# Adaptive Dispatching of Tasks in the Cloud

Lan Wang and Erol Gelenbe, *Fellow, IEEE*

**Abstract**—The increasingly wide application of Cloud Computing enables the consolidation of tens of thousands of applications in shared infrastructures. Thus, meeting the QoS requirements of so many diverse applications in such shared resource environments has become a real challenge, especially since the characteristics and workload of applications differ widely and may change over time. This paper presents an experimental system that can exploit a variety of online QoS aware adaptive task allocation schemes, and three such schemes are designed and compared. These are a measurement driven algorithm that uses reinforcement learning, secondly a "sensible" allocation algorithm that assigns tasks to sub-systems that are observed to provide a lower response time, and then an algorithm that splits the task arrival stream into sub-streams at rates computed from the hosts' processing capabilities. All of these schemes are compared via measurements among themselves and with a simple round-robin scheduler, on two experimental test-beds with homogenous and heterogenous hosts having different processing capacities.

**Index Terms**—Cognitive Packet Network, Random Neural Network, Reinforcement Learning, Sensible Decision Algorithm, Task allocation, Cloud Computing, task Scheduling, Round Robin

## 1 Introduction

Cloud computing enables elasticity and scalability of computing resources such as networks, servers, storage, applications, and services, which constitute a shared pool, providing on-demand services at the level of infrastructure, platform and software [1]. This makes it realistic to deliver computing services in a manner similar to utilities such as water and electricity where service providers take the responsibility of constructing IT infrastructure and end-users make use of the services through the Internet in a pay-as-you-go manner. This convenient and cost-effective way of access to services boosts the application of Cloud computing, which spans many domains including scientific, health care, government, banking, social networks, and commerce [2].

An increasing number of applications from the general public or enterprise users are running in the Cloud, generating a diverse set of workloads in terms of resource demands, performance requirements and task execution [3]. For example, multi-tier web applications composed of several components which are commonly deployed on different nodes [4] impose varied stress on the respective node, and create interactions across components. Tasks being executed in a Cloud environment may be of very different types, such as Web requests that demand fast response and produce loads that vary significantly over time [5], and scientific applications that are computation intensive; they may undergo several phases with varied workload profiles [6], and MapReduce tasks can be composed of different tasks of various sizes and resource requirements [5]. Furthermore, Cloud Computing enables highly heterogeneous workloads to be served on a shared IT infrastructure leading to inevitable interference between co-located workloads [7], while end users require assurance of the quality and reliability of the execution of the tasks that they submit. Therefore, the Cloud service provider must dispatch incoming tasks to servers with consideration for the quality of service (QoS) and cost within a diverse and complex workload environment. Also, energy consumption remains a major issue that can be mitigated through judicious energy-aware scheduling [8].

Thus the present paper focuses primarily on designing and evaluating adaptive schemes that exploit on-line measurement and take decisions with low computational overhead for fast on-line decision making. This work can be relevant to Cloud service providers that use the SaaS model where customers pay for the services, while the service provider sets up the VMs where the required software components are installed to deal with the service requests from the customer.

Our experimental evaluations are conducted on a multiple host test-bed, running with low to high loads that are achieved by varying the types and arrival rates of tasks. To conduct these experiments with greater ease, we have also designed and implemented a portable software module, the Task Allocation Platform (TAP), that is Linux based and easily installed on a Linux based machine. TAP will dynamically allocate user tasks to the available machines, with or without making use of on-line measurements of the resulting performance, and adapt to changes in workload and on-going performance of the Cloud environment, while optimising goals such as cloud provider's profit while maintaining service level agreements (SLAs). TAP is flexible in that it can easily support distinct static or dynamic allocation schemes. It collects measurements on the test-bed, both to report on performance evaluation and also (for certain allocation algorithms) to exploit measurements for adaptive decisions.

Thus in this paper we will report on the performance observed with two well known static allocation algorithms

• *The authors are with the Intelligent Systems and Networks Group, Department of Electrical and Electronic Engineering, Imperial College London*
*Email: {lan.wang12,e.gelenbe}@imperial.ac.uk*

*Manuscript received 29 Nov. 2014; revised 17 July 2015; accepted 19 Aug. 2015. Date of publication 0 . 0000; date of current version 0 . 0000.*
*Recommended for acceptance by T. Fahringer.*
*For information on obtaining reprints of this article, please send e-mal to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TCC.2015.2474406*

(Round-Robin and a probabilistic "equal loading" scheme), and three dynamic algorithms that are described in Section 3.1.

## 2 Prior Work

Extensive research in this challenging area includes work on static algorithms [9], [10], [11] which are simple without excessive overhead; but they are only suitable for stable environments, and cannot easily adapt to dynamic changes in the Cloud. Dynamic algorithms [12], [13], [14], [15] take into consideration different application characteristics and workload profiles both prior to, and during, run-time; however heir complexity can result in computational overhead that may cause performance degradation when implemented in a real system. Thus, many dynamic and adaptive schemes have only been evaluated through simulations [16] rather than in practical experiments, while few have been tested in real environments but with low task arrival rates [3].

Much work on task assignment in the Cloud is based on a detailed representation of tasks to be executed with a rather simplistic representation of the hosts or processing sub-systems, leading to an evaluation based on simulation experiments rather than measurements on a real system. In [17] an application composed of many tasks is represented by a directed acyclic graph (DAG) where tasks, inter-task dependency, computation cost, and inter-task communication cost are represented; two performance-effective and low-complexity algorithms rank the tasks to assign them to a processor in a heterogeneous environment. Related work is presented in [18], [19], while optimisation algorithms based on genetic algorithms [20], ant colony optimisation (ACO) [21], Particle Swarm optimisation [22], Random Neural Network optimisation [23], and auction-based mechanisms [24] have also been studied in this context, with potential applications to workload scheduling in the Cloud [25]. In [26], workload models which reflect the diversity of users and tasks in a Cloud production environment are obtained from a large number of tasks and users over a one month period, and exploited for evaluation in a simulated CloudSim framework.

Other work has used experiments on real test-beds rather than simulations [5] where the characteristics of the typical heterogeneous workloads: parallel batch tasks, web servers, search engines, and MapReduce tasks, result in resource provisioning in a manner that reduces costs for the Cloud itself. Another cost-effective resource provisioning system dedicated to MapReduce tasks [27] uses global resource optimisation. Hardware platform heterogeneity and co-scheduled workload interference are highlighted in [3], where robust analytical methods and collaborative filtering techniques are used to classify incoming workloads in terms of heterogeneity and interference before being greedily scheduled in a manner that achieves interference minimisation and server utilization maximization. The system is evaluated with a wide range of workload scenarios on both a small scale computer cluster and a large-scale Cloud environment applying Amazon EC2 to show its scalability and low computation overhead. However, the arrival rate of incoming workload is low and thus the system performance under saturation state is not examined. Furthermore, the contention for processor cache, memory controller and memory bus incurred by collocated workloads are studied in [28].

Early research that considers the important role of servers in delivering QoS in the Internet can be found in [29], where an architecture is proposed which provides web request classification, admission control, and scheduling with several priority policies to support distinct QoS requirements for different classes of users for multi-tier web applications. However, the scheduling approach is static and in [4], an adaptive feed-back driven resource control system is developed to dynamically provision resource sharing for multi-tier applications in order to achieve both high resource utilization and application-level QoS. A two-tiered on-demand resource allocation mechanism is presented in [30] with local allocation within a server and global allocation based on each local one, so as to achieve better resource utilization and dynamically adjust according to time-varying capacity demands. Energy consumption in computation, data storage and communications is also a challenge in the Cloud. A model for server performance and power consumption is derived in [31] with the potential to predict power usage in terms of workload intensity. In [8], the authors examine the selection of system load that provides the best trade-off between energy consumption and QoS. A heterogeneity-aware dynamic capacity provisioning scheme for Cloud data centers is proposed in [32], which classifies workloads based on the heterogeneity of both workload and machine hardware and dynamically adjusts the number of machines so as to optimise overall energy consumption and scheduling delay.

## 3 Overview of this Paper

The present paper uses experiments to investigate adaptive dynamic allocation algorithms that take decisions based on on-line and up-to-date measurements, and make fast online decisions to achieve desirable QoS levels [33]. The TAP that we have designed to this effect is a practical system implemented as a Linux kernel module which can be easily installed and loaded on any PC with the Linux OS.

TAP runs on a given host, and embeds measurement agents into each host in a Cloud to observe the system's state. These observations are then collected by "smart packets" (SPs) that TAP sends at regular intervals into the system in a manner which favours the search of those sub-systems which are of the greatest interest because they may be used more frequently or because they could provide better performance. The remainder of the paper is organized as follows.

The task allocation algorithms, including three novel approaches, are discussed in Section 3.1. TAP, the task allocation platform that we have designed, is discussed in Section 4, where the dynamic algorithms are introduced. Section 4.1 discusses all the three measurement based allocation schemes, including a mathematical model based scheme presented in Section 4.1.1, the Sensible Algorithm in Section 4.1.2, and the scheme that uses the RNN with reinforcement learning in Section 5.

The experimental results are introduced in Section 6, and first comparison of the different allocation schemes is

presented in Section 7. In Section 7.2 we present further experimental results when the hosts being used have distinctly different processing speeds.

In Section 8 we introduce a "contradictory" performance metric based on the economic cost, as perceived by the Cloud platform, of executing tasks: this cost includes the *penalty* that the Cloud would have to pay to the end user when a SLA (service level agreement) is violated, as well as the intrinsic economic cost of using faster or slower hosts. This same cost function is used for task allocation in view of minimising the overall cost to the Cloud service, and it is then measured and reported for both the Sensible and the RNN based algorithms.

Finally, Section 9 draws our main conclusions and discusses directions for future research.

### 3.1 The Task Allocation Algorithms that are Investigated

In this paper we design, implement in TAP and then experiment with several allocation algorithms:

- (a) round robin allocation of incoming tasks to distinct hosts,
- (b) a scheme that simply dispatches tasks with equal probability among hosts,
- (c) an allocation scheme that uses measurements of the execution times of tasks at hosts to allocate tasks probabilistically, where the probabilities are chosen via a mathematical model prediction so as to *minimise the average response time* for all tasks,
- (d) a Random Neural Network (RNN) [34], [35] based scheme that uses reinforcement learning with a numerically defined goal function that is updated with measurements brought back to TAP by SPs, and
- (e) an on-line greedy adaptive algorithm we call "sensible routing" [36] that selects probabilistically the host whose measured QoS is the best.

To the best of our knowledge, the approaches (d) and (e) have not been used before for task allocation in Cloud or other multi-server environments, though related ideas were suggested for selecting packet routes in multi-hop packet networks [37]. On the other hand, (a) and (b) are well known algorithms that are useful as benchmarks, and a scheme similar to (c) has been proposed in [8] for implementing trade-offs between energy consumption and quality-of-service in multiple-server computer systems.

We evaluate these schemes under varied task arrival rate via experiments on two test-beds: a cluster composed of hosts with similar processing speeds, and another one with where the hosts have significantly distinct processing capacities. The experimental results are then analysed and reported.

## 4 Task Allocation Platform and Test-Bed

TAP carries out online monitoring and measurement constantly in order to keep track of the state of the Cloud system, including resource utilisation (CPU, memory, and I/O), system load, application-level QoS requirements, such as task response time and bandwidth, as well as energy consumption, and possibly also (in future versions of TAP) system security and economic cost. With knowledge learned from these observations, the system can employ the QoS driven task allocation algorithms that we have designed, to make online decisions to achieve the best possible QoS as specified by the tasks' owners, while adapting to conditions that vary over time.

Figure 1 shows TAP's building blocks. The controller, which is the intellectual center of the system, accommodates the online task allocation algorithms, which work alongside the learning algorithm, with the potential to adaptively optimise the use of the Cloud infrastructure. TAP penetrates into the Cloud infrastructure by deploying measurement agents to conduct online observations that are relevant to the QoS requirements of end users, and send back the measurements to the controller. Three types of packets are used [37] for communications between the components of the system: smart packets (SPs) for discovery and measurement, dumb packets (DPs) for carrying task requests or tasks, and acknowledgement packets (ACKs) that carry back the information that has been discovered by SPs. In this section, we present in detail the mechanisms that are implemented in the platform and the algorithms that are used.

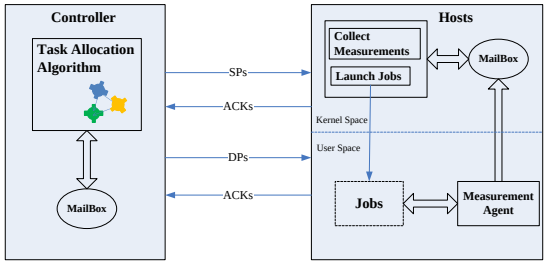



Fig. 1. System Architecture showing the Task Allocation Platform (TAP), which is hosted by a specific computer that receives and dispatches jobs, and which interacts with the measurement system (at the right) which is installed on each host machine that executes jobs. The TAP communicates with each of the measurement systems at the hosts using SPs ("smart packets"), DPs ("dumb packets") and ACKs ("Acknowledgement Packets") as indicated in the text of the paper.

SPs are first sent at random to the various hosts in order to obtain some initial information and inform the measurement agents in the hosts to activate the requested measurement. The task allocation algorithm in TAP learns from the information carried back by the ACKs and makes adaptively optimised decisions which are used to direct the subsequent SPs. Thus, the SPs collect online measurements in an efficient manner and pay more attention to the part of the Cloud where better QoS can be offered, visiting the worse performing parts less frequently.

The incoming tasks or task requests are encapsulated into the DPs, and exploit the decisions explored by SPs to select the host/Cloud sub-system that will execute the task. Once a task (request) arrives at a host in the Cloud, its monitoring is started by the measurement agent which records the trace of the task execution until it is completed and deposits the records into a mailbox which is located in the kernel memory of the host. When an SP arrives at this host, it collects the measurements in the mailbox

and generates an ACK which carries the measurements, and travels back to the controller where the measurement data is extracted and used for subsequent decisions of the task allocation algorithm. As soon as a task completes its execution, the agent also produces an ACK heading back to the controller with all the recorded data, such as the task arrival time at the Cloud, the time at which the task started running and the time at which the task execution completed. When the ACK of the DP reaches the controller, the task response time at the controller is estimated by taking the difference between the current arrival time at the node and the time at which the corresponding task arrives at the controller which is used by the algorithm when the task response time is required to be minimised.

### 4.1 Probabilistic Task Allocation Schemes

The schemes (b), (c), and (e) described in Section 3.1 are examples of probabilistic task allocation schemes. In these schemes, when a task arrives from some user or source outside the Cloud system, TAP decides to allocate it to some host $i$ among the $N$ possible hosts with probability $p_i$ so that *at decision time when the task must be allocated*:

- TAP first calculates $p_i$ for each of the hosts $i$,
- Then TAP uses these probabilities to actually select the host that will receive the task.

In the case of (b) we obviously have $p_i = 1/N$.

Probabilistic schemes have the advantage that a host which is being preferred because, say it is providing better service, is *not* systematically overloaded by repeated allocation since the QoS it offers is only used probabilistically to make a task allocation. In other words, the chance that a given server receives two successive tasks is very small as compared to the case where successive tasks are allocated to distinct servers.

In addition to (b), we experiment with two distinct schemes to calculate $p_i$, Model Based Allocation (c) and Sensible Routing (e).

#### 4.1.1 Model Based Task Allocation

Model Based Allocation (c) uses a mathematical model to predict the estimated performance at a host in order to make a randomised task allocation. This has been used in earlier work concerning task allocation schemes that help reduce the overall energy consumed in a system [8]. In this approach, if $W_i(\lambda, p_i)$ is the relevant QoS metric obtained for host $i$ by allocating a randomised fraction $p_i$ of tasks to host $i$ when the overall arrival rate of tasks to TAP is $\lambda$, then the allocation probabilities $p_1, \ldots, p_N$ are chosen so as to minimise the overall average QoS metric:

$$W = \sum_{i=1}^{N} p_i W_i(\lambda, p_i). \quad (1)$$

At first glance, since each host $i$ is a multiple-core machine with $C_i$ cores, a simple mathematical model that can be used to compute, say the QoS metric "response time" $W_i(\lambda, p_i)$ that host $i$ provides, assuming that there are no main memory limitations and no interference among processors (for instance for memory or disk access), is the $M/M/C_i$ queueing model [38], i.e. with Poisson arrivals, exponential service times, and $C_i$ servers. Of course, both the Poisson arrival and the exponential service time assumptions are simplifications of reality, and more detailed and precise models are also possible for instance using diffusion approximations [39] but would require greater computational effort and more measurement data.

However, a set of simple experiments we have conducted show that the $M/M/K$ model for each host would not correspond to reality. Indeed, in Figure 2 we report the *measured* completion rate of tasks on a host (y-axis) relative to the execution time for a single task running by itself, as a function of the number of simultaneously running tasks (x-axis). These measurements were conducted on a single host (Host 1), and for a single task running on the system, the average task processing time was $64.1$ms.

If this were a perfectly running ideal parallel processing system, we could observe something close to a linear increase in the completion rate of tasks (red dots) when the number of simultaneously running tasks increases, until the number of cores in the machine $C_1$ have been reached. However the measurements shown in Figure 2 indicate (blue dots) a significant increase in completion rate as the number of tasks goes from 1 to 2, but then the rate remains constant, which reveals that there may be significant interference between tasks due to competition for resources. Indeed, if we call $\gamma(l)$ the average completion rate per task, we observed the following values for $\gamma_i(l)/\gamma_i(1)$ for $l = 2, \ldots, 10$ computed to two decimal digits: $0.67, 0.48, 0.34, 0.29, 0.23, 0.20, 0.17, 0.15, 0.13$. From this data, a linear regression estimate was then computed for the average execution time $\mu(i)^{-1}$ when there are $l$ tasks running simultaneously, as shown on Figure 3, yielding a quasi-linear increase. As a result we can quite accurately use the estimate $l.\gamma(l)/\gamma(1) \approx 1.386$. Based on this measured

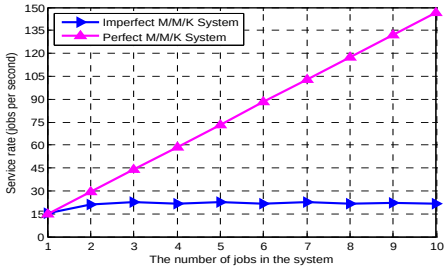



Fig. 2. The ideal service rate provided by the perfect multiple core system (red), compared to the measured task completion rate on Host 1 (blue), plotted against the number of tasks running simultaneously on the host (x-axis).

data, we model the distribution of the number of tasks in a host server $i$ as a random walk on the non-negative integers, where:

- $l = 0$ represents the empty host (i.e. with zero tasks at the host),
- The transition rate from any state $l \geq 0$ to state $l + 1$ is the arrival rate of tasks to the host $\lambda_i$,
- The transition rate from state 1 to state 0 is the $\mu_i(1) = T_i^{-1}$ where $T_i$ is the average execution time of a task (by itself) on the host,

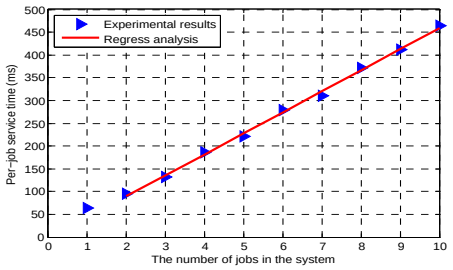



Fig. 3. Measurement of the effective task execution time per task on Host 1, versus the number of simultaneously running tasks on the host (x-axis).

- The transition rate from state $l+1$ to state $l$ if $l \geq 1$ is quasi constant given by $\mu_{i0} \equiv (l.\gamma(l)/\gamma(1))\mu_i(1)$,
- The arrival rate of tasks to Host $i$ is $\lambda_i = p_i^m \lambda$ where $p_i^m$ is the probability with which TAP using the model based algorithm assigns tasks to Host $i$, and $\lambda$ is the overall arrival rate of tasks to TAP.

The probability that there are $l$ tasks at Host $i$ in steady-state is then:

$$p_i(1) = p_i(0)\frac{\lambda_i}{\mu_i(1)},\ p_i(l) = (\frac{\lambda_i}{\mu_{i0}})^{l-1} p_i(1),\ l > 1,$$

$$p_i(0) = \frac{1 - \frac{\lambda_i}{\mu_{i0}}}{1 + \lambda_i \frac{\mu_{i0} - \mu_i(1)}{\mu_{i0}\mu_i(1)}}.$$

Using Little's formula [38] the overall average response times that we wish to minimise, by choosing the $p_i^m$ for a given $\lambda$ is:

$$W^m = \sum_{i=1}^{N} \frac{p_i^m}{\mu_i(1)} \frac{p_i(0)}{(1 - \frac{\lambda_i}{\mu_{i0}})^2}. \quad (2)$$

The appropriate values of the $p_i^m$ for a given system and a given arrival rate $\lambda$ can be then obtained numerically.

To illustrate this approach for the specific service time data regarding the three hosts that we use, in Figure 4 we show the variation of the average task response time with different combinations of $[\lambda_1, \lambda_2, \lambda_3]$, when $\lambda = 20\ tasks/sec$.

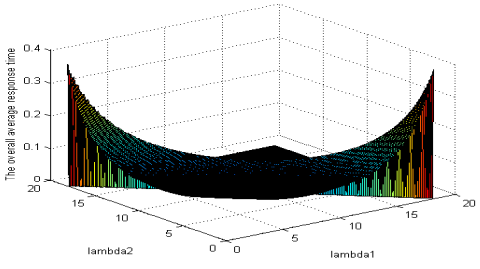



Fig. 4. Variation of the overall average task response time predicted by the infinite server model, with different combinations of $[\lambda_1, \lambda_2, \lambda_3]$, when $\lambda = \lambda_1 + \lambda_2 + \lambda_3$ is set to 20 tasks per second.

#### 4.1.2 Sensible Routing

The Sensible Decision Algorithm (e) uses a weighted average of $G_i$ of the goal function that we wish to *minimise*, which is estimated from on-going measurements at each host $i$, and updated each time $t$ that TAP receives a measurement that can be used to update the goal function. Specifically, when the goal function is the response time, its most recently measured value at time $t$, $G_i^t$, is received at TAP for host $i$, and the $n$-th update of $G_i$ is computed:

$$G_i \leftarrow (1-\alpha)G_i + \alpha G_i^t, \quad (3)$$

where the parameter $0 \leq \alpha \leq 1$ is used to vary the weight given to the most recent measurement as compared to past values. Based on updating this value for each host $i$, the probability $p_i^s$ that will be used to allocate a task to host $i$ is:

$$p_i^S = \frac{\frac{1}{G_i}}{\sum_{j=1}^{N} \frac{1}{G_j}},\ 1 \leq i \leq N. \quad (4)$$

If TAP allocates a task with this approach, it will use the most recent value of the $p_i^S$ which is available. Note that all of the $G_i$ values for different $i$ will not be equally "fresh", though the probing via SPs from TAP to the hosts proceeds at the same rate for all the hosts.

## 5 Random Neural Network Task Allocation with Reinforcement Learning

The Random Neural Network (RNN) has been used in static resource allocation as a "smart oracle" for allocating several resources to a set of tasks so as to minimise task execution times [35]. This earlier approach was based on first computing algorithmically a large set of optimum resource-to-task allocations, and then storing them in the RNN weights through a gradient descent learning algorithm. In order to select the best allocation, the trained RNN is then given an input which represents the set of available tasks, and it outputs the best known allocation.

This earlier work differs completely from the approach used in this paper which is based on on-line search, similar to the search by autonomous robots [40], [41], [42] with reinforcement learning [43] with real-time measurements. The RNN has also been used for packet routing [44]; in that work, an RNN placed at each router to select the next hop for probe (or smart) packets which explore routes and collect quality of service information. Thus the probes are routed to explore the better paths in the network, and bring back the data they collect to each source router. End users then examine the data available at the source nodes, and select the best current paths from the data collected by the by the probes. This approach, where the RNNs serve to route the probes (but not the user traffic) also differs from the approach in this paper, where an RNN is used to decide, for a given task, which server should be used.

In the present work, a RNN is used to select between $N$ hosts to which a task will be allocated, using its $N$ neurons in a fully connected form [23]. Each neuron $i$ is characterised by an integer $k_i(\tau) \geq 0$ which is its "level of excitation", where $\tau$ represents time, and each neuron is connected to other neurons both via excitatory and inhibitory weights. Furthermore, for the specific application for TAP, each

neuron is identified with a particular host, i.e. neuron $i$ is identified with the decision to assign a task to host $i$. The theoretical underpinning of the RNN [45] is a theorem that states that, at the equilibrium state, the probabilities:

$$q_i = \lim_{\tau \to \infty} Prob[k_i(\tau) > 0], \tag{5}$$

are uniquely obtained from the expression:

$$q_i = \frac{\Lambda(i) + \sum_{j=1}^{N} q_j w^+(j,i)}{r(i) + \lambda(i) + \sum_{j=1}^{N} q_j w^-(j,i)}, \tag{6}$$

where the $w^+(j,i)$ and $w^-(j,i)$ are the excitatory and inhibitory weights from neuron $j$ to neuron $i$ with $w^+(i,i) = w^-(i,i) = 0$. $\Lambda(i)$ and $\lambda(i)$ are the inputs of external excitatory and inhibitory signals to neuron $i$, while:

$$r(i) = \sum_{j=1}^{N} [w^+(i,j) + w^-(i,j)] \tag{7}$$

In the present case, a distinct RNN is set up within TAP to cover each distinct goal function $G$. However, these different RNNs need not be created in advance and stored at TAP indefinitely, but instead created when they are actually needed. Thus we will have a distinct RNN that is used to decide about allocations made on the basis of minimising economic cost (as when the end users pay a monetary price for the work they receive), or minimising task response time, or minimising task execution time, and so on.

A given RNN is initialised by setting $w^+(i,j) = w^-(i,j) = 1/2(N-1)$, so that $r(i) = 1$ for all $i$, and $\Lambda(i) = 0.25 + 0.5\lambda(i)$. In particular we can choose $\lambda(i) = 0$ so that all $\Lambda(i) = 0.25$. This of course results in $q_i = 0.5$ for all $i$.

TAP will then use the $q_i,\ i = 1,\ \ldots, N$ to make allocations so that a task is assigned to the host $i$ that corresponds to the highest value of $q_i$. Initially, any one of the hosts will be chosen with equal probability. However with successive updates of the weights, this will change so that TAP selects the "better" hosts which provide a smaller value of $G$.

When TAP receives a value $G_i^t$ of the goal function that was measured at time $t$ at host $i$, and $\frac{1}{G_i^t}$ is the "reward", so that the RNN weights are updated as follows:

- We first update a decision threshold $T_l$ as

$$T \leftarrow \alpha T + (1-\alpha)\frac{1}{G_i^t} \tag{8}$$

  where $0 < \alpha < 1$ is a parameter used to vary the relative importance of "past history".
- Then, if $G_i^t < T$, it is considered that the advice provided by the RNN in the past was successful and TAP updates the weights as follows:

$$\begin{aligned} w^+(j,i) &\leftarrow w^+(j,i) + \frac{1}{G_i^t} \\ w^-(j,k) &\leftarrow w^-(j,k) + \frac{1}{G_i^t(N-2)},\ if\ k \neq i \end{aligned}$$

- $else\ if\ G_i^t > T$

$$\begin{aligned} w^+(j,k) &\leftarrow w^+(j,k) + \frac{1}{G_i^t(N-2)},\ if\ k \neq i \\ w^-(j,i) &\leftarrow w^-(i,j) + \frac{1}{G_i^t}, \end{aligned}$$

- We compute $r^*(i) = \sum_{k=1}^{N} [w^+(i,k) + w^-(i,k)]$ for all $i$ and renormalise all weights so that their values do not grow indefinitely:

$$\begin{aligned} w^+(i,k) &\leftarrow \frac{r(i)}{r^*(i)} w^+(i,k), \\ w^-(i,k) &\leftarrow \frac{r(i)}{r^*(i)} w^-(i,k). \end{aligned} \tag{9}$$

After the weights are updated, the $q_i$ are computed using (6) with the new weights. Since this algorithm will tend to increase the probability $q_i$ of those neurons which correspond to hosts that yield a smaller value of $G_i$, each time TAP assigns a task to a host, it uses the host $i$ that corresponds to the largest $q_i$.

In order to make sure that TAP tries out other alternates and does not miss out on better options, a fraction $f$ of the decisions are made in round robin fashion: thus we are sure that all hosts will be tried out in succession for $f \times 100\%$ of the decisions, and the resulting goal function values will also be collected and updated. In the experiments that we describe below, $f$ was taken to be $0.1$, i.e. 10%. We have actually evaluated this percentage experimentally and found 10% to provide the best value in the setting of our experiments, but depending on the size of the system this percentage may vary.

Note also that this algorithm can be modified to a probabilistic"sensible" version [36] with:

$$p_i^{RNN-S} = \frac{q_i}{\sum_{j=1}^{N} q_j}. \tag{10}$$

## 6 Experiments

We conduct our experiments on a hardware test-bed composed of four nodes that each offer computation, storage and I/O. One node is dedicated to supporting the decision algorithms implemented in TAP, and the other three nodes are used as hosts running task, as shown in Figure 5, with each having a different processing power so that we may observe significant execution time differences for a given task. Since TAP takes decisions based on online measurements, even when there are no incoming tasks, the system maintains awareness of the state of the Cloud by sending SPs periodically. End users are allowed to declare the QoS requirements related to the tasks they submit, which is then translated into one or more QoS metrics which constitute a function called the "goal function" in our system. In this way, the QoS requirements are transformed into a goal function to be minimised, e.g. the minimisation of the task response time. The goal function determines which system parameters need to be measured and how task allocation will be optimised. TAP is implemented as a Linux kernel module which can be easily installed and loaded on any PC with Linux OS. The three hosts (with 2.8GHz, 2.4GHz, and 3.0GHz, respectively, dual-core CPU respectively) are used for task execution, while a separate host (2.8GHz dual-core CPU) supports the controller.

In these experiments we use a small scale test-bed so that we may easily load, and saturate, the system and evaluate the algorithms in both high, medium and low load conditions. However, TAP is scalable because most SPs are sent

to to those hosts which are providing better performance, so that there is no "flooding" of SPs across the system.

A synthetic benchmark is generated with task profiles indicated by using the fields $\{task\ \ ID, QoS\ \ requirement, task\ \ Size\}$, which are packetised into an IP packet and sent to the controller. The task request generator uses this information to forward task requests to TAP. In order to vary the load, in addition to using tasks with distinct CPU and I/O needs, the average time between successive task initialisations is varied, and these times are either of fixed duration (denoted by CR in the figures), or follow a Poisson process denoted by $EXP$.

The first set of experiments we report were run with tasks that were defined as a "prime number generator with an upper bound $B$ on the prime number being generated". Thus the choice of $B$ allowed us to vary both the execution time and the memory requirements of the task. We did not actually "transfer" the tasks from the task controller to the host, but rather installed the task in advance on the host, and the allocation decision by TAP just resulted in arrival of a message from TAP to activate the task with specific value of $B$ on that particular host. The measurement agent resident on that host then monitored the task execution and recorded its measurements into the mailbox. Both the tasks and the measurement agent run in the user's memory space, while the module that receives the SPs and task requests carried by DPs, collects measurements from the mailbox, and generates ACKs with the collected measurements runs in the kernel space of memory as shown in Figure 5, so that interference between the user program and the system aspects are avoided at least within the memory.

The two QoS goals that were considered were (i) the minimisation of either the execution time (denoted by ET in the figures) on the host, and (ii) the minimisation of the response time (denoted by RT in the figures) at TAP, where RT includes the message sent to activate the task at a host and the time it takes for an ACK to provide information back to TAP, where both the ET and the RT are provided to TAP from the host to the controller.

We first used TAP with the RNN algorithm with Reinforcement Learning (RL) as described above, and TAP with the sensible decision algorithm, and compared their performance. The RNN based TAP was experimented with both (i) and (ii), whereas the sensible decision based TAP only used (ii) the task response time at the controller.

In addition, according to the analytical model based approach was with (ii) task response time computed in terms of the task arrival rate and the system service rate, and then used to determine the optimum values of $\lambda_1, \lambda_2, \lambda_3$ corresponding to the three hosts subject to $\lambda = \lambda_1 + \lambda_2 + \lambda_3$, with an aim to minimise the overall task response time of the system as in (2), and then conducted experiments with task allocation probabilities to the three hosts selected so as to result in the arrival streams to the three hosts having the rates recommended by the analytical solution.

We also compared two static allocation schemes: Round Robin where successive tasks are sent to each host of the cluster in turn, and an equally probable allocation where a task is dispatched to each host with equal probability $0.33$.

All these experiments were repeated for a range of average task arrival rates $\lambda$ equal to $1, 2, 4, 8, 12, 16, 20, 25, 30, 40$ tasks/sec, in order to evaluate performance under load conditions that vary from light to heavy load, including saturation. Each experiment lasted 5 mins so as to achieve a stable state.

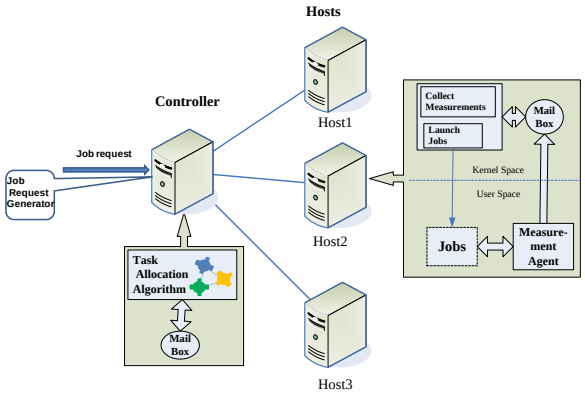


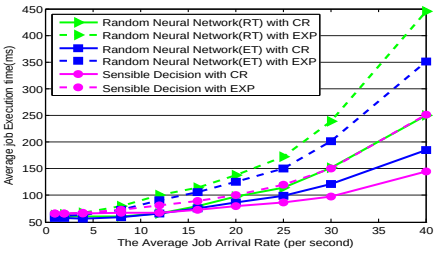

Fig. 5. Schematic description of the task allocation test-bed. Jobs arrive at the controller machine for dispatching to the hosts. The TAP (Task Allocation Platform) software is installed at the controller and takes the dispatching decisions. TAP takes decisions based on data it gathers from each of the measurement systems (at the right) which are installed on each host machine that executes jobs.

## 7 Comparison of the Different Algorithms

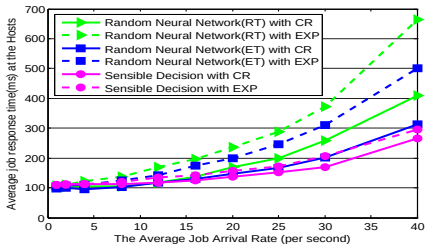
We first compared the two approaches, the RNN and the Sensible Algorithm, based on the measured average task response time observed at the controller, the average task response time at the host and the average task execution time. We see that the three metrics exhibit the same trend as shown in Figure 6.

At low task arrival rates less than $8$/sec, the RNN with RL performs better as shown in Figure 6(d), and it is even clearer with constant task arrival rates. However, as the average task arrival rates grows, the sensible decision algorithm outperforms the RNN, as in Figure 6(c). Also the RNN algorithm with online measurement of the task execution time always performs better than the RNN with the metric of task response time. However, the sensible decision is always best under high task arrival rates, as shown in Figure 6(c) .

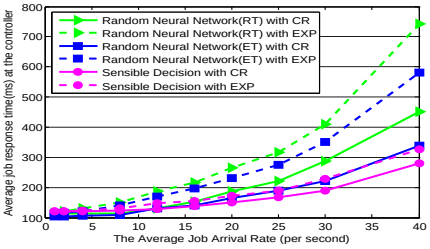
To understand these experimental results, we note that we have used CPU intensive tasks, and *each of them* experiences a longer execution time than when they are executed separately due to the competition for the same physical resource, namely the CPU. Indeed, the hosts are multi-core machines running Linux with a multitasking capability so that multiple tasks will run together and interfere with each other as shown in Figure 3. It can be found that, for example, if four tasks running in parallel, the average execution/response time per task increases two times. That is to say, the fluctuation of the execution time that the tasks experienced under varied number of tasks in the system is quite significant. Since the RNN with RL will send the tasks to the best performing hosts, it will tend to overload them, contrary to the Sensible Algorithm which dispatches tasks probabilistically and therefore tends to spread the load in a better manner.



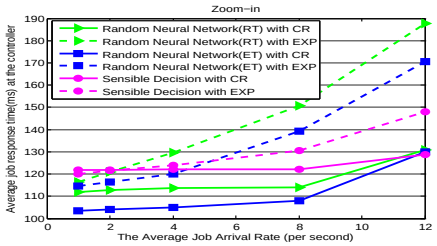

(a)




(b)



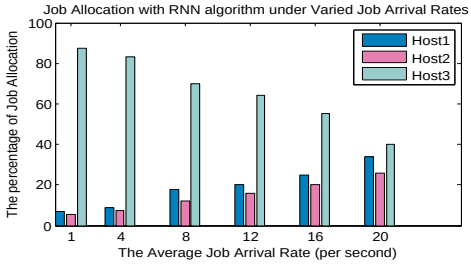

(c)



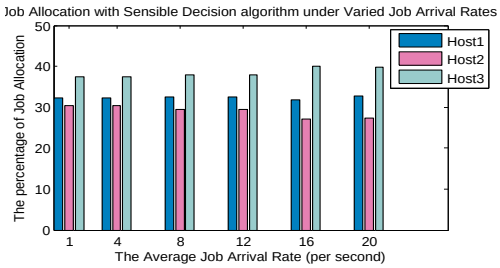

(d)

Fig. 6. Comparison of TAP operating with either the "RNN with reinforcement learning (RL)", or the "Sensible Algorithm" task allocation schemes. The metrics used for comparison are the resulting job execution time (ET) and job response time (RT) which are shown in the $y$-axis. Note that the Goal Function being optimised for each of the schemes, as shown in the legend for each curve, is the Response Time (RT) or the execution time (ET). We vary the rate at which jobs arrive ($x$-axis). Results are shown both for constant job inter-arrival times (CR), and for Poisson arrivals ($EXP$).

When RNN used the task execution time as the QoS criterion, Figure 7(a) shows that it dispatched the majority of tasks correctly to Host 3 which provided the shortest service time. The other two hosts accommodated some tasks because the RNN algorithm was programmed to make 10% of its decisions at random with equal probability. Here, the sensible decision algorithm performed worse because it makes task allocation decision with a probability that is inversely proportional to the task response time/execution time, instead of exactly following the best QoS as the RNN. As shown in Figure 7(b), the proportion of the tasks allocated with the sensible decision algorithm coincides with the proportion of the respective speeds of the three hosts.




(a)



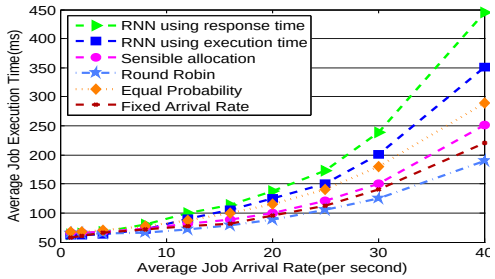

(b)

Fig. 7. The Proportion of task allocations to the three hosts with the RNN and the Sensible Algorithm for different task arrival rates.

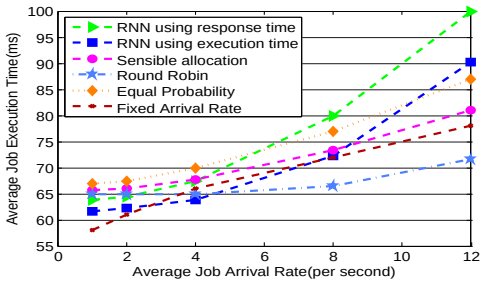
On the other hand, the Sensible Algorithm benefits from the fact that it does not overload the "best" hosts as shown in Figure 6(c) where the tasks may sometimes arrive to a host at rate that is higher than the host's average processing rate, leading to overload or saturation of the host. In Figure 6 we also see that the RNN based algorithm, that uses the task execution time measured at the hosts as the QoS goal, outperforms the RNN with online measurement of the task response time, because the execution time can be a more accurate predictor of overall performance when the communication times between the hosts and TAP fluctuate significantly. However at high task arrival rates, the Sensible Algorithm again performed better.

### 7.1 Comparison with the Model Based and Static Allocation Schemes

Figure 8 shows the average task execution time for the RNN and the Sensible Algorithm, in comparison with the model based scheme, as well as the Round Robin and Equally Probable allocation. The model based scheme performed better than the RNN when the task arrival rate was low, and better than the Sensible Algorithm at high arrival rates. However, the model based scheme can be viewed as an "ideal benchmark" since it relies on full information: it assumes knowledge of the arrival rate, it supposes that arrivals are Poisson, and it assumes knowledge of the task

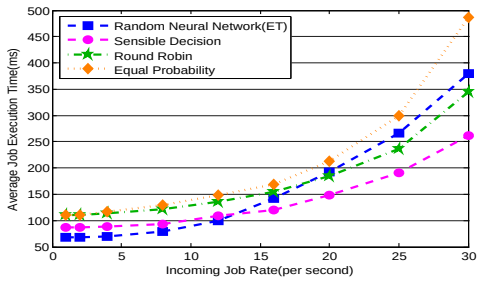



(a)

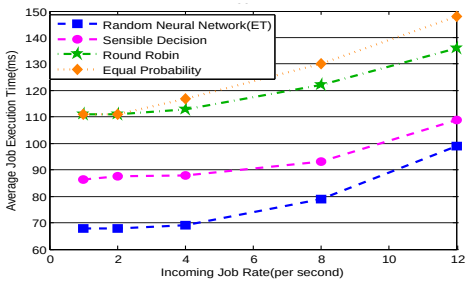



(b)

Fig. 8. The average task execution time experienced under varied task arrival rates and different task allocation schemes when the three hosts have similar performance.




(a)




(b)

Fig. 9. Average execution time experienced in a cluster composed of hosts with non-uniform processing capacities.

service rates at each host, while the RNN based scheme just observes the most recent measurement of the goal function.

As expected the equally probable allocation scheme performed worse. In this case where all servers are roughly equivalent in speed, Round Robin always outperformed the Sensible Algorithm, because it distributes work in a manner that does not overload any of the servers. These results are summarised in Figure 8(a). However the observed results change when the hosts have distinct performance characteristics as shown below.

### 7.2 Performance Measurements when Hosts have Distinct Processing Rates

As a last step, we evaluate the algorithms that we have considered, in a situation where each hosts provides significantly different performance. To strongly differentiate the hosts, we introduce a *background load* on each host which runs constantly and independently of the tasks that TAP allocates to the hosts. This is in fact a realistic situation since in a Cloud, multiple sources of tasks may share the same set of hosts without knowing what their precise workload may be, except for external observations of their performance.

Thus we were emulate three hosts $i = 1, 2, 3$ with relative processing speeds of $2 : 4 : 1$, respectively. The results of these experiments are summarised in Figure 9. We see that TAP with both the RNN and the Sensible Algorithm benefits from the ability of these two schemes to measure the performance differences between the hosts, and dispatch tasks to the hosts which offer a better performance, whereas the two static allocation schemes (Round Robin and the allocation of tasks with equal probabilities) lead to worse performance as a whole.

The performance of the RNN-based scheme clearly stands out among the others as shown in Figure 9(b), confirming that a system such as TAP equipped with the RNN can provide a very useful fine-grained QoS-aware task allocation algorithm.

### 7.3 Multiple QoS Classes

In this section, we will study the effectiveness of TAP when there is greater diversity both in the types of tasks, and in the type of QoS criteria and the SLA that they request. To evaluate the allocation algorithms with two different classes of tasks, we used a web browsing workload generated with HTTPerf which is a well-known web server performance tool.

The first class corresponds to HTTP requests retrieve files from a web server, such as the Apache 2 HTTP server, whereby I/O bound workload is generated on the web server with very little CPU consumption, and the load on the I/O subsystem can be varied with the size of the retrieved files. In our TAP test-bed, the Apache server is deployed on each host in the cluster. HTTPerf generates HTTP requests at a rate that can be specified, while TAP receives the requests and dispatches them to the web servers.

On the other hand, the web services which require a large amount of computation, mainly generate CPU load, are represented by CPU intensive tasks generated by the prime number generator.

In this case we compare the RNN based algorithms with the Sensible Algorithm, both using the Goal of minimising the response time. We also compare them to Round-Robin scheduling. The hosts themselves are stressed differently in terms of CPU and I/O in the cluster to provide different

heterogeneous environments. The workload is generated so as to arrive at TAP following a Poisson process with different average rates of $1, 2, 3, 4$ tasks/sec.

The different performance levels offered by the hosts is implemented by introducing a background load which stresses I/O differently on each host, resulting in relative processing speeds of $6 : 2 : 1$ for Hosts $1,\ 2,\ 3$ with regard to I/O bound services, while a background load which stresses CPU distinctly on each host, resulting in the relative processing speed of $2 : 3 : 6$ (corresponding to Hosts $1, 2, 3$) is used for the CPU bound case.

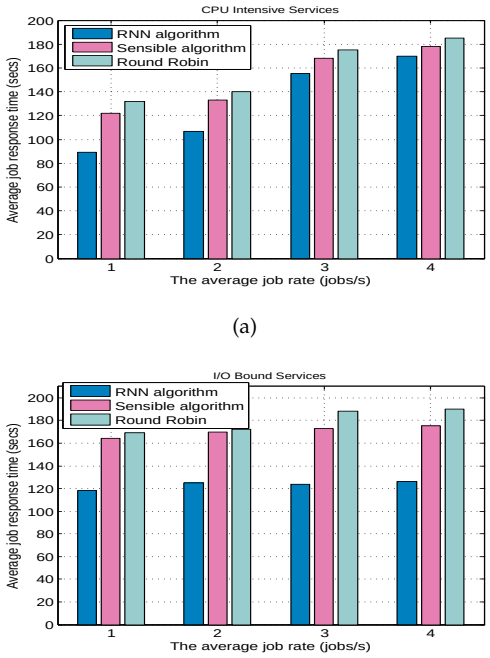



Fig. 10. Average response time experienced by CPU intensive services and I/O bound services in a heterogeneous cluster. We compare Round-Robin with RNN based Reinforcement Learning and the Sensible Algorith.

The results in Figure 10 show that the RNN based algorithm performs better; the reason may be that it is able to detect the best possible host for the task based on its QoS requirement by effective learning from its historical performance experience and make more accurate decisions ( compared with Sensible) which dispatch I/O bound tasks to the hosts where I/O is less stressed and dispatch CPU intensive tasks to the hosts which provide better CPU capacity. During the experiments, we reduced the background load in terms of both CPU and I/O stress on $Host\ 2$ to the lowest level as compared with $Hosts\ 1,\ 3$. It was found that the RNN based algorithm was able to detect the changes and dispatch the majority of subsequent tasks of both types to $Host\ 2$, nevertheless allowing the host where the CPU is heavily stressed to provide good performance to I/O bound tasks. More generally, we also observde that Round-Robin provided worse performance than the two other algorithms.

## 8 Contradictory QoS Requirements

Workloads are often characterised differently in terms of their resource and performance requirements. For example, for an online financial management and accounting software providing SaaS services, the web browsing workloads which retrieve large files from web servers generate I/O bound workload, while accounting and financial reporting applications may need a large amount of computation and require high CPU capacity. Thus, in this section we discuss how we may support multiple tasks with multiple QoS requirements for different QoS classes.

However, distinct QoS classes may also generate different levels of income for the Cloud service, and will also have different running costs on different hosts. In addition, they will have distinct service level agreements (SLAs), and the violation of the SLAs will often have different financial consequences.

*Thus we can consider a QoS Goal which includes two contradictory economic requirements: if we allocate a task to a fast processor, the cost will be higher since a more expensive resource is being used, however the resulting response time will be better resulting in fewer SLA violations and hence a lower penalty (and hence cost to the user). Obviously we have the opposite effect when we allocate a task to a slower machine, resulting in a lower economic cost for hosting the task, but in a higher resulting cost in terms of penalties for the Cloud due to SLA violations.*

Formalising these two contradictory aspects, consider a set of host servers, and different classes of tasks, so that:

- Let the amount paid by the user of a class $j$ task to the Cloud service be $I_j$ when the SLA is respected.
- Also, let us assume that if a task of QoS class $j$ is allocated to a host $m$ which is of type $M_i$, where the type of host includes aspects such as its memory capacity, its I/O devices, and speed, and the services offered by its software, will result in a cost to the Cloud service of $C_{ij}$.
- However, if the SLA is <u>not</u> respected there will *also* be some penalty to be paid by the Cloud service to the user, and this penalty must then be deducted from the income that the Cloud service was expecting to receive. For instance, the penalty will be zero if the response time $T$ of the task is below the SLA upper limit $T_{j,1} > 0$ for class $j$ tasks. More generally, the penalty is $c_{jl}$ if $T_{j,l-1} \leq T < kT_{j,l}$, where $T_{j0} = 0$ and $c_{j0} = 0$ (no penalty).

Using standard notation, let $1_{[X]}$ is the function that takes the value 1 if $X$ is true, and 0 otherwise. Then the *net income* obtained by the Cloud service for running a task of type $j$, after deducing the host operating cost and the eventual penalty for SLA violations, can be written as:

$$I_j^* = I_j - C_{ij}1_{[m=M_i]} - \sum_{l=1}^{n}\{c_{jl}1_{[T_{j,l}\leq T<T_{j,l+1}]}\} + c_{j,n+1}1_{[T\geq kT_{j,n+1}]}.$$

Obviously, the cloud server would like to *maximise* $I_j^*$, while $I_j$ is fixed in advance as part of the service agreement.

Thus in this section we consider task allocation based on a Goal function that will allocate a machine $M_i$ to a task of class $j$ so as to minimise the *net cost* function:

$$C_j = C_{ij}1_{[m=M_i]} + \sum_{l=1}^{n}\{c_{jl}1_{[T_{j,l}\leq T<T_{j,l+1}\}} + c_{j,n+1}1_{[T\geq kT_{j,n+1}]}. \quad (11)$$

Figure 11 shows an example of the penalty function, which is the second term in (11), for two distinct classes of tasks, where the $x$-axis is the value of the response time $T$.

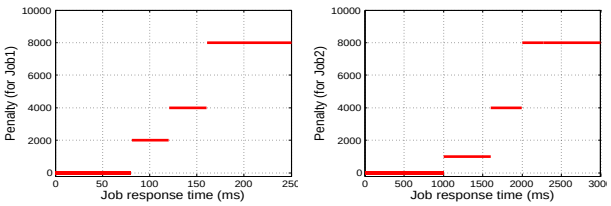
### 8.1 Experiments with Conflicting QoS Objectives

We illustrate the preceding discussion with experiments involving tasks of two classes with distinct QoS requirements, and we emulate a heterogeneous host environment where there is a fast host, a slow host and a medium speed host. This is done by stressing the CPU of each host differently, resulting in the speed-up factor of $1:2:4$ for Host $1, 2, 3$.

We conducted experiments using (11) as the Goal, with two classes of CPU intensive tasks and the two different penalty functions of Figure 11 regarding response time. The RNN based algorithm is compared with the Sensible Algorithm.

With regard to the terms in (11), we have $M_1 = 1000$, $M_2 = 2000$ and $M_3 = 4000$ coinciding with our assumption that the faster machines cost more, and the tasks either are "short" with an execution time of $56$ms, or "long" with an execution time of $190$ms as measured on the fastest host. Tasks were generated following independent and exponentially distributed inter-task intervals (Poisson arrivals), and four experiments with distinct task arrival rates of $1,\ 2,\ 3,\ 4$ tasks/sec were run separately for each class of tasks.

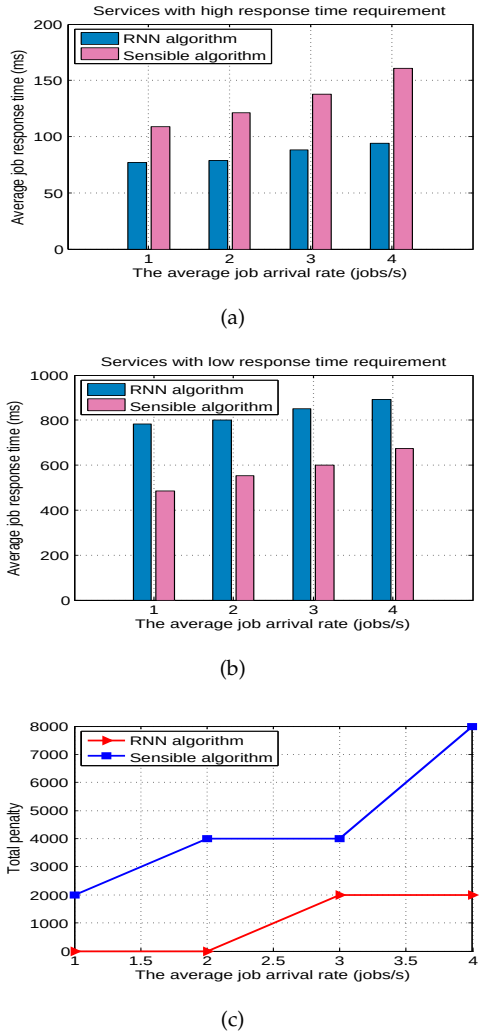
Figure 12 (a) shows that the RNN algorithm is able to do better in reducing the average response time in the case of the shorter tasks, while the Sensible Algorithm is more effective with regard to average response time for the longer tasks as seen in Figure 12 (b), though the RNN based algorithm manages to remain, at least *on average* below the $1000$ penalty threshold . However we also see that the RNN does better in reducing the overall cost plus SLA violation penalty (indicated as the Total Penalty in the $y$-axis) as shown in Figure 12 (c).




(a) (b)

Fig. 11. The penalty function for tasks belonging to Class 1 (left) and Class 2 (right), versus the task response time on the $x$ axis.

We also conduct a separate set of measurements in order to estimate the standard deviation of the response times, and the maximum observed response times as shown in Figure 13. In these cases, each measurement point that we report




(a)

(b)

(c)

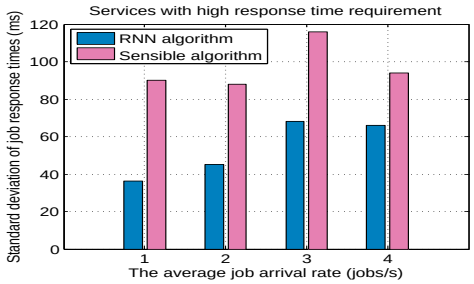
Fig. 12. The average value of the measured total cost for the two classes of tasks, when allocation is based on the Goal function that includes both the economic cost and the penalty as in (11).

is based on five experimental runs with Poisson arrivals, for each of the two classes of tasks, with each experiment lasting $20$ minutes in order to generate ample statistical data. In this setting, we again compare the RNN based algorithm with Reinforcement Learning to the Sensible Algorithm, still using the Goal function with the cost defined in (11).

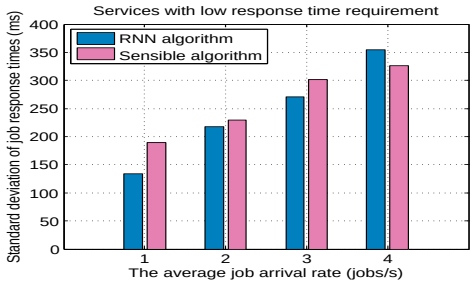
In most of the observed data, we see that the RNN-based algorithm can achieve better performance in terms of reducing the standard deviation of the response times leading to more dependable results, and also to maximum response times that lead to a smaller penalty, as expected from the previous data. The exception to this general observation is when tasks arrive at the highest rate of $4$ tasks/sec. In this case, it appears that the RNN based algorithm is not receiving timely data via the ACKs, leading to a level of performance that is worse than what is achieved by the Sensible Algorithm.

## 9 Conclusions and Future Work

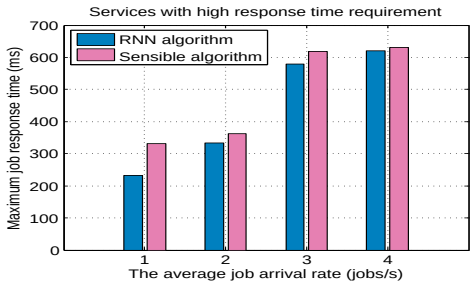
In this paper we have first reviewed the area of task allocation to Cloud servers, and then presented TAP, an



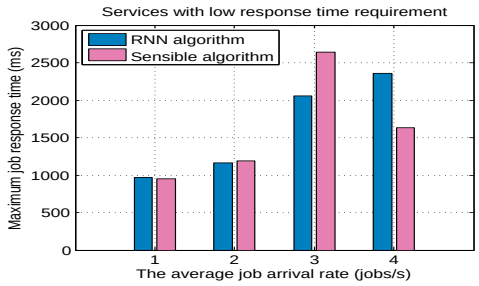

(a)




(b)




(c)




(d)

Fig. 13. The standard deviation of the measured response time, and its maximum value as a function of the task arrival rate ($x$-axis), for the two classes of tasks when task allocation uses the Goal function including both the economic cost and the penalty (11). Note that the performance of the RNN and Reinforcement Learning based algorithm, and the Sensible Algorithm are being compared.

experimental task allocation platform which can incorporate a variety of different algorithms to dispatch tasks to hosts in the Cloud operating in SaaS mode, before reporting on numerous experiments that have used TAP to compare a variety of task allocation algorithms under different operating conditions and with different optimisation criteria.

We consider simple static allocations schemes, such as Round Robin, and a probabilistic allocation which distributes load evenly. We also study a model driven algorithm which uses model based estimates of response time to select distinct allocation rates to different hosts. Two measurement driven adaptive on-line algorithms are also considered : the RNN based algorithm with Reinforcement Leaning, and the Sensible Algorithm that bring intelligence to bear from observations and make judicious allocation decisions.

Numerous experiments with different task profile, and optimisation objectives were considered, with two different sets of host machines: one composed of hosts with similar processing speeds, and another one with hosts having different speeds due to distinct background loads at each host.

Experiments showed Round Robin is effective when the processing rates and loads at each of the hosts are very similar. However when the hosts are quite distinct, the RNN based algorithm with Reinforcement-Learning offered fine-grained QoS-aware task allocation algorithm for accurate decisions, provided that online measurements are frequently updated. We found that the Sensible Algorithm offers a robust QoS-aware scheme with the potential to perform better under heavier load. The fixed arrival rate scheme, with full information of arrival rates and service rates, outperformed both the RNN and "sensible" approach due to the fact that it employs the solution of an analytical model to minimise task response time under known mathematical assumptions. However such assumptions will not usually be known or valid in practice; thus it is useful as a benchmark but cannot be recommended in practical situations.

In future work we will investigate the use of more sophisticated mathematical models such as diffusions approximations [39] to build an on-line measurement and model driven allocation algorithm that exploiting measurements of the arrival and service statistics at each of the hosts in order to estimate the task allocation probabilities. Although we expect that such an approach will have its limits due to the increase of the data that it will need, it may offer a better predictor for more accurate task allocations, and especially it could be used to benchmark other approaches. We would also like to study the Cloud system we have described when a given set of hosts is used by multiple task allocation systems operating with heterogenous input streams (such as Web services, mobile services and compute intensive applications), to see which schemes are the most robust and resilient. One aspect we have not discussed is the order in which tasks are executed, for instance time-stamp order [46]: although this may not be important in some applications, in others which are updated some global system state (such as a bank account), the order in which operations are carried out is critical, and tasks which are related by time-stamp order would have to be carried out in that order to avoid having to reprocess them if that order is violated. Another direction we wish to undertake is the study of the robustness of allocation schemes for Cloud services in the presence of network and service attacks [47] that are designed to disrupt normal operations. Another interesting direction for research will be to study how techniques that are similar to the ones we have developed in this paper, may be exploited in the context of Grid Computing [48].

## REFERENCES


[1] P. Mell and T. Grance, "The nist definition of cloud computing," *NIST Special Publication 800-145*, Aug 2009.

[2] R. Buyya, "Introduction to the ieee transactions on cloud computing," *Cloud Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 3–21, Jan 2013.

[3] C. Delimitrou and C. Kozyrakis, "Qos-aware scheduling in heterogeneous datacenters with paragon," *ACM Trans. Comput. Syst.*, vol. 31, no. 4, pp. 12:1–12:34, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2556583

[4] P. Pradeep, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07. New York, NY, USA: ACM, 2007, pp. 289–302. [Online]. Available: http://doi.acm.org/10.1145/1272996.1273026

[5] J. Zhan, L. Wang, X. Li, W. Shi, C. Weng, W. Zhang, and X. Zang, "Cost-aware cooperative resource provisioning for heterogeneous workloads in data centers," *Computers, IEEE Transactions on*, vol. 62, no. 11, pp. 2155–2168, Nov 2013.

[6] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 931–945, June 2011.

[7] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," *SIGPLAN Not.*, vol. 45, no. 3, pp. 129–142, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1735971.1736036

[8] E. Gelenbe and R. Lent, "Optimising server energy consumption and response time," *Theoretical and Applied Informatics*, no. 4, pp. 257–270, Jan 2013.

[9] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 445–465, 1985.

[10] C. Kim and H. Kameda, "An algorithm for optimal static load balancing in distributed computer systems," *IEEE Transactions on Computers*, vol. 41, no. 3, pp. 381–384, 1992.

[11] H. Kameda, J. Li, C. Kim, and Y. Zhang, *Optimal Load Balancing in Distributed Computer Systems*. Springer, 2011.

[12] J. J. Wolff, "Dynamic load balancing of a network of client and server computers," Feb. 6, 2001, u. S. Patent 6,185,601.

[13] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*. Ieee, 2009, pp. 44–51.

[14] Z. Zhang and X. Zhang, "A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation," in *Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*, vol. 2. IEEE, 2010, pp. 240–243.

[15] W. Tian, Y. Zhao, Y. Zhong, M. Xu, and C. Jing, "A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters," in *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 311–315.

[16] X. Zhu, X. Qin, and M. Qiu, "Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *Computers, IEEE Transactions on*, vol. 60, no. 6, pp. 800–812, June 2011.

[17] H. Topcuouglu, S. Hariri, and M. you Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1109/71.993206

[18] G. Sih and E. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 4, no. 2, pp. 175–187, Feb 1993.

[19] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 5, pp. 506–521, May 1996.

[20] E. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 5, no. 2, pp. 113–120, Feb 1994.

[21] W. Chen and J. 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*, vol. 39, no. 1, pp. 29–43, Jan 2009.

[22] S. Pandey, W. Linlin, S. Guru, and R. 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*, April 2010, pp. 400–407.

[23] E. Gelenbe and J. Fourneau, "Random neural networks with multiple classes of signals," *Neural Computation*, vol. 11, no. 4, pp. 953–963, 1999.

[24] S. Zaman and D. Grosu, "A combinatorial auction-based dynamic vm provisioning and allocation in clouds," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, Nov 2011, pp. 107–114.

[25] C. Lin and S. Lu, "Scheduling scientific workflows elastically for cloud computing," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, July 2011, pp. 746–747.

[26] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "Analysis, modeling and simulation of workload patterns in a large-scale utility cloud," *Cloud Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[27] B. Palanisamy, A. Singh, and L. Liu, "Cost-effective resource provisioning for mapreduce in a cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[28] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," *SIGPLAN Not.*, vol. 45, no. 3, pp. 129–142, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1735971.1736036

[29] N. Bhatti and R. Friedrich, "Web server support for tiered services," *Network, IEEE*, vol. 13, no. 5, pp. 64–71, Sep 1999.

[30] Y. Song, Y. Sun, and W. Shi, "A two-tiered on-demand resource allocation mechanism for vm-based data centers," *Services Computing, IEEE Transactions on*, vol. 6, no. 1, pp. 116–129, First 2013.

[31] E. Gelenbe, R. Lent, and M. Douratsos, "Choosing a local or remote cloud," in *Second Symposium on Network Cloud Computing and Applications, NCCA 2012, London, United Kingdom, December 3-4, 2012*. IEEE Computer Society, 2012, pp. 25–30. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/NCCA.2012.16

[32] Q. Zhang, M. Zhani, R. Boutaba, and J. Hellerstein, "Dynamic heterogeneity-aware resource provisioning in the cloud," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 1, pp. 14–28, March 2014.

[33] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Ieee, 2010, pp. 27–33.

[34] E. Gelenbe, "The first decade of g-networks," *European Journal of Operational Research*, vol. 126, no. 2, pp. 231–232, 2000.

[35] E. Gelenbe and S. Timotheou, "Random neural networks with synchronized interactions," *Neural Computation*, vol. 20, no. 9, pp. 2308–2324, 2008. [Online]. Available: http://dx.doi.org/10.1162/neco.2008.04-07-509

[36] E. Gelenbe, "Sensible decisions based on qos," *Computational management science*, vol. 1, no. 1, pp. 1–14, 2003.

[37] ——, "Steps toward self-aware networks," *Commun. ACM*, vol. 52, no. 7, pp. 66–75, 2009. [Online]. Available: http://doi.acm.org/10.1145/1538788.1538809

[38] E. Gelenbe and I. Mitrani, *Analysis and Synthesis of Computer Systems*. World Scientific, Imperial College Press, 2010.

[39] E. Gelenbe, "Probabilistic models of computer systems," *Acta Informatica*, vol. 12, pp. 285–303, 1979. [Online]. Available: http://dx.doi.org/10.1007/BF00268317

[40] E. Gelenbe and Y. Cao, "Autonomous search for mines," *European Journal of Operational Research*, vol. 108, no. 2, pp. 319–333, 1998.

[41] E. Gelenbe, K. Hussain, and V. Kaptan, "Simulating autonomous agents in augmented reality," *Journal of Systems and Software*, vol. 74, no. 3, pp. 255–268, 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2004.01.016

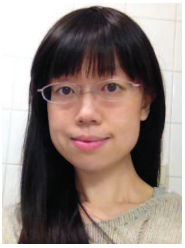
[42] N. Dimakis, A. Filippoupolitis, and E. Gelenbe, "Distributed building evacuation simulator for smart emergency management," *The Computer Journal*, vol. 53, no. 9, pp. 1384–1400, 2010.

[43] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[44] E. Gelenbe, R. Lent, and A. Nunez, "Self-aware networks and QoS," *Proceedings of the IEEE*, vol. 92, no. 9, pp. 1478–1489, Sep. 2004.

[45] E. Gelenbe, "Stability of the random neural network model," *Neural Computation*, vol. 2, no. 2, pp. 239–247, 1990.

[46] F. Baccelli, E. Gelenbe, and B. Plateau, "An end-to-end approach to the resequencing problem," *J. ACM*, vol. 31, no. 3, pp. 474–485, Jun. 1984.
[47] E. Gelenbe and G. Loukas, "A self-aware approach to denial of service defence," *Computer Networks*, vol. 51, no. 5, pp. 1299–1314, April 2007.
[48] D. Batista and N. da Fonseca, "A survey of self-adaptive grids," *Communications Magazine, IEEE*, vol. 48, no. 7, pp. 94–100, July 2010.



**Lan Wang** is currently a PhD student in the Intelligent System and Network group of the Department of Electrical and Electronic Engineering at Imperial College London. She is working on Quality of Service in Cloud computing systems and computer networks. She obtained her Bachelor's degree in Electronic Engineering from Shandong University and her Master's degree in Communications and Signal Processing from Imperial College London. Before studying at Imperial College, she worked as a software development engineer in integrated telecommunication network management systems in China.

**Erol Gelenbe** F'86 FACM is Professor and Head of Intelligent Systems and Networks at Imperial College London, working on energy savings, system security and Quality of Service. He previously founded the System Performance Evaluation groups at INRIA and French universities, chaired the ECE Department at Duke University, created the School of Electrical Engineering and Computer Science at the University of Central Florida, founded the team that designed the commercial Queueing Network Analysis Package QNAP at INRIA, and developed the FLEXSIM Flexible Manufacturing System Simulator, the SYCOMORE multiprocessor switch and the XANTHOS fibre optics local area network. He also invented mathematical models such as G-Networks and the Random Neural Network. Awarded "Honoris Causa" doctorates by the University of Liège (Belgium), Università di Roma II (Italy), and Boğaziçi University (Istanbul), he received the "In Memoriam Dennis Gabor Award" of the Hungarian Academy of Sciences, the Oliver Lodge Medal of the IET (London), the ACM-SIGMETRICS Life-Time Achievement Award and the Grand Prix France Telecom. He is a Fellow of the French National Academy of Technologies, the Hungarian, Polish and Turkish Science Academies, and the Royal Academy of Sciences, Arts and Letters of Belgium.