Performance Studies of a WebSphere Application, Trade, in Scale-out and Scale-up Environments

Hao Yu José E. Moreira Parijat Dube I-hsin Chung Li Zhang

IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598-0218
{*yuh,jmoreira,pdube,ihchung,zhangli*}@*us.ibm.com*

Abstract

Scale-out approach, in contrast to scale-up approach (exploring increasing performance by utilizing more powerful shared-memory servers), refers to deployment of applications on a large number of small, inexpensive, but tightly packaged and tightly interconnected servers. Recently, there has been an increasing interest in scale-out approach. The purpose of this study is to discover advantages or disadvantages of scale-out systems with a typical enterprise workload, IBM Trade Performance Benchmark Sample for WebSphere Application Server (a.k.a. Trade6). In this work, through cross system performance comparison, we show that for such workload, scale-out approach has better performance/cost effect. In term of scalability, we show that WebSphere Application Server packages for distributed environment scale well while the possible bottleneck of the application deployment is the database tier. We present preliminary results to show that both database partitioning feature (DPF) and federated database server approaches are not exactly suitable for providing scale-out solution for the database tier of workloads similar to Trade (small tables and short transactions). In addition, we discuss our on-going effort on further performance study: (1) studies of performance/scalability for larger deployments by adopting the IBM AMBIENCE queuing network modeling tool, (2) performance breakdowns utilizing IBM ACTC hardware counter library.

1 Introduction

The computing approaches used by today's commercial applications can be classified into two large groups. *Scale-up* refers to running applications on large shared-memory servers, whose costs usually increase faster than corresponding performance improvements. *Scale-out* refers to deploying applications on a massive number of small, inexpensive, but tightly packaged and/or tightly interconnected servers.

In the past, there has be a large number of studies on running commercial workloads on large shared-memory server systems [21, 12, 10]. Recently, many of the large-scale web-based enterprises (e.g., Google, Yahoo, eBay, Amazon) have successfully scaled their business by exploring computing systems integrated from large numbers of small, inexpensive computers [17, 16]. In addition, due to cost-performance consideration, traditional commercial companies have shown trend to shift their computing infrastructures from big servers to small-server based systems [19]. Despite the tremendous increase of interest on scale-out approach, we found relevant less in-depth performance study of realistic workloads on such platforms.

The purpose of our study is to gain better understanding of performance impacts of running a typical commercial workload in a scale-out environment. The workload is IBM Trade Performance Benchmark Sample for WebSphere Application Server (a.k.a. Trade), which exercises all three tiers (i.e. client, middle and data storage tiers) of today's enterprise application architecture.

We have deployed the application on both scale-up and scale-out environments to establish comparison. For scale-up system, we used a 16-CPU POWER5 575 system. For scale-out environment, we used a cluster based on IBM BladeCenter solution [4]. In this paper, we report performance results in term of web service throughput and system usages of above systems. Through cross system performance comparison, we show that for such representative commercial workload, scale-out approach has much better performance/cost effect. In term of scalability, we show that WebSphere Application Server packages for distributed environment scale well while the possible bottleneck of the application deployment is the data storage tier. Based on obtained results, we are exploring possibility of scaling out the database tier of the application. In this paper, we present preliminary results on adopting technologies such as DB2

1-4244-0910-1/07/$20.00 ©2007 IEEE.

data partitioning feature and database federation feature. The early results suggest that these techniques are not exactly suitable for scaling the database tier that running on light-weight servers, which makes running the database tier on more powerful scale-up systems a more natural solution.

In the time of writing, we are actively expanding our study. In this paper, we discuss our initial efforts toward modeling the scalability of larger deployments of Trade using an IBM AMBIENCE modeling tool set. In addition, we briefly discuss utilizing Hardware Performance Monitor (HPM) tool of IBM High Performance Computing Toolkit (HPCT) for in-depth performance analysis.

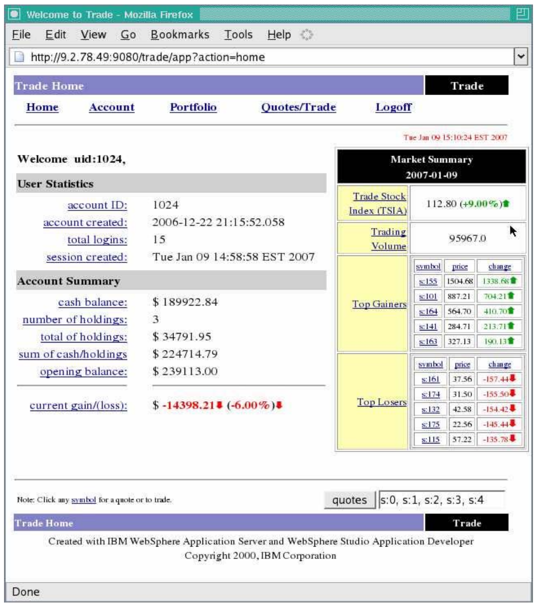
The rest of the paper is organized as following: Section 2 talks about the Trade application and how we have deployed it. Section 3 presents the performance results of running Trade on scale-up and scale-out environments, which show the advantage of scale-out approach in term of performance-cost factor. Section 4 shows preliminary results on the course of exploring natural ways of scale-out the database tier of Trade. Section 5 discusses our efforts on performance modeling and hardware-counter related performance study of Trade. Section 6 concludes the paper and discusses future work.

2 Experiment Setup

Trade [7, 8, 5] is the short name of Trade Performance Benchmark Sample for WebSphere Application Server (WAS). WAS is a family of IBM software products, which provides development and deployment environment for modern multi-tier web applications [7]. The latest WAS products align to J2EE 1.4.

The Trade benchmark is designed and developed to cover the significantly expanding programming model and performance technologies associated with WAS. Trade is a stock trading application and allows a registered user to buy and sell stock, to check his portfolio, and so on. It provides a real-world workload that enabling performance research and verification test of the J2EE 1.4 implementation in WAS. Since Trade version 6 (a.k.a. Trade6), all Trade versions are WebSphere-version dependent. Here we used Trade v6.1 together with WAS v6.1 packages. Fig. 1 shows a sample screen-shot illustrates a user's home page after login. The page has account information, current market summary information, recommendations for buy or sell, etc.

In this study, we have used 3 of various WAS packages. (1) *WebSphere Application Server (WAS)* is the base product package of WebSphere family. It is configured and managed on a single machine. (2) *WebSphere Application Server Network Deployment (WAS ND)* adds the ability to manage multiple application servers and handle clustered environments. WAS ND provides the same programming model support as the base WebSphere Application Server. In addi-


Welcome to Trade - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Home
Account
Portfolio
Quotes/Trade
Logoff
quotes
Done

http://9.2.78.49:9080/trade/app?action=home

Trade Home | Trade

Tue Jan 09 15:10:24 EST 2007

Welcome uid:1024,

User Statistics

account ID:	1024
account created:	2006-12-22 21:15:52.058
total logins:	15
session created:	Tue Jan 09 14:58:58 EST 2007

Account Summary

cash balance:	$ 189922.84
number of holdings:	3
total of holdings:	$ 34791.95
sum of cash/holdings	$ 224714.79
opening balance:	$ 239113.00
current gain/(loss):	$ -14398.21 (-6.00%)

Market Summary
2007-01-09

Trade Stock Index (TSIA)	112.80 (+9.00%)
Trading Volume	95967.0

Top Gainers

symbol	price	change
s:155	1504.68	1338.68
s:101	887.21	704.21
s:164	564.70	410.70
s:141	284.71	213.71
s:163	327.13	190.13

Top Losers

symbol	price	change
s:161	37.56	-157.44
s:174	31.50	-155.50
s:132	42.58	-154.42
s:175	22.56	-145.44
s:115	57.22	-135.78

Note: Click any symbol for a quote or to trade.

s:0, s:1, s:2, s:3, s:4

Trade Home | Trade

Created with IBM WebSphere Application Server and WebSphere Studio Application Developer
Copyright 2000, IBM Corporation

Figure 1. Client Homepage of Trade

tion, it adds support for a variety of topologies and architectures consisting of multiple machines and multiple application servers managed under a single umbrella. Specifically, WAS ND provides deployment optimizations mainly in the form of clustering, workload management, and administration. These features allow larger scale deployments. (3) *WebSphere Application Server Extended Deployment (WAS XD)* is an edition of WebSphere that extends the WAS ND, with additional features for scalability and manageability. We used the latest WAS XD, version 6.1 for most of the experiments on the scale-out environment and the base WAS for the experiments on the scale-up environment.

The scale-up system in our study is a POWER5 575 system with 16 physical CPUs (32 SMT threads). Due to limited resource, we do not have an additional node for database server. We run a single WAS application server and the database server in one LPAR (logical partition) [3] on the system. Fig. 2 shows the deployment of Trade on the scale-up system. The driver node in the figure is a separate Linux node running IBM WebShere Studio Workload Simulator [6], a light-weight workload simulator for web applications, to mimic web requests.

The scale-out environment is a cluster of Linux PC based on the IBM BladeCenter family of products. The building block of the cluster is a BladeCenter-H (BC-H) chassis. The chassis are connected through a 1-Gbit/s Ethernet network. Two different kinds of blades are used in our study: JS21 and HS21, among which, JS21 blades are the focus of our study. JS21 blades are quad-processor (dual-socket, dual-core) PowerPC 970 blades, running at 2.5 GHz. Each blade has 8 GiB of memory. The HS21 blades are quad-processor

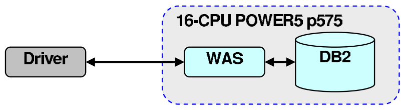


Figure 2. Trade on the p575 system

Intel Woodcrest blades, running at 3.0 GHz. Each blade has 16 GiB of memory.

The architecture of the Trade (version 6 and 6.1) deployment on scale-out environment is representative of most WebSphere applications. Fig. 3 shows one possible deployment topology. In this topology, the driver, which mimics the activities of a collection of traders, initiates transactions through an HTTP request. In the end, a web page, as a reply to the web request, is sent to the driver. Some requests are purely lookup on accounts, portfolios, and stock prices. Other requests actually perform buy or sell operations on stocks. In Fig. 3 the HTTP requests from the driver are sent to a Load Balancer (LB), which then forwards the requests to one or more HTTP servers. The purpose of the load balancer, as the name indicates, is to spread the load between the (potentially) multiple HTTP servers. If there is only one HTTP server, then the load balancer is superfluous. Once an HTTP server receives a request, it passes it to one of application servers for processing. Each application server executes the business logic of the transaction, i.e. it performs the actions implied by the transaction and generates the return response to the driver. Typically, a transaction requires operate on a database. Sometimes the application server does a single query on the database for a transaction (e.g., check the balance in an account). In other cases, a single transaction can generate multiple database operations (e.g., buying a stock requires changing entries in multiple accounts, creating buying and selling records, updating portfolios). Although multiple transactions can be in operation across multiple application server instances at the same time, the operations on the database have consistency requirements. In general, to obtain optimal performance for trade, proper tuning of both DB2 server and WAS is needed. In particular, we set the run-time parameter of Trade to utilize the *Dynamic Caching* feature supported by WAS, which usually speedup the application by a factor of 3.

In term of workload, to stress the application-side systems to their peak capacities, we choose the workload for driving the application with following characteristics. First, for a given session, we specify a fixed number of concurrently active clients. Secondly, we inject 0% thinking delay between contiguous requests of a given active clients.

The data presented in this paper contains two sets of performance information. In term of application performance, we collect average response time and average page-element throughput (also referred to as page throughput in this writing), which are provided by the workload simulator running on the driver node. In term of system resource usage, we have collected usage information of CPU, memory, disk, and network on each participating computer. We used *nmon* [14] for collecting the usage information. nmon is a free performance monitoring tool for AIX and Linux systems, which provides a large amount of information on one screen and can save data to a text file for post-process.

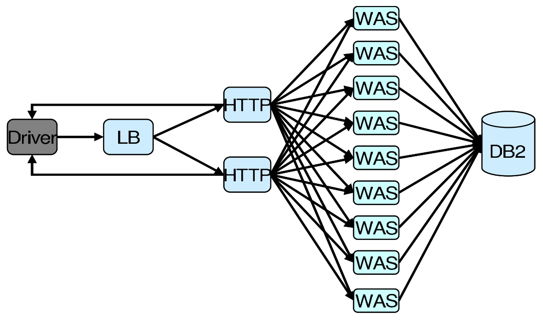


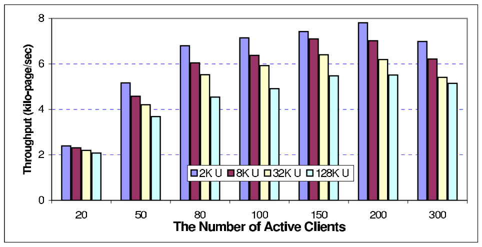
Figure 3. Trade on distributed environment

3 Performance Results

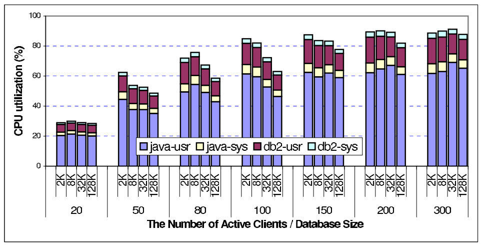
3.1 POWER5 Results

A summary of results for Trade v6 on the p5 575 is shown in Fig. 4 (a). We observe that the average throughput (measured in pages/second) depends on both the number of simultaneous clients performing transactions and the database size (measured in number of users, from 2,000 to 128,000). The throughput first increases with the number of clients, as more clients can better tolerate the latency of transactions, until 200 clients. After that, the system saturates and performance actually starts to decrease. For a given number of active clients, the throughput decreases with an increase in database size. Transaction throughput achieves almost 8000 pages/second for a small database size of 2,000 users and a little over 7000 pages/second for the commonly reported size of 8,000 users.

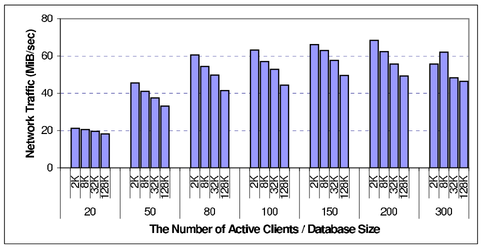
Fig. 4 (b) shows the CPU utilization of the p5 575 system for various experiment configurations. We show the breakdown of CPU utilization between the WAS (java) process and DB2 server processes and within each type of server the breakdown between user and system time. We observe that for the high-utilization cases (200 or more clients), combined user-mode CPU utilization for the WAS and DB2 servers is approximately 80%. Furthermore, the WAS utilizes about three times more CPU than the DB2 server. Here, we aggregated the CPU usages of all *db2sysc* processes to get the CPU usage of the DB2 instance. This way, we might missed some less active DB2 processes. Thus,



(a) Page throughput



(b) POWER5 CPU usage



(c) Driver network traffic

Figure 4. POWER5 results

the actual CPU usage of the DB2 server instance should be higher than reported.

We report the network utilization in Fig. 4 (c), which presents the measured network traffic between the p5 575 and the driver. Not surprisingly, the profile of network utilization as a function of number of clients and database size is very similar to the profiles for throughput and CPU utilization. Network traffic reaches between 60 and 70 MB/s for the high throughput cases, which is about half of the peak performance of a Gigabit Ethernet link (125 MB/s). Here, the packet sizes associated to the page sent from the HTTP server to the driver, as reported by nmon, is in the average of 800 bytes per packet. With an experiment using Netperf sending messages (800 bytes/message) from the p5 system to the driver node with TCP_NODELAY option, the observed network bandwidth is about 75 MB/s. The high network traffic suggests that additional scaling of trade on similar scale-up systems would require a faster network, either through channel bonding or through a 10 Gigabit/second Ethernet.

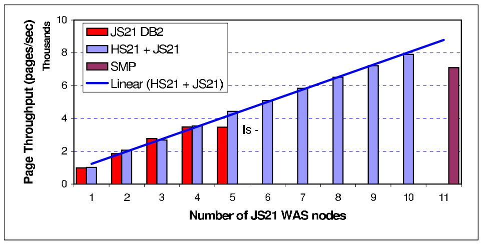


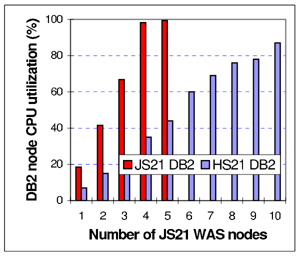
Figure 5. JS21 scaling results

3.2 Scaling Results with WAS on JS21

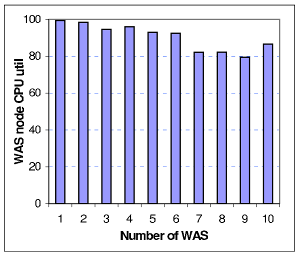
Our scalability experiment with Trade uses JS21 blades for WAS nodes (and HTTP node). We experimented with two variants for the DB2 server: in one case we used a JS21 blade for that role, and in the other case we used an HS21 blade. Results for throughput (pages/second) as a function of the number of WAS nodes is summarized in Fig. 5.

We observe that the configuration with a JS21 blade as the DB2 server reaches saturation at 4 WAS nodes, while the configuration with an HS21 blade as the DB2 server keeps scaling essentially linearly up to 10 WAS nodes. In fact, with 9 WAS nodes, the scale-out configuration matches the performance of the POWER5 575. The cause for the behavior in Fig. 5 can be observed in Fig. 6(a). The CPU utilization of the DB2 server increases with the number of WAS nodes. The JS21 CPUs (PowerPC 970) are not as fast as the CPUs of the HS21 blade (Intel Woodcrest) and therefore they saturate with less work. 4 WAS nodes are enough to bring CPU utilization of the JS21 DB2 server close to 100%. In comparison, CPU utilization of the HS21 DB2 server is under 90% even with 10 WAS nodes. Fig. 6(b) shows that the CPU usages of the WAS nodes are fairly close to capacity. The result also shows that when the number of WAS nodes increases, the average CPU usage of all WAS nodes slightly dropped. At the current stage, we do not have a solution to stablize the CPU usages to the full capacity. The reason is likely to be a mismatch of the HTTP server's policy for distributing the in-come web requests and the dynamic caching optimization of the WAS processes. We plan to look into the two components further for the regard.

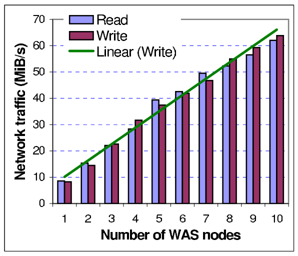
Fig. 6(c) shows that the network traffic of the HTTP server increases linearly with the number of WAS nodes (the figure is for an HS21 DB2 server). That is not surprising since more servers imply more traffic. Similar to the experiment on the SMP system, because the packet sizes of the traffic from the HTTP server to the driver are relatively small (800 bytes per packet), the actual peak bandwidth realizable is only 88 MiB/s (based on experiments using Netperf). Therefore, the network usage is still far from saturation and thus not the bottleneck.



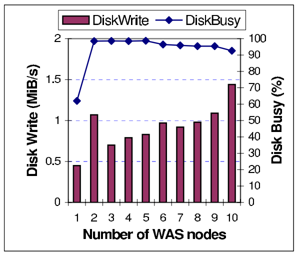
(a) DB2 node CPU usage



(b) WAS nodes CPU usages



(c) HTTP server network traffic



(d) DB2 node disk usage

Figure 6. System usages

Fig. 6(d) is to show the disk access pattern of Trade's database server. The chart shows significant seeking activities (disk controller being busy all the time), and not much data written to the disk. However, we found that the significant disk seeking does not hurt the response of the database server. This effect is because the large amount of write operations are for database logging and they are asynchronousg. In the case of Trade, the database size of Trade is relative small and so as the logging files. We allocated large enough buffer pool to hold the database and observed above 99% buffer pool hit rate.

3.3 Discussion

The above results show that using JS21 blades as WAS nodes and a relatively powerful HS21 node as database server, with 9 JS21 blades, the scale-out deployment of Trade out-performs the scale-up deployment. Here the listed price for JS21 and HS21 blades is about $9,000 each. The price for a fully configured BladeCenter chassis is $51,422. In our scale-out deployment, we have used 1 HTTP server, 9 WAS node, and 1 database node. Including the share for the price for the chassis, the final price for the 11 blades is roughly $139,403, which is lower than the list price for a p5 575 system with 16 processors, $166,988. The results of scale-out deployment show that the middle-tier (WebSphere Application Servers) can deliver quite scalable performance and the bottleneck of the scaling is the database tier that contains single database node.

4 On Scaling the Database Tier

In this section, we describe two preliminary experiments in the effort of scaling out the database tier.

4.1 Partitioned Database Experiment

IBM DB2 Universal Database (UDB) has a Database Partitioning Feature (DPF) [11] since its version 8.1. With DPF, a database can be scalable to available computing resources (CPUs, memory, disks). However, the scalability depends on applications. Because the DPF is distributed together with DB2 version 8.2 and 9.1 packages, it was natural for us to try the feature to scale the database tier of our Trade scale-out deployment. DB2 DPF automatically partitions the database tables onto multiple DB2 servers based on built-in hash functions (no support for range-based partition of data tables across multiple physical nodes). With minor changes of the default database schema of Trade, we distribute relative large tables across the 2 DB2 servers.

Table 1 summarizes a preliminary performance comparison of trade using single-node DB2 server, and dual-node DB2 servers utilizing the DB2 PDF feature. This experiment was done before the release of DB2 version 9.1 and we used DB2 UDB v8.2. In addition, this experiment was run on WAS ND v6.0 (based on IBM Java2-1.4.2), which has lower performance compared to that using version 6.1 (based on IBM Java5-2.0). We used one WAS node for both experiments and we drove the node to its full capacity (100% CPU utilization).

The table shows that the system utilization of the WAS node and DB2 server nodes for both configurations. The column *NET traffic* is for in-coming and out-going network traffic of a node. With both experiments returning the best performance for the application when running on WAS version 6.0: obtaining 750 pages/second page throughput, the results show that the system utilization (CPU usage, network traffic) for the dual-node DB2 servers are much higher. The high utilization of the DB2 servers shows that DB2 PDF has introduced non-trivial overhead. In particular, the *DB server 1*, as the instance owning server that talks to the WAS nodes directly, has 3 times CPU usage of that single server configuration. This indicates that the server will be saturated faster than the database server of the single server configuration will. Therefore, for Trade, it is hard for DB2 DPF to provide scalable support. Nevertheless, this is not a surprise. As indicated by previously published technical papers [20, 9], DB2 DPF is beneficial for databases containing very large tables, and individual transactions taking relative long time. Trade does not fall in the category.

Table 1. Partitioned DB2 server result

setups	Nodes	CPU usage user / sys (%)	NET traffic read / write (MiB/s)
2-nodes DB2 servers	WAS	89 / 7	1.5 / 7.4
	DB server 1	29 / 6	3.6 / 3.3
	DB server 2	14 / 4	2.5 / 1.7
1-node DB2 server	WAS	90 / 7	1.4 / 7.3
	DB server	10 / 2	0.9 / 0.8

Table 2. Federated DB2 server result

Entries	With federation	W/o federation
Avg page throughput	448.3	1117.4
WAS node CPU usage	99.6%	99.8%
App server CPU usage	48%	99%
Federation server CPU	51%	N/A
DB2 node CPU usage	12%	19%

4.2 Federated DB server Experiment

Database federation is intended for data integration to provide uniform access to heterogeneous data sources [15, 2]. In term of implementation, DB2 federation provides such data integration by adding a layer to process the SQL operations that touches multiple tables hosted on different physical servers. In addition, it provides the routing function to send the actual processing of each table to the appropriate database. Database federation can be used to spread a centralized database server hosting multiple tables to multiple backend data sources. Then the data can be served through one or more federated servers, each of which provides a logical view of the original centralized database. In some sense, database federation is a scale-out approach for database management.

We have carried out preliminary study of overhead for scaling Trade database tier utilizing DB2 federation feature. In this study, we used WAS ND v6.1. The deployment topology has 1 WAS node and 1 DB2 node. For the configuration utilizing DB2 federation feature, the role of the DB2 node is a physical data source. On the WAS node, we ran 1 application server instance and a DB2 federated server (providing logical view of the actual database hosted on the DB2 node). In this configuration, the WAS process running on the WAS node sends JDBC requests to federated database server running on the same node. The federated database server serves the requests by communicating with the physical data source running on the DB2 node.

Table 2 gives a performance comparison between the configuration using federated database server and the default configuration, where the WAS node only runs the WAS process that communicates directly to the remote DB2 node. The result suggests that the federation approach not only introduced much overhead to the WAS node and decreased the application performance, but also introduced extra CPU usage on the DB2 node hosting the actual data source. The overhead we witnessed mainly because of two reasons. First, we configured the federated server with DB2 *two-phase commit* feature which introduces non-trivial overhead to ensure transaction integrity for multi-site update database accesses [13]. The second reason is again because Trade has small tables and short transactions. Therefore, the significant overhead we witnessed here may not necessarily prohibit the scaling out of the database tier and further investigation is needed.

5 On Going Studies

5.1 Performance Modeling

Previously, we have show that Trade scales well with the number of WebSphere Application Server nodes, provided having a powerful database server. While our study approach presented previously is empirical, we would like to match the study with a more analytical approach. This should enable us to study the performance and scalability of Trade with both approach and better justify the study via mutual validation, correction, etc. For above purpose, we adopt a queuing network performance modeling toolset developed for IBM AMBIENCE (Automatic Model Building using InferENCE) project [18]. Since AMBIENCE toolset includes comprehensive performance modeling, prediction, and optimization functionalities targeting today's IT infrastructure, it is natural to use it to complement our performance study of Trade. As the first step, we want to use the toolset to identify possible system bottlenecks and to gain better knowledge of the scalability of the scale-out deployment.

From a queuing network modeling perspective, we count following factors for the model abstractions. Naturally, we abstract the Trade scale-out deployment topology as a three-tier web service: HTTP server as the front tier, WAS nodes as the middle tier, and the DB2 node as the third tier. Since a single Trade experiment session has a fixed population of clients, we model the whole system with closed queuing model. Specifically, each client submits a request to the system and waits for the completion of the request to send a new request. In effect, there are always a fixed number of requests in the system. In this abstracted system, different components handle the requests differently. While the front-end and the application Servers process requests from different clients in parallel, the back-end processes the requests in a sequential manner. To count for this, we model the HTTP servers and WAS servers as a *Processor Sharing (PS) system.* In a PS system, all requests being processed by the server get equal share of the processing capacity of the server. The back-end is modeled as a First Come First Serve

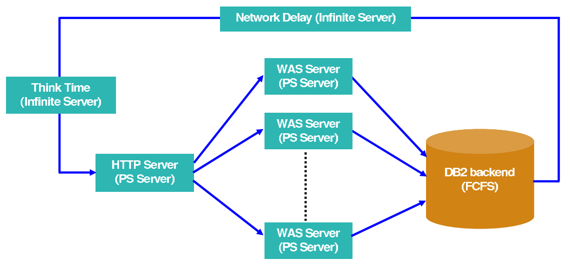


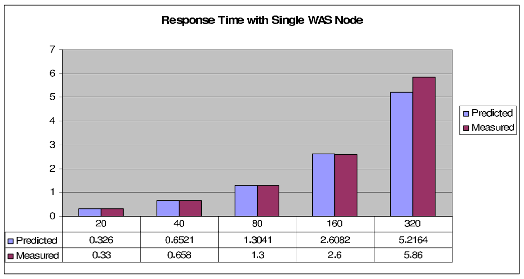
Figure 7. Closed queuing model for Trade

(FCFS) system. Finally, all WAS nodes are homogeneous and we assume that each WAS node handles same fraction of the total requests coming to the HTTP server.

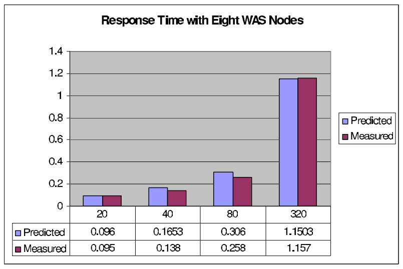
Fig. 7 shows the closed queuing network model for the experiment scenario. The queuing network is symmetric as the delay-centers (think-time, network delay) are modeled as Infinite Server (IS) stations, the HTTP and WAS servers are modeled as PS stations, the back-end is modeled as a FCFS station with infinite waiting room with service times exponentially distributed at the back-end. The distribution of service times at the front-end and WAS nodes can be any arbitrary distribution. As a result, the stationary distribution of the queuing network has a product-form solution. The Mean Value Analysis (MVA) for closed queuing networks with product-form solution gives a recursive relation between performance metrics for the system with the increase of population sizes.

The first step of the model building uses experimental performance data for one experiment configuration (e.g. 20 active clients) to infer the unknown parameters (the service time) of each physical node. The inference involves Mean Value Analysis to obtain an iterative expression for the end-to-end response time and the utilization at each node in terms of the unknown service times at different nodes. Second, the derived response time is compared with measured end-to-end response time and an optimization problem is formulated with constraints on the measured and model predicted utilization of different nodes. The result of this constrained optimization is the mean service time of each node. Finally, the derived mean service times of all nodes are used to predict the performance metrics, end-to-end response time and CPU usage, at all nodes for scenarios with increasing client populations (40,80,160,..).

So far, we have validated above approach on a set of performance data obtained earlier on a cluster of JS20 blades running WAS ND v6.0. Fig. 8 shows the comparison of the predicted performance data and actual data obtained from experiments. The *X* axes represent the different configurations, identified as the total number of active clients. The *Y* axes represent the response time in second. Note that for both charts, we use the data point associated to 20 active clients for deriving the model. The charts show that the



	20	40	80	160	320
Predicted	0.326	0.6521	1.3041	2.6082	5.2164
Measured	0.33	0.658	1.3	2.6	5.86



	20	40	80	320
Predicted	0.096	0.1653	0.306	1.1503
Measured	0.095	0.138	0.258	1.157

Figure 8. Model validation results

predicted response times match the actual value very well, with error only up to 9%. With this set of early results, we show that we can predict the scaling of the number of active clients for a fixed deployment. We have enhanced the model building process to predict the scaling of the deployed WAS nodes and expect to obtain corresponding results shortly.

5.2 Performance Breakdowns

In order to understand the application from a low level hardware point of view, we use the Hardware Performance Monitor (HPM) tool in the IBM High Performance Computing Toolkit [1] to collect hardware level performance information. Concurrently, we are concentrating on the WAS nodes. We collect the counter information while the WAS process running at the system's full capacity and delivering steady high request throughput. Since a newly started WAS process takes certain time to warm up, we use the *hpmstat* command of the HPM tool to collect the hardware counters of the whole system for a short interval while the WAS process is running. Specifically, because there is very limited support for fine-grain multiplexing of counter-groups in the HPM library, we invoke hpmstat multiple times to collect for all the counter groups. Then we collect the derived metrics provided by the tool from the basic counters. At the time of this writing, the study is still undergoing for 3 different platforms (the p5 575 system, JS20 blades, and JS21 blades).

6 Summary and Future Work

To explore the advantages and disadvantages of increasingly popular scale-out computation approach on contemporary commercial applications, we carried out extensive performance studies of a commercial application Trade in both scale-out and scale-up environments. This paper reports some of our preliminary results. Through cross system performance comparison, we show that for such workload, scale-out approach has clear advantage over scale-up approach when considering performance/cost factor. In term of scalability of such workload, we show that WebSphere Application Server packages for distributed environment scale well while the possible bottleneck of the application deployment is the database tier. We have explored straightforward approaches with the effort to scale-out the database tier. The preliminary results suggest that both database partitioning feature (DPF) and federated database server approaches are not exactly suitable for Trade, which has small tables and short transactions. In addition, we have discussed our on-going effort on further performance studies: (1) study of performance/scalability for larger deployments by adopting the IBM AMBIENCE queuing network modeling tool, (2) performance breakdowns utilizing IBM ACTC hardware counter library.

In term of future work, we plan to do similar study for the more popular SPECjAppServer2004 benchmark. In addition, we want to engage study of the TPC benchmarks on the course of exploring scale-out solutions for the database tier of web-based transaction workload. This will by-pass the middle-tier of such workload and thus exposes insights that are more relevant.

7 Acknowledgement

We thank following people from IBM for their helpful discussions, comments, and resource-related helps: Mohammad Banikazemi, Bulent Abali, Trey Cain, Dilma M. Da Dilva, Harold Hall, Joefon Jann, Jack Kouloheris, Robert Krull, Daniel A. Prener, Kyung D. Ryu, Mauricio J. Serrano, Xiaowei Shen, Yefim Shuf, and Garret Swart.

References

[1] IBM Advanced Computing Technology Center Hardware Performance Monitor Users Guide. URL: http://domino.research.ibm.com/comm/research_projects.nsf /pages/actc.hardwareperf2.html.

[2] IBM DB2 Database for Linux, UNIX, and Windows Information Center. URL: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp.

[3] IBM System p and AIX Information Center. URL: http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp.

[4] IBM Systems: BladeCenter. URL: http://www-03.ibm.com/systems/bladecenter/.

[5] IBM Trade Performance Benchmark. URL: https://www14.software.ibm.com/webapp/iwm/web /preLogin.do?source=trade6.

[6] WebSphere Studio Workload Simulator Programming Reference. URL: http://publibfp.boulder.ibm.com/epubs/pdf /c3163082.pdf, October 2003.

[7] WebSphere Application Server V6 Scalability and Performance Handbook. *IBM Redbook number: SG246392*, at URL: http://www.redbooks.ibm.com, May 2005.

[8] Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment. *IBM Redbook number: SG247153*, at URL: http://www.redbooks.ibm.com, June 2006.

[9] R. Ahuja. Introduction to DB2 for Linux in Enterprise and Cluster Environments. IBM DB2 White Papers for Linux, at http://www-306.ibm.com /software/data/db2/linux/papers.html, June 2003.

[10] A. R. Alameldeen et al. Evaluating Non-deterministic Multithreaded Commercial Workloads. In *5th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-02)*, Cambridge, MA, Feb 2002.

[11] R. Chong. Overview of the Database Partitioning Feature. IBM developerWorks technical notes, at http://www-128.ibm.com/developerworks, March 2004.

[12] X. Du, X. Zhang, Y. Dong, and L. Zhang. Architectural Effects of Symmetric Multiprocessors on TPC-C Commercial Workload. *Journal of Parallel and Distributed Computing*, 61(5):609–640, May 2001.

[13] S. Englert, S. Harris, and H. Kache. Performance characteristics of new functionality in WebSphere Federation Server. *IBM developerWorks technical notes*, at http://www-128.ibm.com/developerworks, Dec 2006.

[14] N. Griffiths. nmon Manual. URL: http://www-941.ibm.com/ collaboration/wiki/display/WikiPtype/nmon+Manual, 2006.

[15] L. Hass, E. Lin, and M. Roth. Data Integration Through Database Federation. *IBM Systems Journal*, 41(4), 2002.

[16] J. Layton. How Amazon Works. http://computer.howstuffworks.com/amazon1.htm, 2005.

[17] J. Layton. How eBay Works. http://computer.howstuffworks.com/ebay2.htm, 2005.

[18] Z. Liu, C. H. Xia, P. Momcilovic, and L. Zhang. AMBIENCE: Automatic Model Building using InferEnce. In *Congress MSR03*, Metz, France, Oct 2003.

[19] S. Marlin. Wall Street Firms Embrace Cutting-Edge IT. in *InformationWeek*, March 2005.

[20] F. McArchur. Best Practices for Tuning DB2 UDB v8.1 and its Databases. *IBM developerWorks technical notes*, at http://www-128.ibm.com/developerworks, Apr 2004.

[21] I. M. Steiner and Y. Shuf. A Characterization of a Java-based Commercial Workload on a High-end Enterprise Server. *ACM SIGMETRICS Performance Evaluation Review*, 34(1):379–380, June 2006.