FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 20, no. 1, April 2007, 31-43

Measurement of the Achieved Performance Levels of the WEB Applications With Distributed Relational Database

Dragan Simić, Srećko Ristić, and Slobodan Obradović

Abstract: This paper describes the methods and means used for creating a computer cluster using ordinary PCs. The cluster is running a replicated relational database, and two custom Web applications, used as the database clients. Operating system running all this is Linux 2.4, with Linux Virtual Server (LVS) used as the load-balancing solution, MySQL 4.0 as the replicated database (supporting transactions and referential integrity), and Apache 1.3 as the Web server. PHP4 is used for Web applications development. Additionally, a High Performance Computing (HPC) cluster is implemented, using OpenMOSIX. Measurement and comparison of achieved performance levels is done as the final aim, using two custom applications developed for that purpose, acting as clients of two deployed Web applications. Performance-measurement applications are running under Microsoft Windows, and are developed using Borland Delphi 7.

Keywords: Relational databases, Web applications, Linux Virtual Server, LVS, HPC, Load balancing, MySQL 4.0, PHP4.

1 Introduction

The cluster of independent servers interconnected by a fast computer network, is nowadays a common way used to make systems able of achieving performance levels demanded by deployed applications and day-to-day growing number of actual endusers. Many enterprise or business applications, not only Web sites, are realized as a form of distributed Web application, commonly using a relational database backend for actual data storage. So the actual areas for making performance improvements, always having scalability in mind, are application servers (Web servers), database servers and the network infrastructure.

Manuscript received Jan. 27, 2006.

The authors are with The Advanced School of Electrical Engineering, Vojvode Stepe 283, 11000 Belgrade, Serbia (e-mails: dragan.simic@gmail.com, sreckor@blic.net, sobradovic@vets.edu.yu).

In this paper, the application server performance levels and scalability is achieved by grouping more *real* Apache Web servers into one *large* virtual server, with the ease of adding one or more *real* servers to the virtual *large* one, using LVS as the load balancer. The database subsystem is achieving performance levels and scalability in a similar way. MySQL database is replicated, also grouping more *real* into one virtual *large* database server, keeping the possibility of adding one or more *real* servers into the virtual server. Network infrastructure that was used is a switched 100Mbps Fast Ethernet, which keeps the scalability on the levels achieved by the application servers and database subsystem - Fast Ethernet can be speeded up by using either bonding of more interfaces together or Gigabit Ethernet (possibly bonded, too).

Also, it is important to note that the aim of this work was not to achieve the maximum possible performance levels at the present day, the aim was to get relative relations between performance levels achieved using more solutions and scenarios for simulated real-life situations.

2 Cluster Realization

The actual cluster consisted of five Web servers, five database servers, one LVS load-balancer, and two client machines. All servers were running Slackware 9.0 with custom 2.4.21 Linux kernel (with *hidden* patch applied), and had the identical hardware configuration - Intel Celeron 1100MHz, ASUS motherboard with SiS chipset, 256MB PC133 SDRAM, 20GB ATA-66 5400rpm HDD, 100Mbps integrated SiS NIC (Network Interface Controller) whereas LVS machine had also two additional 100Mbps Realtek-8139C NICs as PCI cards. Both client machines were running Windows 2000 Professional with SP3, and had the following hardware configuration: Intel Pentium 4 1600MHz, ASUS motherboard with Intel chipset, 512MB PC133 SDRAM, 40GB ATA-100 7200rpm HDD, 100Mbps VIA Rhine-II NIC.

Logically, LVS balancer has two Virtual IP addresses (VIPs), first for the virtual Web server and second for the virtual Database server.

Physically, LVS balancer machine has three Fast Ethernet interfaces and they are all connected to a common Fast Ethernet switch because the used way of implementing LVS was LVS/DR (Direct Routing), which requires all real servers and load balancer to have one physical interface connected to the same switch. For an explanation and more information of the way LVS/DR works, and about LVS in general, please consult [1], [2] and [3]. The separation of incoming and outgoing network traffic through the balancer machine was achieved by the appropriate

manipulation of balancer machine responses to address resolution protocol (ARP) queries for virtual IPs (VIPs), where *hidden* patch plays the main role together with appropriate *rp_filter* and *arp_filter* Ethernet interfaces settings. The final flows of traffic through the LVS balancer are shown in Figure 2.

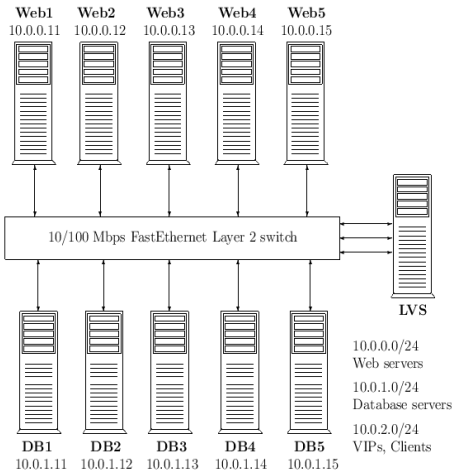

Fig. 1. The used cluster structure, network topology and IP addresses.

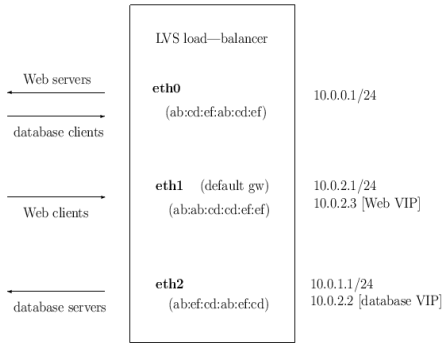

Fig. 2. Configuration of load-balancer interfaces, and the traffic flow.

The relational database replication was implemented through the MySQL's native replication mechanism, please see [4] for detailed information about MySQL replication, and for more information about database replication in general please see [5]. One detail about that replication mechanism is that one database server acts as a master, while the others have slave roles; with a constraint of having all of the non-readonly SQL queries executed only on the master server, which does the propagation of changes on all slave servers. All database servers were running tuned official binary release of MySQL 4.0.15-Max.

One of the Web servers (one that acted as *master* server) was running *rsync* server, which was used by *rsync* clients on the rest of Web servers (*slave* servers) to synchronize the HTML and PHP contents of a shared directory. The reason why some of the more sophisticated methods for sharing a directory contents were not used (like NFS, for example) is a completely static nature of the directory's content (it was actually synchronized just once). All Web servers were running custom built and tuned Apache 1.3.28, with PHP 4.3.3 as a Dynamic Shared Object (DSO) module. For more information about Apache and PHP in general, please consult [6] and [7].

As an addition, an OpenMOSIX (HPC Cluster at the operating-system level)

was also implemented and tested, with the total of eleven nodes in it. All of the servers were connected to a common 100Mbps switch, each server with a single Fast Ethernet interface. All servers were running a custom 2.4.21 Linux kernel, with OpenMOSIX patches applied. HPC Cluster was used to run 22 instances of *distributed.net* (http://www.distributed.net/) client, with the expected almost-linear increase of performances. The reason why OpenMOSIX was not used to create a HPC cluster on which the Web testing applications would run is just because OpenMOSIX does not have the possibility of thread migration between cluster nodes. At the same time, MySQL RDBMS is a true multi-threaded application. For more details about this problem, and more information about OpenMOSIX in general, please consult [8] and [9].

3 WEB Applications

Two custom Web applications were developed, keeping in mind the objective of creating and simulating real-life scenarios and situations using PHP4 as the scripting language.

The first Web application called *Billing*, acts as an application for a hypothetical mobile-devices telecom operator. One interesting detail about this application is that it is using a *dynQuery* system for dynamic creation and execution of SQL queries and PHP scripts, which is developed especially for this Web application. The core idea of *dynQuery* is to also keep SQL queries in the database, of course in special tables used just for that purpose and to get all parameters used to choose the actual SQL queries and substitute the values for their arguments, from the Web client by GET or POST methods.

The database used for *Billing* application has a total of four InnoDB tables, with referential integrity constraints, and a total of about 4,000,000 records. For more details about InnoDB please have a look at [10]. SQL queries that are used for performance testing, include both read-only and update queries.

The second Web application, called *Employees*, is actually a modified test database shipped with Borland's Interbase RDBMS and the actual modifications were targeted to get it in compliance with MySQL 4.0 RDBMS. SQL queries used for performance tests are contained in statically generated PHP scripts (one PHP script - one SQL query) having actual values for their arguments passed from the Web client by using GET method.

The database used for *Employees* application has a total of 10 InnoDB tables with referential integrity constraints and a total of about 400 records. SQL queries used for performance testing are only read-only queries.

4 Performance Measurement Applications

Web Server Load Tool (WSLT) application used for performance measuring, was written in Delphi 7 using Indy component suite (http://www.indyproject.org/). The main goal for this application was the generation of large Web server load simulating real users' behavior.

To achieve this goal, multithreaded programming model was used. Microsoft Windows, of course, has appropriate API calls for the control of processes executed through threads. The most significant call is *CreateThread* which enables parallel execution for created threaded processes. For the actual implementation, Delphi abstract class *TThread* is used. New class *TRobot* is inherited from *TThread*, and it was used to create web server requests. *TRobot* class has following properties:

- Increment step (connections),
- Polling interval (ms),
- Max. number of connections,
- Time to run (sec),
- No. of browsers,
- Use authorization, and
- Method.

Based on these properties values, WSLT performs requests to Web server cluster using URLs stored in a text file. The actual order of URLs simulates a real-user-browsing-Web-site behavior.

While running, WSLT produces a large amount of data that has to be stored for later analysis. WSLT is actually just saving the results data in plain text format, what is not suitable for effective analysis, so a new utility was developed - *Analysis Tool (AT)*.

AT was developed in Delphi 7 and is relying on the Interbase 6 RDBMS. AT reads and processes WSLT output files, stores the actual data into database and produces a set of reports based on stored data. Using the AT application, all of the collected data was loaded into the database and the actual analysis was performed using the reports generated by AT.

Reports generated by AT are stored in HTML files using *TBasePageProducer* component which is shipped as an integral part of Delphi 7. For the generation of graphs *TeeChart* component was used, which is also a part of Delphi 7.

Some parts of the report depend on the actually used application (see the Web test applications section) and some depend on the cluster configuration and WSLT,

so the AT uses an XML based configuration file for the report generation configuration.

The produced report includes the following test results:

- Total number of HTTP requests (greater value is better),
- HTTP requests per second, average (greater value is better),
- Number of good HTTP responses (200 OK) (greater value is better),
- Number of bad HTTP responses (non-200 OK) (lower value is better),
- Total HTML received [MB],
- Average HTML traffic [Mbit/s],
- Minimum HTTP response time [ms] (lower value is better),
- Maximum HTTP response time [ms] (lower value is better),
- Minimum number of HTTP connections (lower value is better),
- Maximum number of HTTP connections (lower value is better),
- Update SQL queries per second (average value),
- Non-update SQL queries per second (average value),
- Total SQL queries per second (average value).

The produced report also includes the following graphs (an example with all eight produced graphs is given on Figure 3):

- Graph 1: Represents the number of active requests in a particular second of testing - for active requests, the requests without server response are counted
- Graph 2: Represents minimal, average and maximal response time in a particular second of testing. In the ideal situation, difference between all three times should be small and without any variations in the whole test time
- Graph 3: Represents the amount of received HTML traffic in a particular second of testing. It is important to highlight the fact that it shows the HTML code traffic speed, as seen by WSLT, and not the actual TCP/IP protocol speed. That is an explanation of the fact that sometimes WSLT (as a plain user space application) sees the HTML traffic rates exceeding the actual network interface speeds, because the Windows probably has some buffers implemented inside the TCP/IP stack that make those traffic bursts possible.
- Graph 4: Represents the number of 'good' responses received from the tested Web server (for the 'good' type, the HTTP /1.1 code 200 responses are counted)

- Graph 5: Represents the number of ’bad’ responses received from the tested Web server (for the ’bad’ type, the HTTP /1.1 responses different from code 200 are counted)
- Graph 6: Represents the distribution of responses by percentage
- Graph 7: Represents the number of update SQL queries in a particular second (this report was created only for the *Billing* application)
- Graph 8: Represents the number of non-update SQL queries in a particular second (this report was created only for *Billing* application)

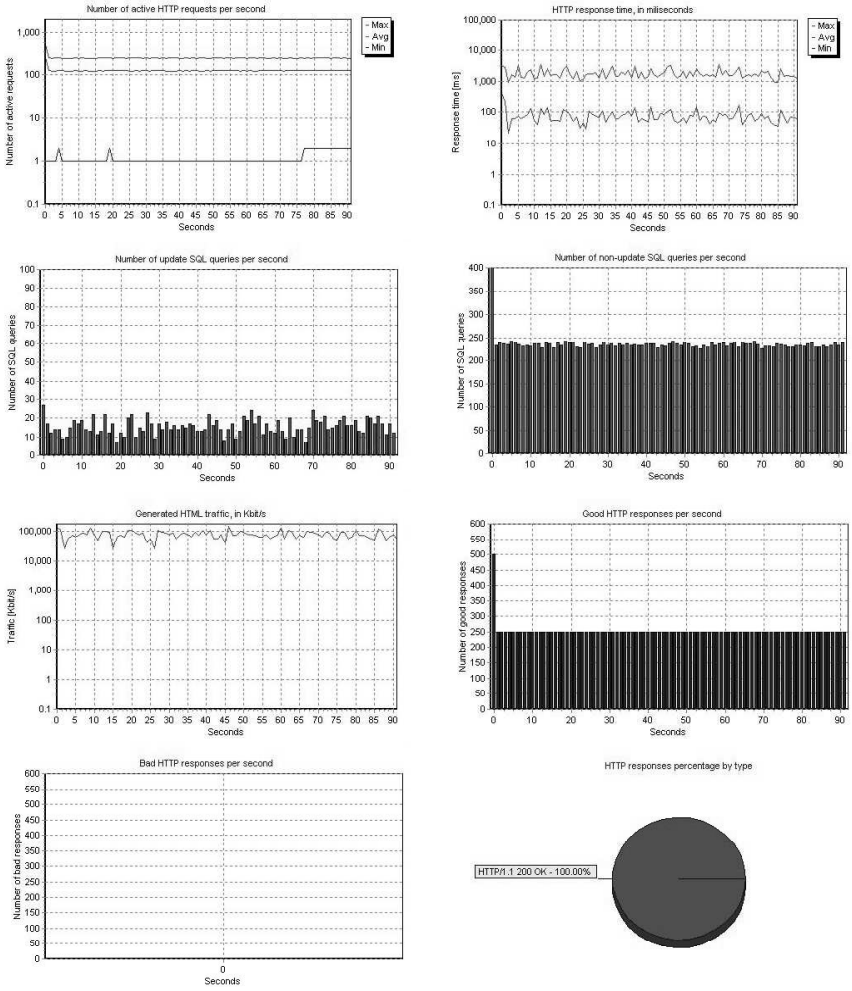

Fig. 3. An example of the graphs produced by WebServer Load Tool.

5 Scenarios and Situations

Various scenarios for actual performance measurements were chosen by two main parameters determining the scenario: Load level (produced by WebServer Load Tool, WSLT), and cluster configuration (number of used Web and database servers, and some other realization details). The following three load levels were chosen and they are listed here using WSLT configuration parameters for simulating those load levels:

Table 1. The parameters of simulated load levels.

Parameter/Load level	Light	Normal	Heavy
Increment step (connections)	50	120	250
Polling interval (msec)	20	50	50
Max number of connections	200	500	1000
Time to run (sec)	90	90	90
No. of browsers	15	25	50

The main background idea used while choosing these parameters for all three load levels, is to make WSLT reach the desired number of simultaneous connections (Max number of connections) in approximately one second. After starting the simulation, WSLT tries to reach and keep up the desired number of connections by checking the number of active connections every Polling interval milliseconds and if the number is lower than desired, increases the number of connections by Increment step; but if the number of connections is equal or greater, WSLT waits for the next Polling interval and the described process is repeated until expiration of Time to run. By using this algorithm, WSLT tries to simulate a real-life scenario of users accessing and using a Web application.

There are a few commercial applications for Web server and database benchmarking and one of them is Quest Software's Benchmark Factory for Databases [11]. Benchmark Factory for Databases is a performance and code scalability testing tool that simulates users and transactions on the database and replays production or synthetic workload in non-production environments.

The following cluster configurations were used for each Web application, using *Light*, *Normal* and *Heavy* load levels, keeping in all cases LVS load-balancer in the *LVS/DR* (LVS/Direct Routing) mode:

- 1 database + 1 Web server, 1 WSLT
- 2 database + 5 Web servers, *wrr* and *wlc* scheduler, 1 WSLT
- 5 database + 5 Web servers, *wrr* and *wlc* scheduler, 1 WSLT
- the configurations listed above, but with two WSLT clients

- some of the configurations were also tested having SQL updates removed, in order to measure the impact of propagating SQL updates to overall performance.

PHP4 has a built-in support for persistent MySQL database connections (please see [7] for more details), and all tests were repeated using persistent database connections but with no noticeable performance gains. Those results of no performance gains are quite surprising as persistent connections eliminate the repeating database connection establishments what should result in significant performance gains. One of the possible explanations for no performance gains while using persistent database connections is that used connection pool is created once, at the beginning of the test (exactly, the pool creation is finished after WSLT reaches the maximum number of simultaneous connections). LVS distributes database connections over the cluster members only while some new connections are added to the pool after that point the whole task of distribution over the cluster members is performed solely by the PHP engine, which either does not distribute them in a performance-efficient manner or distributes them in a way which is in conflict with the initial distribution done by LVS.

6 Results

While analyzing the measured values, it is important to keep in mind that the establishment of each TCP connection involves numerous steps: opening client TCP connection to the virtual Web server, scheduling the request to one of the real Web servers, HTTP request processing, opening a connection to the virtual Database server, scheduling the connection to one of the real Database servers, database request processing, returning the database query result and (optionally) closing the database connection, and finally returning the Web server response and closing the Web client connection. In the other words, measurement results were gathered, produced and analyzed by measuring the user-space HTTP traffic, as seen by the WSLT as a Windows application, keeping out of sight all the latencies, overheads and other parameters irrelevant from the aspect of an average real-world user, what WSLT actually intended to simulate and measure.

Table 2 and Table 3 are showing the measured performance levels and the performance level differences, respectively, for the *Billing* application. Results are shown for all of the cluster configurations and all of the generated load levels. For every category, the best result is shown in **bold** font.

Table 4 and Table 5 show the measured performance levels and the performance level differences, respectively, for the *Employees* application. Results are shown for

Table 2. The parameters of simulated load levels.

Cluster configuration/Results	(1)[a]	(2)[b]	(3)[c]	(4)[d]
1 DB, 1 Web, Light	4003	44.48	2.60	41.88
2 DB, 5 Web, Light, wrr	**6142**	**68.24**	**4.02**	**64.22**
5 DB, 5 Web, Light, wrr	5861	65.12	3.92	61.20
5 DB, 5 Web, Light, wlc	1899	21.10	1.34	19.76
1 DB, 1 Web, Normal	5827	64.74	4.32	60.42
2 DB, 5 Web, Normal, wrr	11108	123.42	**8.01**	115.41
5 DB, 5 Web, Normal, wrr	11114	123.49	7.96	115.53
5 DB, 5 Web, Normal, wlc	**11139**	**123.77**	7.86	**115.91**
1 DB, 1 Web, Heavy	8592	95.47	6.11	89.36
2 DB, 5 Web, Heavy, wrr	22748	252.76	15.07	237.69
5 DB, 5 Web, Heavy, wrr	**23249**	**258.32**	15.99	**242.33**
5 DB, 5 Web, Heavy, wlc	23134	257.04	**16.26**	240.79
5 DB, 5 Web, Heavy, wlc, 2 WSLT	45961	510.68	32.47	478.21
5 DB, 5 Web, Light, wlc, no SQL upd.	6022	66.91	0.00	66.91

[a]The total number of performed HTTP requests during the test
[b]Number of processed HTTP requests per second
[c]Number of performed update SQL queries per second
[d]Number of performed non-update SQL queries per second

Table 3. Cluster performance level differences for Billing application.

Cluster configuration/Differences	(1)[a]	(2)[b]	(3)[c]	(4)[d]
1 DB, 1 Web, Light	34.82	34.82	35.32	34.78
2 DB, 5 Web, Light, wrr	0.00	0.00	0.00	0.00
5 DB, 5 Web, Light, wrr	**4.58**	**4.57**	**2.48**	**4.71**
5 DB, 5 Web, Light, wlc	69.08	69.08	66.67	69.23
1 DB, 1 Web, Normal	47.68	47.69	46.07	47.87
2 DB, 5 Web, Normal, wrr	0.28	0.28	0.00	0.43
5 DB, 5 Web, Normal, wrr	**0.22**	**0.23**	**0.62**	**0.33**
5 DB, 5 Web, Normal, wlc	0.00	0.00	1.87	0.00
1 DB, 1 Web, Heavy	63.04	63.04	62.42	63.12
2 DB, 5 Web, Heavy, wrr	2.15	2.15	7.32	1.91
5 DB, 5 Web, Heavy, wrr	0.00	0.00	**1.66**	0.00
5 DB, 5 Web, Heavy, wlc	**0.49**	**0.50**	0.00	**0.64**
5 DB, 5 Web, Heavy, wlc, 2 WSLT	−98.67	−98.68	−99.69	98.60
5 DB, 5 Web, Light, wlc, no SQL upd.	−217	−217	n/a	−238

[a]Difference of the total number of performed HTTP requests during the test
[b]Difference of the average number of processed HTTP requests per second
[c]Difference of the average number of performed update SQL queries per second
[d]Difference of the average number of performed nonupdate SQL queries per second

Table 4. Measured cluster performance levels for *Employees* application.

Cluster configuration/Results	(1)[a]	(2)[b]
1 DB, 1 Web, Light	6289	69.88
5 DB, 5 Web, Light, wrr	5094	56.60
5 DB, 5 Web, Light, wlc	**13551**	**150.57**
1 DB, 1 Web, Normal	14280	158.67
5 DB, 5 Web, Normal, wrr	14349	159.43
5 DB, 5 Web, Normal, wlc	**14579**	**161.99**
1 DB, 1 Web, Heavy	23176	257.51
5 DB, 5 Web, Heavy, wrr	23000	255.56
5 DB, 5 Web, Heavy, wlc	**23249**	**258.32**

[a]The total number of performed HTTP requests during the test
[b]The average number of processed HTTP requests per second

all cluster configurations and all generated load levels. For every category, the best result is shown in **bold** font.

Table 5. Cluster performance levels differences for *Employees* application.

Cluster configuration/Differences	(1)[a]	(2)[b]
1 DB, 1 Web, Light	**53.59**	**53.59**
5 DB, 5 Web, Light, wrr	62.41	62.41
5 DB, 5 Web, Light, wlc	0.00	0.00
1 DB, 1 Web, Normal	2.05	2.05
5 DB, 5 Web, Normal, wrr	**1.58**	**1.58**
5 DB, 5 Web, Normal, wlc	0.00	0.00
1 DB, 1 Web, Heavy	**0.31**	**0.31**
5 DB, 5 Web, Heavy, wrr	1.07	1.07
5 DB, 5 Web, Heavy, wlc	0.00	0.00

[a]Difference of the total number of performed HTTP requests during the test
[b]Difference of the average number of processed HTTP requests per second

The *Billing* application showed the following performance levels, for all cluster configurations:

- *Light* load level: 50% of performance levels increase for the cluster configurations with more than one Web and Database servers, compared to the 1 DB / 1 Web configuration.
- *Normal* load level: 90% of performance level increase for the cluster configurations with more than one Web and Database servers, compared to the 1 DB / 1 Web configuration. The unexpected behavior of minor performance increase of 0.1% between 5 DB / 5 Web and 2 DB / 5 Web configurations

can be explained by the better optimization of Database servers for multiple concurrent connections.

- *Heavy* load level: 170% of performance level increase for the cluster configurations with more than one Web and Database servers, compared to the 1 DB / 1 Web configuration.

The *Employees* application showed almost the same performance levels for all cluster configurations and load levels, except for the *Light* load level. *Light* load level showed performance level increase of 110% what was caused by an internal error of the WSLT's internal logic, but unfortunately the timeline did not allow the wrong measurements to be repeated. Almost the same performance levels are explained by a load level generated by the *Employees* application that was too low to take noticeable advantages of the cluster environment.

7 Conclusion

This project has three main goals, as follows:

- To design and realize a Web and Database server cluster with the capability of large-scale applications deployment
- To design and implement the Web testing applications and other needed testing tools and strategies for performance measurements
- Prove that the realized cluster actually has the capability of handling heavy loads

One of the main conclusions is a much better optimization of the used Database server (MySQL 4.0) for many concurrent connections, compared to the used Web server (Apache 1.3), which was expected because of the nature and main concerns that were used while the companies were designing all of the actual RDBMS'es. The choice of a load-balancing scheduler did not have noticeable impacts on the achieved performance levels, which can be explained by a quite long duration of client TCP connections (Web and Database connections).

The second of main conclusions is that a Web application needs to be properly designed with the intention of running it on a cluster. As the cluster adds additional overheads that result in greater latencies, but increases the throughput many times, Web application must have the real needs for making the idea of running it on a cluster reasonable. In the other words, running a Web application on the cluster will show a noticeable performance boost only if the Web application generates load that is really high - only in that case the multiple throughput increase makes

more improvement that outperforms the additional latencies which lower the final performance levels.

The last conclusion is based on the measured performance levels that showed performance increase of 150% up to 270%, while the performance increase gets higher as the load generated by the Web application also gets higher.

The conclusions drawn from obtained results are comparable with the conclusions stated in another Web and database benchmarking research paper [12]. The objective of those experiments was to evaluate LVS's ability to distribute Web requests among several servers in a variety of LVS configurations and offer a comparison of LVS's ability to scale on Linux 2.2 versus Linux 2.4.

References

[1] Linux Virtual Server (LVS), *Official LVS on-line documentation*. GPL licensed, 2004. [Online]. Available: http://www.linuxvirtualserver.org/

[2] W. Zhang, S. Jin, and Q. Wu, *Creating Linux Virtual Servers*. National Laboratory for Parallel and Distributed Processing, China, 2003.

[3] W. Zhang, *Linux Virtual Server for Scalable Network Services*. National Laboratory for Parallel and Distributed Processing, China.

[4] MySQL AB, *MySQL Reference Manual for version 4.0.15, official on-line documentation*. GPL licensed, 2004. [Online]. Available: http://www.mysql.com/

[5] R. Vanderwall, *Database Replication Prototype-Master thesis*. Department of Mathematics and Computer Science,: University of Groningen, 2003.

[6] The Apache Web Server, *Apache 1.3 Documentation, on-line documentation*. GPL licensed, 2004. [Online]. Available: http://www.apache.org/

[7] PHP Hypertext Preprocessor, *Official PHP on-line documentation*. GPL licensed, 2004. [Online]. Available: http://www.-php.net/

[8] K. Buytaert, *MOSIX HOWTO, on-line documentation, part of The Linux Documentation Project*. GPL licensed, 2004. [Online]. Available: http://www.tldp.org/

[9] OpenMOSIX, *Official OpenMOSIX on-line documentation*. GPL licensed, 2004. [Online]. Available: http:// openmosix.sourceforge.net/

[10] Innobase Oy Inc, *InnoDB Engine in MySQL-Max 3.23.56 and MySQL 4.0.12, official on-line documentation*. GPL licensed, 2004. [Online]. Available: http:// www.innodb.com/

[11] Quest Software, *Benchmark Factory for Databases*, 2006. [Online]. Available: http://www.quest.com/benchmark_factory

[12] P. O'Rourke, *Performance Evaluation of Linux Virtual Server*, 2005. [Online]. Available: http://www.linuxvirtualserver.org/performance/lvs.ps.gz