Resource Management of Enterprise Cloud Systems Using Layered Queuing and Historical Performance Models

David A. Bacigalupo*, Jano van Hemert†, Asif Usmani‡, Donna N. Dillenberger§, Gary B. Wills* and Stephen A. Jarvis¶
*School of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK
†Data-Intensive Research Group, School of Informatics, University of Edinburgh, EH8 9AB, UK
‡BRE Centre for Fire Safety Engineering, School of Engineering, University of Edinburgh, EH9 3JL, UK
§IBM T.J. Watson Research Centre, Yorktown Heights, New York, 10598, USA
¶High Performance Systems Group, Department of Computer Science, University of Warwick, CV4 7AL, UK
e-mail: db1f08@ecs.soton.ac.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. It is important to quantitatively compare the effectiveness of different prediction techniques for use in cloud infrastructures. 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 $FireGrid$ 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 the cloud infrastructure and cloud customer management systems. 4.) A broad range of criteria for evaluating each technique. However, there have been no previous comparisons meeting these requirements. This paper, meeting the above requirements, quantitatively compares the layered queuing and ("HYDRA") historical techniques – including our initial thoughts on how they could be combined. Supporting results and experiments include the following: $i.$) defining, investigating and hence providing guidelines on the use of a historical and layered queuing model; $ii.$) using these guidelines showing that both techniques can make low overhead and typically over 70% accurate predictions, for new server architectures for which only a small number of benchmarks have been run; and $iii.$) defining and investigating tuning a prediction-based cloud workload and resource management algorithm**

***Keywords*-cloud, performance modelling, HYDRA historical prediction, layered queuing, FireGrid**

I. 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. 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].

It is important to quantitatively compare the effectiveness of different techniques for use in infrastructure (i.e. 'resource') and cloud customer (i.e. 'workload') cloud management systems. 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] 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 comparisons meeting these requirements making it harder to i.) make an informed choice of technique/s; and hence ii.) design effective prediction-based cloud management systems. The following are examples of comparisons which do not meet these requirements. 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 a historical model of the same application. In our previous work [4,5,6] we have compared queuing/historical (i.e. HYDRA) techniques but not met the requirements listed above. 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.

This paper quantitatively compares the HYDRA historical technique and the layered queuing method, meeting the four requirements in the previous paragraph. 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 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 run-time. This has a number of advantages when predicting the performance of dynamic 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).

The contributions of this paper are as follows, focusing on $dynamic$ enterprise systems hosted on a cloud (that may also host $urgent$ systems). The paper: 1.) defines, experimentally investigates and hence provides guidelines on the use of a historical and layered queuing model; 2.) defines and experimentally investigates a prediction-based cloud workload and resource management algorithm; and 3.) comparatively evaluates the layered queuing and HYDRA historical techniques (including our initial thoughts on how they could be combined) for cloud management. The paper is structured as follows: the system model, case study and experimental setup (section 2); the layered queuing method (section 3); the HYDRA historical technique (section 4); cloud workload and resource management (section 5) and the evaluation and guidelines (section 6).

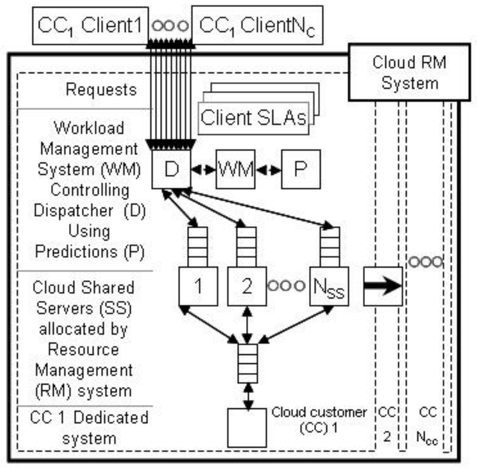
CC1 Client1
CC1 ClientNc
Cloud RM System
Requests
Client SLAs
Workload Management System (WM) Controlling Dispatcher (D) Using Predictions (P)
D
WM
P
Cloud Shared Servers (SS) allocated by Resource Management (RM) system
1
2
NSS
CC 1 Dedicated system
Cloud customer (CC) 1
CC 2
CC NCC

Figure 1. System Model

II. System Model, Case Study and Experimental Setup

Based on [13,14] a cloud customer (CC) pays for shared servers (SS) which may be dynamically shared with other CC and which may be virtualised – see fig. 1. Here, a SS is being transferred from CC1 to CC2; SS transfers are controlled by the 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 workload management system (WM), adjusts the routing of the incoming requests to the application servers. These management systems may use a prediction engine (P) to help make decisions.

The case study is a Eucalyptus [13] cloud 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 e.g. [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 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. For simplicity, 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 embarassingly 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.

The experimental setup contains three application server architectures. Under Websphere v4.0.1. and the typical workload 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). 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.

III. Definition and Experimental Investigation of The Layered Queuing 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].

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. 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.

In the remainder of this section we examine the predictive accuracy and resource usage overhead when parameterising the request processing times under different workloads. 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

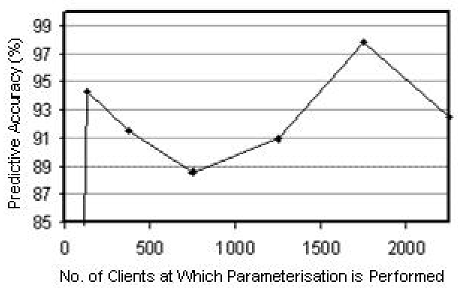

Figure 2. The predictive accuracy when parameterising the layered queuing model at different numbers of clients

Table I
MEAN RESPONSE TIME PREDICTIONS FOR THE NEW SERVER ARCHITECTURE

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

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:

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

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 fig. 2. 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 accuracy sample at this point is very inaccurate, and 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 1. A server capacity of 89 requests/second must be taken offline for this parameterisation 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.

IV. DEFINITION AND EXPERIMENTAL INVESTIGATION OF THE HYDRA HISTORICAL MODEL

The ("HYDRA") 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.

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

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

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.

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

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 a 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 i samples) for the typical workload.

The overall predictive accuracy is defined as the mean of the lower, middle and upper equation accuracy. It is found that accurate predictions can be made even when dp is reduced to 2 and i is reduced to 50. The resulting parameters for the new server architecture are $c_L = 138.9, \lambda_L = 4E^{-06}, c_M = 51.6, \lambda_M = 0.0033, c_U = -7573, \lambda_U = 13.54$. A good level of accuracy of 94% and 78% for the established/new server architectures respectively is achieved (see table 1). For these predictions the $i = 50$ samples for each data point were made sequentially (after a 1 minute warm-up period) using one sampling client. This said, the parameterisation was completed in only 2 minutes.

$Relationship$ 3 can also 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 $max\_thr_E(25) = 158$ (requests/sec) for $AppServ_F$. The resulting predictive accuracy is 74% on the new server architecture.

V. Tuning a Prediction-Enhanced Cloud Workload and Resource Management Algorithm

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. Using the historical and layered queuing models the experiments in this section investigate how a prediction-enhanced workload and resource management algorithm can be tuned using a $slack$ parameter. $\%\_SLA\_failures$ 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 innaccuracy. 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.

```
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 each SS $s$ in $selectCandidateServers(c, ...)$ do:
6.      copy $sim_c$ into $sim_{c,s}$ and use this to simulate
        transfering 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.   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
8.   If (total SSPP for all servers removed from $c$ in $sim_c$)
     $< min(p, spp(c))$ then jump to step 5.
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.
```

Algorithm 1. Distributed resource and workload mangement algorithm

.

Algorithm 1 is our cloud data centre 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. 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 $\%\_server\_usage$. 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. Our WTA simulation can, given data representing the real performance of the servers, calculate the resulting $\%\_SLA\_failures$ and $\%\_server\_usage$. For more details on our WTA and WTA simulation see [4].

Table II
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

The remainder of this section describes the experimental investigation of this algorithm using the sample cloud from section 2. First 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 2. 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 innaccuracy so some $\%\_SLA\_failures$ occur. Table 2 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 2 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].

VI. Model Parameterisation Guidelines and Comparative Evaluation

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 accuracy of the model immediately after parameterisation as it may occassionally 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 fig. 2). 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 fig. 2).

Our guidelines for achieving accurate predictions at a low overhead using the historical technique are as follows based on the experimental analysis in section 4. 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.

When selecting a prediction technique to use we recommend considering the following criteria: parameterisation requirements; the responsiveness of predictions; the systems which can be modelled; the metrics which can be predicted; the ease of use of each technique and performance modelling expertise required; tool support; and documentation. Model parameterisation has been considered above and in the previous two sections; the following is a summary of how the techniques differ in the other categories.

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 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. And 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.

Another option is to create a hybrid of the historical and layered queuing model. One way this can be done is by 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, which is then used to make predictions. This was done in section 5 to increase the responsiveness of layered queuing predictions and hence the speed of algorithm 1. This has the additional advantage of not having to collect real historical performance data. Disadvantages include a reduction in predictive accuracy and having to create and parameterise two models. Hybrids of the layered queuing and historical models will be discussed further in future papers.

VII. Conclusion

This study has comparatively evaluated the layered queuing and ("HYDRA") historical techniques for use in cloud workload and resource management systems. The study has covered the scenario where the systems being cloud-hosted include $dynamic$ enterprise systems and $urgent$ systems (e.g. for emergency response such as our $FireGrid$ application). Based on detailed experimental analysis we have provided guidelines for selecting a technique and obtaining low overhead, high accuracy predictions. The combination of an established system model, a cloud infrastructure (Eucalyptus) compatible with the mainstream Amazon EC2 cloud interfaces; 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 (for research, teaching and assessment) 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 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, G. Beckett, X. Chen, A. Chester, L. Gilbert, 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 and all FireGrid team members 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/