Managing Dynamic Enterprise and Urgent Workloads on Clouds Using Layered Queuing and Historical Performance Models

David A. Bacigalupo[a], Jano van Hemert[b], Xiaoyu Chen[a], Asif Usmani[c], Adam P. Chester[e], Ligang He[e], Donna N. Dillenberger[d], Gary B. Wills[a], Lester Gilbert[a], Stephen A. Jarvis[e]

[a]*School of Electronics and Computer Science, University of Southampton, UK*
[b]*Data-Intensive Research Group, School of Informatics, University of Edinburgh, UK*
[c]*BRE Centre for Fire Safety Engineering, School of Engineering, University of Edinburgh, UK*
[d]*IBM T.J. Watson Research Centre, Yorktown Heights, New York, USA*
[e]*High Performance Systems Group, Department of Computer Science, University of Warwick, UK*

Abstract

The automatic allocation of enterprise workload to resources can be enhanced by being able to make what-if response time predictions whilst different allocations are being considered. We experimentally investigate an historical and a layered queuing performance model and show how they can provide a good level of support for a *dynamic – urgent cloud environment*. Using this we define, implement and experimentally investigate the effectiveness of a prediction-based cloud workload and resource management algorithm. Based on these experimental analyses we: i.) comparatively evaluate the layered queuing and historical techniques; ii.) evaluate the effectiveness of the management algorithm in different operating scenarios; and iii.) provide guidance on using prediction-based workload and resource management.

Keywords: cloud, performance modelling, HYDRA historical model, layered queuing, FireGrid

1. Introduction

Being able to make 'what-if' response time predictions for potential allocations of enterprise workload to resources, can enhance automatic enterprise 'workload to resources' allocation systems [1,2]. For example, these predictions can help provide more cost-effective response time-based Service Level Agreement (SLA) management [22]. Without these predictions, managing the Quality of Service (QoS) levels promised in SLAs can require more (possibly informed) guesswork and over-provisioning of resources. Response time predictions can be very useful when enterprise systems are run on clouds or similar resource-sharing infrastructures. These clouds are often very large, increasing the number of possible workload-resource allocations and providing more opportunities for the use of predictions.

Two common approaches used in the literature for making these response time predictions are extrapolating from historical performance data and solving queuing network models. Examples of the first approach include the use of both coarse [3] and fine [1] grained historical performance data. The former involves recording workload information and operating system/database load metrics, and the latter involves recording the historical usage of each machine's CPU, memory and IO resources by different classes of workload. We have developed and experimentally verified our own ('HYDRA') historical technique [4,5,6]. It is differentiated from other historical modelling work by its focus on simplifying the process of analysing any historical data so as to extract the small number of trends that will be most useful to a management system. Examples of the queuing modelling approach include [7,8,2]. The layered queuing method [9] is of particular interest and will be examined further in this paper as it explicitly models the tiers of servers found in this class of application, and it has been applied to a range of distributed systems (e.g. [10]) including the benchmark used in this paper [11]. A layered queuing model can be solved either by simulation or via an approximate analytical solver.

It is important to quantitatively evaluate and provide guidance on the effectiveness of different approaches to prediction-based cloud infrastructure (i.e. 'resource'), cloud customer (i.e. 'workload') cloud management systems and to consider both the queuing and historical approach whilst doing this. To help make the comparison of relevance to a wide range of possible cloud environments it is useful to consider the following:

1. *Urgent* cloud customers such as the emergency services, that can demand cloud resources at short notice (e.g. for our *FireGrid* [12,24] emergency response software);
2. *Dynamic* enterprise systems, that must rapidly adapt to frequent changes in workload, system configuration and/or available cloud servers;
3. The use of the predictions in a $co-ordinated\ manner$ by both workload and resource management systems;
4. A broad range of criteria for evaluating each technique.

However, there have been no previous studies meeting these requirements making it harder to:

- Make an informed choice of technique/s; and hence
- Design effective prediction-based cloud management systems.

The following are examples of studies which do not meet these requirements. In work such as [2,15] e-Business system operating scenarios and their associated costs are investigated using a queuing model-based system management technique. In [19] the effectiveness of a model-based optimal dynamic server allocation algorithm is evaluated. In [10] a layered queuing model of a distributed database system is created and compared to a Markov chain-based queuing model of the system. In [9] the layered queuing method is compared more generally to other performance modelling techniques. Another recognised queuing technique which has been applied to similar applications is described in [7] and compared with the layered queuing method. However none of these papers include a comparison with an historical model of the same application. In our previous work we have compared queuing/historical (i.e. HYDRA) techniques and investigated the tuning of prediction-based workload and resource management [4,6,18] but not met the requirements listed above or addressed the challenges listed at the end of this section. The historical prediction technique described in [3] is applied to a web-based stock trading application and compared to a queuing modelling approach. However a queuing network model is not created. In work including [1] the effectiveness of an IBM model-based system management technique is investigated, but the requirements listed above are not met.

An important feature of this paper is that the comparison focuses on *dynamic* enterprise systems. These may have to acquire shared cloud servers with *new server architectures* for which only a small number of benchmarks have been run (e.g. to determine request processing speed). We therefore investigate the level of support provided by the historical and layered queuing models for rapidly being parameterised with low overheads on an established server, whilst still obtaining enough data to make accurate predictions on new server architectures. This allows the models to be parameterised prior to making real-time workload and resource management decisions, and removes the need to model the workload and system configuration variables that change less frequently at runtime. This has a number of advantages when predicting the performance of dynamic enterprise systems including: i.) a reduction in model complexity which can dramatically improve the responsiveness of predictions; and ii.) removing the need to consider some variables which may be complex to measure and model (such as the complexity of processing the data in the database for each service class in the workload). See [6] for our detailed discussions on the advantages of the methods used in this paper, and how this kind of method can be applied; and uncertainty in the type of modelling and analysis presented in this paper.

In summary this paper focuses on *dynamic* enterprise systems (with SLAs with financial penalties) hosted in a cloud that may also host *urgent* systems and in which there is a *coordinated use of performance predictions* for QoS management by both the cloud infrastructure and the cloud customer. For brevity, we define this type of system as a '$dynamic - urgent\ cloud\ environment$'. See Section 2 for our dynamic-urgent cloud environment system model, case study and experimental setup.

We present significant extensions over the original paper which describes this work [18]. These contributions can be summarised as follows:

1. In Section 5.3 the paper presents the experimental results of our investi-

gation into the effectiveness of the prediction-enhanced management algorithm in different *$dynamic - urgent$ cloud environment* operating scenarios. This includes the definition, implementation and experimental investigation of our algorithm (algorithm 3) that does not use predictions.
2. In Section 6.2 the paper evaluates the effectiveness of our prediction-enhanced management algorithm in different *$dynamic - urgent$ cloud environment* operating scenarios. This is based on our experimental results. We consider the prediction-enhanced management algorithm with both the layered queuing and historical models plugged in.
3. In Section 6.3 the paper provides guidance on using a prediction-based management algorithms in a *$dynamic-urgent$ cloud environment*, based on our experimental results. This includes suggested techniques and also advice on the potential issues the guidance and techniques can help address.

The paper is structured as follows: the definition of the system model, case study and experimental setup (section 2); the definition, experimental investigation and guidelines for the layered queuing prediction model (section 3); the definition, experimental investigation and guidelines for the HYDRA historical model (section 4); the experimental investigation of using predictions in a cloud workload and resource management system (section 5) and the evaluation and guidelines (section 6).

2. Dynamic-Urgent Cloud Environments: System Model, Case Study, Experimental Setup and Sample Cloud

2.1. System Model

Based on [13,14,17] a cloud customer (CC) pays for shared servers (SS) which may be dynamically shared with other CC and which may be virtualised – see Figure 1. Here, a SS is being transferred from CC1 to CC2; SS transfers are controlled by the cloud infrastructure resource management system (RM). When there is more than one virtual server running on a physical server, each virtual server is allocated a minimum percentage of the physical server's resources. There are N_{CC} CC, each of which can also hire dedicated servers (DS). The internal structure of CC_1 shows the enterprise system model for the paper. Based on established work [11,15] this is modelled as a tier of heterogeneous application SS accessing a database/legacy system. Based on the queuing network in IBM Websphere: a single FIFO queue is used by each application server; the database server has one FIFO queue; and both types of server can process requests concurrently via time-sharing. The workload consists of clients (divided into service classes each with a SLA) which send requests to the system. The dispatcher (D), controlled by the cloud customer workload management system (WM), adjusts the routing of the incoming requests to the application servers. The algorithms in the management systems may use a prediction engine (P) to help make decisions, and may co-ordinate the use of these predictions e.g. between the resource and workload management systems. A cloud customer

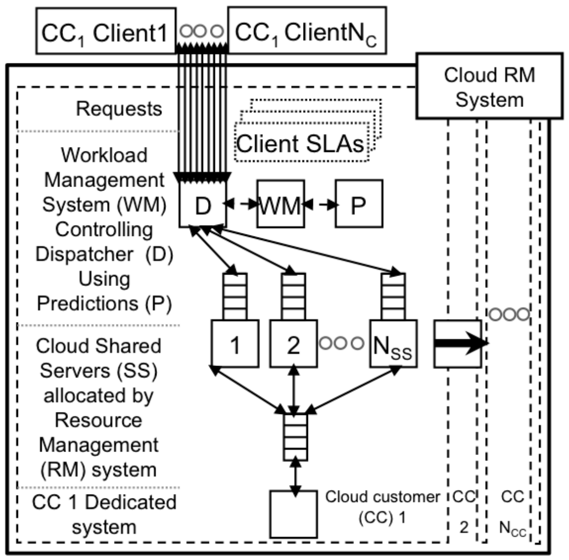
CC$_1$ Client1
CC$_1$ ClientN$_C$
Cloud RM System
Requests
Client SLAs
Workload Management System (WM) Controlling Dispatcher (D) Using Predictions (P)
D
WM
P
1
2
N$_{SS}$
Cloud Shared Servers (SS) allocated by Resource Management (RM) system
CC 1 Dedicated system
Cloud customer (CC) 1
CC 2
CC N$_{CC}$

Figure 1: System Model

can run an urgent system which can make urgent requests for resources (e.g. for emergency response). For a detailed survey and discussion of cloud environments and associated issues see [17].

2.2. Case Study

The case study is based on the Eucalyptus [13] cloud model, with every CC either having this enterprise system model or it is an urgent system. IBM Websphere is selected as the enterprise system middleware as it is a common choice for distributed enterprise applications; the IBM Performance Benchmark Sample *Trade* [16] is selected as it is the main enterprise benchmark for the Websphere platform. The sample enterprise workload is as follows based on Trade and [11]. A service class is created for *browse* users with n_{browse} clients, with the operation (*buy*/*quote* etc) called by a client being randomly selected using the probabilities defined by Trade. A service class with n_{buy} clients is created for *buy* users which involves clients making an average of 10 *buy* requests. The percentage of *buy* clients is defined as b. Think-times are exponentially distributed based on [9] with a mean of 7 seconds for all service classes as recommended by IBM for Trade clients, although heterogeneous think-times are supported by both techniques. Based on the Trade benchmark, a *typical workload* is defined as all *browse* clients.

FireGrid is a command and control application for urgent emergency response during a building fire. In experiments so far the majority of the processing has been to run embarrassingly parallel fire models so it is this which is run on the cloud. *FireGrid* is selected as the urgent system as it is the result of significant and ongoing development by a large coalition including emergency response services and it has been tested in real-time during a live fire test featuring a full-size mock-up of a three room flat.

2.3. Experimental Setup and Sample Cloud

The experimental setup contains three application server architectures. Under Websphere v4.0.1. and the typical workload (defined in the last sub-section) the max. throughputs of the new 'slow' $AppServ_S$ (P3 450Mhz); the established 'fast' $AppServ_F$ (P4 1.8Ghz); and the established 'very fast' $AppServ_{VF}$ (P4 2.66Ghz) are 86, 186 and 320 requests/second respectively. Each server is virtualised using a JVM. The database server architecture is DB2 7.2 on an Athlon 1.4Ghz and 250 clients are simulated by each workload generator (P4 1.8Ghz). The clients are simulated using an Apache JMeter script as described in [6], with the workload being sampled to check it is being generated as required. All servers run Windows 2000, have at least 512MB RAM and are connected via a 100Mbps switch.

In the sample cloud $N_{CC} = 4$ with $FireGrid$ as CC_4, and CC_1, CC_2 and CC_3 are dynamic enterprise systems. CC_1 has one SS of each of the three application server architectures and a workload of 3000 clients. Three service classes are created by dividing the *browse* service class into two service classes with different SLA response time goals (RTG). The workload that is to be allocated to the SS is defined as 10% *buy* clients (RTG=150ms), 45% high priority *browse* clients (RTG=300ms), and 45% low priority *browse* clients (RTG=600ms). The percentages are selected based on the Trade application, which defines 10% of the standard workload to be purchase requests. The response time goals are selected based on the response time of the fastest application server at max throughput (aprox 600ms). CC_2 has 20% more clients, RTG all 20% lower and hence all servers upgraded to the next fastest server architecture available (excepting $AppServ_{VF}$ – the fastest) to $2\times AppServ_{VF}$ and $1\times AppServ_F$. For CC_3 the number of clients and RTG figures are increased/decreased by an additional 20% respectively, and the SS are upgraded again resulting in $3\times AppServ_{VF}$. CC_4 has all the other SS, in this case 5; the minimum number we have executed this part of $FireGrid$ on.

3. Definition and Experimental Investigation of The Layered Queuing Prediction Model

This section describes the experimental investigation of the level of support provided for a $dynamic - urgent\ cloud\ environment$ using a layered queuing prediction model, examining the associated overheads and hence providing guidelines on the parameterisation of the model.

The layered queuing method involves defining an application's queuing network. An approximate solution to the model can then be generated automatically using the LQNS solver. The solution strategy involves dividing the queues into layers (e.g. as in the enterprise system model), generating an initial solution and then iterating backwards and forwards through the layers solving the queues in each layer by mean value analysis and propagating the result to the next layer until the solutions converge. Performance metrics generated include mean response time, throughputs and utilisation information for each service class. A detailed description of the layered queuing method can be found in [9,23].

We have created a layered queuing model of an enterprise system with *app_svr*, *DB_svr* and *DB_svr_disk* layers, each layer containing a queue and a processor. The application server disk is not modelled as the Trade application's utilisation of this resource is found to be almost 0 during normal operation. Workload parameters for each service class v are: the number of clients n_v; the mean processing times (on each processor p), $t_{(p,v)}$ or $t_{(p,v)N}$ for established and new server architectures respectively; and the mean number of requests to the next layer $y_{(app\_svr,v)}$ and $y_{(DB\_svr,v)}$. Processing times are exponentially distributed as standard with the layered queuing method. Communication time is a constant delay d. Queue parameters per layer l are the maximum number of requests each processor can process at the same time via time-sharing m_l.

Model parameterisation is as follows. d is parameterised by subtracting the predicted response time from the actual response time at a small number of clients (250 in the experimental setup) on an established server. Then for each p and for each v take an established server off-line and send a workload consisting only of v for an interval. This overcomes the difficulties that have been found measuring mean processing times (without queuing delay) of multiple service classes, in real system environments [15]. $t_{(p,v)}$ is then calculated by dividing the associated mean interval server (or disk) CPU (or disk) usage by the mean interval throughput (in requests/second). Calculating $t_{(p,v)N}$ involves multiplying a value of $t_{(p,v)}$ by the established/new server request processing speed ratio.

3.1. Experimental Results

In the remainder of this section we examine the predictive accuracy and resource usage overhead when parameterising the request processing times under different workloads. This includes an analysis on the effect of the number of clients used to parameterise the mean processing time variables in the model. In the experimental setup (from Section 2) $m_{app\_svr} = 50$, $m_{DB\_svr} = 20$, $m_{DB\_svr\_disk} = 1$, $y_{app\_svr,buy} = 2$ and $y_{app\_svr,browse} = 1.14$. $y_{DB\_svr,v}$ is modelled as 1 by setting the throughput of *DB_svr_disk* to the throughput of *DB_svr* during parameterisation. The typical (*browse*) workload is parameterised on an established application server with a max. throughput of 213 requests/second; for brevity we refer to the associated processor as E. Each test run involves setting n_{browse} and after a 1 minute warmup recording %CPU/disk usage samples and server mean throughputs for 1 minute. The sampling interval is set at 6 seconds so the increase in the %CPU/disk usage is no more than 5%. We define predictive accuracy as the following. Note that in this paper the term *accuracy* refers to the accuracy of the predictions (i.e. when compared to the measured data), as opposed to the accuracy of the measured data – unless this is explicitly stated.

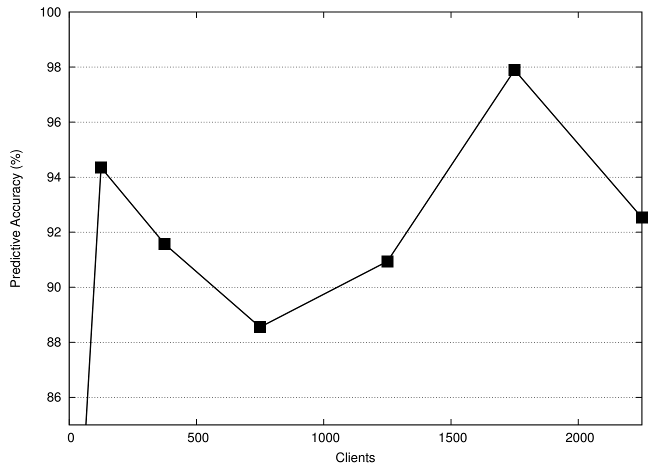

Figure 2: The predictive accuracy when parameterising the layered queuing model at different numbers of clients.

Table 1: The predictive accuracy when parameterising the layered queuing model at different numbers of clients as shown in figure 2.

No. of Clients	63	125	375	750	1250	1750	2250
Predictive Accuracy (%)	84.60	94.35	91.57	88.55	90.94	97.89	92.53

Table 2: Response time predictions for the new server architecture.

No. of Clients	250	500	750
Measured Mean (ms):	122.9	210.4	2060.2
Layered Queuing Predicted Mean (ms):	105.9	137.4	1811.3
Layered Queuing Predictive Accuracy (%):	86.2	65.3	87.9
Historical Predicted Mean (ms):	139.1	268.7	2581.8
Historical Predictive Accuracy (%):	86.8	72.3	74.7

$$accuracy = \frac{|predicted\_value - measured\_value|}{measured\_value} \times 100 \tag{1}$$

As n_{browse} is increased $t_{(E,browse)}$ decreases (from 5.6ms at $n_{browse} = 63$) to a minimum of 4.3ms at $n_{browse} = 750$. It then increases (to 4.7ms at $n_{browse} = 2250$). This pattern is explained as follows. The higher mean processing times at smaller number of clients are due to the larger system and JVM overhead (i.e. for garbage collection) per request. The higher mean processing times at larger numbers of clients are due to the overhead of running a larger number of threads (as Websphere terminates threads that are not needed, at lighter loads). At intermediate numbers of clients these overheads are less significant resulting in lower mean processing times. As a result the predictive accuracy is highest when parameterising the model at a value of n_v between the maximum and minimum mean processing times - see Table 1. However, the higher n_{browse} the more server capacity that must be taken offline to run the workload generators, application and database servers. The maximum predictive accuracy when the model is parameterised at a low overhead is at $n_{browse} = 125$; below this the predictive accuracy drops significantly. This is due to a discontinuity in the rate at which the mean response time increases with number of clients around the point at which max. throughput is reached. The predictive accuracy around this point (one of the predictive accuracy measurements taken at different numbers of clients so as to calculate, via a mean the predictive accuracy) is less accurate than for other numbers of clients. Hence the predictive accuracy at the first maximum is slightly lower than that of the second maximum. When $n_{browse} = 125$ then $t_{(E,browse)} = 4.675ms$, $t_{(F,browse)} = 1.821ms$ and $t_{(G,browse)} = 0.638ms$ where F is the database server processor and G is the database server disk processor. This results in a predictive accuracy of 80% on the new server architecture – see Table 2 and Figure 2. We note that in previous work [4,6] we have shown how percentile response time metrics can be predicted based on the measured exponential distributions. To parameterise the model a server capacity of 89 requests/second must be taken offline for 2 minutes per service class. This is the equivalent of a Pentium III 450Mhz which is likely to be a low parameterisation overhead for a modern resource management system. It is also important to consider the delay when solving the model. Using the standard LQNS solver on a dedicated Athlon 1.4Ghz at a convergence criterion of 20ms this is observed as follows. When the server is lightly loaded and when the server is saturated the model can be solved in around 1 second. However, as you move away from these two extremes the time to solve increases up to a maximum of 5 seconds. This point is reached just before 100% of the number of clients at max. throughput. However, most of the predictions are made in 1-3 seconds. The consequences of these evaluation times are discussed in Sections 5 and 6.

3.2. Guidelines For Parameterising the Model at a Low Overhead

We have found that frequent, rapid model parameterisations can be useful for maintaining predictive accuracy in dynamic enterprise systems – assuming this is used with care. For example correcting for predictive inaccuracy by parameterising again can mask the need for refinements in the configuration of the enterprise system/cloud/model etc. It is also important to check the predictive accuracy of the model immediately after parameterisation as it may occasionally be necessary to repeat the parameterisation, for example after a brief spike of background activity.

Our guidelines for parameterising the layered queuing method are as follows based on the experimental analysis in Section 3. The key variable is n_v, the number of clients at which the layered queuing model is parameterised for each service class v. This is because the layered queuing method assumes that the per-service class request processing times ($t_{(p,v)}$) are constant at different server loads. However this was not found to always be the case due to system (i.e. garbage collection) and thread overheads at small and large numbers of clients, respectively. As n_v and hence the server load is increased, so does the parameterisation overhead. We have found that there tends to be a minimum number of clients that gives a high accuracy and low overhead (e.g. n_{browse} =125 clients in our setup - see table 1). The procedure in Section 3 should be used to identify this point. Alternatively if spare machines are available to parameterise the model the procedure can be used to locate the high overhead parameterisation point that gives the maximum accuracy (see Table 1).

The implications of the layered queuing model experimental results presented in this section are discussed further in Section 6.

4. Definition and Experimental Investigation of The HYDRA Historical Prediction Model

This section describes the experimental investigation of the level of support provided for a *dynamic−urgent cloud environment* using a ('HYDRA') historical prediction model, examining the associated overheads and hence providing guidelines on the parameterisation of the model.

The historical modelling technique [4,5,6] involves sampling performance metrics and associating these measurements with variables representing the workload being processed and the machine's architecture. Historical models then define the relationships (e.g. linear/exponential equations) between the variables and metrics. In the case study, the server workload variables are the number of clients n and the percentage of *buy* clients b. The main server architecture variable is request processing speed (measured as max_thr, the max. throughput under the typical workload, in these experiments). Let $no\_of\_clients_{mt}$ be the number of clients at max_thr. *Relationship* 1 models the effect of $no\_of\_clients$, the number of typical workload clients, on the mean response time. It has been found that this relationship is best approximated using separate lower and upper equations - see equations 2 and 3 respectively. A middle equation (not shown for brevity) the same as equation 2 with L replaced by M.

$$mrt_L = c_L e^{(\lambda_L \times no\_of\_clients)} \tag{2}$$

$$mrt_U = \lambda_U \times no\_of\_clients + c_U \tag{3}$$

Where mrt_L/mrt_U is the mean response time for when $no\_of\_clients < (0.75 * no\_of\_clients_{mt})$ and $no\_of\_clients > (1.1 * no\_of\_clients_{mt})$ respectively, and c_L, c_M, c_U, λ_L, λ_M and λ_U are parameters that must be parameterised from historical data. $no\_of\_clients_{mt}$ is calculated using a relationship between *no_of_clients* and the server throughput (see [4]). We have also found, although it is not covered in this paper, that using a transition relationship to phase between the lower and middle, or middle and upper equations can increase predictive accuracy [5].

Relationship 2 models the effect of *max_thr* on *relationship* 1 as follows:

$$c_L = \Lambda(c_L) \times max\_thr + C(c_L) \tag{4}$$

$$\lambda_L = C(\Lambda_L) \times max\_thr^{\Lambda(\lambda_L)} \tag{5}$$

Where $\Lambda(c_L)$, $C(c_L)$, $C(\lambda_L)$ and $\Lambda(\lambda_L)$ are parameters that must be parameterised from historical data. For brevity equations for the middle equation (equations 4 and 5 with L replaced by M) are not shown. The equations for parameters for the upper (linear) equations are calculated as follows. Given an increase/decrease in *max_thr* of $z\%$, λ_U is found to increase/decrease by roughly $1/z\%$, and c_U is found to be roughly constant.

Relationship 3 models the effect of b on *max_thr*; this is found to be a linear relationship. This is used to extrapolate $max\_thr_E(b)$, the max. throughput of an established server under b. The max. throughput on a new server under b, $max\_thr_N(b)$, is then calculated using equation 6, where $b = 0$ represents the typical workload.

$$max\_thr_N(b) = \frac{max\_thr_E(b)}{max\_thr_E(0)} \times max\_thr_N(0) \tag{6}$$

4.1. Experimental Results

The remainder of this section experimentally investigates the parameterisation of historical models on a live system, using the experimental setup from Section 2. This allows an historical model to be parameterised at a significantly smaller resource usage overhead than the layered queuing method as the only additional requirement on the system is to process the one (or more) response time sampling clients. The parameters in *relationship* 1 and *relationship* 2 are parameterised by fitting least squares trend-lines to historical data from the established $AppServ_F$ and $AppServ_{VF}$ servers. The historical data consists of the *max_thr* of each server and *dp* data points for each equation q (either 'upper', 'middle' or 'lower') of *relationship* 1 respectively. Each data point records *no_of_clients* and the mrt_q (averaged across s samples) for the typical workload.

The overall predictive accuracy is defined as the mean of the lower, middle and upper equation predictive accuracy. It is found that accurate predictions can be made even when *dp* is reduced to 2 and s is reduced to 50. The resulting parameters are shown in Tables 3 and 4. A good level of predictive accuracy of 94% and 78% for the established/new server architectures respectively is achieved (see Table 2). *Relationship* 3 can be rapidly parameterised as this only requires one additional item of data; $max\_thr_E(b)$ for a $b > 0$. This is tested using LQNS predictions for historical data, specifically $max\_thr_E(25) =$

Table 3: Calculated parameters for the new server architecture.

c_L	λ_L	c_M	λ_M	c_U	λ_U
138.9	$4E^{-06}$	51.6	0.0033	-7573	13.54

Table 4: Calculated parameters used to calculate the parameters in table 2.

$\Lambda(c_L)$	$C(c_L)$	$C(\lambda_L)$	$\Lambda(\lambda_L)$	$\Lambda(c_M)$	$C(c_M)$	$C(\lambda_M)$	$\Lambda(\lambda_M)$
-0.55	186.05	6E-14	4.05	712019	-2.14	0.023	-0.431

158 (requests/sec) for $AppServ_F$. The resulting predictive accuracy is 74% on the new server architecture. Once equations 2-6 have been parameterised predictions can be made almost instantaneously.

Having calculated the predictive accuracy, it is necessary to calculate the overheads that were incurred. The total sampling time t required to parameterise the model (that is, the time to parameterise the model using one instrumented client) is estimated in equation 7 where s_i, r_i and h_i are the number of samples, mean response time and mean think-time when recording data point i, and DP is the total number of data points. Since r_i will not be known (and cannot be predicted) until after the model has been parameterised it is not normally possible to predict in advance exactly how long the parameterisation will take. However, r_i can be estimated, for example by using the measurements/predictions from the previous parameterisation or, in the case of a new application, from when the application was being tested prior to deployment. Since distributed enterprise think-times can be quite long (for example 7 seconds as recommended by IBM for Trade clients) t can be quite large. For example in this case the parameters are $DP = 6$ for each of the two server architectures, and $s_i = 50$ and $h_i = 7000ms$ for each data point. The historical model parameter values are used as the r parameters. This results in $t = 76$ minutes.

However, by using multiple simulated clients to sample the response times this can be dramatically reduced. When using c simulated clients to record each data point the resource usage overhead o on the system can be calculated by equation 8 where l_i is the total number of clients for data point i. For these predictions c is set to 12 so that the maximum resource usage overhead for a data point is 5% (for data points recorded at 250 clients). The resulting overall resource usage o is calculated at 1.8% for $AppServ_F$ and 0.91% for $AppServ_{VF}$; a low overhead. The overhead is less for $AppServ_{VF}$ as it is a more powerful server architecture. Alternatively the value of c could be increased so as to parameterise the model more rapidly, albeit at a higher resource usage overhead. The time to parameterise the model when $c > 1$ is calculated as equation 9.

In this case this results in a time to parameterise of 3 minutes 12 seconds for $AppServ_F$ and 3 minutes 9 seconds for $AppServ_{VF}$. In these experiments the system was warmed up for 30 seconds prior to recording each data point, creating an additional overhead of 3 minutes for each server. However, due to the high throughputs of the systems under parameterisation (35-320 requests/second) it is likely that the system reaches a steady state more rapidly than this and so this warm-up interval could be reduced. Indeed it is likely to be particularly effective to base the duration of this warm-up interval on the server throughput, with shorter warm-up times for higher throughput data points. Overall these calculations show that the historical model can be rapidly parameterised at a low resource usage overhead.

$$t = \Sigma_{i=1}^{DP} \left[s_i(r_i + h_i) \right] \tag{7}$$

$$o = \frac{\Sigma_{i=1}^{DP} \left[s_i(\frac{c}{l_i}) \right]}{\Sigma_{i=1}^{DP} s_i} \times 100 \tag{8}$$

$$t = \Sigma_{i=1}^{DP} \left[\lceil \frac{s_i}{c} \rceil (r_i + h_i) \right] \tag{9}$$

4.2. Guidelines For Parameterising the Model at a Low Overhead

Our guidelines for achieving accurate predictions at a low overhead using the historical technique are as follows based on the experimental analysis in this section. It is necessary to sample the response times of established servers at a range of loads for the lower, middle and upper equations of *relationship* 1. This is due to the changing shape of the response time graph as server load increases (see [4]). It is also necessary to use two (or more) established application server architectures for parameterisation so as to be able to extrapolate a trend-line (unlike the layered queuing method which only requires one). Further, when sampling the response times of an established server for the lower, middle or upper equations it is necessary to include samples at both low and high numbers of clients so as to get a sufficient spread of data points from which to draw a trend-line. Initial experiments have shown that exponential predictive accuracy increases significantly if three or more (number of clients, mean response time) data points are used for parameterisation as opposed to the current two. We therefore recommend that a minimum of two/three data points (with at least 50 samples per data point) be used when parameterising linear/exponential trend-lines, although in practice the more data points used the better.

The implications of the historical model experimental results presented in this section are discussed further in section 6.

5. Experimental Investigation of Using Prediction Models in a Cloud Workload and Resource Management System

Sections 3 and 4 have experimentally investigated the level of support for dynamic-urgent cloud environments provided by the layered queuing model and HYDRA historical model respectively. This has included showing that rapid predictions can be made for an enterprise application on a new server architecture in a dynamic-urgent cloud environment with a good level of predictive accuracy.

These sections have also shown that this can be achieved despite rapidly parameterising the model at a low overhead. We now move on to experimentally investigate the use of these prediction models by the management systems of both the cloud (i.e. the resource management system) and cloud customer (i.e. the workload management system). Using the historical and layered queuing models in this section we define a cloud resource and workload management algorithm that uses simulations as part of the decision-making process. We then experimentally investigate via simulation the effectiveness of this algorithm in different operating scenarios. This includes how a prediction-enhanced workload and resource management algorithm can be tuned, and the definition, implementation and experimental investigation of an algorithm that does not use predictions.

SLA-based cloud customers running enterprise systems (as defined in Section 2) incur two main types of cost. The first involves paying penalties for SLA failures e.g. missing SLA response time goals; and the second is the cost of using the servers in the system. $\%\_SLA\_failures$ is defined as the percentage of clients rejected from the server/s of a cloud customer (CC) and $\%\_server\_usage$ is defined as the percentage of the total cloud shared server (SS) processing power (SSPP) allocated to a CC, with processing power defined as max_thr. A generic strategy to compensate for predictive inaccuracy and balance the cloud customers' costs involves running the resource and workload management algorithm, as if the cloud customers' workloads are larger than they actually are. In these experiments this involves multiplying the number of clients in each service class in each CC by $slack$.

Some service classes may have insufficient servers for $\%\_SLA\_failures = 0$ to be achieved, e.g. due to predictive inaccuracy. To deal with this the enterprise system model in Section 2 is extended so application servers reject clients at runtime if response times are within a threshold of missing SLA goals. It is assumed these clients miss their SLA goals. This prevents all the existing clients on a server from also missing their SLA goals. In practice, it is likely that the rejected workload would be handled by a second set of servers in the cloud for each CC, that accept all workload for that customer.

The historical and layered queuing models are used in these experiments as follows. The layered queuing method can require significant CPU time to make each mean response time prediction (i.e. up to 5 seconds on an Athlon 1.4Ghz under a convergence criterion of 20ms). Further, multiple predictions must be made to make the reverse prediction i.e. searching for the maximum number of clients an SLA compliant server can support. This is information that may be requested frequently by workload and resource management algorithms and can be provided almost instantaneously by the historical model. A workaround to this problem is to fit trend-lines to the layered queuing mean response time predictions (using historical model $relationship$ 1 only) and use these to make all predictions almost instantaneously, albeit at the cost of a 1.7% reduction in predictive accuracy. This reduction is very small in part because of the absence of experimental noise. In the experiments in this section these 'hybrid' predictions are used as the predictions and the more accurate historical model is used as the real system response times so as to be able to provide the workload and resource management algorithm with all the required data.

5.1. Defining the Workload and Resource Management Algorithm

Algorithm 1 is our cloud resource and workload management algorithm for use when there is no suitable unallocated server processing power available to process an urgent application. The algorithm reassigns servers from one CC to another and assigns workload to servers. In the experiments in this paper the function $selectCandidateCC(CCx, p, ...)$ returns all the cloud customers except CCx and function $selectCandidateServers(c, ...)$ returns all SS in c unless a SS has already been allocated from this CC in which case no SS are returned. Step 9 is implemented by the resource management algorithm (RM) repeatedly searching the set of tuples generated by step 7 for the tuple with the lowest value of m, and removing that tuple and the corresponding shared server; this repeats until $((SSPP\_removed) \geq p)$ or there are no more tuples, where $SSPP\_removed$ is the shared server processing power removed.

Algorithm 1 Distributed resource and workload management algorithm

```
1: Cloud customer (CC) $CCx$ requests $p$ shared server processing power (SSPP) from the cloud resource management system (RM).
2: RM executes $cc = selectCandidateCC(CCx, p, ...)$ to select a set of CC it might take SS from, using the policy set by the cloud administrators.
3: RM gathers information by asking the workload management system (WM) in each (CC $c$ in $cc$) to predict the effect on their $\%\_SLA\_failures$ of losing: 1 SS; 2 SS; up to $min(p, spp(c))$ SSPP. $spp()$ is a function returning the total SSPP of a CC or server. The WM for each CC $c$ in $cc$ sends the information by:
4: parameterise predictions model, copy configuration of $c$ into new workload transfer algorithm (WTA) simulation $sim_c$.
5: for all SS $s$ in $selectCandidateServers(c, ...)$ do
6:     copy $sim_c$ into $sim_{c,s}$ and use this to simulate transferring the workload from $s$ onto the other SS in $c$, removing $s$ from $c$, and then (assuming at this stage accurate predictions) predict the resulting $\%\_SLA\_failure$ $f_{c,s}$.
7: end for
8: The server $s$ with the lowest value of $m = f_{c,s}/spp(s)$ is selected. delete $sim_c$ and rename $sim_{c,s}$ to $sim_c$. Send $(c, s, f_{c,s}, m)$ to RM.
9: RM selects set of CC and decides on SSPP to take from each.
10: For each selected CC $c$, the WM tells its dispatcher, for each SS it is to lose, to transfer workload off it as per $sim_c$. As SS become idle they are transferred to $CCx$ by RM.
```

The workload transfer algorithm WTA provides, for a cloud customer c, an allocation of workload to a set of servers using information provided by a performance model, calculating the resulting $\%\_SLA\_failures$ and $\%\_server\_usage$.

Table 5: Workload and Resource Management Results

Cloud Customer (CC):	1	2
Server Architecture Taken	$AppServ_S$	$AppServ_F$
Server Speed (Requests/sec)	86	186
$\%\_SLA\_failures$ $(slack = 1.0)$	0.0	2.64
$\%\_SLA\_failures$ $(slack = 1.1)$	0.0	0.25
$\%\_SLA\_failures$ $(slack = 1.2)$	0.0	2.80

Since there is no priority queuing or processing in the system model, our WTA aims to avoid workload with different SLA response time goals on the same server. To improve execution time our WTA executes as follows: i.) work through the service classes in order of priority; ii.) for each service class compile a set bp of the best possible servers for the workload w in the current service class; and then allocate w to the servers in bp being considerate to other service classes. The algorithm is shown in algorithm 2. Our WTA simulation can, given data representing the real performance of the servers, calculate the resulting $\%\_SLA\_failures$ and $\%\_server\_usage$. Each service class consists of a number of clients, each of which is initially unallocated. Application servers are considered to have available capacity unless the performance model predicts that adding an extra client from the current service class would result in some clients missing SLA response time goals.

Algorithm 2 Workload transfer algorithm WTA.

```
1: Sort the service classes in order of increasing response time goal.
2: Let current_service_class = first service class in list
3: repeat
4:     if all clients in current_service_class allocated to an application server then
5:         current_service_class = next service class in list
6:     end if
7:     app_server = application_server_selection_algorithm()
8:     repeat
9:         allocate clients from current_service_class to app_server
10:    until maximum capacity is reached on app_server OR all clients in current_service_class are allocated to an application server
11: until application servers have no available capacity and all clients allocated
```

5.2. *Example Use of Predictions in a Cloud Workload and Resource Management System*

This sub-section describes an example use of Algorithm 1 using the sample cloud from Section 2. We use WTA to allocate each CC workload onto the SS belonging to that CC. We observe that for all CC there are no $\%\_SLA\_failures$. $FireGrid$ (CC_4) then requests the equivalent of $3{\times}AppServ_S$ worth of shared server processing power (SSPP) (i.e. $p = 258$ *requests/second*). This is to take *FireGrid*'s number of SS up to 8 – our standard number for a small fire (e.g. in one room). We run algorithm 1 with $slack = 1$ (i.e. no predictive accuracy compensation). For each of CC_1 and CC_2 there is one SS for which $m = 0$ - see table 5. This means the predictions indicate *FireGrid* can be given the required servers without causing any $\%\_SLA\_failures$, once WTA is used for each CC to re-allocate the displaced workload. (This is not the case for CC_3; $m = 0.022$ so CC_3 is not considered further.) However there is a (non-uniform) predictive inaccuracy so some $\%\_SLA\_failures$ occur. Table 5 shows how setting $slack = 1.1$ (which results in the same m values of 0) significantly improves this. However increasing the slack too far can put too little workload on some of the SS and hence be less effective at reducing SLA failures. An example of this on the selected two servers is shown in the last line of Table 5 for when $slack = 1.2$. The value of $slack = 1.1$ is found by manual experimentation. This approach is selected as we have found experimentally that if predictive accuracy is uniform it can be straightforward to calculate a good value of *slack* but in more realistic scenarios manual experimentation is often a good approach [4].

5.3. Experimental Investigation of the Effectiveness of Prediction-Enhanced Workload-Resource Assignment in Different Operating Scenarios.

The remainder of this section experimentally investigates the effectiveness of prediction-enhanced workload-resource assignment in different operating scenarios. We have found in previous experiments that there are three important operating scenario variables to cover when investigating the effectiveness of a prediction-based management system: 1) the extent to which the system is loaded; 2) the scenario heterogeneity; and 3) whether the performance predictions are tuned to compensate for predictive inaccuracy [4,20]. We therefore investigate these variables. This study builds on our analysis of tuning SLA failure versus resource usage in a prediction-based management system [4]. The experimental setup is extended to include a total of 16 application servers. Eight of the servers have a new architecture ($AppServ_S$) and eight have the same architectures as existing servers (four $AppServ_F$ and four $AppServ_{VF}$) 100% resource usage is defined as using all 16 servers.

Algorithm 3 shows our workload-resource assignment algorithm that does not use predictions, based in part on one of the algorithms that comes as standard in the IBM Websphere platform. In these experiments we compare this algorithm to the prediction-enhanced algorithm. We note that the weighted allocation of workload to servers could be further refined by also weighting by service class mean request processing time. However this is not done here due to the difficulties that have been found measuring mean processing times (without queuing delay) of multiple service classes, in real system environments [15]. The processing power of each server is measured as the maximum throughput of the server under the typical workload.

We begin the experiments using the workload from Section 2, which we refer to as the standard workload. The average predictive accuracy of the (non-

Algorithm 3 No prediction workload-resource assignment algorithm.

```
1: Let total_processing_power = sum of processing powers for all servers.
2: Calculate the proportion of total_processing_power allowed to go to each
   service class by weighting each service class in inverse proportion to its SLA
   goal. Assign this to service class variable allowed_processing_power.
3: Let unallocated_processing_power = server_processing_power
4: Let servers = all servers ordered by processing power
5: for all server in servers do
6:     Select the highest priority service class with allowed_processing_power ≠
       0.
7:     Allocate as much processing power to this service class as it is allowed
       (or until the server runs out of unallocated processing power) and hence
       proportionally allocate clients from that service class to the server.
8: end for
```

Table 6: % SLA failure per % resource usage at different numbers of clients for the standard workload as shown in figure 3

No. of Clients	Using predictions (not tuned)	Using predictions (tuned)	No predictions
3000	0.00	0.00	0.00
6000	0.18	0.00	0.00
9000	0.00	0.00	0.00
12000	0.06	0.00	0.04
15000	0.06	0.01	0.12
18000	0.02	0.00	0.18
21000	0.10	0.11	0.22
24000	0.23	0.24	0.29

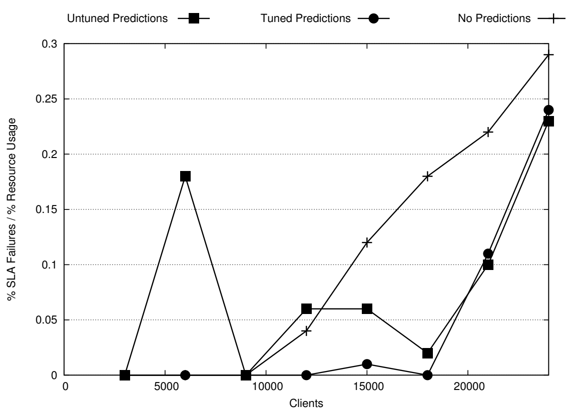

Figure 3: % SLA failure per unit resource usage at different numbers of clients for the standard workload

uniform) predictions (weighted by the number of servers in the server pool) is 92.5% (corresponding to a slack of 1.075). However the minimum slack that results in 0% SLA failures before 100% resource usage is 1.1. The difference is due to some predictions being used more by the prediction-enhanced algorithm than others. For example the predictive accuracy of $AppServ_F$ is the highest of the three servers at 97.04%, but due to the design of the prediction-enhanced algorithm the middle servers tend to be used less frequently. We set the slack at 1.075 as we have found that allowing for a small number of SLA failures at about this level can result in a disproportionately large improvement in the % resource usage [4].

Table 6 and Figure 3 show the results. It can be seen that the tuned prediction-enhanced algorithm significantly outperforms the no prediction algorithm. However if the prediction-enhanced algorithm is not properly tuned its performance is much more variable and at some numbers of clients performs worse than the no prediction algorithm. It is noted that the irregular shape of e.g. the 'not tuned' line is because runtime optimisations allow the system to use any available capacity the algorithm leaves on a server. So, once the total workload crosses a threshold and a small number of clients are allocated to an additional server, the performance will temporarily improve (as can be seen at 9000 clients).

For the next set of results we create a workload in which the mean buy service class SLA response time goal (RTG) is divided by two (so RTG=75ms). In addition, the low priority browse service class RTG is doubled (so RTG=1200ms). We refer to this as the more heterogeneous workload. The tuning is not modified since being set based on the standard workload. The results for the more heterogeneous workload are shown in Table 7 and graphed in Figure 4. It can be seen that at small numbers of clients the tuned prediction-enhanced algorithm does significantly worse than the no prediction algorithm. This illustrates the importance of considering a range of likely operating scenarios when manually tuning the prediction-enhanced algorithm. In this instance the value of *slack* set using the standard workload is too low and should be increased – albeit at the cost of a small increase in % resource usage. Table 7 also shows how the no prediction algorithm does significantly worse under the more heterogeneous workload. This is discussed further in Section 6.

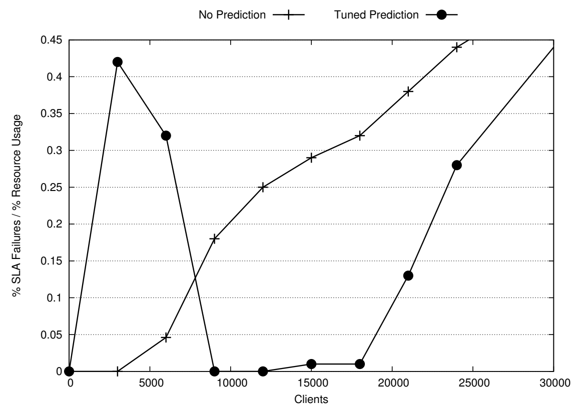

Figure 4: % SLA failure per unit resource usage at different numbers of clients, for the more heterogeneous workload.

Table 7: % SLA failure / % resource usage at different numbers of clients, for the more heterogeneous workload as shown in Figure 4.

No. of Clients	No prediction	Prediction (tuned)
3000	0	0.42
6000	0.046	0.32
9000	0.18	0
12000	0.25	0
15000	0.29	0.01
18000	0.32	0.01
21000	0.38	0.13
24000	0.44	0.28

An important difference between a properly tuned) prediction-enhanced algorithm and a no prediction algorithm is as follows. With the prediction-enhanced algorithm as the number of clients increases, first all the lowest priority service classes fail their SLAs, then the next lowest priority service class etc. In contrast there is only limited control of the order (and extent) to which the service classes are effected as the number of clients increases, in the no prediction algorithm. That is, higher priority service classes can be put on faster servers with established server architectures, but this does not give any guarantee. The SLA failure per service class results for the standard workload are shown in Tables 8 and 9. Service classes with no SLA failures for the data range shown on each graph are removed for clarity. For the no prediction algorithm the SLA failures for smaller numbers of clients are from the low priority browse service class. However once the % SLA failures for this service class reaches just under 50% the high priority browse service class starts to suffer SLA failures. In contrast it is not until the prediction-enhanced algorithm reaches 100% SLA failures for the low priority browse service class, that the high priority browse service class starts to suffer SLA failures.

In summary this section has experimentally investigated the effectiveness of using performance predictions in different operating scenarios. These operating scenarios have been selected as those our previous work has identified as important when investigating the effectiveness of performance predictions [4,20]. The implications of the experimental results presented in this section are discussed further in section 6.

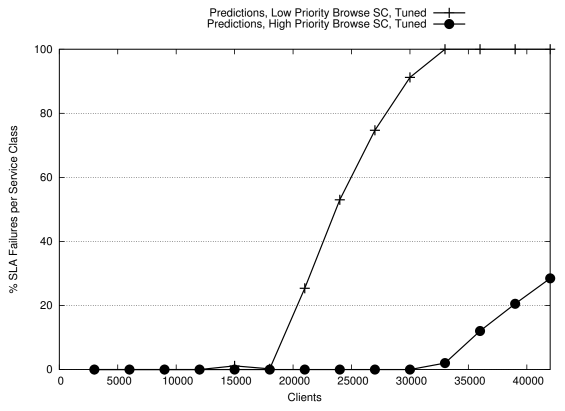

Figure 5: % SLA failure per service class at different numbers of clients, for the standard workload with the prediction-enhanced algorithm.

6. Evaluations, Guidelines and Discussion

In this section, based on the experimental investigation in Sections 3 to 5, we: i.) provide a criticism of the layered queuing and historical methods and the application of these methods in this paper; ii.) evaluate the effectiveness of the prediction-enhanced workload and resource management algorithm from Section 5 in different *dynamic−urgent cloud environment* operating scenarios; and iii.) provide guidance on using a prediction-based workload and resource management algorithm in a *dynamic − urgent cloud environment*.

Table 8: % SLA failure per service class (SC) at different numbers of clients, for the standard workload with the prediction-enhanced algorithm as shown in Figure 5.

No. of Clients	Predictions (tuned) (low priority browse SC)	Predictions (tuned) (high priority browse SC)
3000	0.00	0.00
6000	0.00	0.00
9000	0.00	0.00
12000	0.00	0.00
15000	1.16	0.00
18000	0.23	0.00
21000	25.37	0.00
24000	53.00	0.00
27000	74.69	0.00
30000	91.21	0.00
33000	100.00	2.03
36000	100.00	12.04
39000	100.00	20.52
42000	100.00	28.47

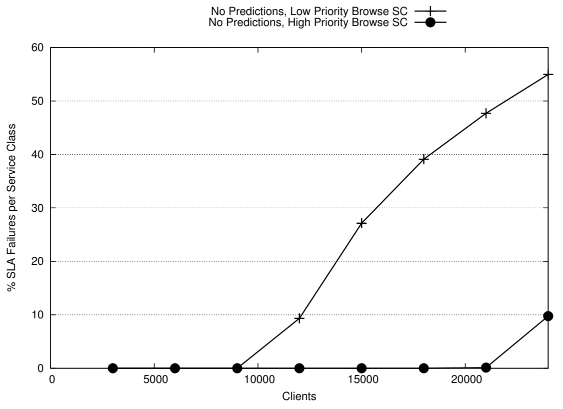

Figure 6: % SLA failure per service class at different numbers of clients, for the standard workload with the no prediction enhanced algorithm.

6.1. Criticism of the Layered Queuing and Historical Methods and the Application of these Methods in this Paper

6.1.1. Criticism of the Layered Queuing and Historical Methods

We have provided a detailed criticism of the historical and layered queuing methods and their application under five evaluation criteria in [6]. This includes both model description and analysis. Overall, there is no clear modelling paradigm of choice as the historical method and layered queuing method

Table 9: % SLA failure per service class at different numbers of clients, for the standard workload with the no prediction enhanced algorithm as shown in Figure 6.

No. of Clients	No predictions (low priority browse SC)	No predictions (high priority browse SC)
3000	0.00	0.00
6000	0.00	0.00
9000	0.00	0.00
12000	9.33	0.00
15000	27.14	0.00
18000	39.11	0.00
21000	47.69	0.12
24000	54.97	9.76

are strong under different evaluation criteria. Four of these evaluation criteria (systems that can be modelled, metrics that can be predicted, ease of use given a minimal level of performance modelling expertise and prediction responsiveness) are evaluated qualitatively based on the underlying characteristics of the two methods.

Some highlights are as follows. There are a number of functional limitations with the layered queuing method that should be taken into account when selecting a technique. It is more difficult to model applications that cache significant amount of database data at the application server (as opposed to applications such as the *Trade* benchmark which access the majority of database data directly so as to avoid data inconsistencies if the application server crashes). This is because the number of calls to the database must be a constant in the layered queuing model, whereas if a cache is used this value will depend on the cache miss rate. Using the historical technique the size of the application server's cache can be recorded as an extra variable. Relationships can then be added to approximate the historical relationship between the performance metrics, this new variable and the existing variables, using the techniques presented in section 4. Another limitation of the layered queuing method is that the important class of percentile response time metrics cannot be predicted directly. In contrast the historical technique can extrapolate from and hence make predictions for a wide range of metrics.

However layered queuing models are also significantly easier to create with a minimum level of performance modelling expertise than a historical model. This is because creating a historical model involves specifying and validating how predictions will be made, whereas once a system's queuing network configuration is specified layered queuing models can be solved automatically. The layered queuing method may therefore be preferable when there is a shortage of either time or performance modelling expertise when creating the model. The main conclusion under the 'prediction evaluation delay' criteria is that the historical method can be significantly more responsive than the layered queuing method. This is based primarily on the fact that the historical method supports runtime predictions made by a single equation such as (once parameterised) equations 2 or 3. However the procedure for solving a layered queuing model, as outlined in section 3, can require significantly more resources to solve via e.g. iteration or simulation. Indeed, this is what we have found in our experimental experience (see results in section 3.1).

We also note that the historical technique has tool support primarily to help collect the historical data – however this is aimed at expert users (see the HYDRA Toolkit [6]). The layered queuing method has tool support for both beginners and more experienced users, including a model validator, solver and GUI editor. Further, although both techniques are well documented (e.g. [4,5,6,9,10,11]), it is once again only the layered queuing method that provides material for both beginner and more expert users.

Our criticism of the methods under the 'high accuracy new server architecture predictions given low overhead parameterisation' evaluation criteria is based in part on the underlying characteristics of the methods and in part on the quantitative experimental results from this study. This is discussed further in section 6.1.2.

6.1.2. Criticism of the Application of the Layered Queuing and Historical Methods in this Paper

This sub-section provides a critical discussion on the application of the layered queuing and historical methods in this paper (i.e. for model description and analysis). We have shown in this paper it is possible to make accurate new server architecture predictions after only rapid low overhead parameterisation using both the layered queuing and historical methods. The following qualifies this statement. A historical model may perform poorly if there are problems with the historical data e.g. due to the samples per data point being too small; the historical data not covering the full range of likely system states; there being insufficient data points per model relationship; or due to experimental error. Both the analysis (for parameterisation) and model description rely on this historical data. We refer the reader to our analysis of the effect on the historical model of insufficient historical data [4, 6]. We have found that some layered queuing models may be best solved via a significantly slower solver tool as opposed to the faster but potentially less accurate analytical approximation tool (LQNS) used in this paper (see also the note below about this tool). During layered queuing model analysis sources of parameterisation uncertainty include experimental error and the number of clients at which the model is parameterised - see the analysis in section 3.1. See also [6] for further qualifications for both methods.

For a justification and discussion of the main assumptions we make see [6] and elsewhere in this paper. We note in particular the following. We have validated our exponential service times assumption experimentally for the experimental setup - and this is supported by e.g. [10,11]. Our future publications will study metrics based on quantiles or higher moments. This will be based in part on our initial results predicting percentile response times with a good level of accuracy, based on predicted exponential response time distributions [6]. We have validated our experimental performance data and experimental setup using manual checking, our layered queuing model and comparison to related studies [1,11,15].

See [6] for our detailed discussion of alternative performance predictions methods and alternative ways of applying the layered queuing and historical models. For example, in our model description an alternative would be experimentally investigating variations (e.g. approximations) on the models in this study based on our initial results and guidance on this in [6]. This is particularly important with the layered queuing method as modelling variations can occasionally prevent the LQNS solver producing a prediction. Examples of potential alternative analysis include introducing additional noise artificially to parameterisation data; or doing extra analysis on the historical model by creating a hybrid of the historical and layered queuing model (see [6]). This hybrid involves generating 'pseudo' historical (workload, performance) data points for a server architecture using a layered queuing model, and using these data points to parameterise the relationships in a historical model. This is then used to make predictions which can be tested against the layered queuing prediction even if real historical data is not available. Disadvantages include a reduction in predictive accuracy and having to create and parameterise two models.

We conclude by noting that we have chosen the historical and layered queuing methods - and the application of the methods described in this paper - because they are effective and we are familiar with them. We expect this effectiveness to deteriorate the further a potential system and usage scenario is from that outlined in this paper, as discussed above. Future work includes doing sensitivity and transient analyses. This will be based on our previous analyses of the effect of key variables which can change predictive accuracy. These include [4,6], section 3.1 and our work on the effect of predictive innaccuracy in workload and resource management [21].

6.2. Evaluation of the Effectiveness of the Prediction-Enhanced Workload and Resource Management Algorithm in Different Operating Scenarios

This section considers the effectiveness of the prediction-enhanced workload and resource management algorithm in different *dynamic−urgent cloud environment* operating scenarios. This considers the prediction-enhanced workload and resource management algorithm with both the layered queuing and historical models plugged in.

The more heterogeneous an operating scenario is (e.g. in terms of the workload and resources), the more unique options there are likely to be for how workload is allocated to resources. This is likely to give the prediction-enhanced algorithm an increasing edge over the no prediction algorithm as the system becomes more heterogeneous. For an illustration of this effect see Figure 4. This shows how the improvement the prediction-enhanced algorithm provides over the no prediction algorithm is greater in the more heterogeneous operating scenario. It must also be considered that as the number of workload-resource allocation decisions to consider (and hence the number of predictions required) increases, the management algorithm will require more resources to calculate predictions. This will be particularly noticeable when using layered queuing predictions, and may be almost unnoticeable when using historical predictions on small clouds. For example, in the experiments in this paper, assigning a workload with three service classes to a set of 16 servers using the historical model took only a maximum of a couple of seconds to complete. An additional consideration is that in operating scenarios that are consistently very homogeneous there may be a shortage of historical data for the historical model to extrapolate from. For example, if predictions are required urgently to evaluate unusual system changes being considered by the workload and resource management system.

In increasingly lightly loaded operating scenarios, the costs of using predictions for workload and resource management are increasingly likely to out-weigh the benefits. In an operating scenario that is so heavily loaded the system is almost saturated, predictions may not be able to stop the system having undesirable levels of performance. However, they are likely to be useful in other regards – for example, in more precisely controlling which parts of the workload suffer. An example of these behaviours is presented in the per-service class results in Section 5.

Predictions can be particularly useful if a cloud and/or cloud customer are interested in packing workload tightly onto machines (e.g. if the system is heavily loaded or to make financial savings on server/resource usage costs). However the more tightly the workload is packed the more carefully the predictions must be tuned to avoid unexpected effects (see for example table 7). Predictions can also be useful for helping control the balance between competing costs in a system. For example between the costs of SLA failures and server/resource usage using the slack tuning parameter.

6.3. Guidance on Using a Prediction-Based Workload and Resource Management Algorithm in a Dynamic Urgent Cloud Environment

If predictions are to be used in a cloud, it may be advantageous to be selective about the workload-resource allocation decisions for which they are used. This level of selectivity is likely to be made at a coarse-grained level. For example, this decision could be made by the cloud adjusting how/when the resource manager uses predictions based on the current cloud-level operating scenario. For example, only using predictions when the cloud is heavily loaded. The decision about whether to use predictions at all is also likely to be made by each cloud customer. Further, the current cloud customer operating scenario for each cloud customer can be used to adjust this decision at runtime. For example, if a particular cloud customer has a significant amount of low-priority workload it may conclude there is no need to use predictions at this point. Being selective in these ways can help address the potential issues identified in section 6.1, such as the range of systems that performance predictions can be applied to. It can also help with the ease of use given a limited level of performance modelling expertise issue, by reducing the required number of IT staff with model-based system management skills.

At a finer level of granularity a workload and resource management algorithm could be set so that workload-resource allocations with poor potential are discarded, with limited use of predictions. This is particularly useful when using predictions that take seconds as opposed to milliseconds to complete. For example in algorithm 1 this can be achieved by ruling out servers that are considered low potential candidates to be given away to the urgent application. We have found a promising strategy is one in which the estimated potential of each server is calculated based on: i.) an estimate of the cost of giving the server away (based on the number of clients on the server for each service class); and ii.) the potential benefit (based on the processing speed of the server). Being selective at a finer-level of granularity can help improve the responsiveness of the workload and resource management system, especially when using layered queuing predictions.

In this paper we have shown that the historical model can make accurate predictions with only a small amount of historical data. In some operating scenarios (e.g. if there is a limited range of historical data due to the system being fairly homogeneous) it can be useful to take actions to increase the range of historical data. This can be achieved by moving workload between resources during parameterisation so as to artificially increase the heterogeneity of the system. This can give a wider range of collected historical data and hence increase the predictive accuracy. This is at the cost of the associated overheads and delays (as discussed in section 4).

Workload and resource management systems without predictions have the advantage that it is often easier for IT staff to understand and hence predict how they should behave in different operating scenarios. This is because a more straightforward decision-making process is used, making it potentially easier for IT staff to correctly diagnose problems. The experiments in section 5 have illustrated how even if a prediction-enhanced workload and resource management system is working effectively overall, there can still be a small number of operating scenarios where it performs very badly. This can be because the tuning (e.g. via the slack parameter) cannot be perfect in all operating scenarios. We recommend using the process from section 5 to identify operating scenarios in which this effect can be expected.

7. Conclusion

This paper documents the following studies.

- An experimental investigation of the level of support provided for a *dynamic–urgent cloud environment* using our historical and layered queuing prediction model. This includes examining the associated overheads and hence providing guidelines on parameterising the models at a low overhead.

- The definition, implementation and via simulation experimental investigation of the effectiveness of our historical/layered queuing prediction-based cloud workload and resource management algorithm, in different *dynamic* $-$ *urgent cloud environment* operating scenarios. This includes the definition, implementation and experimental investigation of an algorithm that does not use predictions.

- The creation of evaluations and guidance based on the experimental investigations. This includes our evaluation of the effectiveness of our prediction-enhanced management algorithm in different *dynamic* $-$ *urgent cloud environment* operating scenarios. We consider our prediction-enhanced management algorithm with both the layered queuing and historical models plugged in. Our guidance on using a prediction-based management algorithm in a *dynamic* $-$ *urgent cloud environment* is then presented. This includes suggested techniques, and also advice on the potential issues the guidance and techniques can help address. We also include a criticism of the layered queuing and historical methods and the application of these methods in this paper.

The combination of an established system model, popular enterprise middleware (IBM Websphere); and an enterprise benchmark based on best practices (the Websphere Performance Benchmark Sample $Trade$), should make this work of relevance to a wide range of enterprise cloud systems. Future work is likely to include an investigation into the obstacles faced by universities (teaching and assessment as well as research) and businesses in adopting cloud computing, and the extent to which early adopters are overcoming these obstacles to realise reliable and effective cloud solutions.

Acknowledgment

This work was sponsored in part by the JISC TeciRes project, the Technology Strategy Board FireGrid Project, The University of Edinburgh School of Informatics, the EPSRC DOPCHE Project (GR/S03058/01) and IBM UK Ltd. We gratefully acknowledge the assistance during current and earlier versions of this work of: M. Atkinson, T. Berman, G. Beckett, A. Chester, L. He, J. Koetsier, S. Koo, T. Liu, I. Mitrani, A. van Moorsel, G. Nudd, S. Potter, G. Pringle, J. Slegers, C. Smith, D. Spooner, N. Thomas, J. Torero, S. Welch, W. J. Xue, S. J. Pennycook and all FireGrid team members and IBM employees not mentioned here.

References

[1] J. Aman, C. Eilert, D. Emmes, P. Yocom, D. Dillenberger, Adaptive Algorithms for Managing a Distributed Data Processing Workload, IBM Systems Journal, vol. 36(2), 1997, pp. 242-283

[2] Z. Liu, M.S. Squillante, J. Wolf, On Maximizing Service-Level-Agreement Profits, Proc. ACM Conference on Electronic Commerce (EC01), Florida, USA, October 2001

[3] M. Goldszmidt, D. Palma, B. Sabata, On the Quantification of e-Business Capacity, Proc. ACM Conference on Electronic Commerce (EC01), Florida, USA, October 2001

[4] D.A. Bacigalupo, S.A. Jarvis, L. He, D.P. Spooner, D.N. Dillenberger, G.R. Nudd, An Investigation into the Application of Different Performance Prediction Methods to Distributed Enterprise Applications, The Journal of Supercomputing, vol. 34, 2005, pp. 93-111

[5] D.A. Bacigalupo, S.A. Jarvis, L. He, G.R. Nudd, An Investigation into the Application of Different Performance Prediction Techniques to e-Commerce Applications, Workshop on Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems, Proc. 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS04), New Mexico, USA, April 2004

[6] D.A. Bacigalupo, Performance Prediction-Enhanced Resource Management of Distributed Enterprise Systems, PhD Thesis, Dept. Computer Science, Uni. of Warwick, UK, 2006

[7] D. Menasce, Two-Level Iterative Queuing Modeling of Software Contention, Proc. 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS02), Texas, USA, October 2002

[8] Y. Diao, J. Hellerstein, S. Parekh, Stochastic Modeling of Lotus Notes with a Queueing Model, Proc. Computer Measurement Group Int. Conference (CMG01), California, USA, December 2001

[9] C.M. Woodside, J.E. Neilson, D.C. Petriu, S. Majumdar, The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software, IEEE Transactions On Computer, vol. 44(1), 1995, pp. 20-34

[10] F. Sheikh, M. Woodside, Layered Analytic Performance Modelling of a Distributed Database System, Proc. Int. Conference on Distributed Computing Systems (ICDCS97), Maryland USA, May 1997

[11] T. Liu, S. Kumaran, J. Chung, Performance Modeling of EJBs, Proc. 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI03), Florida USA, July 2003

[12] A. Cowlard, W. Jahn, C. Abecassis-Empis, G. Rein, J. Torero, Sensor Assisted Fire Fighting, Fire Technology, Springer Press, 2008. ISSN:0015-2684

[13] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov, Eucalyptus: an open-source cloud computing infrastructure, Journal of Physics: Conference Series, vol. 180, 2009

[14] P. Mell, T. Grance, The NIST Definition of Cloud Computing v15, US National Institute of Standards and Technology ITL Technical Report, 2009. Available at csrc.nist.gov

[15] L. Zhang, C. Xia, M. Squillante, W. Nathaniel Mills III, Workload Service Requirements Analysis: A Queueing Network Optimization Approach, Proc. 10th IEEE Int. Symposium on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems, Texas, USA, October 2002

[16] IBM Websphere Performance Sample: Trade. Available at www.ibm.com/software/info/websphere/

[17] X. Chen, G.B. Wills, L. Gilbert, D. Bacigalupo, JISC TeciRes Project Report, Using Cloud for Research: A Technical Review. June 2010. Available at: tecires.ecs.soton.ac.uk.

[18] D.A. Bacigalupo, J. van Hemert, A. Usmani, D.N. Dillenberger, G.B. Wills and S.A. Jarvis, Resource Management of Enterprise Cloud Systems Using Layered Queuing and Historical Performance Models, Proc 24th IEEE International Parallel and Distributed Processing Symposium (IEEE Press), as part of The 9th International Workshop on Performance Modeling, Evaluation, and Optimization of Ubiquitous Computing and Networked Systems (PMEO-UCNS'10), 2010

[19] J. Slegers, I. Mitrani and N. Thomas, Optimal Dynamic Server Allocation in Systems with On/Off Sources, Proc. Formal Methods and Stochastic Models for Performance Evaluation, Lecture Notes in Computing Science Volume 4748 pp. 186-199, 2007.

[20] L. He, S.A. Jarvis, D.A. Bacigalupo, D.P. Spooner, G.R. Nudd, Performance-aware Load Balancing for Multiclusters, Proc. 2nd International Symposium on Parallel and Distributed Processing and Applications (ISPA'04), Lecture Notes in Computer Science 3358, 2005. Springer Press.

[21] S.A. Jarvis, L. He, D.P. Spooner, G.R. Nudd, The Impact of Predictive Inaccuracies on Execution Scheduling, International Journal of Performance Evaluation, vol. 60(1-4), 2005, pp. 127-139

[22] J. Fito, I. Goiri and J. Guitart, SLA-driven Elastic Cloud Hosting Provider, Proc. 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, Pisa, Italy, February 2010

[23] G. Franks, T. Al-Omari, M. Woodside, O. Das, S. Derisavi, Enhanced Modeling and Solution of Layered Queueing Networks, IEEE Transactions on Software Engineering, vol. 35(2) pp. 148-161, 2009

[24] L. Han, S. Potter, G. Beckett, G. Pringle, S. Welch, S. Koo, G. Wickler, A. Usmani, J.L. Torero, A. Tate, FireGrid: An e-infrastructure for next-generation emergency response support, Journal of Parallel and Distributed Computing, Vol. 70(11) pp. 1128-1141, 2010