable of reasoning about these variations.

4.4 Application Response Time Data

To generate a realistic simulation for our real world resource allocation problem we utilise response time data provided to us from real cloud deployments by Cedexis. Cedexis are specialists in multi-cloud deployment strategies which are inferred through the analysis of real-time cloud performance data (bandwidth, response time, availability). Cedexis measures cloud performance from disparate geographical locations across multiple cloud providers observed from real end users. This enables their customers to get up to the minute performance measures of how particular clouds are performing to requests, localised to specific geographical locations and regions. Using this information customers can then load balance and scale their applications to ensure the highest levels of service possible. Cedexis value their real-time data the most and are thus happy to open up their historical data to the research community for

experimental and simulation purposes.

Customers of Cedexis agree to the placement of a script on a busy page in their website i.e. *www.bbc.co.uk*. When users visit this page, the script is automatically downloaded and runs client side[2]. The script contains a list of URLs pointing to applications hosted in different cloud regions. The client randomly chooses one or many URLs each time, to measure the performance from the client to the cloud.

Each URL points to a location which contains one of two possible objects, a 50 byte object and a 100 Kbyte object. Requests for the 50 byte object are executed twice in quick succession, this is useful as the difference between the response times can determine the delay from DNS lookup or SSL negotiation. Importantly, a dynamic application is used to service the requests and not a static object. This is to prevent caching and edge servicing via content delivery networks which will skew the results. In this manner actual server response times can be observed from a variety of locations.

The http response time parameter measures the length of time it takes the server to respond to the request. It is a close approximation of server round trip time from the client to the virtual server. This technique offers a more holistic view of response time performance from the user's perspective.

A Bitcurrent[3] report in 2011 [Bit11] compiled from data provided by Cedexis over a seven day period, offers insights into the variability associated with disparate user http response time across different regions of the globe. This report was generated using almost 300 million data points, ranging over nine different globally distributed cloud providers. This report gives a comprehensive and detailed analysis of the performance of the regions, performance per country and even the performance of individual ISPs within each of these countries.

Table 4.2: Detailed performance breakdown per cloud region

EC2 Region	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
APAC (Syd)	905	2050	30728	18224	30048	9108	3885	2548	1933	1049
EU (Ire)	43215	15243	7437	3424	1553	1033	600	359	282	173
US (N. Vir)	3633	37655	11161	7313	2742	1527	1469	748	426	362
SA (S. Pao)	953	2161	17062	6179	2968	1275	746	526	337	197
APAC (Sing)	511	573	5678	8556	4544	1377	793	495	345	209
APAC (Tok)	1314	10014	18688	23174	5638	2963	2265	930	528	355

[2] The procedure only executes when it has detected the browser to be in an idle state

[3] Is now CloudOps research

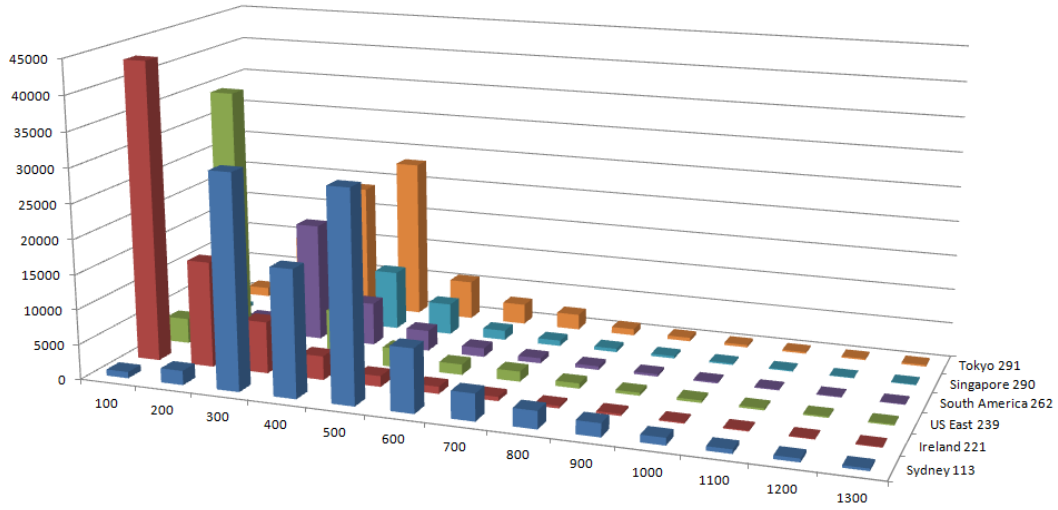


Figure 4.6: Performance histogram by cloud provider (global)

For the purposes of our simulations we sample from a much smaller data set consisting of 350k+ data points recorded over a single day (11^{th} of March 2013) focussing on a single cloud provider, Amazon EC2. Figure 4.6 details http response time over a single day across 6 separate regions of EC2. We split up the response times observed into buckets of $0 - 100$ milliseconds, $100 - 200$ etc, meaning a response of 50ms will be placed in the first bucket and so on. We then count the number of tests observed within each bucket and visualise them as a histogram.

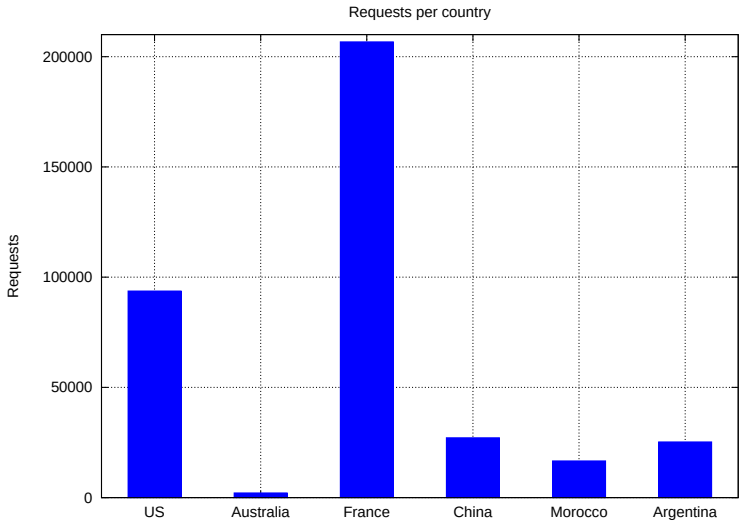


Figure 4.7: Requests by country of origin

The histogram shows that the best performing regions $(0 - 100)$ are Europe (Ireland) followed by US East (Northern Virginia), with Asia Pacific (Tokyo) in 3^{rd}. The full breakdown of performance for all the regions tested over the day is given in Table 4.2. It's clear from the results that proximity to cloud region plays an important role in server round-trip times (response times).

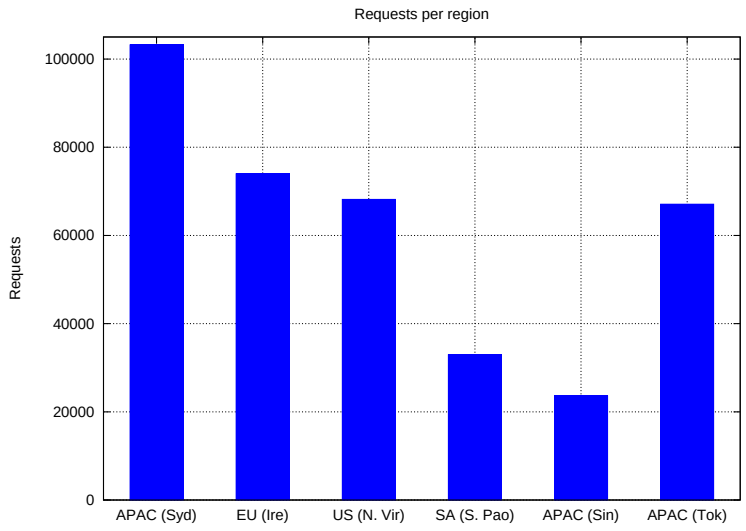


Figure 4.8: Requests by region

Figure 4.7 shows the number of requests executed on a per country basis. Currently Cedexis's biggest customers are large French media sites which result in a disproportionate number of requests from the France, $206,702$ in total. Requests serviced by EU EC2 region originating from France would likely experience reduced latency due to lower physical distances from client to provider. Likewise for the US region $(93,701)$, the portion of requests handled by US East (Northern Virginia) would have also had reduced distances to travel when compared to a region such as Asia Pacific (Sydney). As Cedexis' gain greater presence worldwide, more holistic results from a global perspective will be possible.

Figure 4.8 details the number of requests handled per region, interestingly the Asia Pacific (Sydney) region handled the most requests followed by EU (Ireland) and the US East (Northern Virginia). It is important to understand that the apparent poor performance observed by the Asia Pacific (Sydney) region is in part due to the physical distances which must be traveled as opposed to the actual IaaS cloud suffering performance difficulties itself. It is obviously not a good idea to host your applications/services in Australia if the majority of your requests

originate in Europe and the Americas.

To use this data for the purposes of our simulations we sample each time using a simple random sampling method with replacement. The distribution of the data allows us to model response times according to region and locale, generating performance models most suited given our simulation configuration each time. This allows for a much more realistic and accurate modeling of the actual variations observed from the end user offering a more detailed simulation model than ever before.

4.5 Summary

This section described the data sets we use for our experiments in Chapters 5, 6 and 7. Full details of both the Cedexis data and the disk I/O performance data is detailed in the Appendix. Whilst this data is useful for evaluating our techniques in a cloud "like" environment it is not intended to be a direct replica. In addition neither is it a comprehensive performance analysis of the EC2 platform. This research is not concerned with evaluating EC2, but is primarily concerned with the performance of the proposed algorithms in a virtualised environment.

Chapter 5

A Learning Architecture for Workflow Scheduling in the Cloud

5.1 Introduction

Scientific experiments in fields such as astronomy [Berriman *et al.* 04], epigenomics [Illumina 11] and neuroscience [Zhao *et al.* 05] 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 significant 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. Advancements in grid computing technologies led to the development of large computational grids, where through virtual organisations the scientific community could share resources among each other in a large distributed grid network. In response to the sharing of their computational resources, members of the virtual organisation were allocated pre-reserved slots for their own experiments.

The recent advancement of Cloud computing technologies has led to speculation regarding the possibilities of executing scientific workflows on the cloud [Juve & Deelman 10]. A recent

study highlighted the cost effectiveness of executing scientific workflows on Amazon EC2[1] [Deelman *et al.* 08]. However this study focused more so on the storage costs within platforms such as Amazon EC2 which account for only part of the costs associated with data management within these infrastructures. These platforms highlight the importance of cost based scheduling solutions.

Previous workflow scheduling approaches for computational grids and clusters focussed on resources which were reserved in advance. These resources were either leased for a fixed time period such as a year or else leased for a set time period specified by the terms of the virtual organisation such as within large science grids. The on-demand computing model available through cloud infrastructures facilitates real time procurement practises in a pay use model. Under this model resource consumption is billed depending on usage for a variety of conditions

- Virtual machines are procured in an on demand fashion, billable on an hourly basis. Yearly leasing is also possible.
- Data transfer costs between regions are also billable and are dependent on the volume consumed per month.
- Data storage is billed per month depending on volume being utilised.
- I/O read and writes to disks are billed based on the number executed per month.

It's clear that regarding said cost criteria, a market oriented model of clouds must be considered when examining how existing workflow management tools need to be adapted to better utilise cloud platforms. In addition scheduling approaches for clouds should be dynamic (online) and adaptive to changes in system behaviours. They should be cognisant of regional vagaries, the costs of transmission between these regions, the costs of instantiation within each region, the location of the data and be capable of adjusting to fluctuating costs for communication and computation.

This chapter proposes a novel cloud workflow scheduling architecture capable of scheduling application workflows in a cloud computing environment. We schedule workflow applications

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

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. In addition we employ a model based Bayesian learning technique to approximate hidden aspects of the environmental model.

5.2 Motivations and Aims

- **H1** : Through analysis of the environmental state and in particular observing temporal changes in performance, one can improve the schedule selection process to achieve a more optimal solution.

The primary goal of this chapter is to describe our workflow scheduling approach which combines techniques from evolutionary computation and dynamic programming to determine high performing schedules depending on the current state of the environment. Our goal is to determine the appropriate schedules with regard to cost and makespan under variable resource conditions. We focus on optimising schedules for cloud environments and include the performance variations within the model. Variable performance only becomes a significant factor when the deadline for execution is monolithic and there exists certain cost constraints associated with this execution, meaning that if one has access to resources for an unlimited time at no cost then there is nothing to optimise. Whilst a number of previous studies have considered cost based scheduling on utility grids maximising defined QoS constraints, these studies have often focused on pre-reserved instances [Singh *et al.* 07, Zhao & Sakellariou 07], focusing on either deadline constraints or cost constraints [Yu & Buyya 06], some multi-objective optimisation [Yu *et al.* 07], some limited MDP based approaches [Yu *et al.* 05] and a market oriented scheduling solution [Wu *et al.* 11].

In order to effectively answer the proposed research question, we must examine how approaches can be adapted to handle the regional effects and cloud resource variability. In contrast to previous work we must consider variable resource performance as part of the optimisation procedure as it is an important factor within clouds and grids. We must also consider how best to mitigate against this variability through the discovery of observable parameters which can aid in the prediction process.

5.3 Workflow Architecture

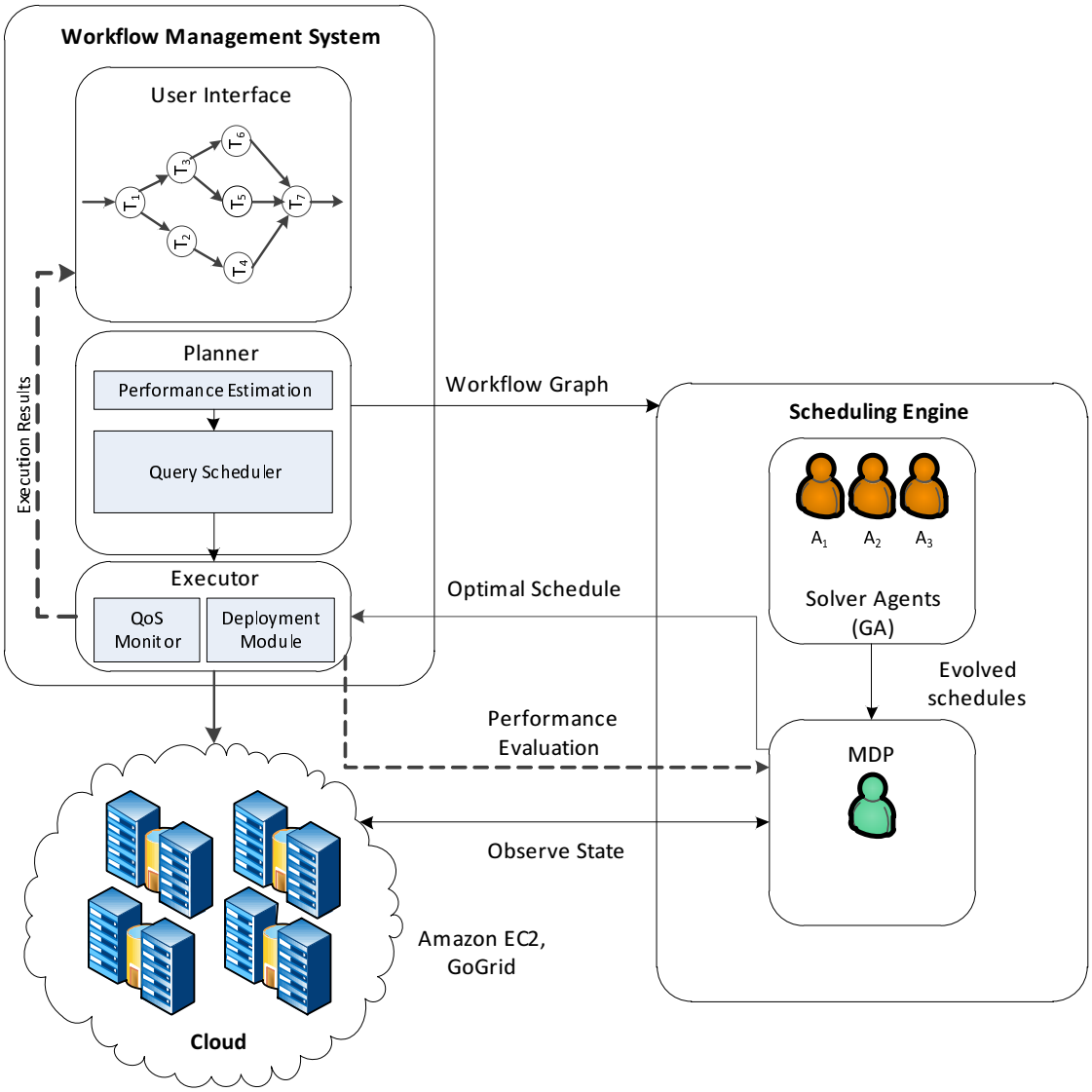


Figure 5.1: Proposed Workflow Management System Architecture

Figure 5.1 details the architecture of the cloud workflow management system. Workflow execution is a multi stage process consisting of a number of steps leading up to and after workflow execution. Firstly users submit their application workflows through the user interface, along with their non-functional QoS constraints. Typically these are cost and makespan constraints determining how much the user wishes to spend and the associated deadline within which the workflow must be completed by. The next stage of the process involves performance estimation on the submitted workflow. The performance estimation phase calculates the average execution time that a given task will take to run on a specified resource. As stated previously,

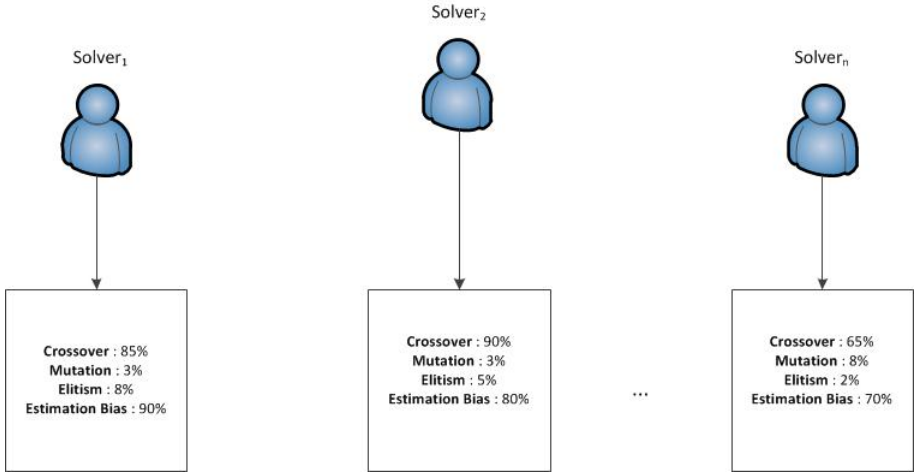


Figure 5.2: Individual solver agent configurations

many authors [Yu & Buyya 06, Pandey *et al.* 10] utilise estimation techniques such as analytical modeling [Nudd *et al.* 00] or historical data [Smith *et al.* 98] to estimate the running time of a task on a particular resource. One of the advantages of our approach is its ability to reason over the probability of these being accurate given the present state of the environment. For the purposes of our work we utilise a simple historical data modeling solution by calculating averages of previous executions on similar tasks for a given resource. We assume that estimates calculated on a given resource can be applied across a whole family of resources, as the capabilities of virtual machines are inversely proportional to their cost i.e the greater the cost the greater the capabilities in terms of CPU, RAM and Disk. Whilst in cloud environments such as Amazon EC2 this may not strictly be true, i.e. certain families such as the compute intensive instances (c1.medium, c1.xlarge ...) or the general purpose instances (m1.small, m1.medium ...) may contain instance types which are not inversely proportional to each other. However for the purposes of this research we assume that this condition holds, to simplify our analysis.

Under the proposed system architecture decision points consist of choosing between multiple workflow schedules for a given application deployment. These schedules are generated using a genetic algorithm to evolve a solution to the scheduling problem. Each instance of the genetic algorithm is known as a solver and each solver has an independent configuration unique within the environment, as is described by figure 5.2. This means that each individual solver will likely generate different schedules from their counterparts as they have all been configured differently. The differences in configuration between solvers applies to both the genetic operators of crossover, mutation, elitism and also regarding estimated task completion times.

Each solver is initialised with a fixed bias placing greater or lesser emphasis on the predicted execution times when compared to its counterparts. With the temporal variability which exists within cloud infrastructures, solvers will generate solutions of varying accuracy at different times.

The selected schedule is then executed via the Executor component. This module queues the tasks from the schedule with the associated resources. Once a processing schedule has completed, 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 function accordingly. This completes a cycle of learning known as an episode, where the outcome from execution forms an integral part of the instructional feedback required to train the system.

5.4 Cloud Workflow Scheduling

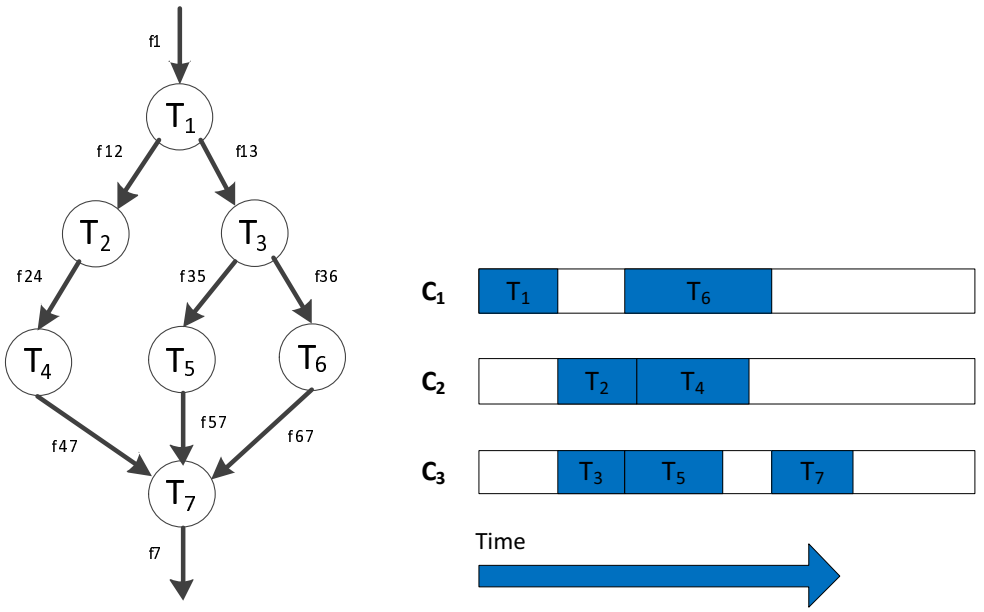
This section describes our algorithmic approaches with respect to the scheduling of workflow applications in the cloud. It outlines Directed Acyclic Graphs (DAGs) which can neatly be used to represent the execution order and data dependencies between tasks in a workflow application. In addition we detail our Bayesian model learning approach which is capable of inferring about hidden aspects of the environment using past experiences.

5.4.1 Directed Acyclic Graph

Workflow applications can generally be modelled as a Directed Acyclic Graph (DAG) $G = \{V, E\}$. In mathematics a DAG structure is considered to be a directed graph with no directed cycles. This structure ensures that if starting at a particular node in the graph, one cannot loop back around to this starting point by traversing a sequence of edges [Christofides 75].

Mapping workflow applications into a DAG structure involves representing the individual workflow tasks as the set of vertices $V = \{T_1, ..., T_n\}$ in the graph. E the set of directed edges represents the various precedence constraints between task nodes. A directed edge $E_{i,j}$ states that task T_i is the parent of task 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.

Figure 5.3a describes a sample DAG graph with 7 tasks, defining the dependencies between



(a) Application Workflow (b) Valid Schedule

Figure 5.3: Sample workflow application with feasible schedule

each task. The file dependencies between these are given as f_{ij} where i is the parent task and j is the child. Figure 5.3b describes a valid schedule for the corresponding DAG workflow, consisting of a finite set of compute services $C = \{C_1, ..., C_n\}$ and the tasks scheduled for execution on the said resources. The execution of task T_i on a compute service C_j incurs a monetary 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 $S = \{S_1, ..., S_n\}$ associated with each resource. Files must be transferred to the corresponding storage site prior to execution of a particular task. 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 across all virtual machines. Let M_{C_i} be equal to the total makespan of compute service C_i. Then the total makespan for the entire workflow is

$$M_{total} = max(C_i) \forall i \in C \tag{5.1}$$

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

$$Ex_{total} = D_{total}(C)_i + P_{total}(C)_i \forall i \in C \tag{5.2}$$

The objective of the cloud workflow scheduler is to find a mapping from tasks to compute services that concurrently minimises the cost of execution and the overall makespan of the workflow.

$$Min(Ex_{total} + M_{total}) \tag{5.3}$$

In general, workflow scheduling onto distributed resources is an NP-Complete problem [Ullman 75]. Meta-heuristic algorithms such genetic algorithms have demonstrated very good performance at generating solutions to these problems and incur very little computational overhead.

5.4.2 Scheduling Using a Genetic Algorithm

In our scheduling architecture each solver agent employs a Genetic Algorithm in order to create valid schedules for the workflow scheduling task. In our approach evolution occurs over the entire population, with offspring from the fittest solutions replacing only the weakest schedules in the population. Firstly an initial population of feasible schedules is created. Then each solution is assigned a fitness value based on Equation 5.3. A solutions total cost and execution time infers its fitness in the environment. For selection purposes we normalise fitness according to the following equation.

$$F(i) = \frac{f_{max} - f_i}{f_{max} - f_{min}} + \mu \tag{5.4}$$

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 $\mu = 0.3$. Normalising the fitness in this manner ensures that the highest probability is apportioned to the fittest solutions during the selection process. 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 parents is required. Each pairing results in the reproduction of two offspring.

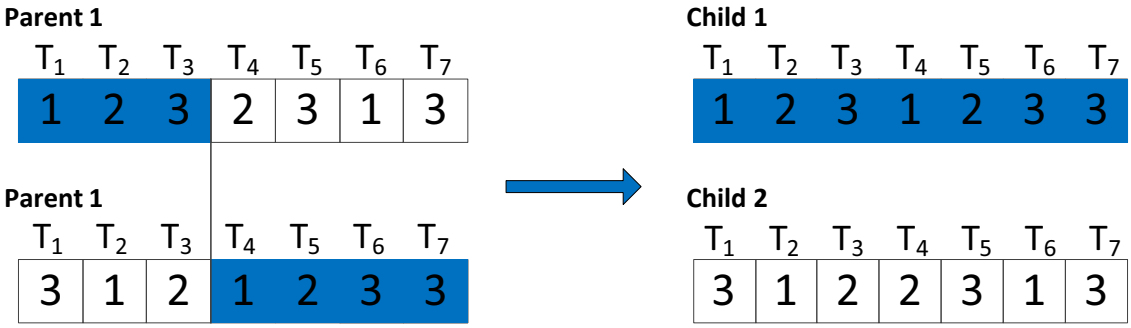


Figure 5.4: Single point crossover on two valid schedules

Figure 5.4 details the crossover of two parents and the 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 that this can be achieved, 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.

The mutation process is depicted in figure 5.5 and involves randomly altering a single bit on the chromosome. Mutation is applied probabilistically throughout the evolutionary process and has a low probability of occurrence.

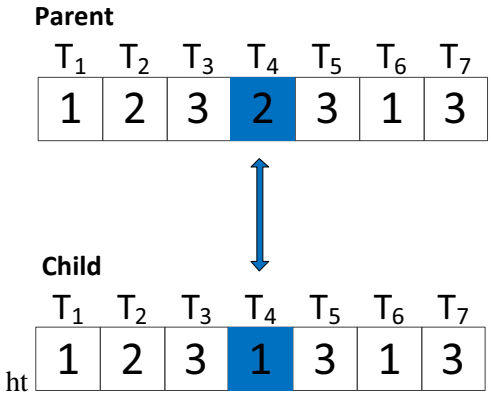


Figure 5.5: Mutation of a single valid schedule

Both the crossover and mutation operations depicted in figures 5.4 and 5.5 contain valid schedules which are either combined or mutated to create additional valid schedules. However these operations (crossover & mutation) can also result in the creation of schedules which are invalid. For instance a schedule would be deemed invalid if a task is scheduled to execute on a given resource at a later time than the execution of one of its parent tasks on the same resource. This will result in deadlock as the precedence constraints between tasks defined by the DAG structure must be upheld for a successful execution. To convert an invalid schedule into a valid one we apply a schedule adjustment process which rectifies any invalid allocations, and is defined by [Oh & Wu 04]. Since Elitism only involves taking the fittest individuals from the previous generation and transporting them directly (unaltered) into the next generation, no conversion to feasible schedule adjustments are required.

The probabilistic values of the genetic operators are fixed for the entirety of the evolutionary process. Once the required number of offspring have been created they form the next generation and the process begins once more. The algorithm terminates when the stopping condition is met, for example after 1000 iterations. For a more general description of the genetic algorithm and their associated parameters, please refer to section 3.7.1.

5.4.3 Choosing Schedules via Learning

The key novelty proposed by our architecture is the modeling of the scheduling problem as a finite MDP and solving using techniques from dynamic programming. Framing the problem in this manner allows a learning agent to solve the MDP and reason about the possible uncertainty surrounding the various schedule generated by the solver agents. In solving a finite MDP two algorithms, value iteration or policy iteration algorithms from dynamic programming are popular choices. In this work we choose the value iteration algorithm to calculate the state value functions as it is much more computationally efficient, requiring fewer iterations than policy iteration. For a more detailed explanation of dynamic programming methods please see section 3.3.

Approximations of $V^{\pi}(s)$ which are indicative as to the benefit of being in state s under policy π 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.

The value iteration algorithm is outlined in section 3.3.2 and is used to compute the value function estimates $V(s)$. For the purposes of choosing schedules from the respective solvers, we propose a state signal comprised of the following three variables, **Time**, **Resource Load** and **Execution Result**. 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 a significant effect on the workflow execution performance. By including time into the state space the learning agent 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 three distinctions, *light*, *moderate* and *heavy*. Coarse interpretations such as this when applied previously have demonstrated surprisingly good results for approximating the system load from a learning perspective [Tesauro *et al.* 05]. The final variable *Execution Result* defines whether or not a given selection was successful in executing the workflow, within the specified constraints (*Success, Failure*). This information is returned via the QoS monitor.

There are many other possible metrics that could, if added to the state signal, give even greater insights (*CPU utilisation, disk reads, detailed bandwidth models or even a finer granularity with regard to resource load*). 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), combined with the three distinctions for resource load and the two terminal states (*Success and Failure*), then the total state space consists of 144 states.

The set of all possible actions within the MDP are defined by the number of solver agents present in the system. Thus the learning agent at each given decision point must choose from amongst the valid schedules generated from n number of solver agents. The rewards within a learning context are defined as the total monetary costs incurred as a result of the execution of a given workflow. If a schedule results in the violation of the user specified deadline (QoS constraints), then additional penalties are added to the reward. The goal of the learner is to ensure the successful completion of the workflow execution within the defined budget and deadline constraints.

5.4.4 Bayesian Model Learning

The Bayesian update rule presented here is a simplified version of Bayes theorem (detailed in section 3.5.1) and similar to what was proposed by David Spiegelhalter [Spiegelhalter *et al.* 93] and further extended by Prashant Doshi [Doshi *et al.* 05]. Using a modified Bayes rule, all that is required to compute the posterior probability is an initial prior probability and subsequent environmental experience. The approach involves maintaining an experience counter $Expc$ for each state variable. To explain how the probabilities are calculated we focus on one aspect of the missing MDP model, the transition probabilities T. Each time the reinforcement learning agent chooses an action a within state s and observes the next state s', the experience associated with the transition is incremented by 1. Knowledge is accumulated iteratively over time for each action selected within a state. Through observing the responses the agent builds up a model of the underlying dynamics. Equations 5.5 and 5.6 define the update rules for approximating the transition probabilities from experience.[2]

$$P'(s = s'|a, s = s) = \frac{P(s = s'|a, s = s) \times Expc + 1}{Expc'} \tag{5.5}$$

Equation 5.6 ensures that the probability distribution over the total number of states sums up to 1.

$$P'(s = s'|a, s = s) = \frac{P(s = s'|a, s = s) \times Expc}{Expc'} \tag{5.6}$$

To illustrate the workings of this approach we consider the simple 4×4 stochastic GridWorld problem from Sutton and Barto [Sutton & Barto 98] shown in figure 5.6. In this world there exists a total of 14 possible "normal" states and two "goal" states highlighted in blue at opposite corners of the grid. Within each state a set of four possible actions (Up, Down, Left, Right) are available which allow one to navigate about the grid. Each action causes a state transition in the direction of that action 80% of the time and at right angles to that action 10% of the time i.e. $P^{up}_{9,5} = 0.8$, $P^{up}_{9,8} = 0.1$ and $P^{up}_{9,10} = 0.1$. These represent the true probabilities i.e. that of which we are attempting to approximate. All actions which result in movement off the grid leave the state unchanged. The goal is to find the shortest path (lowest number of steps) to either of the goal states from any state in the grid. All state transitions result in a reward of

[2] $Expc'$ is the incremented counter, $Expc' = Expc + 1$

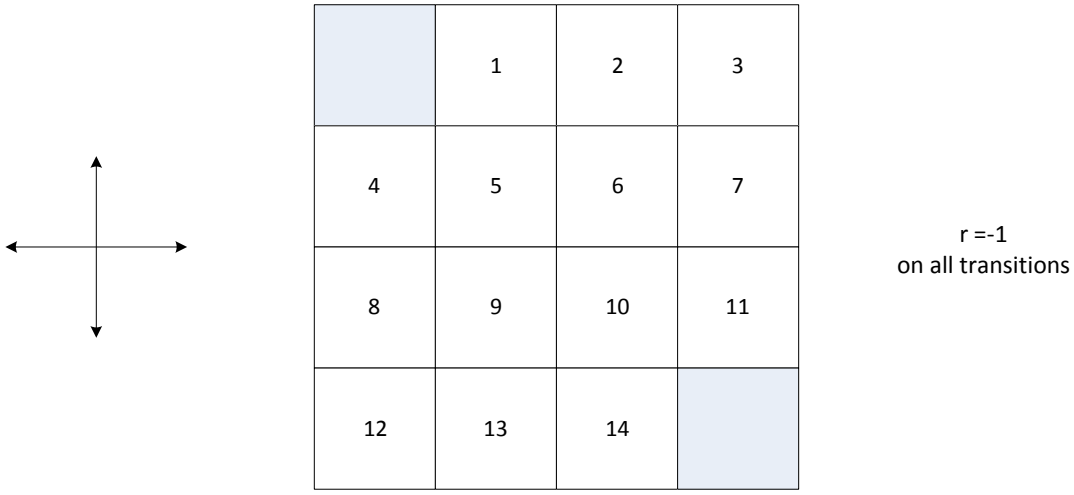


Figure 5.6: Stochastic GridWorld

-1 except those that result in a terminal (goal) state which carry no penalty (reward = 0). It is assumed initially that the transition probability $P^a_{s,s'}$ for all states and actions is unknown. Thus the prior probability for transitions between all states is initially given an equal probability of occurrence $P^a_{s,\forall s'} = \frac{1}{S}$, remembering $P^a_{s,s'} \equiv P(s'|b,s,a)$. Through experience the agent adjusts its estimates of the transition probabilities. A truncated example is given on the next page, showing the adjustment in beliefs over time as the agent updates its experience of choosing action *up* in state 9. The experience counter is initialised to 1.

The updated posterior probability P' from the previous episode serves as the new prior probability for the next learning phase as shown in $t = 1$. As the agent gains greater experience its beliefs concerning the transition probabilities begin to converge towards the true probabilities. In the illustrated example this process can clearly be seen as the same action *up* is repeatedly chosen (in state 9) and the probabilities are adjusted from time step to time step. Additionally the sequence also illustrates the uncertainty in the problem regarding state transitions. After time steps 0 and 1, the action up has led to state 5 each time but on the third occasion (t=2) it shifted to the left visiting state 8 instead. The updated belief now has to be adjusted accordingly to consider the new evidence concerning transition, resulting in a jump from 0.0236 to 0.27. In addition the learner is also blissfully unaware that there is a 10% probability of reaching state 10 by choosing action *up* in state 9. Without experience to the contrary the probability $T(9, Up, 10)$ will continually decrease towards 0 until an observation is made. Additional information regarding this method can be found here [Doshi *et al.* 05].

t = 0

$$P'(s'=1|a=up,s=9)=\frac{0.071\times 1}{2}=0.0355$$
$$\vdots$$
$$P'(s'=5|a=up,s=9)=\frac{0.071\times 1+1}{2}=0.5355$$
$$P'(s'=6|a=up,s=9)=\frac{0.071\times 1}{2}=0.0355$$
$$\vdots$$
$$P'(s'=14|a=up,s=9)=\frac{0.071\times 1}{2}=0.0355$$

t = 1

$$P'(s'=1|a=up,s=9)=\frac{0.0355\times 1}{3}=0.0236$$
$$\vdots$$
$$P'(s'=5|a=up,s=9)=\frac{0.5355\times 2+1}{3}=0.69$$
$$P'(s'=6|a=up,s=9)=\frac{0.0355\times 2}{3}=0.0236$$
$$\vdots$$
$$P'(s'=14|a=up,s=9)=\frac{0.0355\times 2}{3}=0.0236$$

t = 2

$$P'(s'=1|a=up,s=9)=\frac{0.0236\times 3}{4}=0.018$$
$$\vdots$$
$$P'(s'=5|a=up,s=9)=\frac{0.69\times 3}{4}=0.52$$
$$P'(s'=6|a=up,s=9)=\frac{0.0236\times 3}{4}=0.018$$
$$\vdots$$
$$P'(s'=8|a=up,s=9)=\frac{0.0236\times 3+1}{4}=0.27$$
$$P'(s'=9|a=up,s=9)=\frac{0.0236\times 3}{4}=0.018$$
$$\vdots$$
$$P'(s'=14|a=up,s=9)=\frac{0.0236\times 3}{4}=0.018$$

5.5 Experimental Results

To evaluate our approach which combines techniques from both dynamic programming and evolutionary computation, we use Cloudsim, a simulator developed by the CLOUDS group at the University of Melbourne [Calheiros *et al.* 09]. Cloudsim is a discrete event simulation tool which allows you to model a typical IaaS cloud scenario on a single processor. Developed on top of Gridsim [Buyya & Murshed 02], 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.

Within Cloudsim one can create individual workflow tasks, setting their input and output files and the size of the task in Million Instructions Per Second (MIPS). The length of time it takes to completely execute the task[3] is a function of the size of the task in MIPS and the size of the processor in MIPS (a 1000 MIPS task will take 1 second to complete on 1000 MIPS processor). You can also create virtual machines and spawn them within separate data centers. To execute the evolved schedule, Cloudsim facilitates the binding of specific tasks to virtual machine instances, ensuring that they are only executed on those machines. Through space shared scheduling of tasks on the simulator we can schedule the tasks to execute on the resources only after the previous executions have completed. To determine the time spent on data operations we sample the data performance model for each virtual machine instance at each moment in time. It is constructed by plotting consecutive gaussian distributions over the mean and standard deviation of the observed data from micro-benchmarks detailed in chapter 4. Sampling this distribution provides us with a realistic measure of performance each time. Combining these results we can then calculate the cost of execution and the time taken per task.

Data between regions is transferred via a Content Delivery Network (CDN) such as Amazon CloudFront[4]. Two tasks which have a file dependency must transfer the file between the relevant storage sites to commence execution. If two tasks execute on the same resource then there is no data transfer required. We assume a fixed bandwidth model between regions for the purposes of our experiments. This means that the time taken for transfers is a function of the

[3] Tasks are known as Cloudlets in Cloudsim

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

size of the file and the bandwidth between the two sites. This time is included in the overall task processing time.

Table 5.1: Communication costs ($cents) 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

1. 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 and one in Japan. Each data centre supports a single host capable of instantiating multiple virtual machines. We simulate three virtual machines, a small (500 MIPS), medium (1000 MIPS) and large (2000 MIPS) virtual machine. Similarly the costs per virtual machine are, small ($0.1), medium ($0.15) and large ($0.2) per hour.

2. The communication costs between regions are specified in table 5.1. These represent a typical cloud hosts costs per unit data transfer between the four geographic regions. For the purposes of our simulations we consider C_1 to be in the US, C_2 in Europe, C_3 in Hong Kong and C_4 in Japan.

3. The deadline in terms of the time by which the workflow should complete is defined separately for each experiment.

4. For each experiment we use three separate solvers to produce different schedules for the workflow scheduling problem. Each solver configuration is listed in table 5.2. The configurations with respect to *Crossover*, *Mutation*, *Elitism* and *Estimation Bias* are specified. These parameters effect the search capabilities of algorithm, where higher values of mutation tend towards greater exploration. *Estimation Bias* determines the emphasis placed upon the task execution time estimates, with a value close to 1 placing a high degree of confidence in the predicted task execution times, while a value close to 0 indicates low confidence. Collectively these parameters ensure a diverse range of solutions with respect to cost and makespan are returned. It's worth noting that further analysis of these parameters would potentially yield more optimal configurations, however this

would not offer any significant value towards further validating our hypothesis. This is because a more optimal configuration with regards to the parameters specified in table 5.2 could not be optimal all of the time for all of the different states of the environment. It would most likely be optimal from a maximum likelyhood perspective for most of the states some of the time (on average). Our concept is to increase the probability of choosing optimal schedules by selecting a solver based on the current environmental state, of which the goal is to choose the best schedule available all of the time.

Table 5.2: Separate solver configurations

	$Crossover$	$Mutation$	$Elitism$	$EstimateBias$
$Solver_1$	80%	2%	8%	95%
$Solver_2$	85%	3%	5%	90%
$Solver_3$	90%	7%	3%	80%

5. Workflows arrive at the system for scheduling and execution at a rate determined by a Poisson process. An example of the shape of the distribution for various mean inter-arrival rate parameter values (λ) is depicted in figure 5.7. A Poisson process is a discrete probability distribution and is useful for modeling a finite number of events which occur within a discrete time interval. In the past it has been used to model a variety of problems such as the arrival rate of calls in telecommunications, photons arriving at a telescope in astronomy and even the arrival of cars at traffic lights.

Table 5.2 displays the configuration of the three solvers. In summary, $Solver_1$ could be deemed exploitative, $Solver_3$ exploratory and $Solver_2$ somewhere in between the two. $Solver_1$ places a high degree of emphasis on the estimated execution times with a value of 95% for *Estimation Bias*. It also tends towards the exploitation of the schedules which it has already found, by moving a higher proportion of them directly into the next generation with an *Elitism* value of 8%. The evolved schedules produced from this configuration tend towards greater accuracy when the environment is relatively stable. On the other hand $Solver_3$ is the opposite, it has the lowest value of *Estimation Bias* at 80% and the highest values of *Crossover* and *Mutation*, at 90% and 7% respectively. This encourages greater exploration which promotes the discovery of better schedules when there is greater uncertainty within the environment. $Solver_2$ operates between the two, producing schedules that are better than $Solver_3$ in dynamic environments

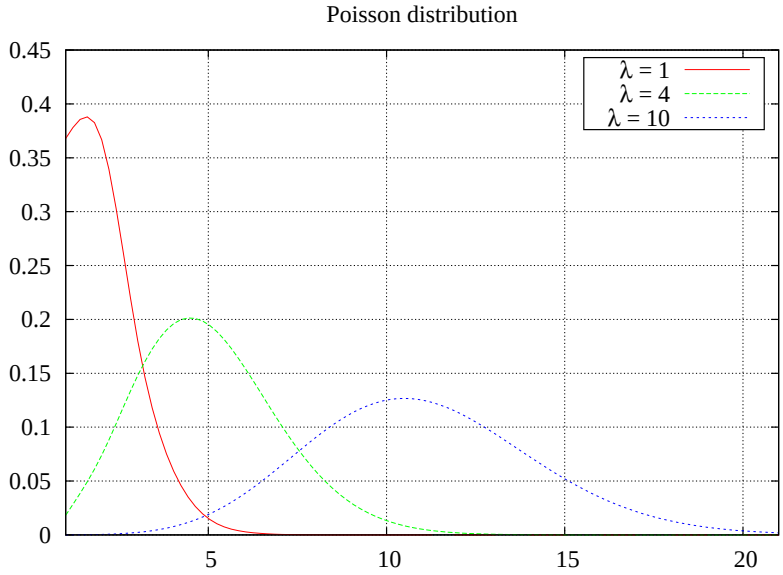


Figure 5.7: Poisson process varying rate λ parameter values

but worse than $Solver_1$.

Therefore to briefly summarise, given three separate workflow schedules generated by the three separate solvers on the same workflow, which schedule should be selected, given the current state of the environment. The goal of the learning approach is to choose the best one available.

In the past a number of grid and cluster workflow management tools have been proposed to address scheduling onto distributed resources such as grids and clouds. Many of these have been devised to facilitate the execution of scientific workflows. Tools such as Pegasus [Deelman *et al.* 05], Kepler [Ludäscher *et al.* 06] and Taverna workbench [Oinn *et al.* 04] have all successfully executed scientific workflows on computational grids and clusters. These generally schedule tasks based on an earliest finish time, earliest starting time or high processing capability heuristic set. Collectively these can be considered as a "***Best Resource Selection***" (BRS) [Pandey *et al.* 10] approach, where a resource is selected solely based on its performance. Throughout our experiments we compare our approach to these BRS selection mechanisms.

In addition, numerous other scheduling schemes have been proposed from both a heuristic and meta-heuristic perspective. These include a number of Particle Swarm Optimisation implementations [Pandey *et al.* 10, Wu *et al.* 10], Ant Colony Optimisation [Chen & Zhang 09], list scheduling schemes [Topcuoglu *et al.* 02] and simpler heuristics such as Min-min, Max-min

and Sufferage [Maheswaran *et al.* 99]. The reason why don't include comparisons of all of these approaches is that we are not strictly proposing a new scheduling solution. Instead we are adopting existing solutions and applying learning over those solutions to determine which one is the best given the current environmental state. This means that any of these approaches can theoretically be included within the framework and not just strictly a genetic algorithm. The core idea being that certain approaches will devise better plans given the specifics of the domain, such as the application being modeled and the nature of the distributed resources. The ability to be able to learn this in a domain independent manner is the principle contribution of this chapter.

5.5.1 Variable Workloads

The first experiment we examine investigates the performance of the proposed solution for a variety of workflow sizes, with the frequency of arrivals governed by a Poisson process. We wish to demonstrate the efficacy of the technique at choosing schedules from the various different solvers under these variable conditions.

To simulate a variable workload on the system, we fix the mean inter-arrival rate of the Poisson process at $\lambda = 10$. This governs the frequency of workflows arriving for execution on the system. With Poisson distributions the bulk of the probability resides around the rate parameter (λ). This is typified by the bell shaped curves in figure 5.7. For a rate parameter of $\lambda = 10$ the probability of 10 workflows arriving per time period is roughly 13%.

Within this experiment the number of compute services is limited to a range of 4 to 16 virtual machines. Thus the minimum at any time is 4 and the maximum is 16. The maximum number of virtual machines allowed in any one region is also restricted to a total of 4 for all instance types (small, medium and large). This ensures a good spread between the regions with regard to resource utilisation. The number of tasks within each workflow varies from 20 to 50 and is sampled according to a uniform distribution. In addition the size of each task is randomly selected, with the maximum total processing size for the entire workflow restricted to 500,000 MIPS. The total data processed by each workflow is fixed at 512Mb per workflow. This is randomly distributed amongst the tasks in the workflow. The deadline for each workflow is fixed at 100 minutes. The entire experiment lasts for 24 hours. During this time, peak time

will shift between the different data center regions in accordance with the local time. During this time, the performance model is sampled from the times which demonstrated the poorest performance with respect to disk I/O found during micro-benchmarking (section 4.3.1).

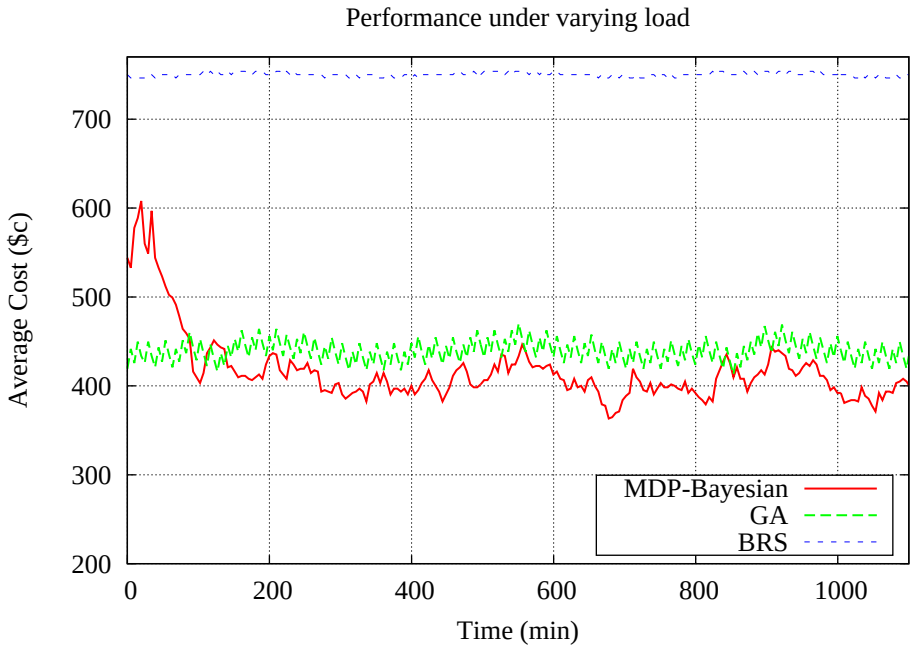


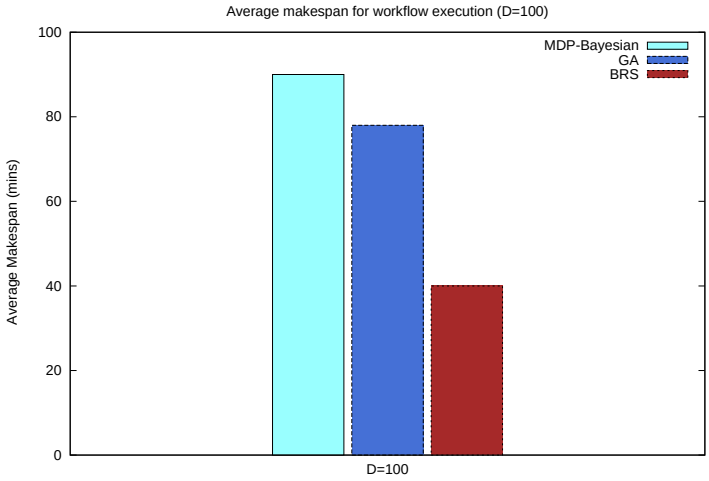
Figure 5.8: Performance of MDP as system load varies

Figure 5.8 shows the performance of our *MDP-Bayesian* learning approach through 24 hours of workflow executions. We compare the performance of the *MDP-Bayesian* approach against the *BRS* approach and a single genetic algorithm solver configuration. The *genetic algorithm* parameter configuration is the configuration of $Solver_2$, 85%, 3%, 5% & 90% for *Crossover, Mutation, Elitism and Estimation Bias*. Through repeated experiments it was observed that this configuration yielded high quality schedules across a wide range of system loads.

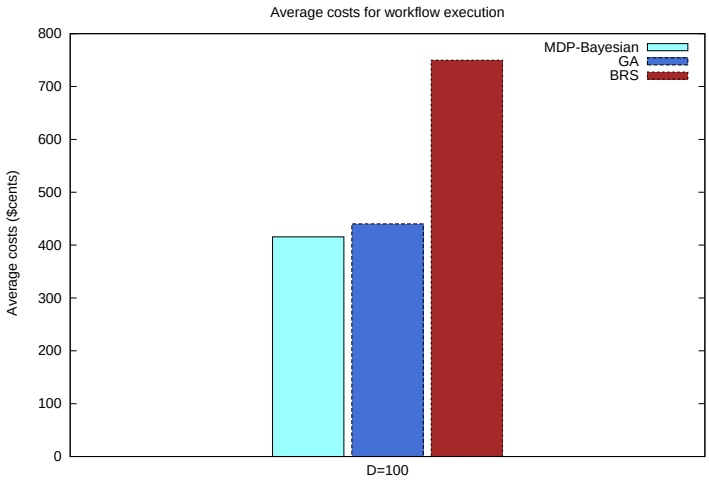
In the initial stages the *MDP-Bayesian* approach assigns equal probabilities to all actions. This results in the selection of schedules which are suboptimal, meaning it doesn't perform as well as the *genetic algorithm* solution ($Solver_2$ configuration) in the beginning. It takes time to explore the different choices presented to it for each workflow scheduling task. This results in a higher proportion of suboptimal actions initially and achieves higher costs than compared to the *genetic algorithm* during this phase.

As the experiment progresses the *MDP-Bayesian* learning approach explores the state space further, building up its knowledge representation regarding each of its selection choices. After

150 minutes of learning time, it has already started to choose schedules better than either of its counterparts, outperforming both the *genetic algorithm* and *BRS* approaches. Since the data size for each workflow is fixed and the *BRS* approach always chooses the best resources available to schedule tasks, the costs incurred each time is relatively static. The costs depicted



(a) Average makespan for workflow execution fixed deadline (100 mins)



(b) Average cost for workflow execution fixed deadline (100 mins)

Figure 5.9: Demand curves for generating variable user requests

in figure 5.8 are inclusive of both the communication costs between regions and any penalties incurred as a result of violation. The magnitude of the penalty could be defined by the user in the Service Level Agreement when submitting the workflow application for execution.

Figure 5.9a displays the average makespan for each of the approaches incurred throughout

the course of this experiment. The average makespan for workflow execution for the *MDP-Bayesian* approach was 89 minutes. This was higher than both the genetic algorithm approach and the *BRS* approach however it did not violate the deadline, on average. The longer makespan, was largely due to the processing of workflow tasks on cheaper resources. As it learned to cherry pick schedules based on the environmental state, the *MDP-Bayesian* learner was able to choose lower cost schedules, which did not violate the deadline constraint, on average. Whilst some violations may still have occurred, overall the average cost depicted in figure 5.9b is still lower than both the *genetic algorithm* and *BRS* approach.

This experiment successfully demonstrates the performance of our Bayesian learning approach, with regard to a fixed deadline constraint. Our next experiment investigates performance under a variety of deadlines.

5.5.2 Variable Workflow Deadlines

Certain workflow applications may require execution completion within a specified time frame. For instance, a workflow application which gathers weather data from various weather stations distributed around the country, must process this data and generate an appropriate forecast, in time for the main evening news [Ramakrishnan *et al.* 09]. Cognisant of this, we analyse the effects of three separate workflow deadlines (D=90, D=70 and D=50 minutes), applied to the incoming workflows. Throughout the experiments the chosen workflow deadline is fixed for the duration of the deadline, but the number of tasks, performance of the resources etc, varies as per the previous experiment. Our goal is to investigate how each technique performs under this deadline constraint, with all other parameters from the previous experiment remaining.

Figure 5.10 demonstrates the average makespan for each approach over 50 runs of the experiment. For the 90 minute deadline, on average the *MDP-Bayesian* and the *genetic algorithm* have longer makespans, but they are still within the defined deadline constraint. They choose to use instances which are of lower cost but take longer to complete the tasks. For the next deadline of 70 minutes both the *MDP-Bayesian* approach and the *genetic algorithm* have longer makespans than the *BRS* approach but they do not violate the deadline threshold. This is also important from a cost perspective as additional penalties are applied for workflow violation.

Interestingly for the shortest deadline of 50 minutes the *MDP-Bayesian* learning approach

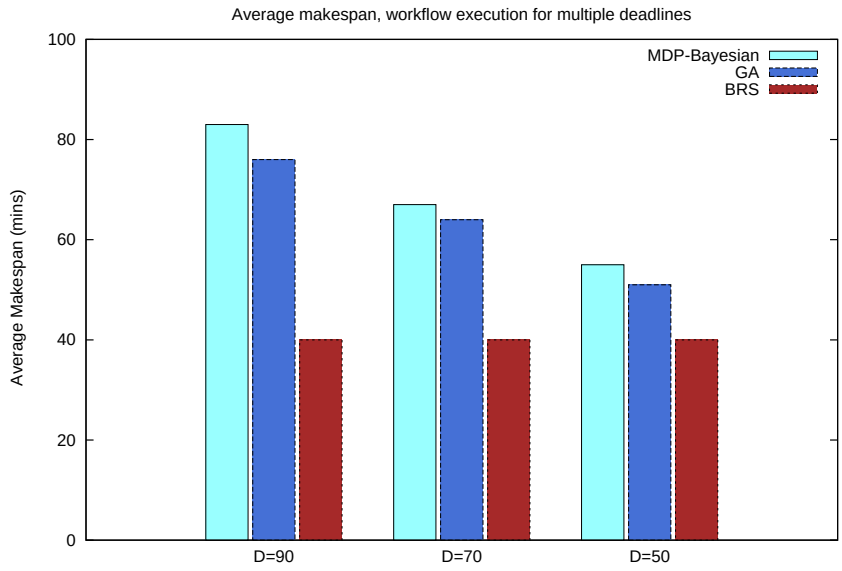


Figure 5.10: Average makespan for workflow execution multiple deadlines (90, 70, 50 mins)

violates the deadline slightly, averaging five minutes over the defined deadline. The reason for this can be explained through analysis of the initial learning phase. During this period the *MDP-Bayesian* learner chooses schedules with limited knowledge of the dynamics of its environment. This results in a high proportion of sub-optimal selections initially which causes more violations during this period. When the deadline is a larger value, such as when it is at 100 minutes, suboptimal selections rarely cause a violation, given experimental configurations. However as the deadline is reduced, poor selection choices begin to result in greater numbers of constraint violations.

In contrast to both the *MDP-Bayesian* and *genetic algorithm* methods, the *BRS* method is not impacted by the varying deadlines at all. This is obvious as it chooses the best resources available each time irregardless of the costs. This is quite inefficient from a cost perspective but consistently achieves a lower makespan than either of the other two approaches. It is also far less likely to violate this deadline when compared to the *MDP-Bayesian* and the genetic algorithm approach. If cost is not a concern then *BRS* can execute the workflows in a much shorter time, but rarely is cost not of concern.

If the initial learning time is responsible for the deadline violations of the *MDP-Bayesian* approach (highlighted by figure 5.10 on the constrained deadline of (D=50)), what could possibly be done to mitigate against the effects of this? A number of approaches have previously

demonstrated success in this regard, these include, intialising the value function estimates with specific domain knowledge provided by an expert system, executing offline training prior to deployment in the real world [Tesauro *et al.* 06] or employing parallel learning methods which reduce the overall learning time [Kretchmar 02]. To reduce the impact of the initial learning time, we carry out a simple offline training measure, after which we deploy the learner within the real system. Offline training replicates the dynamics of the environment, allowing for the discovery of good policies prior to interacting directly with the environment.

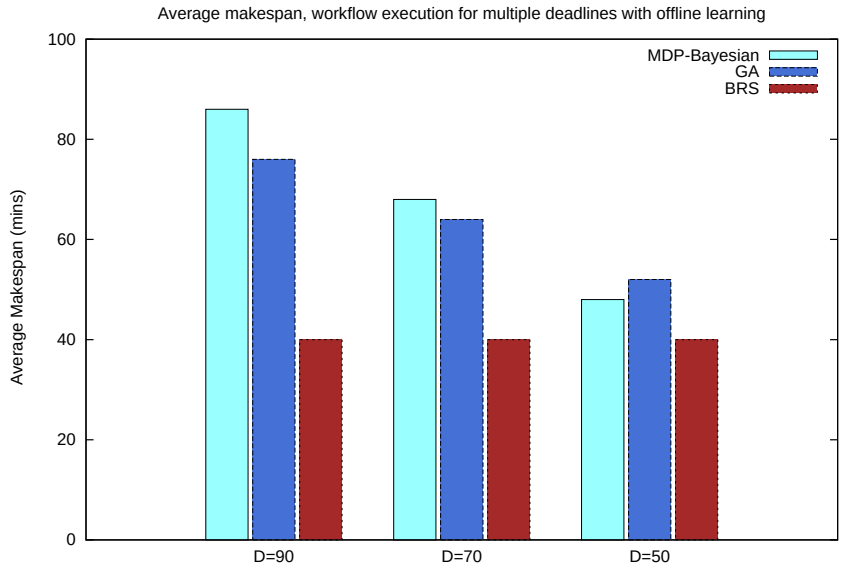


Figure 5.11: Average makespan for workflow execution per deadline (90, 70, 50 mins) initialised with offline training

Figure 5.11 displays the results for the three separate deadlines of 90, 70 and 50 minutes. This time no violation occurs for the D=50 deadline. Interestingly there is also an increase in makespan for the other two deadlines by a factor 4.48% and 3.56% respectively for D=90 and D=70. However analysis of the corresponding costs reveals that a similar drop was not observed.

Figure 5.12 plots the corresponding costs associated with each deadline. For the most part it can be observed that the greater the makespan the lower the resulting incurred costs tend to be. However this is not always the case given the non-linearities inherent within the system, such as data transfer costs between regions, variable virtual machine performance and penalties applied for deadline violation. The genetic algorithm approach tends to have proportionally

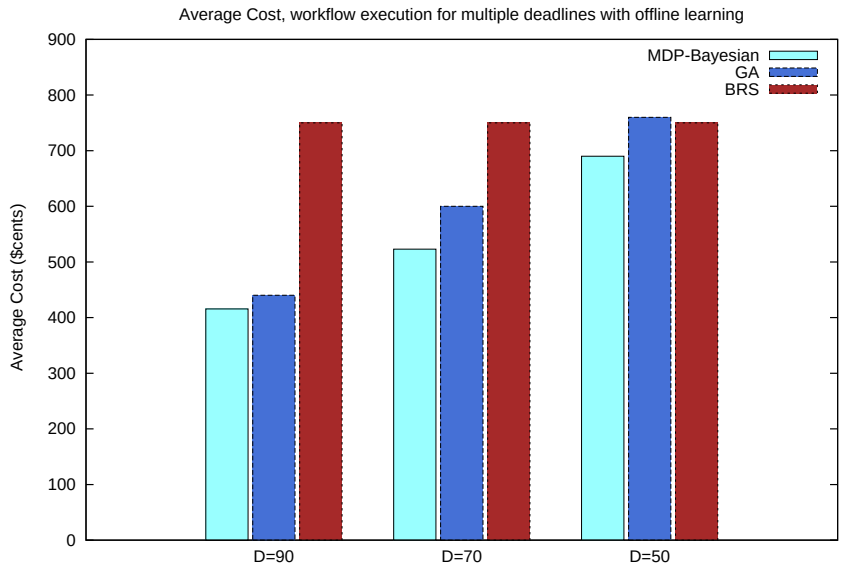


Figure 5.12: Average costs for workflow execution per deadline (90, 70, 50 mins) initialised with offline training

higher costs than the *MDP-Bayesian* approach throughout. This is largely due to a lack of observation over state, as the *MDP-Bayesian* can determine the likelyhood a lower cost schedule has of violation, meaning it can select the cheaper alternative given the present state. Whilst the genetic algorithm performed very well in our first experiment where the deadline was fixed at 90, it tended to evolve solutions which proved slightly more expensive as the process progressed and even for the D=50 produced a schedule which was more expensive than the *BRS* approach (largely due to violations).

Whilst previously we considered that the size of the data within the workflow to be fixed, one important aspect of scheduling on infrastructure platforms such as clouds is the cost involved in data transfers between different regions.

5.5.3 Increasing Data Size

The costs of data transfer between regions is an important consideration for any workflow scheduling approach. The previous two experiments integrated the costs of data transfers into the overall cost of execution but the actual size of the data being processed by each workflow was fixed. This meant that the data transfer costs incurred per workflow would on average be relatively similar from workflow to workflow. Considering this we now examine the effects of

data size on the approaches as the costs incurred with workflow executions. We understand that for certain executions it does not make sense to move the data to the processing site and instead move the business logic of the task to the data itself. However this can often prove problematic, for instance the processing platform adjacent to the data might have limited computational capacity or might not have to correct configuration of hardware/software requirements.

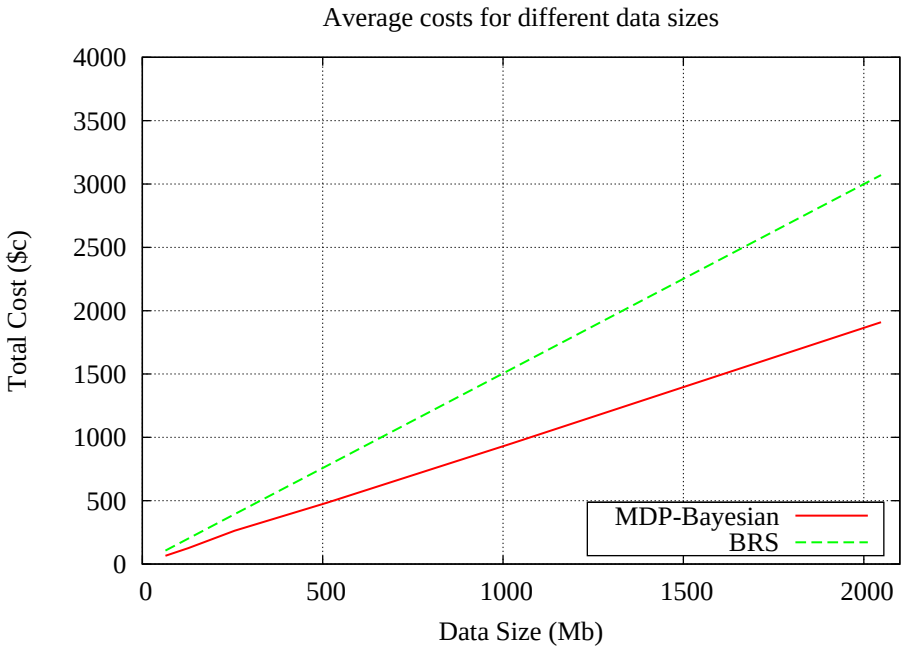


Figure 5.13: Average costs as workflow data size increases from [64Mb - 2048Mb]

With the size of the workflow fixed at 512Mb the previous experiments did not examine the impact different data sizes from a cost perspective. Figure 5.13 displays the costs of the *BRS* approach and the *MDP-Bayesian* approach as the size of the workflows increase from [64Mb - 2048Mb]. The *BRS* approach does not care about the location of the resources and will always choose the best resource regardless. This means that as the data sizes increase, greater costs are incurred as a result of execution. The *MDP-Bayesian* approach in contrast, grows at a much slower rate than the *BRS* approach as the workflow data sizes increase. Whilst it also does not consider the location of the resources, it does however attempt to minimise the overall execution costs. Thus the fitness of a schedule from an evolutionary perspective which proposes many data transfers between regions will be low. This is particulary important when dealing with larger task output files. Evolved schedules which allocate dependent tasks which proportionally generate larger output files onto the same resources will be fitter than those that don't. However it is not possible given the experimental configuration to run all tasks on the

same resource or region as you will most likely violate the makespan constraint due to lack of computational resource.

Whilst allocating child tasks to resources in regions where their parent tasks have executed will reduce the data transmission costs, it can pose a number problems. For instance, in the case of Amazon EC2, they specify that you should distribute your applications and resources between regions to prevent against failures and outages. Thus if you decide to execute all your workflows in the same region you are vulnerable to possible failures. Another reason is the cost of resources in the various geographical regions can vary, meaning cheaper executions can be achieved through distributing the processing requirements among resources from different regions. With our solution we have demonstrated that schedules can have a higher probability of success in regard to reduced costs and makespan when deployed in a distributed multi-region manner.

5.5.4 Kullback-Liebler Divergence

The Kullback-Leibler Divergence (KLD), sometimes referred to as information gain or relative entropy is a technique capable of measuring the distance between two probability distributions. Given two probability distributions P and Q, the KL divergence is given by

$$D_{KL}(P \parallel Q) = \sum_{i} ln\frac{P(i)}{Q(i)}P(i) \tag{5.7}$$

Note that the KL divergence is not a true metric and is not symmetrical, meaning that the KL divergence from P to Q is not equal to the KL divergence from Q to P. If P and Q are identical then $D_{KL} = 0$.

The learning technique proposed in this chapter utilises Bayesian inference to update its knowledge based on observed experiences while operating within the environment. A typical environmental interaction and corresponding observation could consist of the following, suppose a workflow has been submitted to the for execution. The current environmental state for the system as a whole is

$$s = \{12:30, (s(3), m(1), l(2), util(74\%)), Unknown\} \tag{5.8}$$

this is translated as, current time of 12:30, current total number of active resource instances (*3 small, 1 medium and 2 large*), with the utilisation of these resources currently at 74% and the

outcome of workflow execution is currently *Unknown* as it hasn't yet been deployed. Given the present system state, the system chooses a schedule from a relevant solver. It then deploys the workflow within the system, instantiating resources as necessary in accordance with the schedule. Once the execution has completed, it observes the outcome, i.e. the action of choosing a particular solver will result in the changing of the state from having an *Unknown* **Execution Result** initially, to having either a *Success* or *Failure* result depending on the outcome, constraint violations etc. Thus this allows the learning system to build up a representation of the probability of successful workflow execution from choosing a particular action given the state of the environment. This probability is referred to as the transition probability in the MDP framework and careful approximation of this probability is required, in order solve for good polices using dynamic programming.

In a simulated environment using real data such as what we use for our experiments, it is often difficult to know apriori what the real transition function is. One method for approximating it in the simulated environment is to apply a uniform probability distribution over the various action choices ensuring that each action is selected a large number of times within every state. By observing the outcomes of the actions one can compute maximum likelyhood estimates for the real transition function. This involves simulating thousands of learning trials to ensure that as close an approximation as possible is achieved. In a real environment this would not be feasible but for the purposes of our experiments within the simulated environment this is acceptable. To determine the true transition function we run the simulations for 50,000 time steps. This is a controlled simulation where we visit each state a large number of times and execute each action an equal number of times to observe the outcome each time. This allows us to build up a good approximation of the what the true underlying model is, although theoretically this can really only truly be determined on the limit as the number of trials tends towards infinity.

Figure 5.14 plots the Kullback-Liebler divergence between the true transition probabilities, estimated via the method described in the previous paragraph and the observed transition functions as learning normally progresses within the system. We plot two independent runs of Bayesian model learning. What this experiment demonstrates is the variability within the environment from a learning perspective. The goal is to approximate as closely as possible the true values over time.

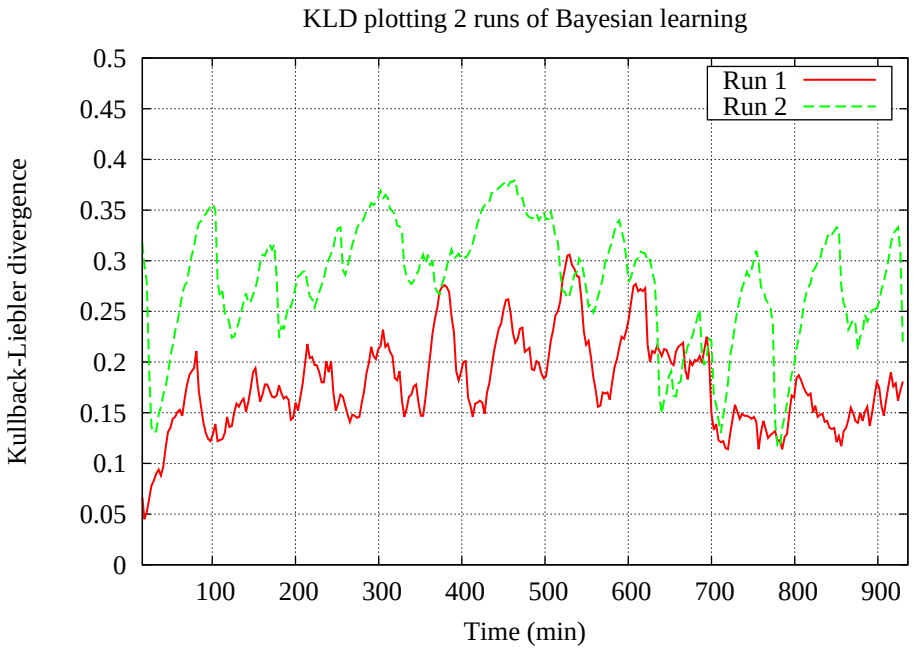


Figure 5.14: Kullback-liebler divergence

For both runs the KLD value between the true transition function and the current estimation tends to fluctuate quite significantly. This occurs as it observes outcomes to its actions, which differ from its previous estimation of that outcome. This continues for about 600 time steps until it begins to converge somewhat on the first run, but the second run is still variable, although it has dropped somewhat also. Eventually given enough learning experience this will converge to 0, but this may take a very long time. A number of factors such as the frequency of state visits and actions selected play an important role in building up a better approximation of the workings of the underlying environment.

Whilst the Bayesian learner has achieved a relatively close approximation of the underlying model through learning, this evaluation only takes a preliminary look at the possibilities of Bayesian model learning. A more thorough examination of Bayesian learning and how it can be applied to reduce the convergence times is discussed in chapter 7.

Chapter 6

Dynamically Scaling Applications in the Cloud

6.1 Introduction

Dynamically scaling applications on large IaaS clouds in response to workload or performance changes represents a vital challenge for resource planning techniques and application management. An effective scaling solution must allocate resources to optimise factors such as cost, performance and reliability. The solution itself has to be scalable i.e. capable of dealing with large workloads and complex resource allocation decisions. The current approach favoured for allocating resources to applications based on fluctuating numbers of requests and application performance is to define threshold based policies [Padala *et al.* 07, Padala *et al.* 09]. These are rule based approaches, where upper and lower bounds are defined based on an observable metric such as application response time. With this approach applications can scale up to meet demand at peak times and scale back once demand has subsided. Determining appropriate thresholds however requires expert domain and application knowledge and must often be redefined based on application updates or workload changes. Since one cannot guarantee the performance of the underlying resource, naively adding similar resources to ensure compliance may not be an optimal strategy.

6.2 Motivations and Aims

- **H2** : Employing multiple learners which share their experiences and are capable of observing specific environmental conditions, will improve performance over previous solutions for dynamically scaling applications in the cloud.

This chapter proposes utilising multiple reinforcement agents, learning optimal application scaling policies in parallel and sharing their observations with each other, throughout the agent environment interaction. This mechanism takes advantage of the inherent parallelism associated with distributed computational platforms such as computational clouds to offer improved performance over the single agent learner case. Parallel reinforcement learning serves two functions, firstly it decreases the length of time its takes agents to determine optimal resource allocations. Secondly the approach is scalable as the number of resources grows, due to the increasing numbers of learners as a function of the number of resources.

We focus on applying our learning approach to scaling data intensive applications in cloud environments[1]. Using the disk I/O throughput data model depicted in chapter 4 we analyse how reinforcement learning can be uniquely adapted to scale applications executing on these resources. In addition the chapter also proposes a novel state action space formalism which effectively deals with the temporal performance variability observed in computational clouds and described in chapter 4.

The majority of previous work in this domain has focussed on two approaches, the first approach involves defining performance thresholds and the corresponding corrective action once a breach has occurred. The second involves using techniques from decision theory and machine learning to automate the scaling of the application based on the feedback in the form of application response time or some other observable metric. The focus of the contributions proposed in this chapter are of the later decision theoretic approach.

With respect to decision theoretic scaling on clouds, recently efforts have been made to develop adaptive policies towards improving resource allocation decisions [Dutreilh *et al.* 11] over strict threshold models. Autoscaling policies should ideally focus on long term strategic objectives, such as costs and application performance even if this involves forgoing short

[1] It's worth noting that the approach is general enough to scale applications or services of any type.

term gains to realise greater gains overall. Policies should also be adaptive to variations in the underlying resource performance and scale in the presence of new or unseen workloads combined with large numbers of resources. With regard to these decision theoretic solutions, significant work has focussed on stochastic planning techniques such as Markov Decision Processes, combined with reinforcement learning techniques [Rao *et al.* 09, Tesauro *et al.* 06]. One of the strengths of these techniques is their ability to reason under uncertainty. However there are a number of issues that have not been addressed by existing research. One of the major drawbacks associated with reinforcement learning techniques when dealing with real world problems, is the length of time it takes to converge to optimal or near optimal policies. In a dynamic scalability context this is the time it takes to learn the best scaling decisions for each given instance. One approach aimed at addressing this problem is to develop a hybrid [Tesauro *et al.* 07] mechanism in which the learning technique is trained using a good external policy, devised by a domain expert or inferred via offline learning. One problem with using domain experts is that the performance of the learning approach following this policy is bound by the skill/knowledge of the individual who proposed the policy. Determining a good initial policy in complex environments with a multitude of variables at play is a difficult challenge.

The key difference between our approach and previous works is that they focus on determining policies for allocating resources to match a variable workload or user request model. Our approach supports multiple criteria in that the outputted policy considers both the variable workload and the underlying performance model. The approach also facilitates learning across geographical regions where it is capable of reasoning about the temporal performance variances associated with the cloud. In addition, to improve the time taken to approximate optimal or near optimal policies we have also devised a parallel learning approach. This is first time a parallelised reinforcement learning approach has been applied in this context. Previous approaches to reducing the state space size and improving convergence times involve hybrid [Tesauro *et al.* 07] learning approaches and utilising function approximation [Vengerov 07] techniques.

6.3 Q-learning

In the absence of a complete environmental model, model free reinforcement learning algorithms such as Q-learning [Watkins 89] can be used to generate good policies in difficult circumstances. Q-learning belongs to a collection of algorithms called Temporal Difference (TD) methods. Not requiring a complete model of the environment, TD methods possess a significant advantage. TD methods have the capability of being able to make predictions incrementally and in an online fashion.

The update rule for Q-learning is defined as

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{6.1}$$

and calculated each time a state is reached which is nonterminal. Approximations of $Q^{\pi}(s,a)$ 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 action selection selection strategy we employ is an ϵ-greedy policy to decide what action to select whilst occupying a particular state. This means that the agents choose the action which presents it with the greatest amount of reward, most of the time. Let $A'(s) \subseteq A(s)$, be the set of all non-greedy actions. The probability of selection for each non-greedy action is reduced to $\frac{\epsilon}{|A'(s)|}$, resulting in a probability of $1 - \epsilon$ for the greedy strategy.

Estimated action values of each state action pair $Q^{\pi}(s,a)$ are stored in lookup table form. 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 agent's 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. This represents a key benefit of policies determined through reinforcement learning compared with threshold based policies. The reinforcement learning based approaches are capable of reasoning over multiple actions, choosing only those which yield the greatest cumulative reward over the entire duration of the episode.

One limitation of reinforcement learning methods such as Q-learning is that they can often require significant experience within a given environment in order to learn a good policy. Whilst

it is easy to implement, is computationally efficient and can operate successfully in the absence of a complete environmental model, it does not make efficient use of the observations that it gathers throughout the course of learning [Littman 96]. In an environment where computational resources are relatively cheap and gathering real world experience costly, an viable alternative solution is to extend Q-learning to operate in parallel.

6.3.1 Parallel Reinforcement Learning

An agent learner can dramatically speed up the time it takes to learn an approximate model of the environment if it does not have to visit every state and action in the given environment. If instead it can learn the value of states it had not previously visited from neighbouring learners, then the time taken to approximate Q^* (optimal value function) could be greatly reduced. This hypothesis is the premise for our parallel learning architecture, where we define a solution to allow agents to share information whilst engaging in dynamic application scaling task.

Parallel learning techniques generally comprise one of the following two approaches. The reinforcement learning agents learn individually operating on the same task or learn on a subset of the given task and their findings are aggregated as part of a global solution. Our approach is an example of the former, where all agents attempt to allocate resources to support the scaling of the same application type. Whilst the agents operate on the same learning task, they will all have different learning experiences due to the stochastic nature of the environment i.e. they will visit different states, choose different actions and observe different rewards. Each Q-learning agent independently maintains a local Q_l and global estimate Q_g of the approximate Q-values. Q_g is the agent's global representation of Q. It consists of the combined experience of all other learning agents exclusive of their own. This separation of personal experience from that of all the other agents facilitates a weighted aggregation of experience. In environments exhibiting a high degree of randomness an agent may weight its own experience over that of the global experience. Q_g is calculated by aggregating the weighted sum of Q-value estimates of all other agents according to Equation 6.2.

$$Q(s,a) = \frac{Q(s,a)_l \times Expc_l + Q(s,a)_g \times Expc_g}{Expc_l + Expc_g} \tag{6.2}$$

Expc represents the learners experience counter. It is a simple incremental counter which

increases by 1 each time the learner observes a particular state and executes a particular action. $Expc_l$ and $Expc_g$ represent the local and global experience counters for all agents respectively.

The agent learner makes decisions based on Q(s,a) the weighted aggregation of the local and global estimates of Q. Algorithm 6 depicts the steps involved in our parallel learning approach. Firstly both the local $Q(s,a)_l$ and global $Q(s,a)_g$ value estimates are initialised to 0. This is an optimistic initialisation and encourages exploration in the early stages of learning. The communications arrays $comms_{in}$ and $comms_{out}$ are initially set to $\emptyset$. For all states visited the agent chooses an action a using an ϵ-greedy policy π with respect to Q the combined global and local estimate. This policy ensures that not all the agent's actions are greedy with respect to Q. Sometimes the agent will choose to act randomly, this balances the tradeoff between exploration and exploitation. A high value of epsilon will bias the agent's decisions towards exploration and a low value allows the agent to exploit its current knowledge.

Based on the policy π the agent executes the action a, observes the reward r and next state s'. The agent then updates its estimate of $Q(s,a)_l$ in accordance with Equation 6.1. If the difference between the Q value estimates are greater than a predefined threshold θ then agent's local estimate is added to the outgoing communications array $comms_{out}$. This information is then transmitted to all other learning agents. Initially quite a lot of data is transmitted between agents, but as the local estimates converge to the global estimates and the difference between them becomes less than θ then no more information is transmitted.

6.4 Model for the Auto-scaling of Applications in Clouds

To facilitate agent learning for a cloud resource allocation problem one must define an appropriate state action space formalism. The revised state action space formalism is designed specifically for obtaining better policies within computational clouds. Our state space representation is tailored to suit the performance related variabilities and the geographical distribution of resources. We define the state space S as the conjunction of three state variables $S = \{u, v, time\}$.

- u is the total number of user requests observed per time period. This value varies between time steps.

Algorithm 6 Our Algorithm for Parallel Q-Learning

Initialise $Q(s,a)_l = 0$, $Q(s,a)_g = 0$, $Q(s,a) = 0$
$comms_{out}, comms_{in} \leftarrow \emptyset$
$\pi \leftarrow$ an arbitrary ϵ-greedy policy w.r.t to Q

repeat
  for all $s \in S$ **do**
    Choose a from s using policy π
    Take action a, observe r, s'
    $Q(s,a)_l \leftarrow Q(s,a)_l + \alpha[r + \gamma \max_{a'} Q(s',a')_l - Q(s,a)_l]$
    $s \leftarrow s'$;
    if $\| (Q(s,a)_l - Q(s,a)_g) \| > \theta$ **then**
      Add $Q(s,a)_l$ to $comms_{out}$
    end if
    Transmit $comms_{out}$ to all agents
    Receive $comms_{in}$ from other agents
    if $comms_{in} \neq \emptyset$ **then**
      for all $Q(s,a)_g \in comms_{in}$ **do**
        $Q(s,a) = \frac{Q(s,a) \times Expc_l + Q(s,a)_g \times Expc_g}{Expc_l + Expc_g}$
      end for
    end if
  end for
until s is terminal

- v is the total number of virtual machines allocated to the application, where each virtual machine instance

  $V_i \in \{\{ty_1, gr_1\}, ...\{ty_n, gr_m\}\}$. n represents the total number of virtual machine types and m is the total number of geographic regions. ty is the virtual machine type and gr is the geograpical region.

- $time$ is UTC time. It allows the agent to reason about possible performance related effects such as peak time of day in a data centre in a specific region.

The agents action set A contains the set of all possible actions within the current state. The agent can choose to add, remove or maintain the number of virtual machines allocated to the application. Rewards are determined based on a defined Service Level Agreement (SLA) which is related to performance. The overall reward allocated per time step is given by the following equations.

$$C(a) = C_r \times V_a + \{ \sum_{i=1}^{v} (C_r \times V_i) \} \tag{6.3}$$

$$Pe(a) = P_c \times \begin{cases} (1 + \frac{p' - sla}{sla} & if \;\; p' > SLA \\ 0 & else \end{cases} \tag{6.4}$$

$$R(s, a) = C(a) + Pe(a) \tag{6.5}$$

C_r is the cost of the resource, this is variable depending on the type, specific configuration and region. V_i represents an individual virtual machine instance, with V_a representing the specific virtual machine allocated, deallocated or maintained as a result of action a. H is the overall penalty applied as a result of violating the specified SLA. $P_c \in \Re$ represents a defined penalty constant incurred through violation of the SLA. The total reward $R(s, a)$ for choosing action a, in s is the combination of the cost of execution and any associated penalties. Whilst many more state observations could be included in this model (CPU utilisation, Memory utilisation, average response time), the approach works surprisingly well given relatively modest state information. In fact previous allocation approaches have had comparable performance to heavily researched open-loop queuing theoretic models, using only current demand as the single observable state variable [Tesauro 05].

Reinforcement learning approaches generally suffer from curse of dimensionality problems, as the size of the state space grows exponentially with each new state variable added. This limitation prevents reinforcement learning techniques from handling environments consisting of very large state and action spaces. To prove the viability of using reinforcement learning to handle application scalability in clouds in lieu of potentially large state and action spaces we have devised a learning architecture aimed at parallelising Q-learning in the context of auto-scaling resources.

Figure 6.1 presents a high level architecture of parallel Q-learning in a typical cloud environment. Each agent makes its own decisions about the incoming user requests and experiences penalties and rewards independently. Each agent's environment is insular, this means that multiple independent agents do not introduce non-stationarity in each others environments as a direct result of learning in parallel. Based on the allocated numbers of requests the agent must attempt to learn an approximate individually optimal policy. The agents then share information regarding their observations while operating in the environment. Each agent communicates directly with all the other agents in the system. Actions of whether to add, remove or maintain the existing amount of allocated VMs are executed by the instance manager based on

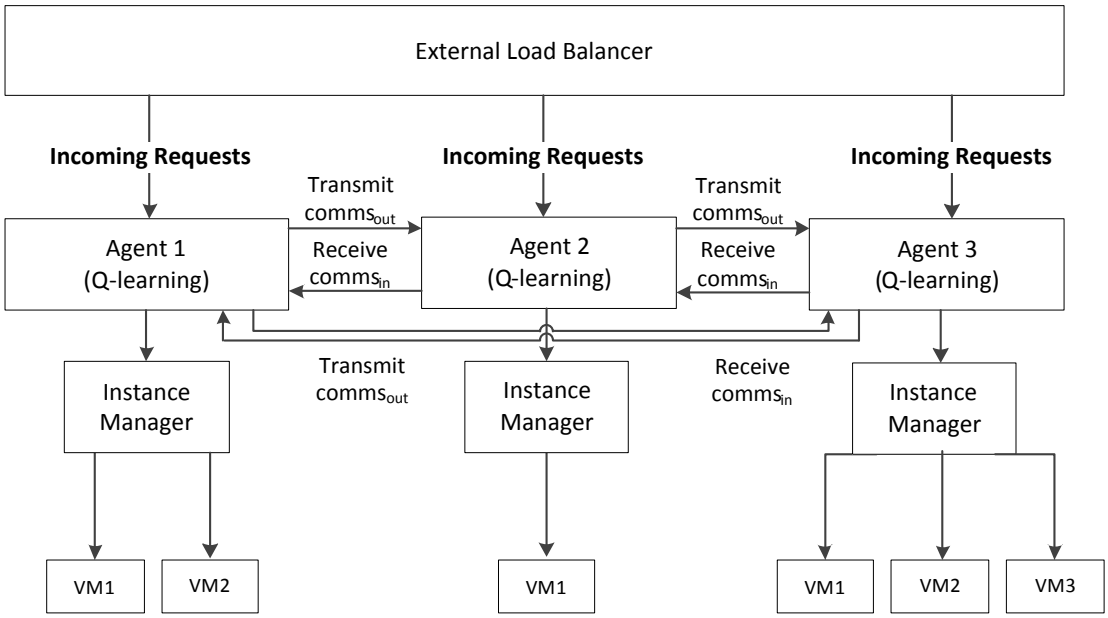


Figure 6.1: Parallel Q-learning Architecture

the instructions of the learning agent.

6.5 Experimental Results

In this section we examine algorithmic performance from two different perspectives. Firstly we investigate the performance of our proposed formalism in the presence of a variable underlying resource and workload model. Secondly we evaluate our parallel reinforcement learning approach and examine its performance with respect to the time taken to converge to optimal policies.

6.5.1 Experimental Setup

We develop an experimental testbed in Matlab to evaluate our results. Unless stated in the individual experimental sections the following parametric settings are consistent across all the experiments.

1. The user request models are generated by a workload simulator. The simulator is capable of generating user requests from either a Poisson distribution or from a sinusoidal curve. The open-loop mode Poisson simulator generates requests with an adjustable

mean arrival rate (λ) ranging from $20 - 120$ requests per time period. The sinusoidal user demand curve has a maximum of 40 requests per time period. Figures 6.2a and 6.2b describe the type of user request models (albeit with different parameter configurations) we employ to drive our simulations of varying incoming requests.

2. The application modeled is a data intensive application. Each incoming user request executes a data operation (read/write) of a $5Mb$ fixed size. It is assumed that an external load balancer distributes requests evenly between the available VMs in a round robin fashion. The goal is to learn the optimal VM configuration (count, type and region) to satisfy the incoming requests and maintain the agreed SLA at the lowest possible cost. The performance of each VM with respect to its instance and storage type is sampled from the data model (disk I/O) described in chapter 4.

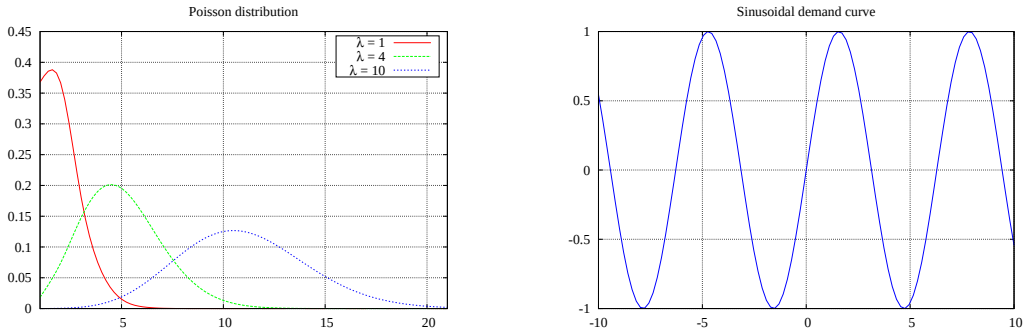
3. A performance model is constructed by plotting consecutive gaussian distributions over the mean and standard deviation of the observed data on a per instance, temporal basis. We can then generate the performance model for any VM instance at any time of day by simply sampling this distribution. Alternative sampling techniques such as simple random sampling with replacement are not sufficient in this case as our sample size is just too small. The experiments are restricted to alternating sequential read and writes as this is a more accurate representation of how large file read/writes are handled on disk.

4. Four separate data centres are simulated in four disparate geographic regions, closely emulating the data centre regions supported by Amazon's EC2. Whilst the performance model constructed in chapter 4 is based entirely on observations taken from the US East N. Virginia data centre, we contend that similar temporal effects as result of peak usage times would also be observed across the other regions as well. Our simulations also emulate EC2's instance pricing model, where prices per VM vary between types and regions. In our experiments we define the price of the VM as directly proportional to its configuration, in terms of CPU, memory and disk size i.e. the greater the size of the configuration the greater the cost.

5. A Service Level Agreement (SLA) is defined in msecs for each experiment individually and it governs the maximum allowed response time per request. The application response time is calculated as a function of the number of requests executed on a VM multiplied by their respective read/write file sizes divided by the throughput in Mb/s for

that time period. So if the sequential read performance is currently at 80 Mb/s for VM1, then two requests totaling 10 Mb executed concurrently will complete within 125 msecs. Requests exceeding the specified SLA are deemed in violation and incur a penalty defined as $P_c = 1$. The value of the penalty P_c has a direct impact on the distance the policy maintains from the SLA.

6. Q-learning is initialised with the following parametric settings. A value of $\alpha = 0.6$ for the learning rate, ensures that the majority of the error in the current estimate is backed up. The discount factor $\gamma = 0.85$, discounts the value of future states. A value of $\epsilon = 0.15$ is chosen, to facilitate adequate environmental exploration. The Q values are stored stored in lookup table form for all the agent learners in the population.



(a) Poisson distribution

(b) Sinusoidal demand

Figure 6.2: Demand curves for generating variable user requests

6.5.2 Optimising for Variable Resource Performance

To analyse the benefit of the state space formalism proposed in section 6.4 over previous work which doesn't consider the temporal fluctuations in I/O performance, the performance of Q-learning with respect to the two MDP formalisms is evaluated. All other aspects of the state space such as observability over the number of virtual machines, their type, location and the incoming user requests remains.

VMs are instantiated to support the application with respect to three different types (*m1.small, c1.medium, m1.large*) and four region configurations (*EU, US, APAC, SA*). Since the data centres are globally distributed, no two data centres will experience peak interference at the same time. Learning intervals are discretised into time steps, with each learning epoch split

into 10 minute intervals. Each interval constitutes a decision point, where the agent chooses an action a, is presented with a reward r and the next state s'. The 10 minute interval is quite large and in a non-simulation environment this level of interaction would be too lengthy, lacking the precision to deal with sudden environmental fluctuations. It would also take far longer to learn a good policy in real time when compared to smaller interval sampling such as per minute. However in the simulation environment we are not concerned with how quickly the policy can be discovered in a real time sense, more so the number of time steps it takes to learn. In a real environment one could triple the speed of the environmental sampling to make sure that greater amounts of environmental experience can be observed speeding up policy convergence considerably. The reason why we choose the 10 minute interval is to do with the sampling of our data model. Our data model measures hourly fluctuations in the data. Whilst technically sampling the distribution at a smaller granularity level than which the data was gathered is still valid, we feel that since we are only concerned with the effects of performance fluctuations for this experiment, a 10 learning interval gives us a more accurate representation of the underlying data model. Finally the agent's action set is limited to the addition or substraction of a single virtual machine each time. The number of user requests is fixed at a static 20 requests per ten minute period with the SLA fixed at 250 msecs.

The first approach (hereafter referred to as *RL-time*) reasons across both user requests and resource variability. *RL-time* incorporates current UTC time into its state space, which allows it to consider the possible effects of peak time in the data centre causing interference from a disk I/O perspective. The second approach does not reason about the variability of the resource, instead presuming that each additional resource gives a defined performance gain based on its configuration, consistent with previous work [Yu & Buyya 06, Dutreilh *et al.* 11]. We refer to this simply as *RL-basic* and it considers all VMs of a specific type to be homogenous with respect to performance i.e. each additional VM added to the system brings a defined resource gain which is fixed for the lifetime of that VM. Since the RL-basic learner can not associate the environmental state with performance deviations, the best it can hope to do is compute maximum likelyhood estimates of its observations. It cannot make the same accurate predictions about the actions selected as the RL-time approach can. As a result of this the policy tends towards the allocation of greater numbers of VMs at a higher cost.

The value specified for P_c has a direct impact on how closely the learned policy approximates

the given SLA value. A high penalty will encourage policies that produce lower response times, resulting in increased numbers of VMs and greater execution costs. This will result in higher overall resource costs as the learning agent attempts to avoid SLA violations. In contrast a low value for P_c will result in greater numbers of SLA violations as the cost of the resources will outweigh the penalty for not meeting the target SLA. We specify a penalty of 1 and proportionally apply it depending on the distance from the SLA. The objective of each approach is to choose resources which facilitate the combined goals of cost reduction in conjunction with maintaining the SLA as closely as possible.

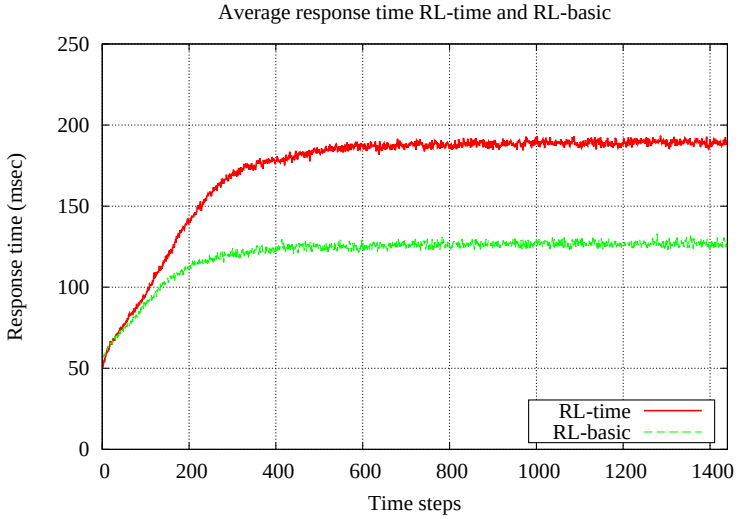


Figure 6.3: Average Response Time : RL-time vs RL-basic

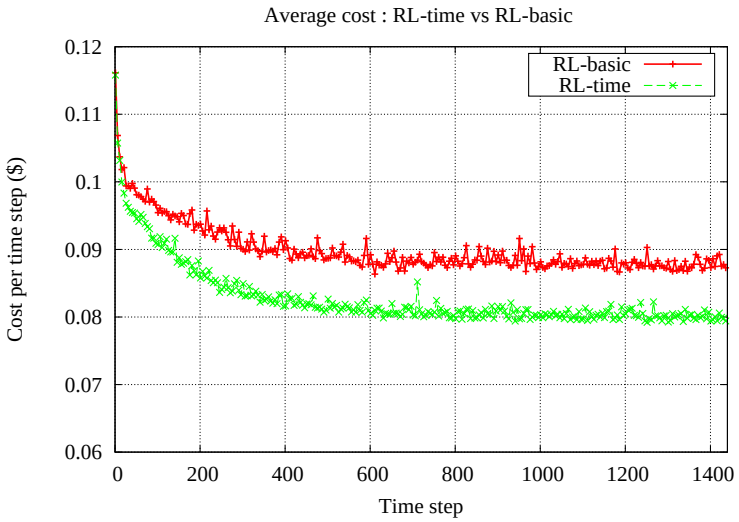


Figure 6.4: Average Costs : RL-time vs RL-basic

The learning agent observes the state of the environment after each time step and makes decisions based on actions which in theory have performed best in the past. Figure 6.3 demonstrates the performance comparison between *RL-time* and *RL-basic* with respect to average response time. The *RL-time* approach demonstrates higher average response times per request, standing at approximately 190 (msecs) at convergence, however it is still considerably below the SLA of 250 (msecs). *RL-basic* is unable to reason about the underlying performance of the virtual machines deployed. This results in a greater probability of choosing suboptimal resources for a given time period. As a result of this the final policy opts to maintain a much higher number of VMs allocated to support the application, to avoid the costly penalties resulting from SLA violations. Whilst this policy ensures that it has a lower response time than the *RL-basic* approach, it also incurs higher average costs as more resources are allocated to the application. This is evident in figure 6.4, where the average cost per time step is much higher for the *RL-basic* formalism than the temporal observant approach (*RL-time*) which maintains on average a 47% cost saving over the *RL-basic* approach, in the simulated cloud environment.

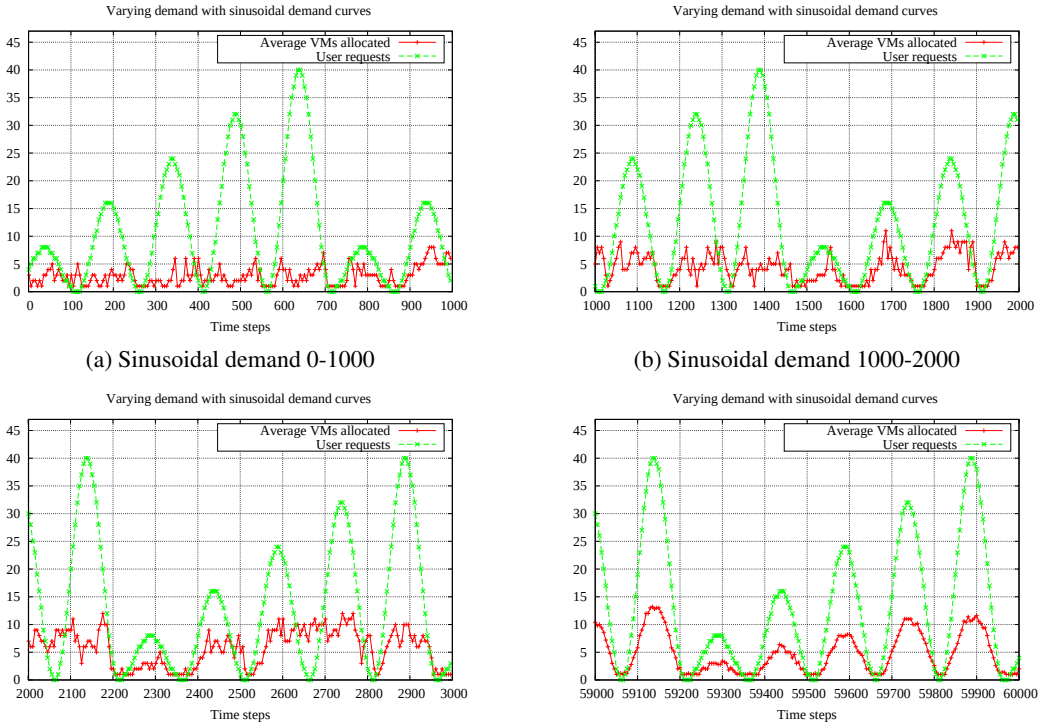
This experiment illustrates the importance of choosing a suitable state space representation capable of learning for the specific problem. The state space must be sufficiently informative to allow the learner to make predictions and associate observations with the relevant state information. During the learning process, the *RL-basic* approach will have experienced the exact same distribution of rewards as the *RL-time* learning agent. The difference though is that the *RL-basic* can not associate why a certain a phenomenon is being observed and can only compute maximum likelyhood estimates as a function of its observable state space at that time. Going forward its ability to make predictions based on this knowledge is much reduced in comparison to the *RL-time* learning agent.

More information about the environment may have also been useful such as the current spin speed of the disks, the number of other VMs residing on the host machine, the nature of the VMs within the co-located environment. Whilst being informative it is also important that the state space is sufficiently concise to ensure the problem remains within certain bounds of tractability and does not require excessive computational overhead to solve. It is somewhat debatable from a theoretical view point whether or not something like time should be observed in the state space because technically you cannot choose actions to transition time, i.e. it automatically progresses in a linear fashion regardless of action selection choices. However it

does present an associative link between an observation and the corresponding state. For this reason we feel it is theoretically justifiable to include it as a state variable in our approach.

6.5.3 Q-learning with Variable User Demand Patterns

Next we analyse the performance of the Q-learning (*RL-time*) approach as the number of incoming user requests is varied throughout the course of the learning process. We investigate two separate demand patterns, a sinusoidal demand function and an open loop queueing model driven by a Poisson process. Figure 6.2 depicts both curves with variable rate (λ) parameters for the poisson process and a fixed range for the sinusoidal curve. Our first set of experiments focusses on the sinusoidal pattern which is representative of daily or seasonal fluctuations in demand for the application. We vary the periods over time which creates a repeating pattern growing to a maximum height of 40 requests before dropping back and repeating the process again.



(a) Sinusoidal demand 0-1000

(b) Sinusoidal demand 1000-2000

(c) Sinusoidal demand 2000-3000

(d) Sinusoidal demand 59000 - 60000

Figure 6.5: Demand curves for generating variable user requests

Figure 6.5a demonstrates the *RL-time* Q-learning approach over the first 1000 time steps of

learning within the system. The learner chooses actions and is penalised and rewarded based on the immediate value of the selection, however at this stage it does not have sufficient environmental knowledge to determine the best action to support the incoming request numbers. During the initial exploratory stages many actions are sampled as the learner attempts to build up the necessary representation of its surroundings. If maintaining high levels of application response time is critical for the service provider then the initial learning phase executing numerous sub-optimal actions would produce unacceptably low levels of performance.

Figure 6.5b depicts the number of VMs allocated from 1000-2000 learning steps. After 1500 steps the approach is beginning to identify the pattern and track the oscillations with the appropriate number of VMs to ensure the requests are satisfied. However it is still learning as figure 6.5c shows continued exploratory behaviour. Figure 6.5d shows the VM allocation policy after $59,000$ time steps and the approach has finally converged and now tracks the demand pattern with the corresponding number of virtual machines.

If the learning algorithm is focussed on strictly optimising to meet the user demand without a variable resource performance model then the problem is much more straightforward and can be learned in fewer time steps. In our example the performance of the VMs fluctuates according to the performance model which depends on the time. We do not consider the geographical spread of the resources for this experiment as we wish to focus on the matching demand patterns first. So all VMs act as though they have been spawned in the same data centre. This combination of variable resource performance which varies according to the current time and the variable user requests, presents a complex challenge for the learning agent. This is why it takes such time to discover the optimal policy for the given conditions.

Figure 6.6 depicts the actual numbers of requests generated by a Poisson process with a rate parameter $\lambda = 10$ over the course of 100 learning time steps.

Figure 6.7 plots the average number of VMs allocated each time, with the SLA per time step increased to 500 (msecs). For the fixed workload ($\lambda = 20$) problem, the Q-learning approach converges after a relatively short number of time steps (circa 800) determining the appropriate configuration of VMs given the number of requests and variability associated with the performance. The rate parameter for the Poisson process apportions a higher degree of probability over the generation of values which are close in value to the rate parameter. This can easily

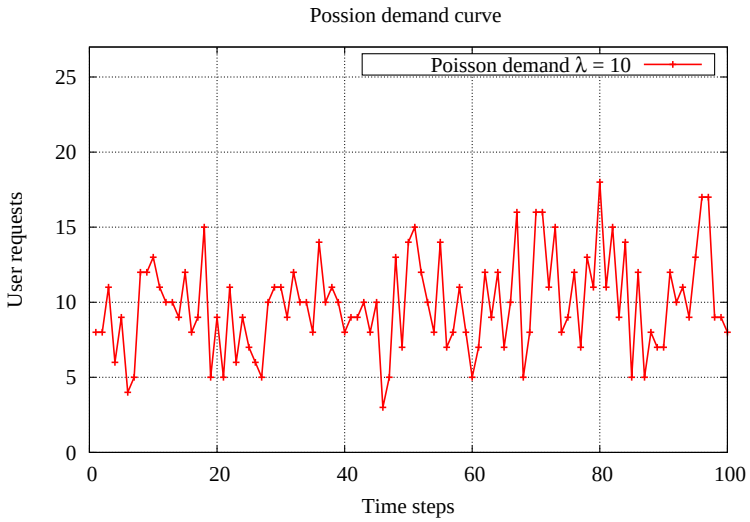


Figure 6.6: Poisson process depicting user requests per time step

been seen by the bell shaped curve pictured in figure 6.2. As the rate parameter increases the variance about this point also increases, leading to a flattening effect of the curve and posing a more difficult problem from a workload estimation point of view.

The second plot in figure 6.7 shows the average number of VMs allocated as the inter-arrival rate adjusts from 40 to 120 requests per time period every 2000 time steps. This demonstrates heightened variability in the environment and displays the performance of the learning approach as workloads change. After 2000 time steps the first adjustment is made where the number of user requests switches from 40 to 60. Figure 6.7 clearly demonstrates the speed at which the approach can determine the appropriate amount of resources to allocate given the change in the mean number of user requests proving that once an initial policy has been discovered it can quickly adapt to model changes for this specific problem. However there is a higher degree of fluctuations for the average number of VMs allocated when compared with the lower rate parameter of 20. This is due to the number of user requests becoming more spread out about the mean. After 4000 time steps the rate parameter (λ) shifts again to 120 requests per time period. The approach now takes longer to converge to the larger request model. This due to its limited experience of the newly observed states and the greater resource fluctuations as a result of the larger numbers of allocated VMs.

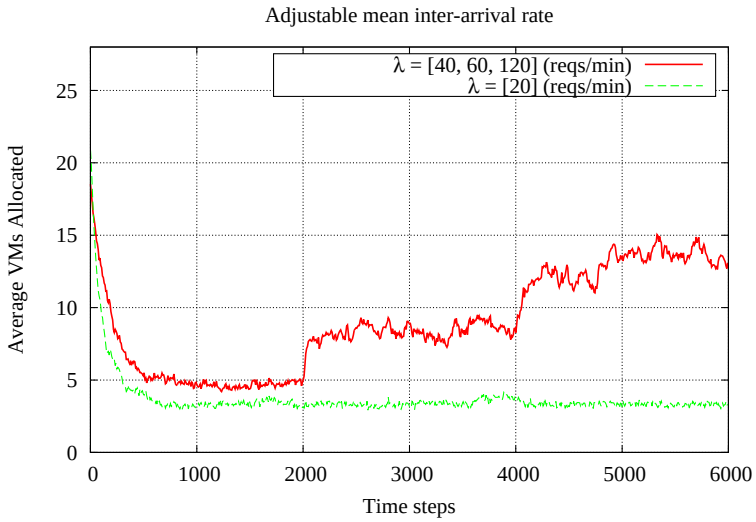


Figure 6.7: Q-learning performance under varying mean-arrival rates

6.5.4 Agents Learning in Parallel

One of the challenges commonly faced when dealing with real world problems with large state spaces is the time it takes to converge to an optimal or even good policy. In certain environments the time it takes to determine good policies can restrict the value of the reinforcement learning proposition. Usually a substantial number of state visits are required to achieve asymptotic convergence to $Q^{\pi}(s, a)$, however in reality often good policies can be determined with far fewer visits. In many real world problems this level of performance is unacceptable. In an application scaling context this would potentially lead to expensive allocations in the learning phase that would inhibit the commercial viability of the approach. In order to improve the length of time it takes to converge to good policies we examine a novel parallel learning approach for improving convergence times through the sharing of Q-value estimates.

Whilst each agent attempts to optimise allocations for their respective numbers of user requests, they will encounter similar states. By sharing estimates of the values of these states amongst each other, they can reduce the length of time it takes to learn an optimal policy. The stochastic nature of the workload request model ensures each agent will have a different learning experience but they will all eventually converge on the same result. This experiment empirically analyses the performance of the parallel learning architecture proposed in section 6.3.1. The goal of the proposed architecture is to speed up the time it takes to converge to

a given policy using parallel learning methods. The learning agents employ instances of the Parallel Q-learning algorithm depicted previously in algorithm 6.

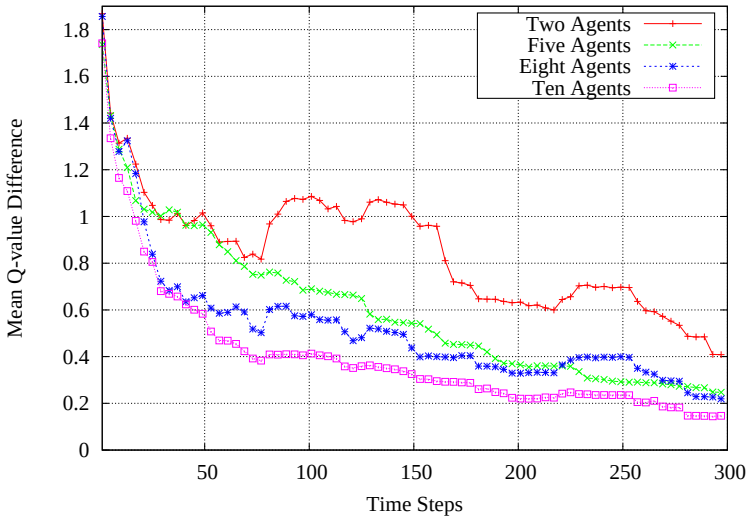


Figure 6.8: Average Q-value distance between learning agents

Figure 6.8 plots the average distance between Q-function estimates as the number of parallel agents is increased from 2 to 10 agents. The action selection policy π is an ϵ-greedy policy. In order to facilitate a strict analysis of the parallel learning performance, we homogenise the VMs allocated to the application with respect to location and performance. We employ a fixed Poisson process with the rate parameter $\lambda = 20$ fixed throughout the entire learning episode.

The graph depicts the average distance between agents' approximation of the value of $Q(s, a)$ over a single run of parallel learning. In the case of the two agent learners their valuations tend to fluctuate considerably more as they experience a variety of different states and rewards over the relatively short number of trials of learning. If one were to execute this experiment and average the results over multiple runs these fluctuations would become less dramatic, but it is important to recognise the inherent variability in any learning process. The graph depicts the reduction in time it takes to converge towards the shared value function estimate as the number of agents is increased. Each approach will eventually converge towards 0 given enough time and learning trials. Both 8 and 10 agent learners converge much faster than the 2 and 5 agent learners, clearly demonstrating the improvements that can be made. However even over a single run one can see that performance is still probabilistically driven. Whilst 8 agent learners does converge faster than 5 for much of the learning episode, it can be seen that after 220 trials it is overtaken briefly by the lesser number of agents (5 learners). Thus

it is important to point out that even though the experiment irrefutably demonstrates that 10 agent learners is better than two, as the numbers are increased the gains tend to diminish. Thus a natural threshold exists wherein the addition of subsequent agent learners does not substantially improve performance and may increase costs relating to communication or even computational overhead.

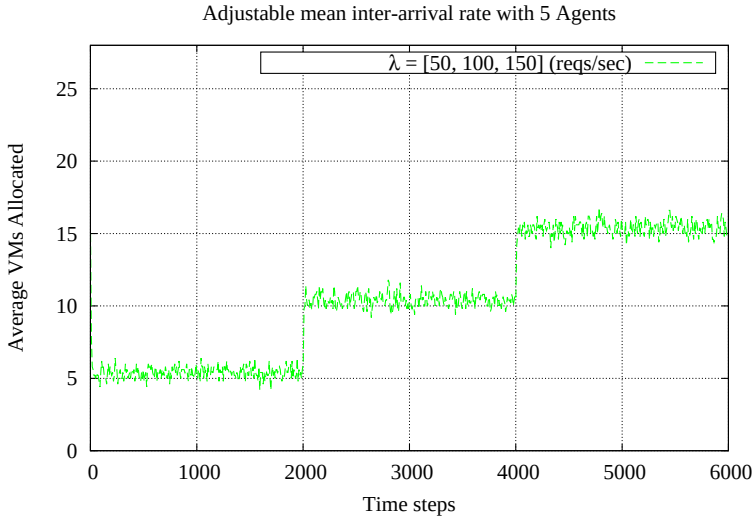
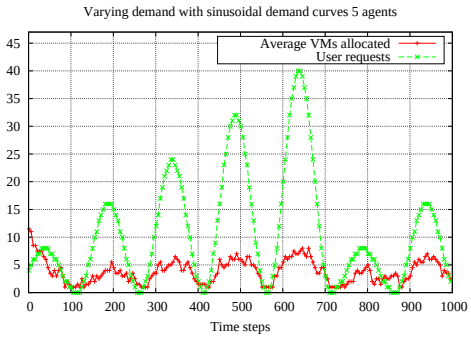


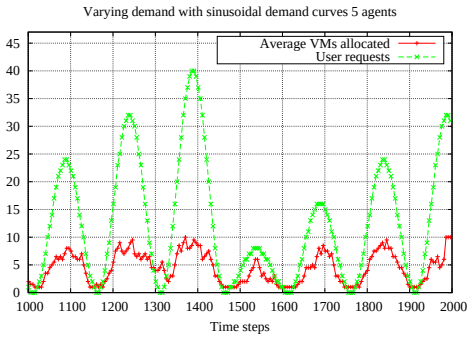
Figure 6.9: Q-learning performance with 5 parallel agents with adjustable mean arrival-rate

Figure 6.9 shows the resulting performance of 5 agents learning in parallel where the mean inter-arrival rates are adjusted every 2000 time steps. For the first 2000 time steps the inter-arrival rate parameter is equal to 50 requests per time period. With this setting the initial convergence to the average optimal allocation of VMs per time step takes about 80 timesteps. If you compare this to the single agent learning in Figure 6.7 albeit with a variable performance model, the convergence time as a result of learning in parallel has dropped significantly. As the rate parameter is shifted to 100 and 150 requests per time period at 2000 and 4000 time steps respectively the time taken to converge is also dramatically reduced as a result of both the combination of prior knowledge and learning in parallel.

Figures 6.10a and 6.10b show a similar pattern for the sinusoidal demand curve given five agents learning in parallel. Convergence is much quicker than the previous single learning agent experiment shown in figure 6.10. Overall parallel learning can have a significant impact on the length of time it takes to converge to good policies even under challenging conditions.



(a) Sinusoidal demand 0-1000



(b) Sinusoidal demand 1000-2000

Figure 6.10: 5 agent learners sinusoidal demand pattern

Chapter 7

Parallel Framework for Bayesian Reinforcement Learning

7.1 Introduction

- **H3** : Using the Kullback-leibler divergence will render the sharing of probability density estimates amongst agent learners possible, and will result in a reduction in learning convergence times overall.

One principle challenge encountered in reinforcement learning is the ability to effectively deal with tasks containing large state and action spaces. MDP based formalisms generally suffer from curse of dimensionality problems where each additional state variable causes an exponential increase in problem complexity. This chapter presents a parallel learning framework suitable for model based reinforcement learning algorithms which distributes large learning tasks by executing multiple instances of learning in parallel. The solution proposes an effective strategy for sharing posterior probabilities online, reducing the convergence times dramatically for both small and more importantly large learning tasks. This is particulary useful for real-world problems such as application scaling in clouds, which consist of large state and action spaces.

Much of the focus by the research community towards solving so called large reinforcement learning tasks has concentrated on function approximation techniques [Sutton *et al.* 99, Melo *et al.* 08, Baird & *et al.* 95, Boyan & Moore 95]. Function approximation techniques reduce the

problem space by generalising over states and actions not yet visited based on their proximity to ones that have. Utilising machine learning techniques which perform regression analysis on the available experience, the learning system does not have to directly sample every state in order to learn an effective policy. These methods have greatly enhanced the effectiveness of reinforcement learning algorithms enabling them to derive solutions in domains which previously proved intractable due to a combination of problem size and representations of state value estimates in tabular format.

The increase in availability of parallel computational resources through recent cloud and utility computing paradigms has facilitated an alternative approach to solving for large state and actions spaces. Known as parallel learning methods [Kretchmar 02], these approaches can potentially offer effective solutions when dealing with large agent learning tasks [Chu *et al.* 07] in a much simpler and more intuitive way. To date a number of approaches have been developed for parallelising learning algorithms including Stochastic Gradient Descent [Zinkevich *et al.* 10], SARSA [Grounds & Kudenko 08] and the dynamic programming techniques, value iteration and policy iteration [Chu *et al.* 07]. Within this domain two main approaches towards the parallelisation of learning techniques have predominated thus far. The first approach involves de-constructing the learning task into subtasks, executing learning on a specific part of the problem [Littman 94] and aggregating the results to finalise a complete solution. The second approach involves executing learning on the same task and sharing the experience between instances to reduce the complexity [Kretchmar 02].

Our parallel approach is derived from the latter where learning instances execute in parallel on the same task. Previous work by Kretchmar [Kretchmar 02] demonstrated the convergence speedups made possible by executing multiple reinforcement learning instances in parallel and sharing action value estimates. This was demonstrated empirically using multi-armed bandit tasks. Due to the stochastic nature of the environment each instance has a unique learning experience and through information sharing allows other instances to benefit from this experience reducing the need for each specific instance to sample all aspects itself. In addition Grounds and Kudenko [Grounds & Kudenko 08] developed an approach for solving a single instance reinforcement learning task by concurrently executing instances of the task using parallel hardware.

These previous approaches have focussed on sharing value estimates or linear function approximation representations and combining the results to learn better policies in a shorter time period. These approaches are model free learners where a control policy is approximated directly through environmental interactions when a complete environmental model is not available. In contrast model based methods attempt to approximate aspects of the hidden model and once approximated solve using techniques from dynamic programming. These methods can yield better policies, particularly if the estimations of the missing model are precise, but they require greater quantities of learning experience in contrast to model free methods which can generate good policies in relatively quick time. In this chapter we further these works to provide a framework for parallelising model based reinforcement learning methods which has previously been ignored. The approach involves the communication of estimates of probability density functions between instances sharing their observations and beliefs of the true model values. The approach is general enough to handle any probabilistic component of the missing model such as the transition probabilities and the distribution of rewards. By parallelising the model learning process, we demonstrate significant reductions in the length of time taken to approximate good policies.

7.2 Bayesian Reinforcement Learning

Reinforcement learning [Kaelbling *et al.* 96] is a trial and error learning framework which learns how to act optimally within an environment by observing direct responses to actions over time. Not requiring a complete MDP model representation such as the complete transition probabilities or reward distribution, model free reinforcement learning approaches such as Q-learning can be powerful solutions for solving problems in high dimensional space and complexity. The relative ease and simplicity of their implementation offers an online learning mechanism capable of learning good policies, usually with a low computational overhead.

To decide what action to choose within each state, model free methods base their action selections on maximum likelyhood estimates (averages) of the rewards achievable from taking said action within the current state. Through knowledge exploitation e.g. by greedily choosing actions which maximise the average or discounted returns achievable from the current state, a good control policy mapping actions to states can be discovered. However the purely

greedy policy approach proves to be sub-optimal if there is any uncertainty in the reward structure [Strens 00] with a high likelyhood of convergence to a local minimum. The conventional approach to mitigating against this problem is to introduce a small probability of exploration which results in the selection of a non-greedy action every so often. This aids in the discovery of better actions which would not have been possible through purely exploitative policies. Policies such as $\epsilon - greedy$ or softmax action selection offer alternatives to the purely greedy solution ensuring better exploration of the state space.

An alternative approach to model free methods is to retain a notion of uncertainty in the model parameters and make decisions according to hypotheses over the true model parameters as opposed to averaging returns from actions. This is known as model based reinforcement learning and is theoretically as justifiable a technique as model free learning. Bayesian reinforcement learning is a model based reinforcement learning approach proposed by Malcolm Strens [Strens 00] which combines Bayesian inference with reinforcement learning to compute the posterior probability over the model parameters and integrate observations within this framework. Whilst this approach is guaranteed to discover optimal policies, actually generating the Bayesian optimal solution involves computing an intractable integral over all courses of actions [Martin 67] which is not possible for all but the simplest of models. Thus in order to integrate conditional planning with reinforcement learning we require a sampling approach to approximate action selection within the environment. First however we discuss how Bayesian inference is combined with reinforcement learning to generate a model based learning algorithm.

7.2.1 Bayesian Inference

Chapter 3 section 3.5.1 discusses Bayes theorem, a mathematical framework which allows for the integration of ones observations into ones beliefs. The posterior probability $P'(X = x|e)$, denoting the probability that a random variable X has a value equal to x given experience e can be computed via Bayes theorem

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{7.1}$$

which requires a conditional probability $P(X|Y)$ and two unconditional probabilities $(P(Y), P(X))$ to compute a single conditional posterior probability $P(Y|X)$ [Russell *et al.* 95].

Bayesian reinforcement learning algorithms combine Bayesian inference with reinforcement learning to generate a model based reinforcement learning solution. Model based reinforcement learners focus on approximating the underlying missing model, by incorporating prior knowledge the learner can avoid the costly exercise of repeating steps in the environment [Dearden *et al.* 99]. In contrast model free methods attempt to directly approximate the optimal policy through computing maximum likelyhood estimates via online environmental interactions. Whilst model based methods make better use of the information available, this can also result in the addition of increased complexity and computational overhead making them a less viable option when compared to model free methods for online learning.

Malcolm Strens [Strens 00] describes a Bayesian reinforcement learning approach for representing the MDP model parameters as a posterior distribution strictly for the probabilistic model components, the transition and reward distributions. Using a hierarchal version of the Dirichlet formalism proposed by Friedman & Singer [Friedman & Singer 99] as the prior probability, Strens applied Bayesian inference to compute posterior distributions over the parameters in an iterative manner. By limiting the sampling of the posterior distribution to a sample size of 1 each time Strens generates hypotheses in an unbiased manner. In comparison with an action selection policy such as ϵ-greedy, this approach is a much more natural way of moving from a purely exploratory policy to a more exploitative one as the posterior probabilities begin to converge to the true values. Bayesian methods can optimally trade-off exploration and exploitation [Poupart *et al.* 06] offering an advantage over model free learning methods such as Q-learning [Watkins 89].

An interesting theoretical framework which has received much attention by the research community with regard to Bayesian reinforcement learning is the Partially Observable Markov Decision Process or POMDP. Suited to mapping problems where one only has partial observability over the state space, POMDPs are solved by maintaining a distribution of belief states which is a multinomial distribution over a collection of real world states. Through observations the learner adjusts its beliefs accordingly and computes the likelyhood of being in one particular state over another. A finite POMDP will only have a finite number of belief states. Since the process is Markovian b' the next belief state can easily be determined as a function of the previous belief state b, the action a and the observation o, similarly to how next states s' are determined in a standard MDP ($T(s, a, s')$). Bellmans equation can be modified to incorporate

the belief structure over the state space and is given as

$$V^*(b,s) = \max_a + \sum_{s'} P(s'|b,s,a)[R^a_{ss'} + \gamma V(b',s')] \tag{7.2}$$

denoting the relationship between states and the learners beliefs. Interestingly this modification affects planning as the learner now considers for each action how its knowledge is affected by its choices [Kawaguchi & Araya 13a, Poupart *et al.* 06], thus providing a logical balance between exploration and exploitation.

7.2.2 Action Selection Strategies

Every reinforcement learning algorithm has an action selection strategy denoting the action to choose whilst in a particular state. This is known as its policy and is critical towards determining performance within the system. A wide variety of action selection strategies for reinforcement learning have been devised and a brief selection is listed below:

ϵ-greedy Choose the greedy (best) action the majority of the time but with a small fixed probability of ϵ choose a non-greedy random exploratory action. This ensures that a fixed quotient of exploratory actions are always executed, preventing convergence to a local minimum.

Softmax selection Actions are selected as a graded function according their values, thus the better the action values the more likely they are to be chosen. *Gibbs* or *Boltzmann* distributions are commonly used to choose actions according to the computed value estimates [Sutton & Barto 98].

Unbiased sampling Generates unbiased samples from the posterior distribution of the model parameters. It then chooses actions based on these samples, as learning progresses the approach converges towards the greedy strategy [Strens 00].

Bayesian sparse sampling *(myopic)* This sampling approach is considered myopic because it does not consider the affects of its actions on its future beliefs [Wang *et al.* 05] i.e. $\gamma V^*(b)$ as opposed to $\gamma V^*(b')$. The myopic approach does however produce a sampling method capable of operating as an online reinforcement learning algorithm.

Interval estimation Actions are selected according to $arg\max_a[\hat{Q}(s,a) + U(s,a)]$ where $U[s,a]$ is a $(1-\delta)$ upper confidence interval on the point estimate $\hat{Q}(s,a)$ [Wang *et al.* 05, Kaelbling 94].

These selection strategies possess advantages and disadvantages depending on a number of factors such as the specific problem, the information available and whether or not we are learning online or offline (via simulation). In a parallel learning context how do these strategies compare. Do some have a better performance over others or could even a combination of policies provide better solutions than a single selection strategy?

The next section describes a simplified model learning procedure which we utilise to update the posterior probabilities given the observed information.

7.3 Learning in Parallel

Our approach to learning in parallel, involves multiple reinforcement learning agents sharing estimates of probability densities over time in order to speedup convergence to the true model parameters. Each learner attempts to approximate the model parameters independently of one another based on their own individual observations. We assume that the learners cannot interfere with each others experiences, in other words, transitions between states or rewards received cannot be affected or interfered with, by the decisions of the other agents learning in parallel.

Employing a Bayesian learning algorithm, each learner maintains a local posterior probability estimate P' and has access to a global estimate G representing the collective knowledge of all other independent learners. This allows each agent to separate their own personal experience, from that of all others. In order to facilitate learning in parallel the learners need some measure of the relative difference between their estimate of a given model parameter and the global estimate. In order to calculate the distance between the various probability estimates from respective agent learners we employ the Kullback-Liebler divergence mechanism which was first introduced in section 5.5.4. It provides a non-negative measure which we utilise for comparative purposes.

Reinforcement learning maintains action value estimates of states and actions $Q(s, a)$ observed. $Q(s, a)$ for the MDP model is specified by

$$Q(s,a) = E[R(s,a)] + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a') \tag{7.3}$$

where the reward $R(s, a)$ is scalar achievable upon from transition between states.

Algorithm 7 Parallel Bayesian Reinforcement Learning

Initialise priors $P^a_{s,\forall s'}$,$G^a_{s,\forall s'} = \frac{1}{S}$
$comms_{out}, comms_{in} \leftarrow \emptyset$
repeat
Choose a from s using policy π (unbiased sampling of the posterior)
Take action a, observe r, s'
Update P' using Eqs (5.5) and (5.6)
$P \leftarrow P'$ (posterior becomes new prior for next iteration)

$$T = \frac{P' \times P'_{Expc'} + G \times G_{Expc'}}{P'_{Expc'} + G_{Expc'}}$$

$$Q(s, a) = E[R(s, a)] + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

if $\parallel D_{kl}(T, G) \parallel > \theta$ **then**
Add P' to $comms_{out}$
end if
Transmit $comms_{out}$ to all agents
Receive $comms_{in}$ from other agents
if $comms_{in} \neq \emptyset$ **then**

$$G = \frac{\sum_i P^i \times Expc^i}{\sum_i Expc^i}$$

end if
until s is terminal

Algorithm 7 depicts the steps involved in our parallel Bayesian reinforcement learning approach. Firstly the prior probability P is defined as equi-probable across all possible next states. This is equivalent to assigning probabilities according to a Dirichlet distribution, which has previously been demonstrated viable as an incremental parametric posterior for multinomial probabilities [Friedman & Singer 99]. The global estimate G is also initialised as equi-probable. P forms the initial prior for the Bayesian update rules defined in Equations 5.5 and 5.6. The communications arrays $comms_{in}$ and $comms_{out}$ are set to $\emptyset$.

We repeat the process for each time step of the learning episode. At the end of time step the learning agent occupies a state s and chooses an action a according to its selection strategy. With Bayesian reinforcement learning, determining the appropriate action to select within each state requires the generation of a hypothesis each time. To generate hypotheses over the posterior distribution we employ a simple unbiased sampling method based on the approach defined

by Malcolm Strens [Strens 00]. To ensure a natural balance between exploration and exploitation we obtain a sample of size 1 over the posterior distribution. This ensures that all actions will be selected some of the time, particularly in the beginning and will slowly reduce as the underlying model parameters are revealed. Producing a sample requires mapping the multinomial posterior probability to a cumulative distribution function via numerical integration and generating a random number from the corresponding CDF.

The learner receives a scalar reward r upon execution of the action and transitions to the next state s' with an unknown probability. The agent, observing the next state, computes the posterior probability $P'(s = s'|a, s)$ using Eqs (5.5) and (5.6) in light of the new experience.

A single learner computes the global posterior from all the other learners by combining their experiences to generate a combined posterior T which becomes the transition function estimate to update $Q(s, a)$, Eq 7.3. It is important that the learner always maintains its own separate distribution P' and experience counter $P'_{Expc'}$ to allow for more accurate repeated integration of the vector of posterior probabilities between learners. This can be illustrated by means of a simple example; assume a single learner has observed $P'(s, a, s') = 0.65$ based on 2 transitions to state s upon selecting action a. The combined experience of all the other learners stands at $G(s, a, s') = 0.76$ based on 6 transitions from s to s' when selecting a. To integrate the probabilities we weight them according to the experience, resulting in the integrated combined estimate for $T = 0.65 \times \frac{2}{8} + 0.76 \times \frac{6}{8}$. Thus T is now based on 8 separate experiences and is a much better estimate of the true transition function. However whilst it is useful to compute value function estimates using the experience of all learners it is not useful to assign T as the next prior and $Expc'$ to 8. The reason for this is that estimates will continue to accumulate indefinitely as the learners share information even though this could be based on only a small number of actual experiences. This will lead to large inaccuracies in T and reduce the efficacy of the approach. By maintaining separate experience counters for each learner's own experience we can mitigate against this problem and ensure an effective integration solution.

The KL divergence between T and G gives the learner a measure of the distance between the two distributions. If this distance is greater than a predetermined threshold θ (a very small number) then it adds P' to the outgoing communications array $comms_{out}$ to be shared with the other learners. This information can then be incorporated by the other learning agents to improve their own estimates. Finally all information received from other agents is accessed

via the $comms_{in}$ array and during each learning step the agent integrates this information into the global estimate value.

7.4 Experimental Results

To demonstrate the effectiveness of our parallel learning method we validate our approach using a stochastic GridWorld example [Sutton & Barto 98] and a virtual resource allocation problem supporting both cost and quality constraints. The focus is to reduce convergence time, improve approximations of the underlying model and ultimately discover better policies. In the stochastic GridWorld problem the transition probabilities belong to a discrete distribution. However in the resource allocation task, the underlying model has a higher degree of complexity where the transition probabilities are affected by a number of variables such as performance/user requests and cannot be easily inferred. In addition we introduce model changes at specific time instances causing shifts in the transition probabilities and analyse the effects on learning.

7.5 Simulations

To evaluate our proposed learning architecture we carry out extensive simulations on a variety of problems. Using Matlab we simulate the performance of the approach on both theoretical problems which are commonly used to benchmark learning algorithms and real world problems which assess suitability and potential in a realistic setting. An example of a theoretical problem is the stochastic GridWorld depicted in figure 5.6 and detailed in section 5.4.4. Although somewhat unrealistic, i.e. rarely would a navigation problem such as robot navigation modeled this way prove useful, it does however serve as a theoretical benchmark to evaluate new learning approaches. If one were to construe this as a real robot navigation problem you would need a much more detailed state space representation, as simply knowing what room one is in may not prove overly useful. For instance, in order to move to other rooms in a real building, the robot would most likely need to observe the doors in the room, navigate around obstacles within the room such as desks and chairs, handle noise from its camera feed and so on. Therefore problems such as the stochastic gridworld, while providing theoretical

importance offer limited value guarantees when dealing with real world settings.

The real world problem we have selected to evaluate our learning approach against, is the problem of virtual machine allocation, supporting a given resource or service with variable underlying performance and user requests deployed in the cloud (IaaS). The varying virtual machine performance is modeled through actual response time data provided by Cedexis [Cedexis 13] and the variable user requests are generated through an open loop queueing process parameterised with a poisson distribution. We have chosen the problem of cloud resource allocation as it provides a multi criteria setting (varying performance and user requests) which increases the challenge in a learning context but also offers a natural parallel resource paradigm which our proposed learning algorithm leverages upon. This means that each instantiated virtual machine has two functions, 1) to support the deployed application (web application) or service, 2) to contribute a portion of its resources to solving the optimisation problem by participating in parallel learning.

7.5.1 Stochastic GridWorld

The Stochastic GridWorld detailed in Section 5.4.4 consists of a fixed number of discrete environmental states represented by the cells of the grid. In each state, the learner can choose from a set of four possible actions (Up, Down, Left, Right). Each chosen action has a probability 0.6 of moving the learner one step in that direction and a probability 0.2 of moving the learner at right angles to the chosen direction. The reward on each transition is 0 and the learner can only occupy one position on the grid at any moment in time. An action that takes the learner off the grid leaves its state unchanged but incurs a negative reward of -1. A number of special states are dispersed randomly throughout the grid, the number of which depends on the overall size of the GridWorld. Once the agent enters a special state, it is deterministically transported to a corresponding state somewhere else on the grid and receives a large numerical reward. The learner must choose actions repeatedly in order to maximise over all return, thus its objective is to attempt to reach the goal states in as short a number of steps possible from anywhere on the grid. The GridWorld we use for our experiments is a 5×5 grid, 25 possible states with two special states resulting in a reward of 10 and 5 if visited.

The core concept behind model based reinforcement learning is that there is more to be gained

from the observations than simply computing Q-values. Thus our first body of results is not concerned with solely computing Q-values but instead we wish to evaluate selection strategies which can combine both model learning and the discovery of good policies. From a strict model learning perspective the optimal solution would be to choose every action an equal number of times throughout the entire learning trial. The resulting observations would be an accurate estimation of the underlying model parameters and a technique from dynamic programming such as Value Iteration would generate the optimal policy for the MDP. This is a perfectly acceptable solution, however for problems with large state and action spaces it would be difficult to visit every state the same number of times and sample every action within those states. It would also likely take a very long time to approximate the underlying model and then solve afterwards. Thus the focus of the selection strategies analysed in the next section is to achieve a high level of model approximation, combined with good reinforcement learning policies.

7.5.1.1 Analysis of Selection Strategies

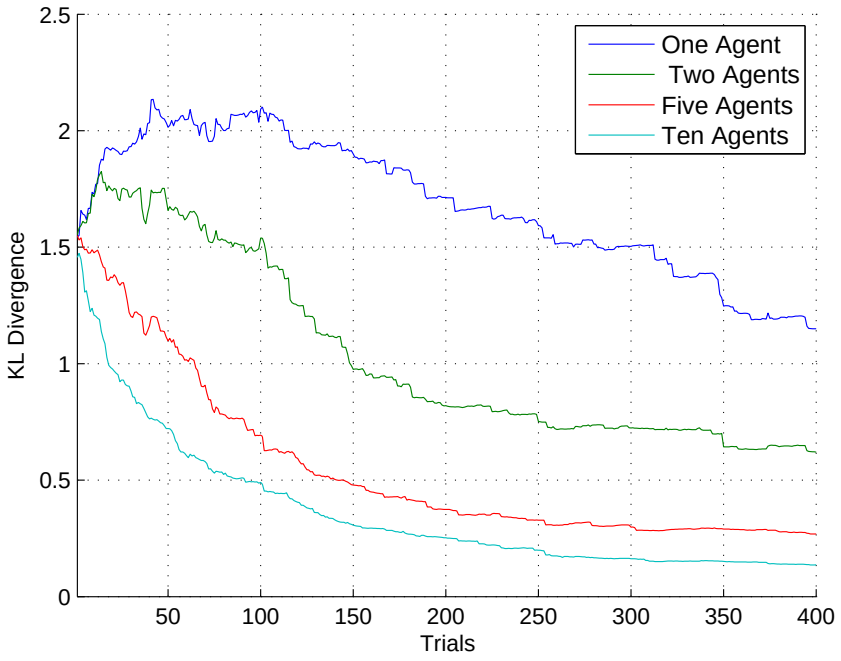


Figure 7.1: Stochastic GridWorld, increasing number of agent learners, $\epsilon - greedy = 0.15$

Figure 7.1 plots the Kullback-Leibler divergence (distance between two probability distributions) between the combined (global) transition probability distribution of parallel learners against the true transition probabilities, over 400 trials of learning. The action selection strategy is an $\epsilon - greedy$ policy where ϵ is fixed at 0.15 and the number of learners is varied from a single learning agent up to ten agents. Clearly as additional learners are added, the distance between the learned transition function and the true function reduces. Interestingly from Figure 7.1 we also see diminishing returns as the number of learners increases. The improvement from the single to two learners is much larger than the improvement from 5 learners to 10. Thus a threshold exists between the computational overhead incurred and the improvement arising from additional learners, which limits the number of learners it is practical to have within the system.

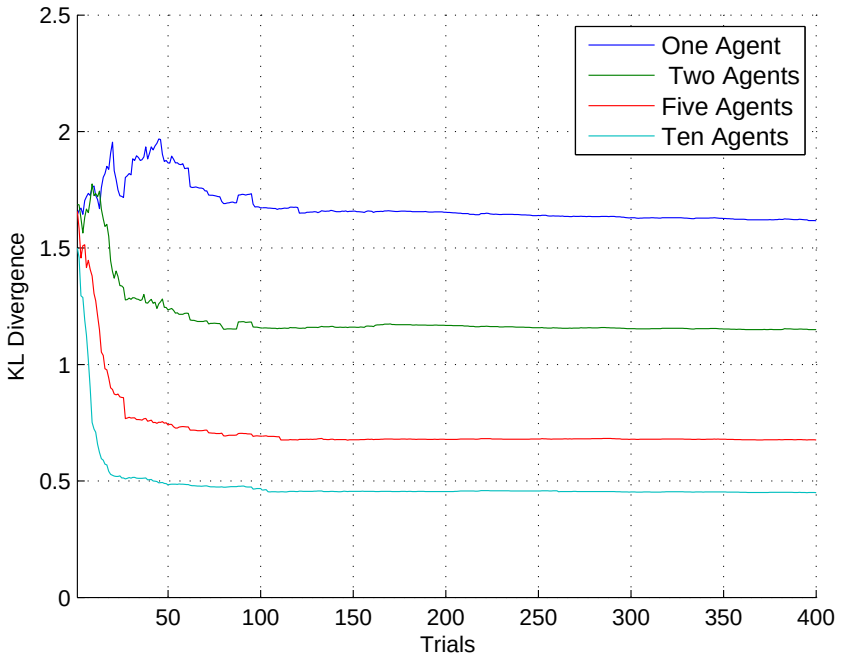


Figure 7.2: Stochastic GridWorld, increasing number of agent learners, softmax selection

Figure 7.2 again plots the Kullback-Leibler divergence between the transition functions approximated by increasing numbers of agent learners and the true transition probabilities. The learners employ the action selection strategy *softmax* which chooses actions according to equation 7.4.

$$\frac{\exp^{Q_t(a)/\tau}}{\sum_{b=1}^{n} \exp^{Q_t(b)/\tau}} \tag{7.4}$$

Softmax selection chooses non-greedy actions proportionately to their value estimates. This is in contrast to ϵ-greedy which is equally likely to choose the worst action as the next best action. Thus as learning progresses, softmax selection focusses on a smaller subset of the action space than $epsilon$-greedy. Not choosing the worst actions some of the time reduces the knowledge of the model parameters, which eventually results in a higher KLD divergence from the true model parameters. Interestingly the performance gain from 2 learners to 5 does not diminish as rapidly as it does with the ϵ-greedy strategy, figure 7.1. However it does not achieve the same proximity to the true transition functions as that of the ϵ-greedy strategy. It's clear that the softmax selection strategy achieves a greater improvement from parallel learning overall, but does not approximate the model as closely as the $epsilon$-greedy policy.

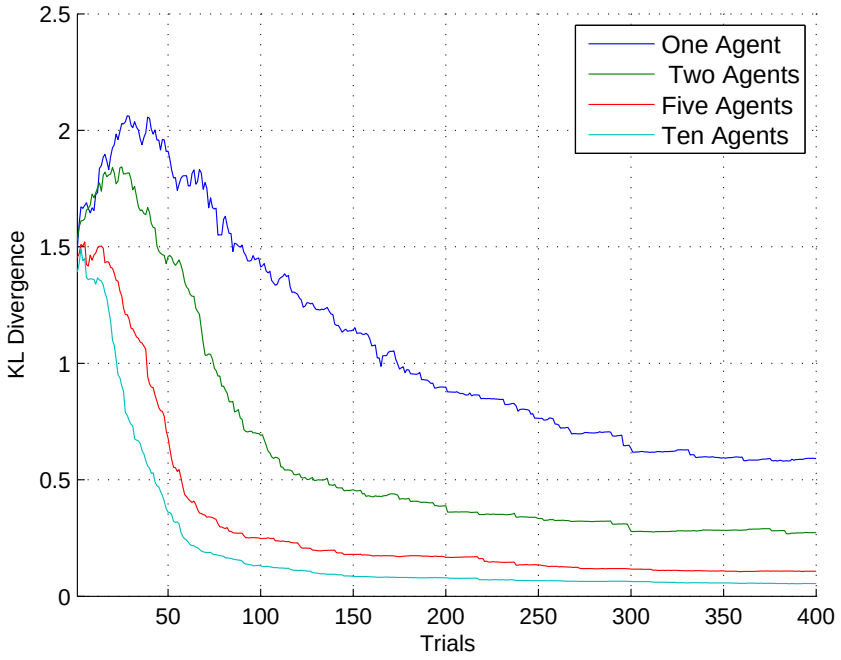


Figure 7.3: Stochastic GridWorld, increasing number of agent learners, unbiased sampling

The third sampling approach we examine is the unbiased sampling solution proposed by Malcolm Strens [Strens 00]. Figure 7.3 plots the Kullback-Liebler Divergence (KLD) between the learners approximation of the model and the actual true values. Unbiased sampling ensures a greater balance between exploration and exploitation, as it considers the effect each action will have on its knowledge and the cumulative discounted reward. It achieves the closest approximation of the true model and also generates the greatest cumulative reward when compared to

the other selection strategies. Unbiased sampling does not achieve the same level of improvement as a result of parallel learning, when compared to softmax selection. This is because the single agent learner employing unbiased sampling has already achieved a very good approximation of the transition probabilities, the equivalent of five agents learning with *softmax* in parallel. Notwithstanding all of this, Figure 7.3 still demonstrates significant improvements over the single agent learner as additional learners are added to the system.

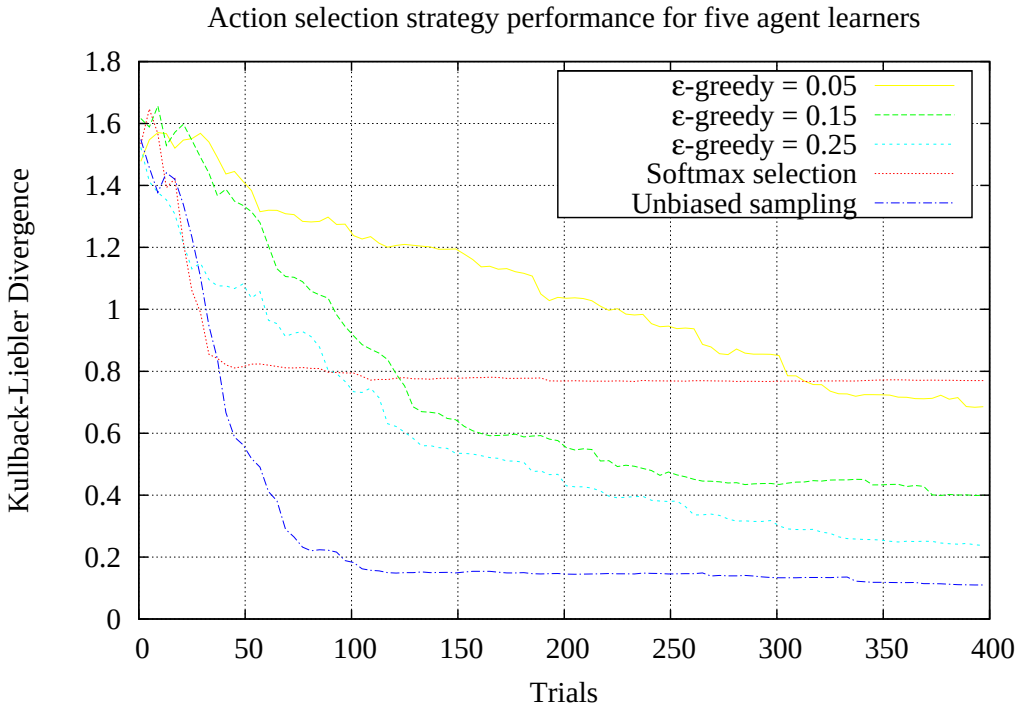


Figure 7.4: Comparison of selection strategies for five agent learners

Figure 7.4 compares the selection strategies for five agents learning in parallel over 400 trials. We alter the ϵ-greedy strategy by varying the ϵ parameter which controls the selection of non-greedy actions from 5% to 25%, where the lower the value of ϵ the more greedy the strategy becomes. Determining an optimal setting for ϵ requires some domain knowledge and some degree of experimentation. This is one of disadvantages with this type of selection strategy. The best performing strategy for 5 agents learning in parallel is unbiased sampling, achieving a KLD value of 1.8 after only 100 trials of learning and overall achieving a much lower KLD value than the nearest ϵ-greedy strategy($\epsilon = 0.25$). However with reinforcement learning the goal is also to achieve large amounts of discounted reward over the course of interaction with the environment.

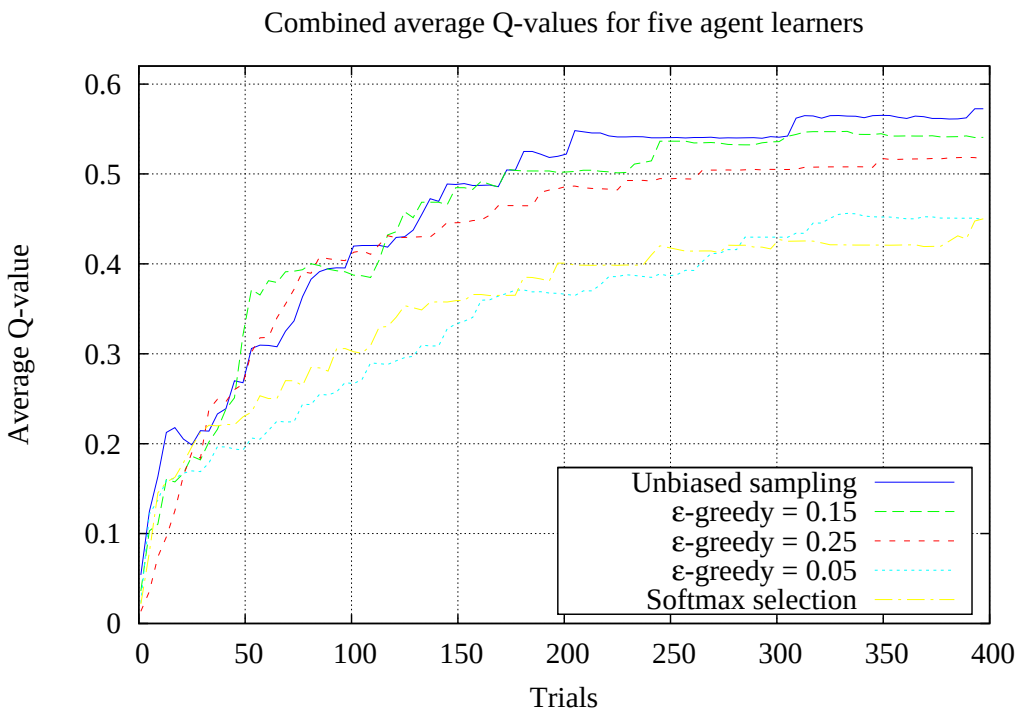


Figure 7.5: Average Q values for five agent learners with varying selection strategies

Figure 7.5 compares the average Q-values over 400 trials for each strategy, again with 5 agents learning in parallel. The unbiased sampling technique outperforms the other strategies as it naturally exploits the closest approximation of the underlying model to achieve a high value policy. It is interesting to note that the ϵ-greedy policy with $\epsilon = 0.15$ has a higher average Q-value than the $\epsilon = 0.25$ strategy. Although as can be seen from Figure 7.4 the $\epsilon = 0.25$ strategy achieves a closer approximation of the underlying model, it does not exploit this knowledge to good effect and is outperformed by the more exploitative $\epsilon = 0.15$ strategy. Determining the optimal setting for ϵ would require extensive sensitivity analysis over the entire range of parameter settings. However this work is not concerned with determining the optimal setting of this parameter, but instead the improvements that are achievable through parallel methods utilising these strategies as a whole, which we have demonstrated empirically.

7.5.1.2 The Effects of Underlying Model Changes

This section investigates the parallel learning performance of the various action selection strategies, when underlying model parameters are non-stationary and shift periodically. This type of scenario would frequently occur in a real world world setting, such as in application

deployments where the addition of a software patch, an Operating System update or variance in user request characteristics could lead to model changes. We simulate it in the Stochastic GridWorld by varying the transition function over all states and actions.

The model is altered at fixed points over the entire learning episode. The initial values of the transition probabilities are set to having a 90% chance of moving in the direction of your choice and 5% chance of moving at right angles to that choice. Every 100 trials the probability of moving at right angles is increased by 5% for each direction. Thus after 100 trials of learning choosing action *Up* will have an 80% chance of moving you *Up*, a 10% chance of moving you left and a 10% chance of moving you right. This continues every 100 trials for a total of 300 trials at which point it stops and remains at that level for the remainder of the trials.

This experiment examines the vulnerability of selection strategies to fundamental shifts and changes in the underlying model parameters. Some strategies reduce the number of random actions as learning begins to converge, choosing instead to exploit their knowledge once they have a decent grasp of the workings of the system. However a sudden shift in the model parameters renders this strategy suboptimal as it will not pursue an exploratory course of the actions to learn about this change in the model. From a parallel learning perspective we wish to study the effects that this technique has on the chosen selection strategies.

Figures 7.6, 7.7 and 7.8 detail the performance of the ϵ-greedy strategy with ϵ varying from 0.05 to 0.25 and model changes every 100 trials until the last change at trial 300. Clearly the strategy with the greatest degree of exploration ($\epsilon = 0.25$) outperforms the others as it still pursues non-greedy actions 25% of the time, helping it to cope more easily with changes in the model. Again even in this circumstance executing multiple instances in parallel and updating using our method realises substantial improvements over the single learner case. Figure 7.9 analyses how the unbiased sampling method performs in an environment where the underlying model parameters are non-stationary. It again demonstrates convergence speedups with the KLD between the true probabilities and learned approximations reducing substantially as learning progresses. All the experiments show some degree of improvement upon model change but Figure 7.10 compares their performance over 600 trials with 10 agents learning in parallel. This time the best performing selection strategy with 10 agents sharing information is the $\epsilon - greedy = 0.25$ strategy followed by the $\epsilon - greedy = 0.15$ and unbiased sampling in third. The two worst performing $\epsilon - greedy = 0.05$ and softmax selection represent the least

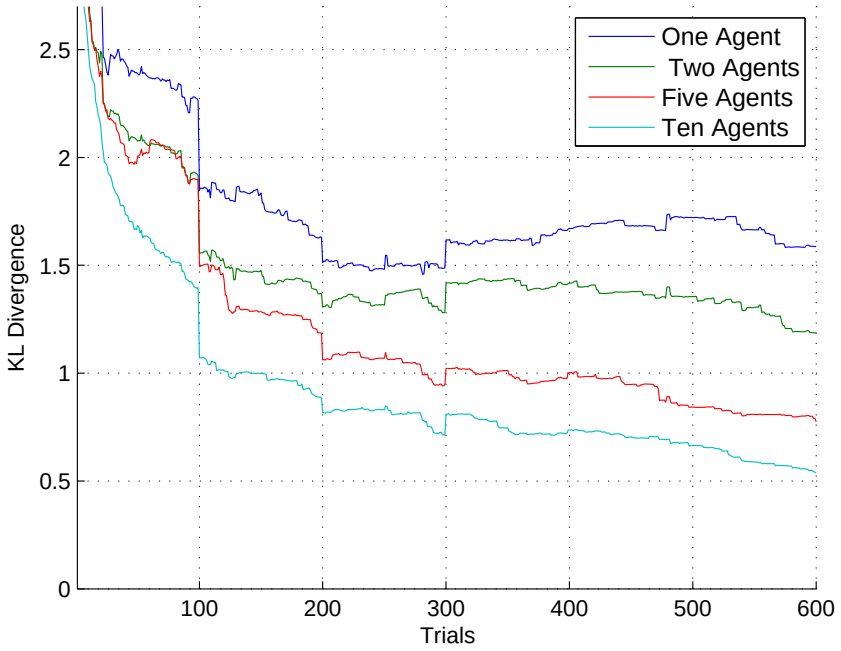


Figure 7.6: Effect of model shift on ϵ-greedy selection ($\epsilon = 0.05$)

exploratory of the approaches and do not perform as well as they do not engage in sufficient exploratory behaviours to discover the new transition function.

7.5.2 Virtual Machine Allocation

Our second learning task examines performance in a real world setting using data traces gathered from virtual servers in the cloud. The focus is on allocating resources to support application deployments, responding to variable user requests and performance. Multiple learners are deployed in parallel to decrease the time taken to learn good policies. The experiment uses real http response time data sets and an open loop queueing user request model to simulate a cloud resource allocation problem.

7.5.2.1 MDP Model and Simulation parameters

To facilitate agent learning for a cloud resource allocation problem we define a simple state action space formalism. The state space S consists of two state variables $S = \{q, v\}$.

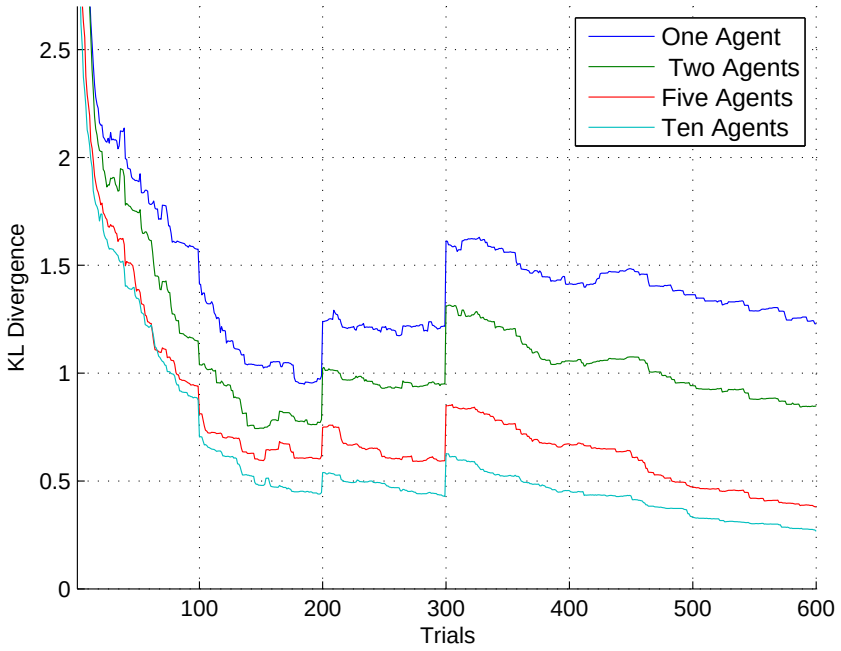


Figure 7.7: Effect of model shift on ϵ-greedy selection ($\epsilon = 0.15$)

- q is the average http response time for the service over the period under consideration. This is indicative as to the quality of the service and is defined in the SLA.
- v is the total number of virtual machines currently allocated to the application.

The learners action set A is to $\{add, remove, maintain\}$ an n number of virtual machines each time. Rewards are determined based on a defined Service Level Agreement (SLA) which is directly related to performance. The overall reward allocated per time step is given by the following equations.

$$C(a) = C_r \times V_a + \{\sum_{i=1}^{v}(C_r \times V_i)\} \tag{7.5}$$

$$H(a) = P_c \times \begin{cases} (1 + \frac{p' - sla}{sla} & if \;\; p' > SLA \\ 0 & else \end{cases} \tag{7.6}$$

$$R(s, a) = C(a) + H(a) \tag{7.7}$$

C_r is the cost of the resource, this is variable depending on the type, specific configuration and region. V_i represents an individual virtual machine instance, with V_a representing the specific virtual machine allocated, deallocated or maintained as a result of action a. H is the

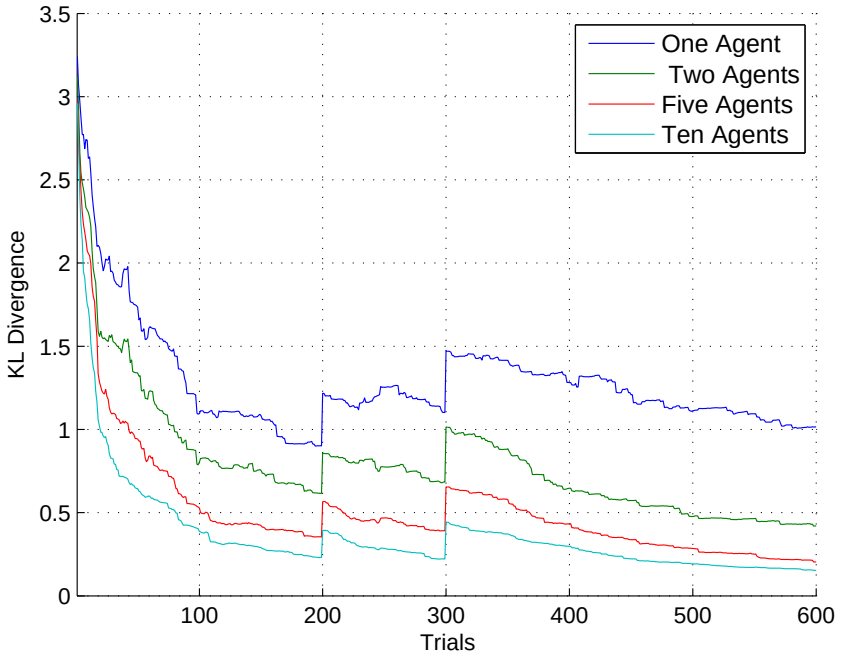


Figure 7.8: Effect of model shift on ϵ-greedy selection ($\epsilon = 0.25$)

overall penalty applied as a result of violating the specified SLA. $P_c \in \Re$ represents a defined penalty constant incurred through violation of the SLA. The SLA comprises a guaranteed level of service response time agreed between the service provider and consumer. The total reward $R(s, a)$ for choosing action a, in s is the combination of the cost of execution and any associated penalties.

Each virtual machine can support a maximum of 100 concurrent requests each time for the application. Thus the response time for the application is calculated by utilising random sampling with replacement over the entire learning period. This problem has a state space 16 times the previous GridWorld example.

User requests are generated via an open loop poisson process with an adjustable mean inter-arrival rate. The origin of the requests are restricted to the 6 countries detailed in figure 4.7 chapter 4 section 4.4. The application response time is proportionally sampled with replacement, as a function of the number of incoming requests and the number of virtual machines currently supporting the application. This value governs the number of user requests that can be handled prior to significant performance degradation. The application response time is split into 50 separate bins. The maximum number of VMs allocated to the application at any given

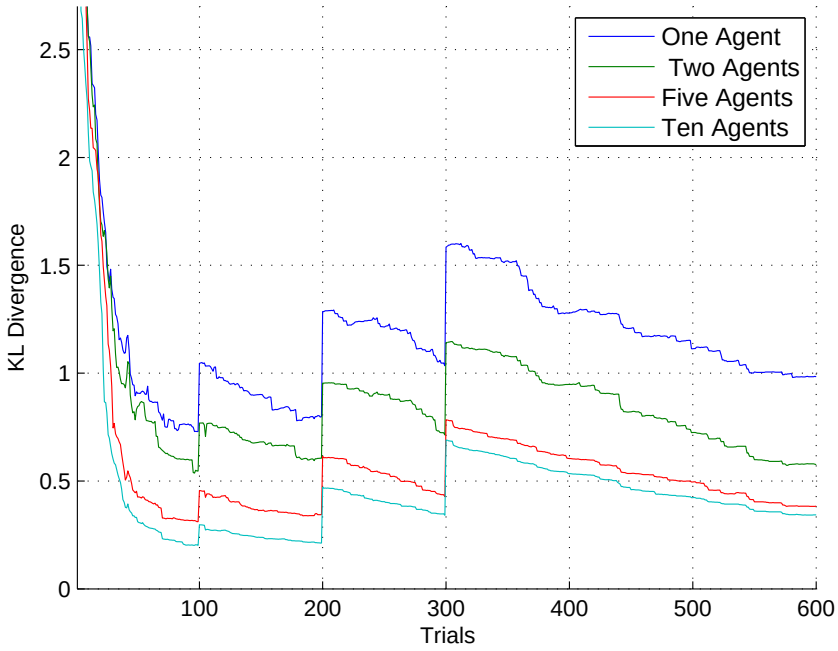


Figure 7.9: Effect of model shift on unbiased sampling

time is restricted to 10.

7.5.2.2 Performance on Virtual Resource Allocation Problem

Figure 7.11 plots the global K-L divergence between "ground truth" and the learned probability estimates over 1000 trials. The results are averaged over ten runs with the learning agent numbers increasing from 2 to 10 agents. As the number of agents increases there is a marked improvement in the distance between the true model parameters and the learned parameters.

Due to the size and nature of this task, the learning challenge is much greater than the previous *Stochastic Gridworld* problem. Figure 7.11 highlights the sizeable adjustments with respect to the approximation of the underlying transition function as the number of agents increase. Clearly parallel learning has a significant impact for this type of real-world problem which presents a much more challenging and dynamic environment for the agent learner. As the model approximations improve so too do the policies derived from this information. This is a validation of our hypothesis that a parallel learning mechanism is capable of reducing policy times considerably as additional agents are added to the system.

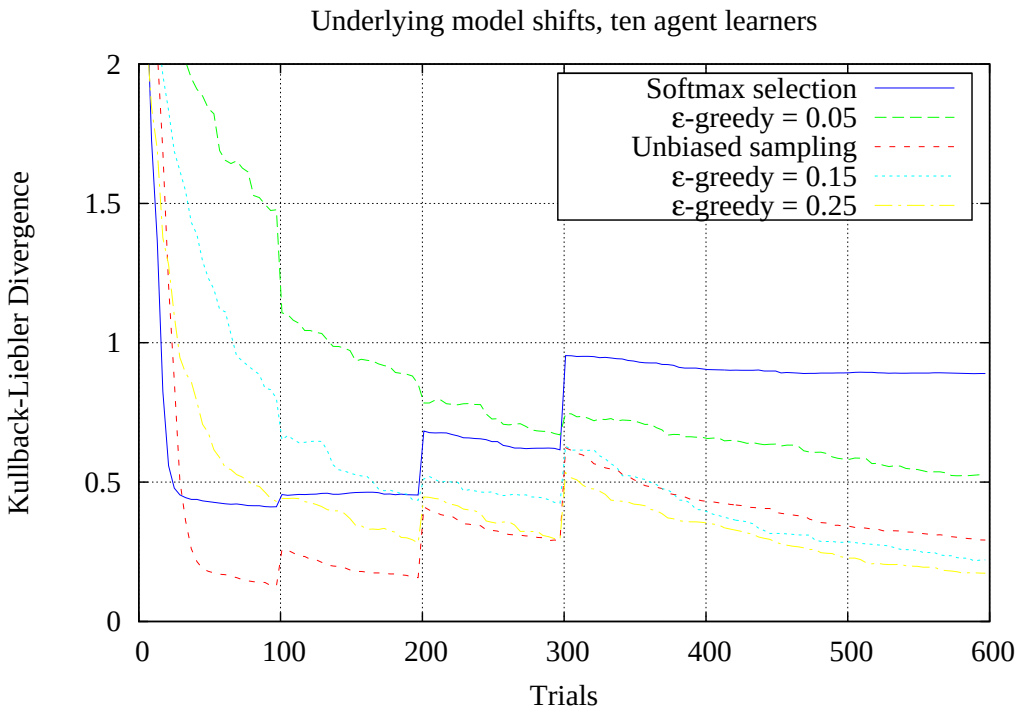


Figure 7.10: Comparison of the performance of the selection strategies for 10 agents in parallel

Figure 7.12 shows the Kullback-Liebler divergence for three individual runs for a single agent attempting to approximate the true transition probabilities. The variation in learning experience can clearly be seen as each agent observes different responses to its actions and adjusts its estimates accordingly. This demonstrates that whilst the agents are learning on the same problem each agent's experience is unique and is what makes learning in parallel so effective.

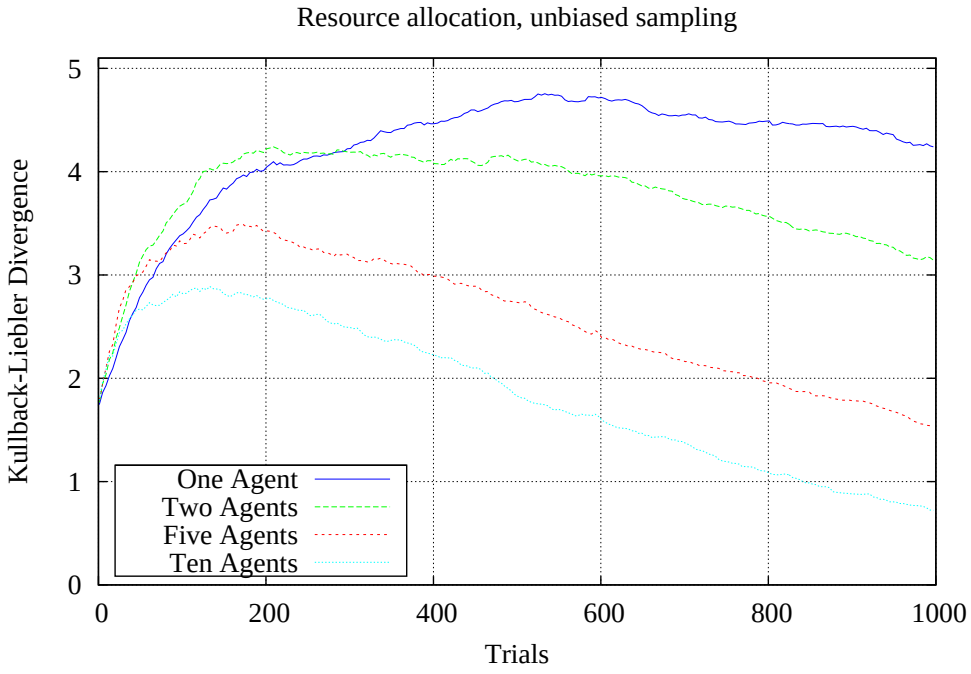


Figure 7.11: Virtual resource allocation, unbiased sampling, multiple agent learners, learning in parallel

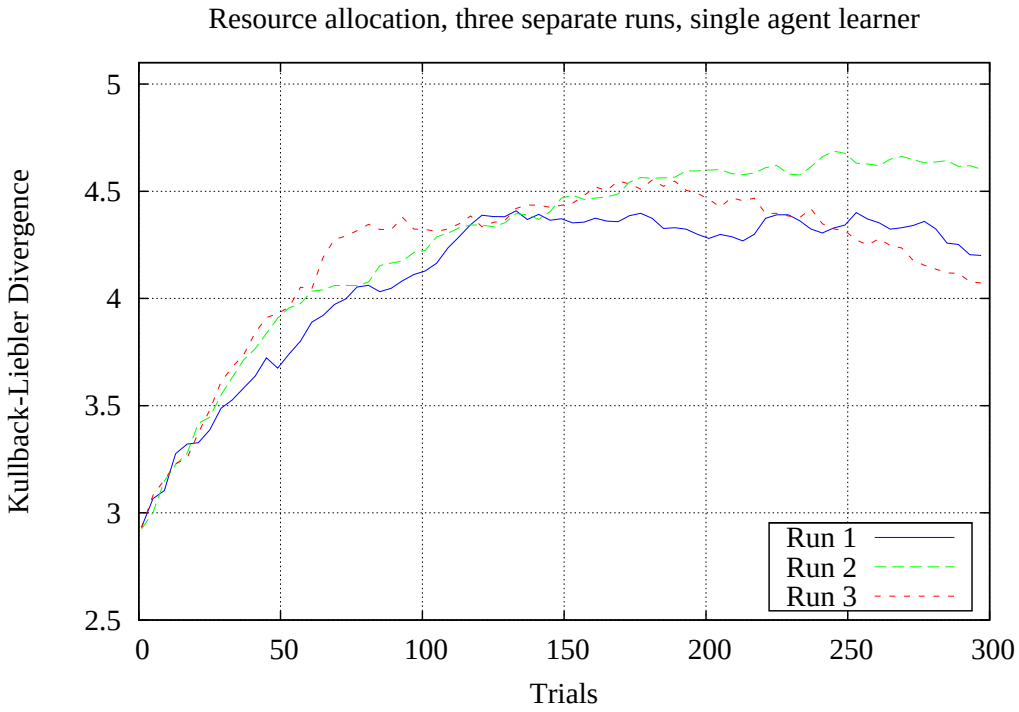


Figure 7.12: A single run of unbiased sampling for 300 trials

Chapter 8

Conclusions

The research presented within this thesis has focused on applying techniques from machine learning known as reinforcement learning, to address two critical issues prevalent within modern day IaaS clouds, *Performance Unpredictability* and *Scaling Quickly*. Previously cited as critical obstacles preventing the greater growth and adoption of cloud computing infrastructures [Armbrust *et al.* 10], this dissertation has proposed a number of different novel solutions utilising learning techniques to address these two obstacles.

To examine the efficacy of our proposed solutions, we carried out extensive empirical evaluations, including simulations of workflow scheduling, dynamic application scaling and stochastic GridWorld domains. Within each of these experimental problems a variety of conditions and settings were simulated. These included, variable incoming workflows and user request arrival rates, variable resource performance and application response times.

Using data models generated from micro-benchmarking real cloud virtual machine instances we were able to simulate variability with respect to performance, allowing us to model the performance unpredictability applied to workflow execution and application scaling in the cloud. The core focus of our empirical analysis and contributions within this thesis are based on three principle areas

- Workflow scheduling
- Dynamic application scaling
- Parallel Bayesian reinforcement learning

Within each of these areas, we conducted an examination into how learning techniques such as reinforcement learning could be applied to solve the principle problems of performance unpredictability and scaling quickly. Throughout the examination of these areas we proposed novel algorithms and architectures to extend upon existing techniques and approaches, providing detailed analysis in the process.

8.1 Workflow Scheduling

Many workflow scheduling solutions from both a meta-heuristic [Yu *et al.* 05, Pandey *et al.* 10, Wu *et al.* 10] and heuristic perspective [Maheswaran *et al.* 99] have previously been proposed. Extending upon these works, this dissertation represents the first attempt at devising a learning architecture suitable for workflow scheduling in clouds. Our approach is capable of assessing the suitability of deploying certain schedules given the present state of the environment. This offers a significant advantage in dynamic environments where the state of the environment is constantly changing. The ability to be able to predict the likely outcomes of certain events allows for greater accuracy with regard to deployments in the field.

Scheduling workflows by mapping individual tasks on to distributed resources is a NP-Complete problem [Ullman 75]. In the past meta-heuristic algorithms have demonstrated effective performance with relatively little computational overhead. For this reason we employed a genetic algorithm with different configurations to evolve solutions to the workflow scheduling problem. The various configurations included different settings for the genetic operators of crossover, mutation and elitism as well as the emphasis placed on the execution estimation times for specific tasks. This provided us with a simple framework which we could use to analyse the performance of our learning architecture at choosing schedules from multiple independent solver agents based on the environmental state. The approach is flexible enough to facilitate the inclusion of a variety of different learning algorithms and scheduling mechanisms to optimise the process even further.

To infer aspects of the hidden underlying transition function we also applied a Bayesian model learning algorithm. Through repeated environmental interactions the approach demonstrated a capacity to build up a close approximation of the hidden underlying model. Once a good

approximation is achieved, the problem can be solved using methods from dynamic programming such as value iteration. Using real data from micro-benchmarking live cloud instances through the *filebench* suite, we emulated variable disk I/O performance which is commonly observed within virtualised platforms. This allowed us to capture some of the performance unpredictability inherent within the underlying infrastructures and generate more realistic simulations. In addition we also directed workflow requests at the system using a variable open loop queuing Poisson process to simulate a realistic incoming workflow request model.

To demonstrate the efficacy of our approach over previous approaches and to demonstrate its viability at sampling the environmental state and choosing the appropriate action each time we conducted a number of experiments. The first experiment evaluated the effects with regard to variable numbers of incoming user workflows and task sizes within the system. By comparing our proposed *MDP-Bayesian* solution against a *genetic algorithm* and a *"Best Resource Selection"* (BRS) scheme we demonstrated significant cost savings over the other approaches. In addition we also demonstrated excellent results with regard to optimising workflow execution from the perspective of both cost and makespan.

The next experiment analysed the system performance with variable deadline constraints imposed upon workflow executions. Whilst outperforming both the *genetic algorithm* and *BRS* approaches for the majority of deadlines (D=90 and D=70), for the lowest deadline constraint of D=50 the *MDP-Bayesian* solution did violate the specified constraint. The reason for this was due to the initial learning period required where the learner explored a number of sub-optimal actions which resulted in poorer performance during this initial learning phase. To address this problem we proposed a solution via offline training where an initial policy is determined via trials of learning offline prior to being deployed on the real system. Once we applied the offline policy to the same experiment we were able to demonstrate excellent performance on the lower deadline constraint ensuring the deadline was not violated and the costs were lower than the corresponding *BRS* solution.

Finally we examined the impact of increasing data sizes within the system and the effects of Bayesian learning. Our analysis demonstrated that as the data sizes for the various workflows increased across the system, the costs incurred by the *BRS* approach in contrast to the *MDP-Bayesian* solution were significantly higher on average. With regard to Bayesian learning we demonstrating promising initial performance with regard to approximating the true transition

probabilities within the simulated dynamic environment.

In summary we have presented our initial architecture for a learning oriented cloud workflow scheduler. The ability to reason over environmental uncertainty through Bayesian learning is a significant advancement over previous approaches. In addition the proposed state action space formalism represents an additional novelty with regard to handling temporal fluctuations and performance unpredictability within the environment. In addition, learning techniques such as Bayesian model learning also allow the system to "keep up" with the constantly changing underlying environment.

8.2 Dynamic Application Scaling

Dynamically scaling applications in computational clouds in response to changes in user requests or underlying performance variations, presents a key challenge for resource planning techniques. In an effort to optimally scale the resources quickly and cater for fluctuations in performance such as application response time while maintaining quality of service commitments, approaches must be adaptive and capable of responding in a timely manner to achieve the objectives in light of changing conditions.

Cognisant of this fact this dissertation proposed a dynamic scaling approach capable of automatically generating scaling solutions in response to changing environmental conditions. Using a reinforcement learning algorithm known as Q-learning we successfully demonstrated excellent performance whilst allocating resources to support the simulated application. One of the contributions of our solution was devising a novel state action space formalism capable of dynamically allocating resources in response to changing conditions and making predictions about the temporal variations present within computational clouds. By sensing the state of the environment at any given moment in time our solution can make more informed decisions about the potential performance of the resources it allocates to support the application.

Utilising reinforcement learning techniques to automatically control the scaling of virtual resources offers advantages with respect to reliability, adaptability and autonomy. Threshold based approaches have predominated industry scaling solutions such as Amazon Auto Scaling and RightScale. However the development of effective thresholds which govern allocation

decisions requires extensive domain and application knowledge. Learning mechanisms are capable of operating within the environment without any prior knowledge. This represents a significant advantage over more domain specific measures such as threshold based models.

In addition to our solution we also proposed a parallel learning framework to mitigate against the effects of curse of dimensionality problems which are often associated with real world learning problems. With this approach each agent learner attempts to learn optimal policies independently on the same task and share their experiences with all the other agents learning within the system. The combination of parallel agent learning with our novel state space solution, enables advanced uncertainty reasoning capabilities over cloud resources.

8.3 Parallel Bayesian Reinforcement Learning

Bayesian reinforcement learning is a technique which combines Bayesian inference with reinforcement learning to make better use of the knowledge acquired through environmental interactions. Whilst mechanisms such as Q-learning can operate on large problems from a computationally efficient perspective, it does not always make the most efficient use of the information available to it throughout the learning process.

Bayesian reinforcement learning provides mechanisms to address this in a reinforcement learning context, producing better solutions than Q-learning or other temporal difference mechanisms. However whilst it can produce better solutions over these approaches there are some issues that need to be addressed. Determining the Bayesian optimal action each time involves the computation of an intractable integral across all possible future states and actions. For this reason many authors have proposed a variety of sampling mechanisms which can reduce the sample space to ensure good performance combined with a smaller computational overhead.

Cognisant of this fact we successfully applied a parallel learning mechanism to Bayesian reinforcement learning concurrently analysing the effects on a variety of different action selection strategies. For each selection strategy we examined how it performs in a parallel context and demonstrated the significant performance speedups possible using this approach. Using the Kullback-Liebler divergence metric we were able to share probability density function estimates amongst agents learning in parallel on the same problem. To the best of our knowledge

this is the first time that such as technique has been devised and we have provided a number of empirical results demonstrating the performance increases possible.

We carried out a number of empirical evaluations using a stochastic GridWorld example and a virtual machine resource allocation problem, proving in both scenarios the convergence speedups made possible through the described parallel learning approach. In addition we proved the efficacy of the technique at handling model changes within the environment which represents a significant challenge facing reinforcement learning algorithms, where often a significant model shift can result in the agent learner having to learn from scratch again. The approach we have presented is also general enough to apply to any missing probabilistic component of the underlying model, such as the distribution of rewards or transition functions.

8.4 Future Work

Arising from the work presented in this thesis a number of possible advancements could be made to further enhance the techniques and solutions proposed. One of the first extensions relates to the fact that the empirical evaluations of the techniques proposed in this work were conducted in simulated environments with real data models. It would be interesting to investigate the performance of scheduling workflows or allocating resources to support application scaling in a real IaaS setting. This would allow for an evaluation of performance with actual live resources and real applications.

Within the context of workflow scheduling we evaluated our hypothesis by deploying solver agents which utilise a *genetic algorithm* to generate schedules. Whilst each solver was configured different which ensured diversity in the population of evolved schedules, it would be an interesting extension to analyse the performance of different algorithmic approaches. Techniques such as Particle Swarm Optimisation, Ant Colony Optimisation or List scheduling schemes could be integrated into the system. Arising from this, some interesting hypotheses could be investigated, such as which techniques perform best under which environmental conditions or for which application types such as data intensive, memory intensive, CPU intensive or other type, does a particular approach perform best on. It would be interesting to analyse how the present observed sensory data would perform given these new approaches. Could the inclusion of additional state variables enhance the performance further?

From an application scaling perspective one important question often arises in relation to the cost-benefit [Vaquero *et al.* 11] of the execution of resource dimensioning actions. For example what effect will an improvement in response time have on sales for my e-commerce website. It is conceivable that the inclusion of additional resources to boost performance may not have a corresponding impact on sales. Thus an important extension from a scaling perspective would be to frame the returns perhaps not from an application response time perspective but from a more business centric perspective.

Within Bayesian reinforcement learning many selection strategies have been proposed by the research community. Within this thesis we analysed the performance of three strategies, *epsilon greedy*, *unbiased sampling* and *softmax selection*. Additional sampling strategies have also been proposed such as, the Probably Optimistic Transition [Kawaguchi & Araya 13b], the Variance-Based Reward Bonus [Sorg *et al.* 12] and the Bayesian Exploration Bonus [Kolter & Ng 09]. Further analyses of the application of parallel methods to Bayesian reinforcement learning with these selection strategies could provide interesting observations. If one were to utilise multiple agent learners, each with their own separate selection strategies, could better solutions be found through a combination of learning strategies?

8.5 Summary

At the beginning of this dissertation we proposed a number of hypotheses

- **H1** : Through analysis of the environmental state and in particular observing temporal changes in performance, one can improve the schedule selection process to achieve a more optimal solution.
- **H2** : Employing multiple learners which share their experiences and are capable of observing specific environmental conditions, will improve performance over previous solutions for dynamically scaling applications in the cloud.
- **H3** : Using the Kullback-leibler divergence will render the sharing of probability density estimates amongst agent learners possible, and will result in a reduction in learning convergence times overall.

Each results chapter dealt specifically with each hypothesis defining the contributions of our research as we examined each hypothesis. Within this context the contributions of each of these chapters with respect to the proposed hypotheses are

- **C1** : Through a combination of evolutionary computation and reinforcement learning, we proposed a novel workflow scheduling framework capable of meeting defined user constraints, in cost and makespan. Through a Bayesian model learning framework the approach is capable of updating its estimates of the underlying transition functions without prior knowledge.
- **C2** : Our second contribution proposes a parallel learning architecture capable of scaling applications in the cloud in response to the changing conditions of resource performance and incoming user requests. We demonstrated the benefits of parallel learning mechanisms at reducing the time to learn good policies.
- **C3** : Using the Kullback-Liebler divergence mechanism we successfully facilitated information sharing amongst learning agents. We also proved the convergence speedups possible through our parallel Bayesian reinforcement learning algorithm when operating on both a *Stochasic GridWorld* problem and a *Cloud Resource Allocation*.

Bibliography

[Aarts & Lenstra 97] Emile Aarts and Jan K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[Abramovici *et al.* 92] Alex Abramovici, William E Althouse, Ronald WP Drever, Yekta Gürsel, Seiji Kawamura, Frederick J Raab, David Shoemaker, Lisa Sievers, Robert E Spero, Kip S Thorne, *et al*. Ligo: The laser interferometer gravitational-wave observatory. *Science*, 256(5055):325–333, 1992.

[Akhter & Roberts 06] Shameem Akhter and Jason Roberts. Multi-core programming. 2006.

[Amazon 13] Amazon. Amazon elastic compute cloud (amazon ec2), http://aws.amazon.com/ec2/. 2013.

[Amazon-WS 13] Amazon-WS. Amazon web services, http://aws.amazon.com/. 2013.

[Amin *et al.* 04] Kaizar Amin, Gregor Von Laszewski, Mihael Hategan, Nestor J Zaluzec, Shawn Hampton, and Albert Rossi. Gridant: A client-controllable grid workflow system. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10–pp. IEEE, 2004.

[Araújo *et al.* 01] Mário César Ugulino Araújo, Teresa Cristina Bezerra Saldanha, Roberto Kawakami Harrop Galvão, Takashi Yoneyama, Henrique Caldas Chame, and Valeria Visani. The successive projections algorithm for variable selection in spectroscopic multicomponent analysis. *Chemometrics and Intelligent Laboratory Systems*, 57(2):65–73, 2001.

[Armbrust *et al.* 09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A

berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[Armbrust *et al.* 10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, *et al*. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[AutoScale 13] AutoScale. Autoscale, http://aws.amazon.com/autoscaling/. 2013.

[Baird & *et al.* 95] L. Baird *et al*. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning-International Workshop then Conference*, pages 30–37. Morgan Kaufmann Publishers, Inc., 1995.

[Baker & Ayechew 03] Barrie M Baker and MA Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[Banerjee *et al.* 11] Prith Banerjee, Richard Friedrich, Cullen Bash, Patrick Goldsack, Bernardo A Huberman, John Manley, Chandrakant Patel, Partha Ranganathan, and Alistair Veitch. Everything as a service: Powering the new information economy. *Computer*, 44(3):36–43, 2011.

[Barham *et al.* 03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[Barrett & Weld 94] Anthony Barrett and Daniel S Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[Barrett *et al.* 11] Enda Barrett, Enda Howley, and Jim Duggan. A learning architecture for scheduling workflow applications in the cloud. In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, pages 83–90. IEEE, 2011.

[Barrett *et al.* 12] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 2012.

[Bellman 57] Richard E. Bellman. *Dynamic Programming*, volume 1. Dover Publications, 1957.

[Benson *et al.* 11] Theophilus Benson, Aditya Akella, Anees Shaikh, and Sambit Sahu. Cloudnaas: a cloud networking platform for enterprise applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 8. ACM, 2011.

[Bernheim & Whinston 86] B Douglas Bernheim and Michael D Whinston. Menu auctions, resource allocation, and economic influence. *The quarterly journal of economics*, 101(1):1–31, 1986.

[Bernholdt *et al.* 05] David Bernholdt, Shishir Bharathi, David Brown, Kasidit Chanchio, Meili Chen, Ann Chervenak, Luca Cinquini, Bob Drach, Ian Foster, Peter Fox, *et al*. The earth system grid: Supporting the next generation of climate modeling research. *Proceedings of the IEEE*, 93(3):485–495, 2005.

[Bernstein *et al.* 05] Daniel S Bernstein, Eric A Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized pomdps. In *Workshop on Multiagent Planning and Scheduling*, pages 52–57, 2005.

[Berriman *et al.* 04] GB Berriman, JC Good, AC Laity, A Bergou, J Jacob, DS Katz, E Deelman, C Kesselman, G Singh, M-H Su, *et al*. Montage: A grid enabled image mosaic service for the national virtual observatory. In *Astronomical Data Analysis Software and Systems (ADASS) XIII*, volume 314, page 593, 2004.

[Berriman *et al.* 10] G Bruce Berriman, Ewa Deelman, Paul Groth, and Gideon Juve. The application of cloud computing to the creation of image mosaics and management of their provenance. In *SPIE Astronomical Telescopes and Instrumentation: Observational Frontiers of Astronomy for the New Decade*, pages 77401F–77401F. International Society for Optics and Photonics, 2010.

[Bertsekas & Tsitsiklis 89] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[Bertsekas & Tsitsiklis 96] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996.

[Bhardwaj *et al.* 10] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[Bhargava & Sundaresan 04] Hemant K Bhargava and Shankar Sundaresan. Computing as utility: Managing availability, commitment, and pricing through contingent bid auctions. *Journal of Management Information Systems*, 21(2):201–227, 2004.

[Bianchi *et al.* 09] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: an international journal*, 8(2):239–287, 2009.

[Bit11] Cloud performance from the end user. Technical report, 2011.

[Blaha *et al.* 08] P Blaha, K Schwarz, GKH Madsen, D Kvasnicka, J Luitz, and K Schwarz. An augmented plane wave plus local orbitals program for calculating crystal properties: Wien2k user's guide. *Techn. Universitat Wien*, 2008.

[Blum & Furst 97] Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1):281–300, 1997.

[Blum & Roli 03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[Blythe & Veloso 97] Jim Blythe and Manuela Veloso. Analogical replay for efficient conditional planning. *Proceedings of the American Association of Arti. cial Intelligence (AAAI-97)*, pages 668–673, 1997.

[Blythe *et al.* 05] Jim Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 759–767. IEEE, 2005.

[Bock & Lengauer 08] Christoph Bock and Thomas Lengauer. Computational epigenetics. *Bioinformatics*, 24(1):1–10, 2008.

[Boyan & Littman 94] Justin A Boyan and Michael L Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, pages 671–671, 1994.

[Boyan & Moore 95] Justin Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995.

[Brandic *et al.* 05] Ivona Brandic, Siegfried Benkner, Gerhard Engelbrecht, and Rainer Schmidt. Qos support for time-critical grid workflow applications. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. IEEE, 2005.

[Brandic *et al.* 08] Ivona Brandic, Sabri Pllana, and Siegfried Benkner. Specification, planning, and execution of qos-aware grid workflows within the amadeus environment. *Concurrency and Computation: Practice and Experience*, 20(4):331–345, 2008.

[Braun *et al.* 01] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, *et al*. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[Buyya & Murshed 02] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[Buyya 09] Rajkumar Buyya. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *CCGRID*, page 1, 2009.

[Calheiros *et al.* 09] Rodrigo N. Calheiros, Rajiv Ranjan, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *CoRR*, abs/0903.2525, 2009.

[Canyurt & Öztürk 06] Olcay Ersel Canyurt and Harun Kemal Öztürk. Three different applications of genetic algorithm (ga) search techniques on oil demand estimation.

Energy conversion and management, 47(18):3138–3148, 2006.

[Cao *et al.* 03] Junwei Cao, Stephen A Jarvis, Subhash Saini, and Graham R Nudd. Gridflow: Workflow management for grid computing. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 198–205. IEEE, 2003.

[Carpenter & Grossberg 88] Gail A. Carpenter and Stephen Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3):77–88, 1988.

[Casanova *et al.* 00] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, and Francine Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pages 349–363. IEEE, 2000.

[Cassandra *et al.* 97] Anthony Cassandra, Michael L Littman, and Nevin L Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 54–61. Morgan Kaufmann Publishers Inc., 1997.

[Catlett & *et al.* 02] Charlie Catlett *et al*. The philosophy of teragrid: building an open, extensible, distributed terascale facility. In *Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002*, pages 5–5, 2002.

[Cedexis 13] Cedexis. Cedexis, http://www.cedexis.com/. 2013.

[Chen & Zhang 09] Wei-Neng Chen and Jun Zhang. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(1):29–43, 2009.

[Choudhary 07] Vidyanand Choudhary. Software as a service: Implications for investment in software development. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 209a–209a. IEEE, 2007.

[Chow & Lin 71] Gregory C Chow and An-loh Lin. Best linear unbiased interpolation, distribution, and extrapolation of time series by related series. *The review of Economics and Statistics*, 53(4):372–375, 1971.

[Christofides 75] Nicos Christofides. *Graph theory: An algorithmic approach (Computer science and applied mathematics)*. Academic Press, Inc., 1975.

[Chu *et al.* 07] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, Cambridge, MA, 2007.

[Cockcroft 11] Adrian Cockcroft. Performance architecture for cloud. In *http://www.cloudconnectevent.com/cloud-computing-conference/cloud-performance-summit.php*, 2011.

[Colorni *et al.* 91] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, *et al*. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991.

[Crites & Barto 96] Robert Crites and Andrew Barto. Improving elevator performance using reinforcement learning. pages 1017–1023, 1996.

[Daw 03] Nathaniel D Daw. *Reinforcement learning models of the dopamine system and their behavioral implications*. Unpublished PhD thesis, University College London, 2003.

[Dean & Ghemawat 08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[Dearden *et al.* 99] Richard Dearden, Nir Friedman, and David Andre. Model based bayesian exploration. In *Proceedings of the fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 150–159. Morgan Kaufmann Publishers Inc., 1999.

[Deb *et al.* 00] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Lecture notes in computer science*, 1917:849–858, 2000.

[Deelman *et al.* 03a] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard

Cavanaugh, *et al*. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[Deelman *et al.* 03b] Ewa Deelman, Raymond Plante, Carl Kesselman, Gurmeet Singh, Mei-Hui Su, Gretchen Greene, Robert Hanisch, Niall Gaffney, Antonio Volpicelli, James Annis, *et al*. Grid-based galaxy morphology analysis for the national virtual observatory. In *Supercomputing, 2003 ACM/IEEE Conference*, pages 47–47. IEEE, 2003.

[Deelman *et al.* 05] Ewa Deelman, Gurmeet Singh, Mei hui Su, James Blythe, A Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13:219–237, 2005.

[Deelman *et al.* 08] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.

[Della Croce *et al.* 95] Federico Della Croce, Roberto Tadei, and Giuseppe Volta. A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1):15–24, 1995.

[Denardo 03] E.V. Denardo. *Dynamic programming: models and applications*. Dover Publications, 2003.

[Dorigo 92] Marco Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[Doshi *et al.* 05] Prashant Doshi, Richard Goodwin, Rama Akkiraju, and Kunal Verma. Dynamic workflow composition using markov decision processes. *International Journal of Web Services Research*, 2:1–17, 2005.

[Draper *et al.* 94] Denise Draper, Steve Hanks, and Daniel S Weld. Probabilistic planning with information gathering and contingent execution. In *AIPS*, pages 31–36. Citeseer, 1994.

[Dutreilh *et al.* 10] Xavier Dutreilh, Nicolas Rivierre, Aurélien Moreau, Jacques Malenfant, and Isis Truck. From data center resource allocation to control theory and back. In

Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pages 410–417. IEEE, 2010.

[Dutreilh *et al.* 11] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, pages 67–74, 2011.

[Eddy 04] Sean R Eddy. What is dynamic programming? *Nature biotechnology*, 22(7):909–910, 2004.

[Ekanayake & Fox 10] Jaliya Ekanayake and Geoffrey Fox. High performance parallel computing with clouds and cloud technologies. pages 20–38, 2010.

[Emeakaroha *et al.* 11] Vincent C Emeakaroha, Paweł P Łabaj, Michael Maurer, Ivona Brandic, and David P Kreil. Optimizing bioinformatics workflows for data analysis using cloud management techniques. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, pages 37–46. ACM, 2011.

[Fahringer *et al.* 05] Thomas Fahringer, Alexandru Jugravu, Sabri Pllana, Radu Prodan, Clovis Seragiotto, and Hong-Linh Truong. Askalon: a tool set for cluster and grid computing. *Concurrency and Computation: Practice and Experience*, 17(2-4):143–169, 2005.

[Fallenbeck *et al.* 06] Niels Fallenbeck, Hans-Joachim Picht, Matthew Smith, and Bernd Freisleben. Xen and the art of cluster scheduling. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, page 4. IEEE Computer Society, 2006.

[Fang *et al.* 93] Hsiao-Lan Fang, Peter Ross, and David Corne. *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems*. University of Edinburgh, Department of Artificial Intelligence, 1993.

[Fikes & Nilsson 72] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[Fisher & Thompson 63] Henry Fisher and Gerald L Thompson. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, pages 225–251, 1963.

[Foster & Kesselman 03] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Morgan Kaufmann, 2003.

[Foster *et al.* 01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International journal of high performance computing applications*, 15(3):200–222, 2001.

[Foster *et al.* 08] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[Frey *et al.* 02] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[Friedman & Singer 99] Nir Friedman and Yoram Singer. Efficient bayesian parameter estimation in large discrete domains. *Advances in Neural Information Processing Systems 11: Proceedings of the 1998 Conference*, 11:417, 1999.

[Friedman 94] Jerome H Friedman. An overview of predictive learning and function approximation. In *From Statistics to Neural Networks*, pages 1–61. Springer, 1994.

[Galstyan *et al.* 04] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource allocation in the grid using reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1314–1315. IEEE Computer Society, 2004.

[Geladi & Kowalski 86] Paul Geladi and Bruce R Kowalski. Partial least-squares regression: a tutorial. *Analytica chimica acta*, 185:1–17, 1986.

[Gendreau *et al.* 94] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290, 1994.

[Gharooni-fard *et al.* 10] Golnar Gharooni-fard, Fahime Moein-darbari, Hossein Deldari, and Anahita Morvaridi. Scheduling of scientific workflows using a chaos-genetic algorithm. *Procedia Computer Science*, 1(1):1445–1454, 2010.

[Gil *et al.* 07] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers.

Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, 2007.

[Goldberg & Deb 91] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[Goldberg & Holland 88] D.E. Goldberg and J.H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[Goldberg 89] David E Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.

[Gonçalves *et al.* 05] José Fernando Gonçalves, Jorge José de Magalhães Mendes, and Mauricio GC Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, 167(1):77–95, 2005.

[Greschler & Mangan 02] David Greschler and Tim Mangan. Networking lessons in delivering software as a servicepart i. *International Journal of Network Management*, 12(5):317–321, 2002.

[Grounds & Kudenko 08] Matthew Grounds and Daniel Kudenko. Parallel reinforcement learning with linear function approximation. pages 60–74, 2008.

[Grzes & Kudenko 10] Marek Grzes and Daniel Kudenko. Online learning of shaping rewards in reinforcement learning. *Neural Networks*, 23(4):541 – 550, 2010. ¡ce:title¿The 18th International Conference on Artificial Neural Networks, ICANN 2008¡/ce:title¿.

[Guan *et al.* 11] Qiang Guan, Ziming Zhang, and Song Fu. Proactive failure management by integrated unsupervised and semi-supervised learning for dependable cloud systems. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 83–90. IEEE, 2011.

[Harberger 54] Arnold C Harberger. Monopoly and resource allocation. *The American Economic Review*, 44(2):77–87, 1954.

[Hartigan & Wong 79] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[Hastie *et al.* 05] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[Heckerman 08] David Heckerman. *A tutorial on learning with Bayesian networks*. Springer, 2008.

[Holland 92] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence*. MIT press, 1992.

[Hu & Wellman 03] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *The Journal of Machine Learning Research*, 4:1039–1069, 2003.

[Huang & Sundararajan 05] Ke-Wei Huang and Arun Sundararajan. Pricing models for on-demand computing. *CeDER Working Paper No. 05-26*, 2005.

[Hwang 08] Kai Hwang. Massively distributed systems: From grids and p2p to clouds. In *Advances in Grid and Pervasive Computing*, pages 1–1. Springer, 2008.

[Illumina 11] Illumina. Illumina: , http://www.illumina.com. 2011.

[Iosup *et al.* 11] Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Trans. Parallel Distrib. Syst.*, 22:931–945, June 2011.

[Jaeger *et al.* 08] Paul T Jaeger, Jimmy Lin, and Justin M Grimes. Cloud computing and information policy: Computing in a policy cloud? *Journal of Information Technology & Politics*, 5(3):269–283, 2008.

[Jain *et al.* 84] Raj Jain, Dah-Ming Chiu, and William R Hawe. *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*. Eastern Research Laboratory, Digital Equipment Corporation, 1984.

[Jang *et al.* 04] S Jang, Xingfu Wu, Valerie Taylor, Gaurang Mehta, Karan Vahi, and Ewa Deelman. Using performance prediction to allocate grid resources. *Texas A&M University, College Station, TX, GriPhyN Technical Report*, 25, 2004.

[Jiao & Wang 00] Licheng Jiao and Lei Wang. A novel genetic algorithm based on immunity. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(5):552–561, 2000.

[Joachims 98] Thorsten Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[Juve & Deelman 10] Gideon Juve and Ewa Deelman. Scientific workflows and clouds. *ACM Crossroads*, 16(3):14–18, 2010.

[Kaelbling 94] Leslie Pack Kaelbling. Associative reinforcement learning: Functions ink-dnf. *Machine Learning*, 15(3):279–298, 1994.

[Kaelbling *et al.* 96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Karlsson 01] Lars Karlsson. Conditional progressive planning under uncertainty. In *IJCAI*, pages 431–438. Citeseer, 2001.

[Kasahara & Narita 84] Hironori Kasahara and Seinosuke Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers*, 33(11):1023–1029, 1984.

[Kawaguchi & Araya 13a] K. Kawaguchi and M. Araya. A Greedy Approximation of Bayesian Reinforcement Learning with Probably Optimistic Transition Model. *ArXiv e-prints*, March 2013.

[Kawaguchi & Araya 13b] Kenji Kawaguchi and Mauricio Araya. A greedy approximation of bayesian reinforcement learning with probably optimistic transition model. *CoRR*, abs/1303.3163, 2013.

[Kennedy & Eberhart 95] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.

[Kivity *et al.* 07] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[Kiyanclar 05] Nadir Kiyanclar. A survey of virtualization techniques focusing on secure on-demand cluster computing. *arXiv preprint cs/0511010*, 2005.

[Knowles & Corne 99] Joshua Knowles and David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1. IEEE, 1999.

[Koh *et al.* 07] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. An analysis of performance interference effects in virtual environments. In *In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS*, 2007.

[Kolter & Ng 09] J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 513–520. ACM, 2009.

[Koza *et al.* 99] John R Koza, Forrest H Bennett III, and Oscar Stiffelman. *Genetic programming as a Darwinian invention machine*. Springer, 1999.

[Kramer & Sangiovanni-Vincentelli 89] Alan H Kramer and A Sangiovanni-Vincentelli. Efficient parallel learning algorithms for neural networks. pages 40–48, 1989.

[Kretchmar 02] R Matthew Kretchmar. Parallel reinforcement learning. 2002.

[Kumar & Varaiya 86] P. R. Kumar and P. P. Varaiya. *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[Kwok & Ahmad 99] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[Laporte 92] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[Lawton 08] George Lawton. Developing software online with platform-as-a-service technology. *Computer*, 41(6):13–15, 2008.

[Lee & Hughes 10] Kevin Lee and Danny Hughes. System architecture directions for tangible cloud computing. In *Cryptography and Network Security, Data Mining and*

Knowledge Discovery, E-Commerce & Its Applications and Embedded Systems (CDEE), 2010 First ACIS International Symposium on, pages 258–262. IEEE, 2010.

[Li & Schuurmans 12] Y. Li and D. Schuurmans. Mapreduce for parallel reinforcement learning. *Recent Advances in Reinforcement Learning*, pages 309–320, 2012.

[Lim *et al.* 10] Harold C Lim, Shivnath Babu, and Jeffrey S Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, pages 1–10. ACM, 2010.

[Littman & Boyan 93] Michael Littman and Justin Boyan. A distributed reinforcement learning scheme for network routing. pages 45–51, 1993.

[Littman 94] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, page 163, 1994.

[Littman 96] Michael Lederman Littman. Algorithms for sequential decision making. In *Ph.D. Thesis, Department of Computer Science, Brown University*, 1996.

[Liu *et al.* 11] Hongliang Liu, Enda Howley, and Jim Duggan. Particle swarm optimisation with gradually increasing directed neighbourhoods. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 29–36. ACM, 2011.

[Low *et al.* 12] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[Ludäscher *et al.* 06] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[Luss 82] Hanan Luss. Operations research and capacity expansion problems: A survey. *Operations Research*, 30(5):907–947, 1982.

[Maechling *et al.* 05] Philip Maechling, Hans Chalupsky, Maureen Dougherty, Ewa Deelman, Yolanda Gil, Sridhar

Gullapalli, Vipin Gupta, Carl Kesselman, Jihic Kim, Gaurang Mehta, Brian Mendenhall, Thomas Russ, Gurmeet Singh, Marc Spraragen, Garrick Staples, and Karan Vahi. Simplifying construction of complex workflows for non-expert users of the southern california earthquake center community modeling environment. *SIGMOD Rec.*, 34(3):24–30, September 2005.

[Maheswaran *et al.* 99] Muthucumaru Maheswaran, Shoukat Ali, HJ Siegal, Debra Hensgen, and Richard F Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 30–44. IEEE, 1999.

[Majercik & Littman 99] Stephen M Majercik and Michael L Littman. Contingent planning under uncertainty via stochastic satisfiability. In *AAAI/IAAI*, pages 549–556, 1999.

[Mandal *et al.* 05] Anirban Mandal, Ken Kennedy, Charles Koelbel, Gabriel Marin, John Mellor-Crummey, Bo Liu, and Lennart Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 125–134. IEEE, 2005.

[Martin 67] James John Martin. Bayesian decision problems and markov chains. 1967.

[Mayer *et al.* 03] Anthony Mayer, Steve McGough, Nathalie Furmento, William Lee, Steven Newhouse, and John Darlington. Iceni dataflow and workflow: Composition and scheduling in space and time. In *UK e-Science All Hands Meeting*, volume 634, page 627, 2003.

[McGinley *et al.* 11] Brian McGinley, John Maher, Colm O'Riordan, and Fearghal Morgan. Maintaining healthy population diversity using adaptive crossover, mutation, and selection. *IEEE Trans. Evolutionary Computation*, 15(5):692–714, 2011.

[McGough *et al.* 04] Stephen McGough, Laurie Young, Ali Afzal, Steven Newhouse, and John Darlington. Workflow enactment in iceni. In *UK e-Science All Hands Meeting*, pages 894–900, 2004.

[Mell & Grance 11] Peter Mell and Timothy Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800:145, 2011.

[Melo *et al.* 08] F.S. Melo, S.P. Meyn, and M.I. Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*, pages 664–671. ACM, 2008.

[Miller & Goldberg 95] Brad L Miller and David E Goldberg. Genetic algorithms, tournament selection, and the e ects of noise. *Urbana*, 51:61801, 1995.

[Mitchell 97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[Monahan 82] George E Monahan. State of the arta survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[Moore & Atkeson 93] Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

[Moran *et al.* 11] Daniel Moran, Luis M Vaquero, and Fermin Galan. Elastically ruling the cloud: specifying application's behavior in federated clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 89–96. IEEE, 2011.

[Muniswamy 09] V.V. Muniswamy. *Design And Analysis Of Algorithms*. I. K. International Pvt Ltd, 25, Green Park Extn. Uphaar Cinema Market,. New Delhi - 110 016, India, 1 edition, 2009.

[Murtagh 83] Fionn Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[Nadgowda & Sion 10] Shripad J Nadgowda and Radu Sion. Cloud performance benchmark series. Technical report, Technical Report, Cloud computing center of excellence, Stony Brook, University, New York, 2010.

[Nakano & Yamada 91] Ryohei Nakano and Takeshi Yamada. Conventional genetic algorithm for job shop problems. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 474–479. Morgan Kauffman, San Mateo, CA, 1991.

[Nudd *et al.* 00] Graham R Nudd, Darren J Kerbyson, Efstathios Papaefstathiou, Stewart C Perry, John S Harper, and Daniel V Wilcox. Pacea toolset for the performance

prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[Oh & Wu 04] Jaewon Oh and Chisu Wu. Genetic-algorithm-based real-time task scheduling with multiple goals. *J. Syst. Softw.*, 71:245–258, May 2004.

[Oinn *et al.* 04] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, *et al*. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[Onder & Pollack 97] Nilufer Onder and Martha E Pollack. Contingency selection in plan generation. In *Recent Advances in AI Planning*, pages 364–376. Springer, 1997.

[Osman 93] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(4):421–451, 1993.

[Ozturk *et al.* 05] Harun Kemal Ozturk, Halim Ceylan, Olcay Ersel Canyurt, and Arif Hepbasli. Electricity estimation using genetic algorithm approach: a case study of turkey. *Energy*, 30(7):1003–1012, 2005.

[P. Jog & Gucht 89] J. Y. Suh P. Jog and D. V. Gucht. The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman. 1989.

[Padala *et al.* 07] Pradeep Padala, Kang G. Shin, Xiaoyun Zhu Mustafa, Uysal Zhikui Wang, and Sharad Singhal Arif. Adaptive control of virtualized resources in utility computing environments. In *In Proceedings of the European Conference on Computer Systems*, pages 289–302, 2007.

[Padala *et al.* 09] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 13–26, New York, NY, USA, 2009. ACM.

[Panda *et al.* 09] Biswanath Panda, Joshua S Herbach, Sugato Basu, and Roberto J Bayardo. Planet: massively parallel

learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437, 2009.

[Pandey *et al.* 10] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 400–407. IEEE, 2010.

[Pao 89] Yohhan Pao. Adaptive pattern recognition and neural networks. 1989.

[Perez *et al.* 08] Julien Perez, Cécile Germain-Renaud, Balazs Kégl, and Charles Loomis. Grid differentiated services: A reinforcement learning approach. In *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*, pages 287–294. IEEE, 2008.

[Peters *et al.* 03] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003.

[Pineau *et al.* 03] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *International joint conference on artificial intelligence*, volume 18, pages 1025–1032. Lawrence Erlbaum Associates Ltd, 2003.

[Poole & Mackworth 10] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[Pordes *et al.* 07] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, *et al.* The open science grid. In *Journal of Physics: Conference Series*, volume 78, page 012057. IOP Publishing, 2007.

[Poupart *et al.* 06] Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete bayesian reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 697–704. ACM, 2006.

[Powell 07] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[Pu *et al.* 10] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, and Calton Pu. Understanding performance interference of i/o workload in virtualized cloud environments. pages 51–58, 2010.

[Puterman & Shin 78] Martin L Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

[Puterman 94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[Ramakrishnan *et al.* 09] Lavanya Ramakrishnan, Charles Koelbel, Yang-Suk Kee, Rich Wolski, Daniel Nurmi, Dennis Gannon, Graziano Obertelli, Asim YarKhan, Anirban Mandal, T Mark Huang, *et al*. Vgrads: enabling e-science workflows on grids and clouds with fault tolerance. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–12. IEEE, 2009.

[Rao *et al.* 09] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, pages 137–146. ACM, 2009.

[Reeves 03] Colin Reeves. Genetic algorithms. pages 55–82, 2003.

[RightScale 13] RightScale. Rightscale, http://www.rightscale.com/. 2013.

[Rodero-Merino *et al.* 10] Luis Rodero-Merino, Luis M Vaquero, Victor Gil, Fermín Galán, Javier Fontán, Rubén S Montero, and Ignacio M Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, 2010.

[Rolia *et al.* 06] Jerome Rolia, Ludmila Cherkasova, and Clifford McCarthy. Configuring workload manager control parameters for resource pools. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 127–137. IEEE, 2006.

[Rose 04] Robert Rose. Survey of system virtualization techniques. *Collections*, 2004.

[Rudin 76] Harry Rudin. On routing and. *Communications, IEEE Transactions on*, 24(1):43–59, 1976.

[Rummery & Niranjan 94] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

[Russell *et al.* 95] S.J. Russell, P. Norvig, J.F. Canny, J.M. Malik, and D.D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Englewood Cliffs, NJ, 1995.

[Schwartz *et al.* 02] Barry Schwartz, Edward A Wasserman, and Steven J Robbins. *Psychology of learning and behavior*. New York, 2002.

[Seidmann & Ma 04] Abraham Seidmann and Dan Ma. Asps versus enterprise software solutions. In *Workshop on Information Systems and Economics*, 2004.

[Seiffert 04] Udo Seiffert. Artificial neural networks on massively parallel computer hardware. *Neurocomputing*, 57:135–150, 2004.

[Sharpe & Keelin 98] Paul Sharpe and Tom Keelin. How smithkline beecham makes better resource-allocation decisions. *Harvard Business Review*, 76(2):45, 1998.

[Shen *et al.* 05] Zukang Shen, Jeffrey G Andrews, and Brian L Evans. Adaptive resource allocation in multiuser ofdm systems with proportional rate constraints. *Wireless Communications, IEEE Transactions on*, 4(6):2726–2737, 2005.

[Shin *et al.* 08] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on*, pages 181–190. IEEE, 2008.

[Singh *et al.* 07] Gurmeet Singh, Carl Kesselman, and Ewa Deelman. A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In *Proceedings of the 16th international symposium on High performance distributed computing*, pages 117–126. ACM, 2007.

[Sipser 06] Michael Sipser. *Introduction to the Theory of Computation*, volume 27. Thomson Course Technology Boston, MA, 2006.

[Smith *et al.* 98] Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times using historical information. In *Job Scheduling Strategies for Parallel Processing*, pages 122–142. Springer, 1998.

[Sorg *et al.* 12] Jonathan Sorg, Satinder Singh, and Richard L Lewis. Variance-based rewards for approximate bayesian reinforcement learning. *arXiv preprint arXiv:1203.3518*, 2012.

[Sotskov & Shakhlevich 95] Yu N Sotskov and NV Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, 1995.

[Spiegelhalter *et al.* 93] David J Spiegelhalter, A Philip Dawid, Steffen L Lauritzen, and Robert G Cowell. Bayesian analysis in expert systems. *Statistical science*, pages 219–247, 1993.

[Sterling 02] Thomas Lawrence Sterling. *Beowulf cluster computing with Linux*. The MIT Press, 2002.

[Stone *et al.* 05] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[Strens 00] Malcolm Strens. A bayesian framework for reinforcement learning. pages 943–950, 2000.

[Sutton & Barto 98] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[Sutton 88] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[Sutton 90] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, volume 216, page 224, 1990.

[Sutton 91] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.

[Sutton 92] Richard S Sutton. Reinforcement learning architectures. In *Proceedings ISKIT92 International Symposium on Neural Information Processing*, 1992.

[Sutton *et al.* 99] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[Tanenbaum 88] Andrew S Tanenbaum. Computer networks. *Prentice H all PTR (ECS Professional)*, 1(99):6, 1988.

[Taylor *et al.* 07] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The triana workflow environment: Architecture and applications. In *Workflows for e-Science*, pages 320–339. Springer, 2007.

[Tesauro 92] Gerald Tesauro. Practical issues in temporal difference learning. pages 33–53, 1992.

[Tesauro 94] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[Tesauro 05] Gerald Tesauro. Online resource allocation using decompositional reinforcement learning. In *Proc. AAAI-05*, pages 9–13, 2005.

[Tesauro *et al.* 05] Gerald Tesauro, Rajarshi Das, William E Walsh, and Jeffrey O Kephart. Utility-function-driven resource allocation in autonomic systems. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 342–343. IEEE, 2005.

[Tesauro *et al.* 06] Gerald Tesauro, Nicholas K Jong, Rajarshi Das, and Mohamed N Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 65–73. IEEE, 2006.

[Tesauro *et al.* 07] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.

[Theiner & Rutschmann 05] Dieter Theiner and Peter Rutschmann. An inverse modelling approach for the estimation of hydrological model parameters. *Journal of Hydroinformatics*, 2005.

[Topcuoglu *et al.* 02] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002.

[Toth & Vigo 87] Paolo Toth and Daniele Vigo. *The vehicle routing problem*, volume 9. Society for Industrial and Applied Mathematics, 1987.

[Travostino 08] Franco Travostino. Cloud computing. 2008.

[Tsitsiklis & Van Roy 97] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5):674–690, 1997.

[Ullman 75] J. D. Ullman. Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10:384–393, June 1975.

[Urbanek 09] Simon Urbanek. multicore: Parallel processing of r code on machines with multiple cores or cpus. *R package version 0.1-3, URL http://www. rforge. net/multicore*, 2009.

[Vaquero *et al.* 08] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[Vaquero *et al.* 11] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[Vecchiola *et al.* 09] Christian Vecchiola, Suraj Pandey, and Rajkumar Buyya. High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pages 4–16. IEEE, 2009.

[Vengerov 07] David Vengerov. A reinforcement learning approach to dynamic resource allocation. *Engineering Applications of Artificial Intelligence*, 20(3):383–390, 2007.

[VmWare 12] VmWare. Vmware virtualisation software, http://www.vmware.com/. 2012.

[Wagstaff *et al.* 01] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. pages 577–584, 2001.

[Wang *et al.* 05] Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for online reward optimization. In *Proceedings of the 22nd international conference on Machine learning*, pages 956–963. ACM, 2005.

[Watkins & Dayan 92] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[Watkins 89] C. Watkins. *Learning from Delayed Rewards*. Unpublished PhD thesis, University of Cambridge,England, 1989.

[Weld *et al.* 98] Daniel S Weld, Corin R Anderson, and David E Smith. Extending graphplan to handle uncertainty & sensing actions. In *Aaai/iaai*, pages 897–904, 1998.

[Werbos 87] Paul J Werbos. Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *Systems, Man and Cybernetics, IEEE Transactions on*, 17(1):7–20, 1987.

[Whitehead 89] Steven D Whitehead. Scaling reinforcement learning systems. Technical report, Technical Report 304, Dept. of Computer Science, University of Rochester, Rochester, NY 14627, 1989.

[Wu *et al.* 10] Zhangjun Wu, Zhiwei Ni, Lichuan Gu, and Xiao Liu. A revised discrete particle swarm optimization for cloud workflow scheduling. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pages 184–188. IEEE, 2010.

[Wu *et al.* 11] Zhangjun Wu, Xiao Liu, Zhiwei Ni, Dong Yuan, and Yun Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, pages 1–38, 2011. 10.1007/s11227-011-0578-4.

[Yamada & Nakano 92] Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. *Parallel problem solving from nature*, 2:281–290, 1992.

[Yu & Buyya 06] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3):217–230, 2006.

[Yu *et al.* 05] Jia Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-based scheduling of scientific workflow applications on utility grids. pages 8–pp, 2005.

[Yu *et al.* 07] Jia Yu, Michael Kirley, and Rajkumar Buyya. Multi-objective planning for workflow execution on grids. In *Proceedings of the 8th IEEE/ACM International*

conference on Grid Computing, pages 10–17. IEEE Computer Society, 2007.

[Yu *et al.* 08] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Workflow scheduling algorithms for grid computing. pages 173–214, 2008.

[Zaharia *et al.* 10] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278. ACM, 2010.

[Zeng *et al.* 04] Liangzhao Zeng, Boualem Benatallah, Anne HH Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311–327, 2004.

[Zhang & Zhou 09] Liang-Jie Zhang and Qun Zhou. Ccoa: Cloud computing open architecture. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 607–616. Ieee, 2009.

[Zhang *et al.* 01a] Bo Zhang, Qingsheng Cai, Jianfeng Mao, and Baining Guo. Planning and acting under uncertainty: A new model for spoken dialogue systems. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 572–579. Morgan Kaufmann Publishers Inc., 2001.

[Zhang *et al.* 01b] Yanyong Zhang, Anand Sivasubramaniam, José Moreira, and Hubertus Franke. Impact of workload and system parameters on next generation cluster scheduling mechanisms. *Parallel and Distributed Systems, IEEE Transactions on*, 12(9):967–985, 2001.

[Zhang *et al.* 05] Yanyong Zhang, Mark S Squillante, Anand Sivasubramaniam, and Ramendra K Sahoo. Performance implications of failures in large-scale cluster scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 233–252. Springer, 2005.

[Zhao & Sakellariou 03] Henan Zhao and Rizos Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In *Euro-Par 2003 Parallel Processing*, pages 189–194. Springer, 2003.

[Zhao & Sakellariou 07] Henan Zhao and Rizos Sakellariou. Advance reservation policies for workflows. In *Job Scheduling Strategies for Parallel Processing*, pages 47–67. Springer, 2007.

[Zhao *et al.* 05] Yong Zhao, Jed Dobson, Ian Foster, Luc Moreau, and Michael Wilde. A notation and system for expressing and executing cleanly typed workflows on messy scientific data. *ACM Sigmod Record*, 34(3):37–43, 2005.

[Zinkevich *et al.* 10] Martin Zinkevich, Markus Weimer, Alex Smola, and Lihong Li. Parallelized stochastic gradient descent. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2595–2603, 2010.

[Zitzler *et al.* 01] Eckart Zitzler, Marco Laumanns, Lothar Thiele, Eckart Zitzler, Eckart Zitzler, Lothar Thiele, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm, 2001.