Predicting the Effect on Performance of Container-Managed Persistence in a Distributed Enterprise Application

David A. Bacigalupo[1], James W. J. Xue[1], Simon D. Hammond[1], Stephen A. Jarvis[1], Donna N. Dillenberger[2] and Graham R. Nudd[1]
daveb@dcs.warwick.ac.uk

[1] *High Performance Systems Group, Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK*

[2] *IBM T.J. Watson Research Centre, Yorktown Heights, New York 10598, USA*

Abstract

Container-managed persistence is an essential technology as it dramatically simplifies the implementation of enterprise data access. However it can also impose a significant overhead on the performance of the application at runtime. This paper presents a layered queuing performance model for predicting the effect of adding or removing container-managed persistence to a distributed enterprise application, in terms of response time and throughput performance metrics. Predictions can then be made for new server architectures - that is, server architectures for which only a small number of measurements have been made (e.g. to determine request processing speed). An experimental analysis of the model is conducted on a popular enterprise computing architecture based on IBM Websphere, using Enterprise Java Bean-based container-managed persistence as the middleware functionality. The results provide strong experimental evidence for the effectiveness of the model in terms of the accuracy of predictions, the speed with which predictions can be made and the low overhead at which the model can be rapidly parameterised.

1. Introduction

When designing distributed enterprise applications one of the most important decisions that has to be made is whether to use middleware functionality (e.g. container-managed persistence) to manage the application's access to the database, or whether to implement this manually. For example in the J2EE middleware this functionality can be achieved using Entity Enterprise Java Beans (EJBs) [7] and in the .NET middleware this functionality can be achieved using NJDX [22]. However regardless of the middleware being used, it is normally the case that using the middleware functionality makes it easier to develop and maintain the application, whereas the manual version (for example using SQL along with JDBC or ODBC) can be more time-consuming to develop but has the potential to be significantly more efficient. Thus, when deciding whether to use a middleware or manual-based implementation it can be useful to have quantitative data on the potential increase in efficiency. To avoid having to implement and test two full versions of one application, this quantitative data can be provided by a performance model, and the associated performance prediction method.

Various performance prediction methods have been used in the literature to predict the performance of distributed enterprise applications. For example in [15, 16] a two-level queuing network performance prediction model (for software and hardware respectively) is used. In [14] a queuing model is used to maximize the revenue obtained by a workload manager. In [10], workload information and server load metrics are recorded and used to infer the relationship between these variables and the system's performance based on statistical pattern recognition and machine learning techniques. And in [1], the effects of IBM S/390 workload management decisions are predicted by extrapolating from the historical usage of each machine's CPU, memory and I/O resources by different classes of workload, using a combination of performance modelling and statistical pattern recognision techniques. Other examples include our own work on statistical and queuing prediction methods [5, 6, 23, 24]. However, none of these examples can predict the effect on performance of adding or removing middleware functionality to manage the application's access to the database, on a new server architecture for which only a small number of benchmarks have been run (e.g. to determine request processing speed). The work described

1-4244-0910-1/07/$20.00

in [25] goes someway to addressing this gap in the literature, by describing a comparison of the performance of EJB versus JDBC implementations for the same benchmark used in this paper. However there is no evaluation of a performance model that can predict the effect of a change of application implementation. In summary there is a need for distributed enterprise performance prediction methods that model the effect of a change of application implementation from middleware-based to manual code for accessing the data in the database, and test the resulting method on an industry standard distributed enterprise application benchmark.

This paper describes just such a model and associated experimental analysis. In particular this paper looks at how predictions can be made for new server architectures for which only a small number of benchmarks have been run. In addition we parameterise the prediction models rapidly with low overheads. A layered queuing method is selected as the performance modeling technique as it explicitly models the tiers of servers found in this class of application and it has also been applied to a range of distributed systems (e.g. [9,19]). This method has additionally been used by both the High Performance Systems Group [5, 6] and IBM [13] to model distributed enterprise applications. However it *has not* been used previously to model the effect of a change of application implementation from middleware-based to manual code for accessing the database.

The IBM WebSphere middleware [8] is selected as the platform on which this experiment is run as it is a common choice for distributed enterprise applications. Since Websphere is based on the J2EE platform, the middleware functionality used for accessing the data in the database is *container-managed persistence* using Entity EJBs, and the manual version involves using JDBC and SQL. On top of WebSphere, the IBM Websphere Performance Benchmark Sample 'Trade' [11] is run as this is the main distributed enterprise application benchmark for the WebSphere platform.

The contributions of this paper are to: i.) present a performance model for predicting the impact on performance of container-managed persistence, on new server architectures; ii.) investigate the use of the model experimentally, given the requirement that it must be rapidly parameterised at a low overhead; and iii.) to show how the model can be used to make predictions with a good level of accuracy. The combination of a widely used performance modelling method (the layered queuing method), an established system model (see section 2), a popular middleware (IBM Websphere) and a distributed enterprise benchmark based on best practices (the IBM Websphere Performance Benchmark Sample 'Trade') should make this work of relevance to a wide range of distributed enterprise applications.

The remainder of this paper is structured as follows: section 2 depicts the system model, in section 3, commonly used middleware for accessing the data in the database is described along with the manual alternatives, including EJBs and JDBC. Section 4 describes the layered queuing performance model and in section 5, the experimental analysis of the layered queuing model is presented.

2. System Model

Distributed enterprise applications are often installed in dynamic hosting environments (as opposed to being installed on a dedicated set of resources). These are environments consisting of a heterogeneous collection of servers, hired by more than one service provider, with the servers typically being divided across a number of sites. Figure 1 shows the dynamic hosting environment system model used in this paper. Based on the Oceano hosting environment a server can only process the workload from one application at a time to isolate the applications (which may be being hosted for competing organizations); and servers can be dynamically transferred between service providers [4]. The service providers would typically use these servers to process a continuous stream of incoming workload (see e.g. [1, 14]). This workload is modelled as a set of heterogeneous service classes for each service provider, each of which is associated with a performance requirement specified in a Service Level Agreement (SLA). This paper focuses on response time performance requirements as this has been identified by IBM as one of the main performance metrics of interest to e-Business customers [1, 14].

Based on established work, each service provider is modelled as a tier of application servers accessing a single database server [2, 13]. Based on the queuing network in the WebSphere e-Business platform, a single first-in-first-out (FIFO) waiting queue is used by each application server; and all servers can process multiple requests concurrently via time-sharing. There are two types of decision making software which are used in the system: *workload management* [17] –determining the site and server on which incoming requests should run, and *server allocation* – based on the workload and available servers, determining the (dynamic) allocation of servers to service providers. The objective is a trade-off between the quality of service (provided to the server owners, the service providers who hire the servers, the clients who send the requests and the administrators) and the costs of providing this service. Examples of these costs include the cost of hiring servers, the cost of paying penalties for SLA failures, and the cost of re-allocating servers.

In this paper 'No. of clients and the mean client think-time' is used as the primary measure of the workload from a service class. The total number of clients

across all service classes and the percentage of the different service classes are used to represent the system load. Using number of clients (as opposed to a static arrival rate definition) to represent the amount of workload is common when modelling distributed enterprise applications – see for example [5, 10, 13]. This is because it explicitly models the fact that the time a request from a client arrives is not independent of the response times of previous requests, so as the load increases the rate at which clients send requests decreases. In this context 'client' refers to a request generator (i.e. a web browser window) that requires the result of the previous request to send the next request. Users that start several such conversations can be represented as multiple clients.

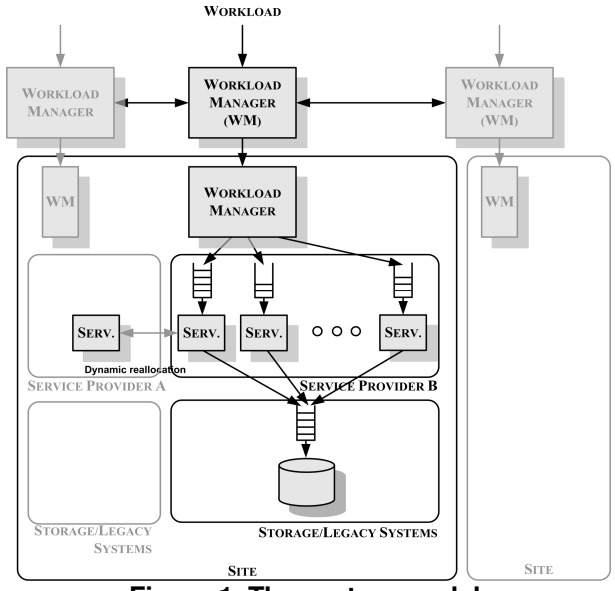

Figure 1. The system model.

Each request from a client calls one of the operations on the application-tier interface. The *typical workload* is defined as the next operation called by a client being randomly selected, with probabilities defined as part of the Trade benchmark as being representative of real clients. The think-time is exponentially distributed with a mean of 7 seconds for all service classes as recommended by IBM for Trade clients [2]

3. Managing Access to the Data in the Database

This section provides an overview of container-managed persistence, and the alternative manual approach to accessing the data in the database. Using container-managed persistence involves creating a set of data structures in the application, and defining the relationship between these data structures and the tables in the database, along with a set of properties that define how the middleware is to synchronise the application data with the database data. For example in the Java platform this often involves creating an entity EJB to represent each table in the database, and when using NJDX for the .NET platform a dedicated object-relational mapping tool is used. The middleware can then provide the services which are typically required for enterprise data access including: transactional access to maintain the integrity of database data; database connection pooling to improve performance; configurable security levels (e.g. per-database or per-user); and error handling facilities to assist system administrators isolate problems.

When container-managed persistence is not used the programmer must manually write the code to access the database, and the data inside it. This would typically involve establishing the connection and setting the connection properties using one language (e.g. JDBC for the Java platform or ODBC for the .NET platform) and then writing the code to access the data in another language (normally SQL, possibly using a vendor's proprietary extensions). In this case code to manage and commit or rollback transactions would have to be written manually, as would much of the code to manage other services including database connections, security and error handling. Typically this would involve a combination of the above languages and the base programming language (e.g. Java or C#). Potentially this could be a major undertaking, although in practice the use of pre-existing libraries can simplify the task. Indeed, the use of pre-existing libraries allows for various 'halfway-houses' between container-managed persistence and the fully manually approach, in which case some of the database access functionality is provided by the libraries and some is implemented manually by the programmer.

The advantages of container-managed persistence include dramatically simplifying the implementation of an enterprise application that requires the services discussed above (e.g. transactions and database connection pooling); and the creation of code which is reusable (e.g. when switching to a different database vendor) and often modular. However container-managed persistence can also impose a significant overhead on the performance of the application at runtime. The advantages of the manual approach include the flexibility that results from having full control over how the application accesses the database (e.g. over transactional boundaries); and that the application can be easier to debug, profile and performance tune. However care must be taken so as not to tie the implementation to the idiosyncrasies of the database software used to test the application, and if the code is poorly written it could in the worst case have even worse performance than if container-managed persistence were used. The reader is referred to [7] for a more

detailed discussion on the advantages and disadvantages of this technology.

4. Layered Queuing Model

A layered queuing performance model explicitly defines a system's queuing network. An approximate solution to the model can then be generated automatically, using the layered queuing network solver (LQNS) [9]. The solution strategy involves dividing the queues into layers corresponding to the tiers of servers in the system model and generating an initial solution. The solver then iterates backwards and forwards through the layers, solving the queues in each layer by mean value analysis, and propagating the result to other layers until the solutions converge. A detailed description of the modeling language and solution algorithm can be found in [9], and a new technique developed by the High Performance Systems Group to allow the solution algorithm to converge rapidly on a solution when there is more than one service class in the workload can be found in [6].

A layered queuing model is created to represent the queuing network in the Websphere middleware. The model has application server, database server and database server disk layers, with each layer containing a queue and a server. In addition the database server layer can include a second queue and server for applications that use a separate session database server, with the associated second server and queue in the database server disk layer. The application server disk is not modelled as the Trade application's utilization of this resource is found to be negligible during normal operation. Workload parameters (per service class) are: the number of clients, the mean processing times on each server, the mean client think-time, the mean number of application and session database requests per application server request and the mean number of database disk requests per database request for the application and session database servers. The other parameters are the maximum number of requests each server can process at the same time via time-sharing.

The following is an overview of how the layered queuing model is used, and the different performance metrics that are involved. First, the resource usage and throughput of each service class under *no_of_clients* number of clients is measured, as well as the maximum throughputs of each application server architecture under the different application implementations under the typical workload as defined in section 2 - or alternatively these values could be calculated using performance data provided in the literature. These performance metrics are then used to calculate the model parameters using equations 1-6. Finally, the model is solved using the solution strategy outlined at the beginning of this section, and the performance metrics generated include response times, throughput and utilization information for each service class. The remainder of this section describes in detail how the model is parameterised using equations 1-6.

The model must be parameterised using a simulated workload consisting of only one service class at a time. This overcomes the difficulties that have been found measuring mean processing times (without queuing delay) of multiple service classes, in real system environments [26]. The number of database requests that a service class makes per application server request (n_D) from the application server to a database D is calculated as:

$$n_D = \frac{throughput_D}{throughput_{appserv}} \quad (1)$$

Where $throughput_D$ is the throughput of the database D and $throughput_{appserv}$ is the throughput of the application server.

The mean processing time for a server on an established server architecture (denoted P_E where the E indicates the server is running on an established server architecture) is calculated for each service class as follows:

$$proc\_time_{P_E} = \frac{\%\_resource\_used_{P_E}}{throughput_{P_E}} \quad (2)$$

Where $\%\_resource\_used_{P_E}$ is, in this model, either the *% CPU usage* or *% disk usage* standard performance metric, depending on the resource the server is modelling. In practice it has been found that the throughput of the database server disk can be set at the same value as that for the database server whilst maintaining a good level of predictive accuracy (if the mean number of database disk requests per database request for that server is also set to 1). This is useful due to the difficulty in measuring the throughput of server disks at a low overhead in real system environments. It is also noted that the model solution strategy (see above) assumes that each call to a server (including 'disk' servers) is composed of multiple low-level requests for service in the underlying hardware processor.

Once the model has been parameterised on an established server architecture the $proc\_time_{P_E}$ parameters can be adjusted (for each server on an established server architecture P_E) so as to give their values for a new server architecture. This is done by calculating the ratio of the established/new server architectures' request processing speeds. For example, request processing speed in the measured performance data collected in section 5 is represented as the maximum throughput of the server architecture under the typical workload (as defined in section 2). However, other

metrics that are easier to measure at a low overhead could also be used. In this paper a maximum throughput metric is used and so the request processing speed ratio is calculated as follows, where $mx\_throughput_{A_E}$ is the maximum throughput of the established server architecture A_E on which the model was parameterised, and $mx\_throughput_{A_N}$ is the maximum throughput of the new server architecture A_N.

$$req\_proc\_speed\_ratio_{A_E,A_N} = \frac{mx\_throughput_{A_E}}{mx\_throughput_{A_N}} \quad (3)$$

The processing times for servers running on the new server architecture (that is just *appserv* in the experiments in section 5) are defined as follows where A_E is the established server architecture on which server P_E is running, and A_N is the new server architecture on which server P_N will be running.

$$proc\_time_{P_N} = proc\_time_{P_E} \times req\_proc\_speed\_ratio_{A_E,A_N} \quad (4)$$

Once the model parameters have been calculated on a server architecture, the values can be input into the model and the model solved to give response time, throughput and % resource usage predictions for any workload on that server architecture using the solution strategy outlined at the beginning of this section. Since the n_D values are constant across server architectures they do not need to be recalculated - the values calculated on the established server architecture can be input into the model for the new server architecture without modification.

The equations used to calculate the processing time parameters on a new application (that is, extrapolating the parameters for a *new* application implemented using container-managed persistence from the same *established* application implemented without container-managed persistence, or vice versa) are as follows:

$$req\_proc\_speed\_ratio_{I_E,I_N} = \frac{mx\_throughput_{I_E}}{mx\_throughput_{I_N}} \quad (5)$$

$$proc\_time_{P_n} = proc\_time_{P_E} \times req\_proc\_speed\_ratio_{I_E,I_N} \quad (6)$$

Where I_E and I_N are the established and new application respectively; $proc\_time_{P_E}$ is, as before the mean processing time of the established application I_E on the established server architecture A_E; and $proc\_time_{P_n}$ is the processing time for the new application I_N on the established server architecture A_E.

In the experimental setup in section 5 there is only one (application) database as the default configuration for the Websphere middleware is to store the session data in-memory as opposed to on a separate database server. (The use of this model to predict performance when session databases are used has been investigated in [24].) It is calculated that in the experimental setup, the typical workload makes 1.14 requests on average to the Trade database using equation 1 (that is n_{DB}=1.14, where *DB* is the database *D*). And the application server and database server (for the application database) can process 50 and 20 requests at the same time, respectively. These values were selected using the WebSphere performance tuning wizard. The database server disk can only process one request at the same time. The mean processing times, which vary depending on whether container-managed persistence is used in the application implementation, and on the server architecture being used, are calculated in the experimental results section (see section 5).

5. Experimental Analysis

Experiments are conducted to examine the predictive accuracy of the model from the previous section. The following software is used: WebSphere Advanced Edition V4.0 [8], DB2 Universal Database (UDB) Enterprise Server Edition (ESE) V7.2 [12] and the 'Trade 2' [11] performance benchmark sample all of which are installed according to the recommended configuration. The experimental setup consists of four tiers of machines for the workload generators, workload manager, application server and database server respectively. The application servers are a P4 1.8Ghz, 256MB JVM heap size and a P4 2.66Ghz, 256MB JVM heap size. The first application server is used to represent the new server architecture and the second the established server architecture. The database server is a P2 355Mhz, 384MB RAM. The IBM HTTP Server v1.3.19 is used as the workload manager to direct the workload either to the established or new server architecture, as required in the experiments. All servers run on Windows 2000 Advanced Server and a number of clients (each generating the typical workload from section 2) are simulated by each workload generator (P4 1.8 GHz, 512MB RAM) using the Apache JMeter tool [3]. It is noted that although, in this paper, only the typical workload (that is, the standard workload provided by IBM) is used, the use of the layered queuing model to make predictions for different, heterogeneous workloads has been demonstrated in [6]. All machines are connected via a 100MB/s fast Ethernet.

5.1. Experimental Results

This section describes the experimental analysis of the predictive accuracy of the model from section 4, using the experimental setup described above. This involves parameterising the model using a server with container-managed persistence turned on, and using that model to make performance predictions for if the application is re-implemented without container-managed persistence, and

if this new application is run on a new server architecture. For clarity, in this section the version of the Trade application implemented using container-managed persistence is referred to as the *established application*, and the version of the Trade application implemented without container-managed persistence is referred to as the *new application*. This investigation is conducted in three steps. First predictions are made for the established application on the established server architecture, then for the new application on the established server architecture, and then for the new application on the new application server architecture. Accuracy calculations are made using the following equation:

$$acc = \left[1 - \frac{predicted\_value - measured\_value}{measured\_value}\right] \times 100 \quad (7)$$

So the first step is making predictions for the established application on the established server architecture. The workload is parameterised on the established server architecture. A number of test runs are conducted to parameterise the model at different values of *no_of_clients*. Each test run involves activating *no_of_clients* number of clients (with *no_of_clients* ranging from 70 to 2520 in increments of 250 with 20 clients instrumented so as to measure the mean response time, in each case) and waiting 30 seconds for the system to reach a steady state. The %CPU/disk usage samples are then recorded for a period of 1 minute along with the mean throughput during the minute, for the application and database servers. The sampling interval is set at 6 seconds so the increase in the %CPU/disk usage (that is, the sampling overhead) is no more than 5%. Equation 2 is then used to calculate the $proc\_time_{P_E}$ parameters for the application server, database server and database server disk. It is found that these values are approximately constant except for very small workloads (i.e. when *no_of_clients* is 70 or 270). As a result the minimum value of *no_of_clients* once the values of the parameters have stabilised is used, so as to minimise the resource usage overhead (i.e. *no_of_clients* is fixed at 520 for the parameterisation). Specifically, the resource usage overhead is that the % CPU usage of the application and database servers is just 27% and 23% respectively, and the database server disk usage is just 1.9% when the model is parameterised at 520 clients. This is likely to be a low resource usage overhead for a modern hosting environment. The resulting $proc\_time_{P_E}$ parameters are shown in table 1. The predictive accuracy of the model for the established application on the established server architecture is then calculated for values of *no_of_clients* from 520 to 2520 in increments of 250. Predictive accuracy before the mean processing time has stabilised (i.e. before 520 clients) is not considered here as performance predictions are less critical when the majority of the server's resources are available, due to the workload being extremely small. The resulting overall mean predictive accuracy is very good at 97.7% and 79.6% for throughput and mean response time respectively. The full details of this calculation are omitted due to space restrictions and because measured performance data is available for IBM Websphere with and without container-managed persistence [25]. The reader is referred instead to the similar calculations for the new application on the established and new server architectures, which follow next.

The second step is to calculate the predictive accuracy of the new application on the established server architecture. The $req\_proc\_speed\_ratio_{I_E,I_N}$ value used is calculated using data provided by the literature and is found to have a value of 1.5 (as calculated using the ratio of the throughput data for container-managed persistence and manual application implementations running on the Websphere middleware in [25]). Using this value the layered queuing model is updated as described in section 4 (using equations 5 and 6) and $proc\_time_{appserv_n}$ is set to 2.43. The predicted mean response times evaluated using this model are shown in table 2, along with the measured mean response times, and the predictive accuracy. Overall a good level of overall mean predictive accuracy is obtained (approximately the same as before at 96.3% for throughput and just a 7.6% reduction for mean response time from the value calculated in step 1).

$proc\_time_{appserv_E}$	$proc\_time_{DB_E}$	$proc\_time_{DBdisk_E}$
3.65	3.75	0.30

Table 1. Model processing time parameters (ms)

Number of Clients	Measured (ms)	Predicted (ms)	Accuracy (%)
520	7.3	9.4	71.23
770	8.9	11.2	74.16
1020	11.0	13.7	75.45
1270	25.5	17.8	69.80
1520	44.3	25.9	58.47
1770	204.1	119.9	58.75
2020	1048.3	572.8	54.64
2270	2131.0	2226.0	95.54
2520	3146.0	2823.0	89.73

Table 2. Measured/predicted mean response time - the new application on the established server architecture.

Number of Clients	Measured (ms)	Predicted (ms)	Accuracy (%)
520	8.8	9.7	89.77
770	8.3	11.6	60.24
1020	27.4	14.2	51.82
1270	37.0	18.4	49.73
1520	117.3	26.8	22.85
1770	514.4	121.0	23.52
2020	1544.0	1306.0	84.59
2270	2692.0	2460.0	91.38
2520	3886.0	3468.0	89.24

Table 3. Measured/predicted mean response times - the new application on the new server architecture.

Finally, the third step is to calculate the predictive accuracy of the new application on the new server architecture. The $req\_proc\_speed\_ratio_{A_E,A_N}$ used is calculated using the max throughputs of the two server architectures for convenience, in this case 255.1 and 235.2 requests/second for the established and new server architectures respectively, resulting in a value of 1.08 when equation 3 is applied. However as discussed in section 4 other request processing speed metrics (for example, metrics which are easier to measure at a low overhead) could be used instead of this particular metric. It is also noted that although a value of 1.08 would seem to imply that the two server architectures are very similar this is in fact misleading, as the response times provided by the established server architecture are on average 1.72 times faster than those provided by the new server architecture. Using the value of 1.08 for $req\_proc\_speed\_ratio_{A_E,A_N}$ the layered queuing model is updated using equation 4 and $proc\_time_{appserv_N}$ is set to 2.63. The predicted mean response times given by this updated model are shown in table 3, as are the measured mean response times and predictive accuracy values. The overall mean predictive accuracy for mean response time and throughput are both good; the value for mean response time being just 9.4% less than for step 2, and the value for throughput being approximately the same at 98.1%. The details of the throughput predictive accuracy calculations are omitted due to lack of space, because the predictive accuracy is higher than that for mean response time and because the shape of the throughput graphs are very similar to the equivalent graphs in [6].

It can be seen that in both tables 2 and 3 the predictive accuracy decreases as the number of clients at which maximum throughput is reached is approached (that is, 2020 clients for table 2 and 1770 clients for table 3), but after that the predictive accuracy reverts to a high level. It is a known problem with the layered queuing method that this type of model underestimates the response time (but not throughput) as the maximum throughput of a system is approached, due to the approximations used to make rapid predictions in the solver [6]. The temporary reduction in predictive accuracy in tables 1 and 2 is consistent with this, and in practice would have been compensated for by increasing the response times predicted by the solution strategy for these samples. However this has not been done here so the predictive accuracy figures give the accuracy of the solution strategy alone.

Overall, it has been shown in this section that the model from section 4 models the effect of container-managed persistence on an application's performance with a good level of accuracy (in terms of the throughput and mean response time metrics). It has been shown that this model is particularly useful as apart from the model of the established application, all that is required to make the predictions for the effect on performance of adding or removing container-managed persistence is an estimate of the overhead of the container-managed persistence (which in this set of experiments has been taken from the literature [25]). It has also been shown that these predictions can be made for a new application server architecture for which only the request processing speed is known. The model also has the advantage that it can be evaluated rapidly to give a prediction (taking 5 seconds or less on an Athlon 1.4Ghz under a convergence criterion of 20ms). And finally, it has been shown that the models can be rapidly parameterised at a low model overhead (as detailed in step 1 above) whilst still providing enough data to make these accurate predictions. Collectively, this provides strong experimental evidence for the effectiveness of the layered queuing model proposed by this paper.

6. Conclusion

This paper examines the effect of using middleware functionality to manage an application's access to the data in the database, on the performance of distributed enterprise applications. A model of the effect of this, on metrics including mean response time and throughput under different workloads and server architectures is presented, using the layered queuing method. Experiments are then conducted to examine this on a popular enterprise computing architecture based on IBM Websphere, using the default EJB-based container-managed persistence as the middleware functionality. The results provide strong experimental evidence for the effectiveness of the model in terms of the accuracy of predictions, the speed with which predictions can be made and the low overhead at which the model can be rapidly parameterised.

In addition to the work described in this paper, we are also investigating hosting environments as part of an

EPSRC e-Science project *Dynamic Operating Policies for Commercial Hosting Environments* (which part sponsors this work under contract no. EP/C538277/1). The reader is referred to other project publications for more information about dynamic hosting environments including performance prediction methods to enhance workload management and dynamic reallocation [5,6,23,24], optimal and heuristic policies for dynamic server reallocation [18,20] and an open software architecture for dynamic operating policies [21]. The work described in this paper will feed into the project by allowing dynamic hosting environment administrators to decide whether it is worth re-implementing an application being hosted for a service provider, so as to add or remove container-managed persistence, based on the predicted effect on application performance.

References

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

[2] Y. An, T. Kin, T. Lau, P. Shum, *A Scalability Study for WebSphere Application Server and DB2 Universal Database*, IBM White paper, 2002. http://www.ibm.com/developerworks/

[3] *Apache JMeter v1.8 User Manual,* available at: http://jakarta.apache.org/jmeter/index.html, 2002

[4] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, B. Rochwerger, *Oceano-SLA Based Management of a Computing Utility*, 7th IFIP/IEEE International Symposium on Integrated Network Management, New York, May 2001

[5] 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, 34:93-111, 2005

[6] D.A.Bacigalupo "*Performance prediction-enhanced resource management of distributed enterprise systems*", PhD Thesis, Department of Computer Science, University of Warwick, 2006

[7] L. DeMichiel, M. Keith, *JSR 220: Enterprise Java Beans version 3.0 Specification*, 2nd May 2006, available from http://java.sun.com/

[8] M. Endrel, *IBM WebSphere V4.0 Advanced Edition Handbook*, IBM International Technical Support Organisation Pub., 2002. Available at: http://www.redbooks.ibm.com/

[9] R.G. Franks, *Performance Analysis of Distributed Server Systems*, Phd thesis, Ottawa-Carleton Institute for Electrical and Computer Engineering, Faculty of Engineering, Department of Systems and Engineering, Carleton University, 20th December 1999.

[10] M. Goldszmidt, D. Palma, B. Sabata, *On the Quantification of e-Business Capacity*, ACM Conference on Electronic Commerce (EC'01), Florida, USA, October 2001

[11] IBM Corporation, *IBM Websphere Performance Benchmark Sample: Trade*. Available at http://www.ibm.com/software/info/websphere/

[12] *IBM DB2 UDB Enterprise Edition v8.0*. http://publib.boulder.ibm.com/infocenter/db2luw/v8/

[13] T. Liu, S. Kumaran, J. Chung, *Performance Modeling of EJBs*, 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI'03), Florida USA, 2003

[14] Z. Liu, M.S. Squillante, J. Wolf, *On Maximizing Service-Level-Agreement Profits*, ACM Conference on Electronic Commerce (EC'01), Florida, USA, October 2001

[15] D. Menasce, *Two-Level Iterative Queuing Modeling of Software Contention*, 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'02), Texas, USA, October 2002

[16] D. Menasce, D. Barbara, R. Dodge, *Preserving QoS of E-commerce Sites through Self-Tuning: A Performance Model Approach*, ACM Conference on Electronic Commerce (EC'01), Florida, USA, October 2001

[17] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum. *Locality-aware request distribution in cluster-based network servers* ACM SIGPLAN Notices, 1998

[18] J. Palmer and I. Mitrani, *Optimal and heuristic policies for dynamic server allocation*, Journal of Parallel and Distributed Computing, 65:10(1204-1211), Elsevier, 2005

[19] F. Sheikh, M. Woodside, *Layered Analytic Performance Modelling of a Distributed Database System*, International Conference on Distributed Computing Systems (ICDCS'97), Maryland USA, May 1997

[20] J.Slegers, I. Mitrani and N. Thomas, *Server Allocation in Grid Systems with On/Off Sources*, Workshop on Middleware and Performance (WOMP'06), Sorrento, Italy, December 2006.

[21] J. Slegers, C. Smith, I. Mitrani, A. van Moorsel and N. Thomas, *Dynamic Operating Policies for Commercial Hosting Environments*, 5th UK e-Science All Hands Meeting, Nottingham, UK, Sept. 2006

[22] Software Tree Inc, *NJDX homepage*, available at: http://www.softwaretree.com/NJDX_index.htm

[23] J.D. Turner, D.A. Bacigalupo, S.A. Jarvis, D.N. Dillenberger, G.R. Nudd, *Application Response Measurement of Distributed Web Services*, Int. Journal of Computer Resource Measurement, 108:45-55, 2002

[24] J.W.J. Xue, D.A. Bacigalupo, S.A. Jarvis, G.R. Nudd, *Performance Prediction of Distributed Enterprise Applications with Session Persistence*, 22nd Annual UK Performance Engineering Workshop (UKPEW'06), Bournemouth University, Poole, UK, 6-7 July 2006

[25] Y. Zhang, A. Liu, W. Qu, *Comparing Industry Benchmarks for J2EE Application Server: IBM's Trade2 vs Sun's ECperf,* 26th ACM Australasian Conference on Computer Science: Research and Practice in Information Technology, Adelaide, Australia, 2003

[26] L. Zhang, C. Xia, M. Squillante, W. Nathaniel Mills III, *Workload Service Requirements Analysis: A Queueing Network Optimization Approach*, 10th IEEE International Symposium on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems (MASCOTS '02), Texas, USA, Oct. 2002