Mini Web Server Clusters for HTTP Request Splitting

Bharat S. Rawal
Department of Computer and Information Sciences
Towson University
Towson, Maryland, USA
brawal@towson.edu

Ramesh K. Karne
Department of Computer and Information Sciences
Towson University
Towson, Maryland, USA
rkarne@towson.edu

Alexander L. Wijesinha
Department of Computer and Information Sciences
Towson University
Towson, Maryland, USA
awijesinha@towson.edu

***Abstract*—HTTP request splitting is a new concept where the TCP connection and data transfer phases are dynamically split between servers without using a central dispatcher or load balancer. Splitting is completely transparent to the client and provides security due to the inaccessibility and invisibility of the data servers. We study the performance of mini Web server clusters with request splitting. With partial delegation in which some requests are split, throughput is better, and response times are only marginally less than for an equivalent non-split system. For example with partial delegation, for a four-node cluster with a single connection server and three data servers serving 64 KB files, and for a three-node cluster with two connection servers and a single data server serving 4 KB files, the respective throughput improvements over non-split systems are 10% and 22%, with only a marginal increase in response time. In practice, the throughput improvement percentages will be higher and response time gaps will be lower since we ignore the overhead of a dispatcher or load balancer in non-split systems. Although these experiments used bare PC Web servers without an operating system/kernel for ease of implementation, splitting and clustering may also be implemented on conventional systems.**

Keywords -Splitting HTTP Requests, Performance, Cluster Computing, Web Servers, Bare Machine Computing.

I. INTRODUCTION

Load balancing is frequently used to enable Web servers to dynamically share the workload. For load balancing, a wide variety of clustering server techniques [20, 21, 22] are employed. Most load balancing systems used in practice require a central control system such as a load balancing switch or dispatcher [20]. Load balancing can be implemented at various layers in the protocol stack [19, 20]. This paper considers a new approach to load balancing that involves splitting HTTP requests among a set of servers, where one or more connection servers (CSs) handle TCP connections and may delegate a fraction (or all) requests to one or more data servers (DSs) that serve the data [23]. For example, the data transfer of a large file could be assigned to a DS and the data transfer of a small file could be handled by the CS itself.

One advantage of splitting is that splitting systems are completely autonomous and do not require a central control system such as dispatcher or load balancer. Another advantage is that no client involvement is necessary as in migratory or M-TCP [13]. In [23], splitting using a single CS and a DS was shown to improve performance compared to non-split systems. Since the DSs are completely anonymous and invisible (they use the IP address of the delegating CS), it would be harder for attackers to access them. In particular, communication between DSs and clients is only one-way, and DSs can be configured to only respond to inter-server packets from an authenticated CS. We study the performance of three different configurations of Web server clusters based on HTTP splitting by measuring the throughput (in requests/sec), and the connection and response times at the client.

In real world applications, some servers may be close to data sources, and some servers may be close to clients. Splitting a client's HTTP request and the underlying TCP connection in this manner allows servers to dynamically balance the workload. We have tested the splitting concept in a LAN that consists of multiple subnets connected by routers. In HTTP splitting, clients can be located anywhere on the Internet. However, there are security and other issues that arise when deploying mini clusters in an Internet where a CS and a DS are on different networks [23].

The remainder of the paper is organized as follows. Section II presents related work; Section III describes splitting and the different cluster configurations used in this study; Section IV presents results of experiments; Section V discusses impacts of splitting; and Section VI contains the conclusion**.**

II. RELATED WORK

We implemented HTTP request splitting on a bare PC with no kernel or OS running on the machine. Bare PC applications use the Bare Machine Computing (BMC) or dispersed OS computing paradigm [7], wherein self-supporting applications run on a bare PC. That is, there is no operating system (OS) or centralized kernel running in the machine. Instead, the application is written in C++ and runs

as an application object (AO) [9] by using its own interfaces to the hardware [8] and device drivers. While the BMC concept resembles approaches that reduce OS overhead and/or use lean kernels such as Exokernel [3, 4], IO-Lite [12], Palacio [10], Libra [1], factored OS [15], bare-metal Linux [14], and TinyOS [18], there are significant differences such as the lack of a centralized code that manages system resources and the absence of a standard TCP/IP protocol stack. In essence, the AO itself manages the CPU and memory, and contains lean versions of the necessary protocols. Protocol intertwining is a form of cross-layer design. Further details on bare PC applications and bare machine computing (BMC) can be found in [5, 6].

Splitting an HTTP request by splitting the underlying TCP connection is different from migrating TCP connections, processes or Web sessions; splicing TCP connections; or masking failures in TCP-based servers. For example, in migratory TCP (M-TCP) [13], a TCP connection is migrated between servers with client involvement; in process migration [11], an executing process is transferred between machines; in proxy-based session handoff [2], a proxy is used to migrate Web sessions in a mobile environment; in TCP splicing [19], two separate TCP connections are established for each request; and in fault-tolerant TCP (FT-TCP) [16], a TCP connection continues after a failure enabling a replicated service to survive. Per our knowledge, no work on splitting connections at a protocol level has been done before.

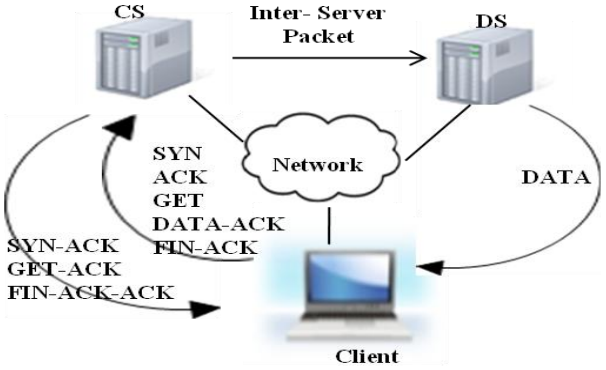


Figure 1. Split architecture

III. CLUSTER CONFIGURATIONS

Figure 1 illustrates generic request splitting [23] and shows the messages exchanged by the intertwined HTTP and TCP protocols. Connection establishment and termination are performed by one or more connection servers (CSs), and data transfer is done by one more data servers (DSs). When a request is split, the client sends the request to a CS, the CS sends an inter-server packet to a DS, and the DS sends the data packets to the client. Inter-server packets may also be sent during the data transfer phase to update the DS if retransmissions are needed. With partial delegation, the CS delegates a fraction of its requests to DSs. With full delegation, the CS delegates all its requests to DSs. The design and implementation details of protocol splitting are provided in [23].

We consider mini Web server clusters consisting of two or more servers with protocol splitting. We then study cluster performance by measuring the throughput and connection and response times of three different server configurations with a varying number of CSs and DSs.

Configuration 1 in Fig. 2a shows full delegation with one CS, one DS, and a set of clients sending requests to the CS. The DS and CS have different IP addresses, but the DS sends data to a client using the IP address of the CS.

Configuration 2 in Fig. 2b shows a single CS with two or more DSs in the system with partial or full delegation. In partial delegation mode, clients designated as non-split request clients (NSRCs) send requests to the CS, and these requests are processed completely by the CS as usual. The connections between the NSRCs and the CSs are shown as dotted lines. With full delegation, clients designated as split-request clients (SRCs) make requests to the CS, and these requests are delegated to DSs. For full delegation, there are no NSRCs in the system. When requests are delegated to DSs, we assume that they are equally distributed among the DSs in round-robin fashion. It is also possible to employ other distribution strategies.

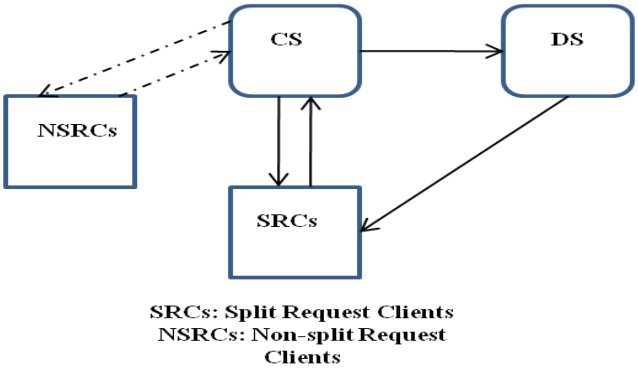


Figure 2a. Split architecture configuration 1

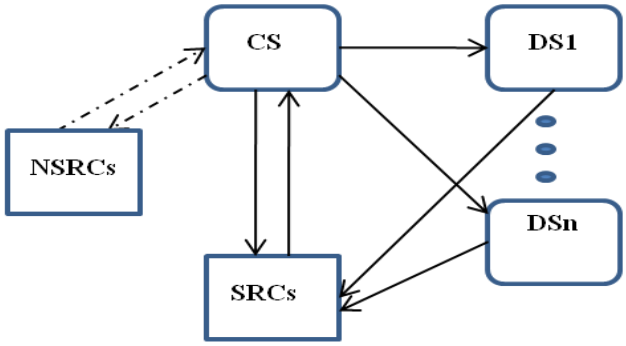
Configuration 3 in Fig. 2c shows two CSs and one DS with both SRCs and NSRCs. For this configuration, we used small file sizes to avoid overloading the single DS. Although we have not done so, multiple DSs could be added as in Configuration 3.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

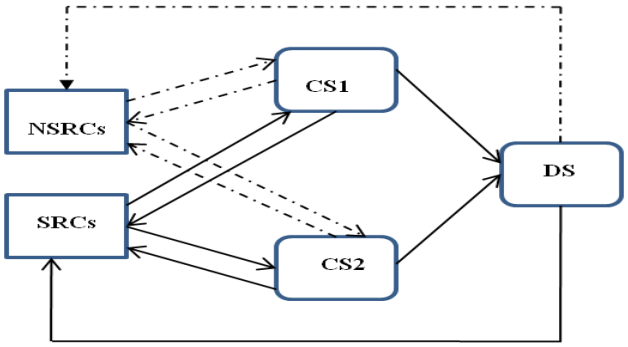
The experimental setup involved a prototype server cluster consisting of Dell Optiplex GX260 PCs with Intel Pentium 4, 2.8GHz Processors, 1GB RAM, and an Intel 1G NIC on the motherboard. All systems were connected to a Linksys 16 port 1 Gbps Ethernet switch. Linux clients were

used to run the http_load stress tool [17], and in addition, bare PC Web clients capable of generating 5700 requests/sec was used to increase the workload beyond the http_load limit of 1000 concurrent HTTP requests/sec per client. The split server cluster was also tested with Internet Explorer browsers running on Windows and Firefox browsers running on Linux.



SRCs: Split Request Clients
NSRCs: Non-split Request Clients

Figure 2b. Split architecture configuration 2



SRCs: Split Request Clients
NSRCs: Non-split Request Clients

Figure 2c. Split architecture configuration 3

B. Configuration 1 (1 CS, 1 DS, full delegation)

In [23], the performance of HTTP splitting with Configuration 1 was evaluated using various file sizes up to 32 KB. Here, we study the performance of Configuration 1 by varying the file size up to 128 KB and measuring the throughput in requests/sec. Fig. 3 shows the results of these experiments. It can be seen that the performance of this configuration is worse than that of a two server non-split system for all file sizes. This is because the DS is overloaded resulting in performance degradation. However, the CS is underutilized since it is only handling connection establishment and termination. For a two server non-split system, we show the theoretical maximum performance (throughput) as being double that of a single (non-split) system, which was determined experimentally to be 6000 requests/sec. In practice, this theoretical limit for non-split systems will not be attained due to the overhead of load balancers and dispatchers.

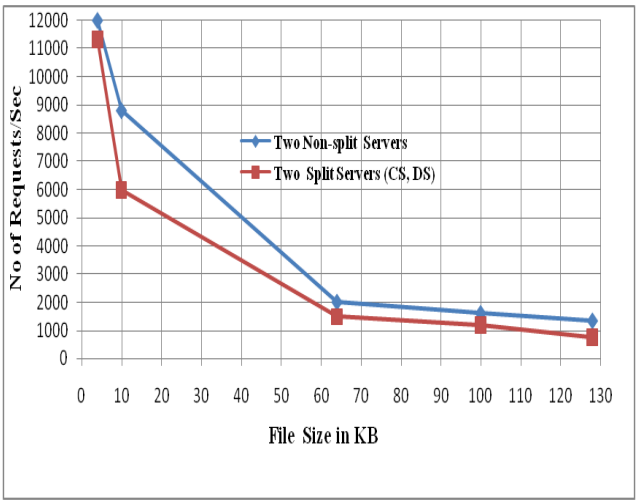


Figure 3. Throughput with increasing file sizes (Configuration 1)

Fig. 4 shows the CPU utilization for the CS and DS in Configuration 1. The DS's CPU utilization for 64 KB files is close to the maximum, indicating that this configuration cannot handle more than 1500 requests/sec. To get further insight into the performance limitations in this case, we determined the impact of connection and response time at the client due to increasing the request rate. The results are shown in Fig. 5. The response time degrades as the number of requests increases starting at 1300 requests/sec and is largest at 1500 requests/sec as expected. These results suggest that performance may be improved by adding more DSs and utilizing the remaining capacity of the CS.

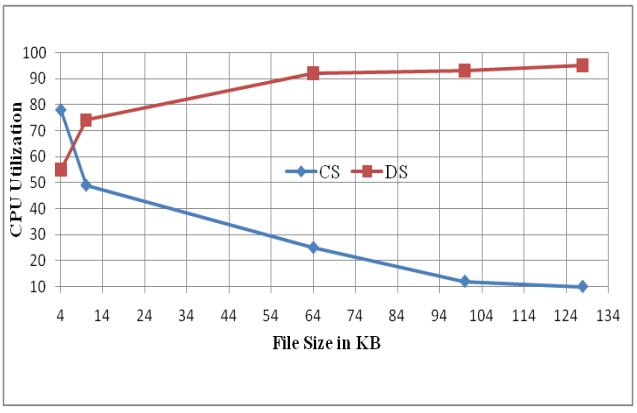


Figure 4. CPU utilization with increasing file sizes (Configuration 1)

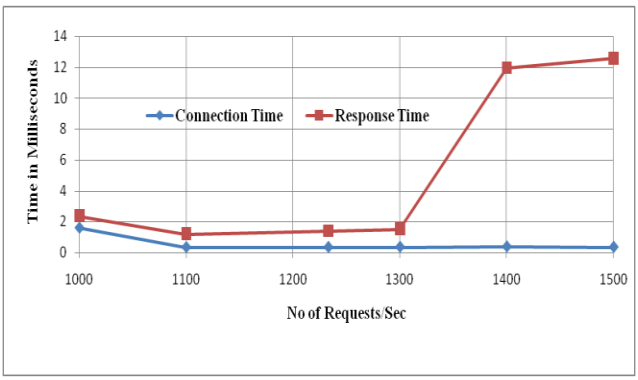


Figure 5. Connection and response times (Configuration 1, file size 64KB)

C. Configuration 2 (1 CS, 1-3 DS, full delegation)

Fig. 6 shows the DS throughput for this configuration by varying the number of DSs with full delegation for 64 KB files. Adding more DSs improves the throughput as seen in the figure. With one DS (i.e. a two-server cluster), 1500 requests/sec can be handled versus the theoretical capacity of 2000 requests/sec for two non-split servers ignoring dispatcher or a load balancer overhead (about 75% of the theoretical non-split performance). With two DSs, the throughput increases to 2500 requests/sec (about 83.3% of the theoretical non-split performance). With three DSs, the maximum throughput is 3700 requests/sec compared to the theoretical limit of 4000 requests/sec for a non-split system (about 92.5% of the theoretical non-split performance).

Fig. 7 shows connection and response times for Configuration 2 with 64 KB files. Although the response time for a single DS is poor, the average response and connection times improve significantly when the number of DSs is increased. A single (non-split) server has connection and response times of 1.62 ms and 2.38 ms respectively, compared to 365 µs and 922 µs respectively for a split system with three DSs and one CS (i.e., connection and response times are improved by factors of 4.4 and 2.6 respectively).

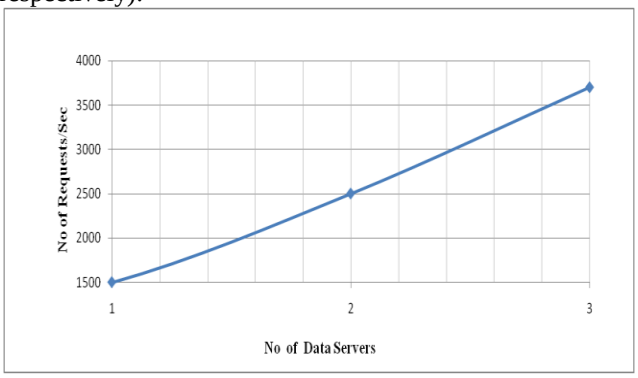


Figure 6. DS throughput (Configuration 2, file size 64KB)

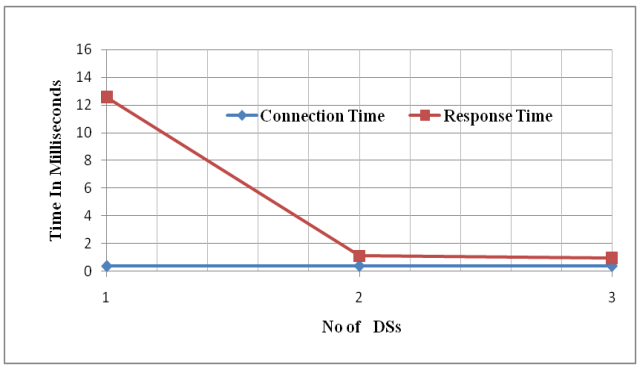


Figure 7. Connection and response times (Configuration 2, file size 64KB)

Fig. 8 shows the CPU utilization in Configuration 2 for the CS and DSs with 64 KB files. The DS utilization drops as expected due to load sharing, while the CS utilization increases due to the increased request rate. However, the CS still has unused capacity to support additional requests.

The preceding experiments show that the performance of a split system with a single CS and three DSs is close to the theoretical limit of a four-server non-split system with respect to both throughput, as well as connection and response times. In addition, the CS is still underutilized.

D. Configuration 2 (1 CS, 1-3 DS, partial delegation)

Configuration 2 with partial delegation and additional clients whose requests are not split (i.e., NSRCs) allows more load to be added in order to efficiently utilize the remaining capacity of the CS. The requests from NSRCs are completely processed by the CS, while the requests from SRCs are split. In this system, we have used 64KB files for requests.

Fig. 9 compares the throughput for split servers with full and partial delegation. The throughput of the split system with partial delegation is more than the theoretical limit for a non-split system due to fully utilizing the capacity of the CS. For a split system with a single CS and a single DS for 64 KB files, partial delegation improves the throughput by 25%. However, this performance gain does not scale up when more DSs are added since the CS is now close to capacity. For example, a split system with 3 DSs improves the throughput only by about 10%. These measurements indicate that a mini-cluster can only have a limited number of DSs if the system is to be self-contained (i.e., without using an external load balancer).

Fig. 10 compares the throughput for split servers with full and partial delegation by varying the file size. The maximum throughput and a performance improvement of 25% are attained for 64 KB files with partial delegation. For 100 KB and 128 KB files, the performance improvements due to splitting are 17.7% and 12.5% respectively with partial delegation. Fig. 11 shows connection and response times with partial delegation for 64

KB files. As with full delegation, response time with partial delegation is poor with only a single DS. However, the response time improves dramatically for split systems with two or three DSs and partial delegation.

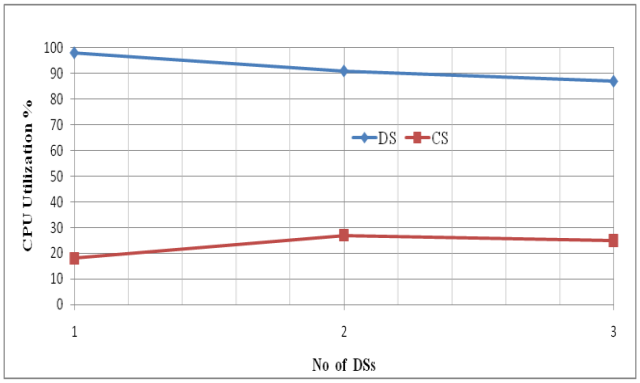


Figure 8. CPU Utilization
(Configuration 2, file size 64KB)

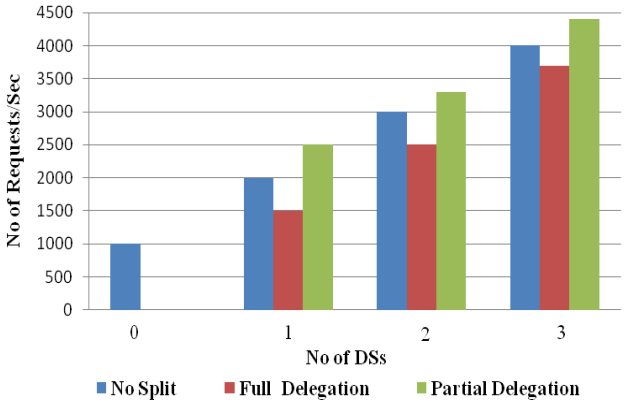


Figure 9. Throughput with full/partial delegation
(Configuration 2, file size 64KB)

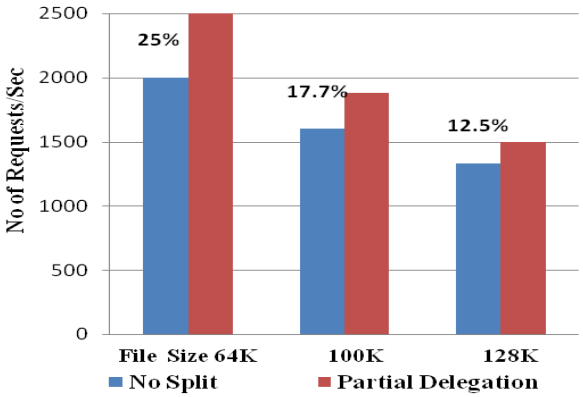


Figure 10. Throughput with full/partial delegation for varying file sizes (Configuration 2)

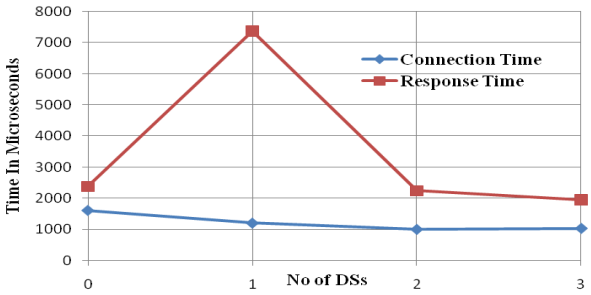


Figure 11. Connection and response times
(Configuration 2, file size 64KB)

E. Configuration 3 (2 CS, 1 DS, partial delegation)

As before, requests are generated by a set of SRCs and NSRCs. For this configuration, 4 KB files were used since larger file sizes will overload the single DS. Figure 12 shows the throughput for three servers with full and partial delegation. Configuration 3 achieves a 6.5% throughput improvement over three non-split servers with full delegation; with partial delegation, it achieves a 22% improvement in throughput compared to three non-split servers.

Fig. 13 shows the connection and response times for Configuration 3. As expected, response time is poor since the single DS gets saturated with the high request rate that be supported with two CSs. With partial delegation, response times improve significantly as the unused CS capacity is used to handle requests from the NSRCs without delegation. While the connection and response times using Configuration 3 are worse than for non-split servers, this disadvantage should be weighed against the increased throughput, cost, and security benefits of using a split system. Also, non-split servers will incur a reduction in response and connection times due to the overhead of using a dispatcher or load balancer.

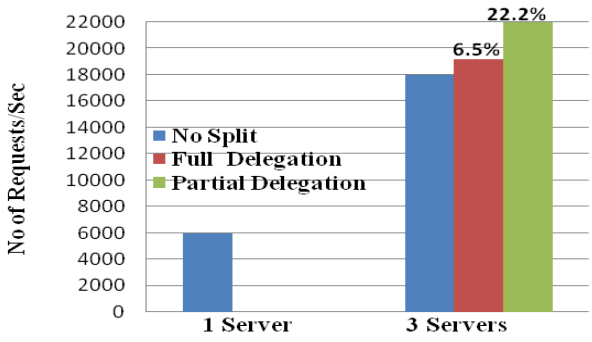


Figure 12. Throughput with full/partial delegation
(Configuration 3, file size 4KB)

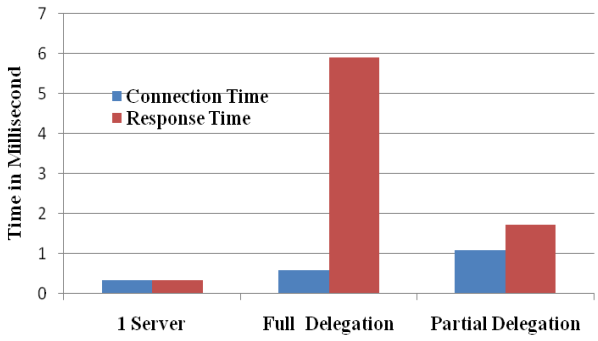


Figure 13. Connection times and response times with full/partial delegation (Configuration 3, file size 4KB)

V. IMPACTS OF SPLITTING

Splitting is a general approach that can be applied in principle to any application protocol that uses TCP (it can also be applied to protocols other than TCP to split the functionality of a protocol across machines or processors). In particular, splitting the HTTP protocol has many impacts in the area of load balancing. We discuss some of these impacts below.

- Split protocol configurations can be used to achieve better response and connection times, while providing scalable performance. Splitting also eliminates the need for (and overhead/cost associated with) external load balancers such as a dispatcher or a special switch.
- Split protocol Configuration 2 (with one CS, one DS) and partial delegation achieves 25% more performance than two homogeneous servers working independently. This performance gain can be utilized to increase server capacity while reducing the number of servers needed in a cluster.
- Split server architectures could be used to distribute the load based on file sizes, proximity to file locations, or security considerations.
- The results obtained in this paper (using specific machines and workloads) indicate that mini-cluster sizes are in the single digits. More research is needed to validate this hypothesis for other traffic loads. However, if we assume that mini-clusters should contain a very small number of nodes, they would be easier to maintain and manage (compared to larger clusters). Using mini-clusters, it is possible to build large clusters by simply increasing the number of mini-clusters.
- Splitting protocols is a new approach to designing server clusters for load balancing. We have demonstrated splitting and built mini-cluster configurations using bare PC servers. However, the general technique of splitting also applies to OS-based clusters provided additional OS overhead can be kept to a minimum (and that undue developer effort is not needed to tweak the kernel to implement splitting).
- When protocol splitting uses two servers (CS and DS) it dramatically simplifies the logic and code in each server (each server only handles part of the TCP and HTTP protocols unlike a conventional Web server that does both protocols completely). Thus, the servers are less complex and hence have inherently more reliability (i.e., are less likely to fail).
- Splitting can also be used to separate the "connection" and "data" parts of any protocol (for example, any connection-oriented protocol like TCP). In general, connection servers can simply perform connections and data servers can provide data. It can also be used to split the functionality of any application-layer protocol (or application) so that different parts of the processing needed by it are done on different machines or processors. Thus, a variety of servers or Web applications can be split in this manner. This approach will spawn new ways of doing computing on the Web.

The configurations studied and the results obtained in this paper can be viewed as a first step to validate the applicability of splitting as a general concept. In future, it would be of interest to investigate its applicability to other protocols and applications.

VI. CONCLUSION

We studied the performance of mini Web server clusters with HTTP request splitting, which does not require a central load balancer or dispatcher, and is completely transparent to the client. Throughput as well as connection and response times with full and partial delegation of requests were measured for a variety of file sizes. A split system with one CS and three DSs, and full or partial delegation, can be used to achieve response times, connection times and throughput close to, or better than, the theoretical limit of non-split systems. For example, this configuration with partial delegation achieves a 10% throughput increase for 64 KB files compared to four non-split servers and response times that are only slightly less. The same configuration with full delegation improves response times for 64 KB files by a factor of 2.6 over the equivalent non-split system, while achieving 92.5% of its theoretical throughput. For a split system with two CSs and one DS and partial delegation, a 22% improvement in throughput over three non-split servers is obtained for 4 KB files with response times that are close to those of a non-split system.

We also discussed the impacts of splitting. When evaluating the tradeoffs of splitting versus non-splitting, it is necessary to consider the overhead and cost of load balancers and dispatchers, which will result in less throughput and worse response times than the theoretical optimum values we have used for non-split systems. The experimental results appear to indicate that scalable Web

server clusters can be built using one or more split server systems, each consisting of 3-4 servers. The performance of split servers depends on the requested file sizes, and it is beneficial to handle small file sizes at the CS and larger files with partial delegation to DSs. It would be useful to study performance of split server systems in which resource files of different sizes are allocated to different servers to optimize performance. More studies are also needed to evaluate the security benefits of split server clusters, and their scalability and performance with a variety of workloads. While these experiments used bare PC Web servers with no OS or kernel for ease of implementation, HTTP requests splitting can also be implemented in principle on conventional systems with an OS.

REFERENCES

[1] G. Ammons, J. Appayoo, M. Butrico, D. Silva, D. Grove, K. Kawachiva, O. Krieger, B. Rosenburg, E. Hensbergen, R.W. isniewski, "Libra: A Library Operating System for a JVM in a Virtualized Execution Environment," VEE '07: Proceedings of the 3rd International Conference on Virtual Execution Environments, June 2007.

[2] G. Canfora, G. Di Santo, G. Venturi, E. Zimeo and M.V.Zito, "Migrating web application sessions in mobile computing," Proceedings of the 14th International Conference on the World Wide Web, 2005, pp. 1166-1167.

[3] D. R. Engler and M.F. Kaashoek, "Exterminate all operating system abstractions," Fifth Workshop on Hot Topics in operating Systems, USENIX, Orcas Island, WA, May 1995, p. 78.

[4] G. R. Ganger, D. R. Engler, M. F. Kaashoek, H. M. Briceno, R. Hunt and T. Pinckney, **"**Fast and flexible application-level networking on exokernel system," ACM Transactions on Computer Systems (TOCS)**,** Volume 20, Issue 1, pp. 49 – 83, February, 2002.

[5] L. He, R. K. Karne, and A. L. Wijesinha, "The Design and Performance of a Bare PC Web Server," International Journal of Computers and Their Applications, IJCA, Vol. 15, No. 2, June 2008, pp. 100-112.

[6] L. He, R.K. Karne, A.L Wijesinha, and A. Emdadi, "A Study of Bare PC Web Server Performance for Workloads with Dynamic and Static Content," The 11th IEEE International Conference on High Performance Computing and Communications (HPCC-09), Seoul, Korea, June 2009, pp. 494-499.

[7] R. K. Karne, K. V. Jaganathan, and T. Ahmed, "DOSC: Dispersed Operating System Computing," OOPSLA '05, 20th Annual ACM Conference on Object Oriented Programming,Systems, Languages, and Applications, Onward Track, ACM,San Diego, CA, October 2005, pp. 55-61.

[8] R. K. Karne, K. V. Jaganathan, and T. Ahmed, "How to run C++ Applications on a bare PC," SNPD 2005, Proceedings of NPD 2005, 6th ACIS International Conference, IEEE, May 2005, pp. 50-55.

[9] R. K. Karne, "Application-oriented Object Architecture: A Revolutionary Approach," 6th International Conference, HPC Asia 2002 (Poster), Centre for Development of Advanced Computing, Bangalore, Karnataka, India, December 2002.

[10] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, R. Brightwell, "Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing," Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010), April, 2010.

[11] D.S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler and S. Zhou. "Process Migration," ACM Computing Surveys, Vol. 32, Issue 3, September 2000, pp. 241-299.

[12] V. S. Pai, P. Druschel, and Zwaenepoel. "IO-Lite: A Unified I/O Buffering and Caching System," *ACM Transactions on Computer Systems,* Vol.18 (1), ACM, Feb. 2000, pp. 37-66.

[13] K. Sultan, D. Srinivasan, D. Iyer and L. lftod. "Migratory TCP: Highly Available Internet Services using Connection Migration," Proceedings of the 22nd International Conference on Distributed Computing Systems, July 2002.

[14] T. Venton, M. Miller, R. Kalla, and A. Blanchard, "A Linux-based tool for hardware bring up, Linux development, and manufacturing," *IBM Systems J.*, Vol. 44 (2), IBM, NY, 2005, pp. 319-330.

[15] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): the case for a scalable operating system for multicores," ACM SIGOPS Operating Systems Review, Volume 43, Issue 2, pp. 76-85, April 2009.

[16] D. Zagorodnov, K. Marzullo, L. Alvisi and T.C. Bressourd, "Practical and low overhead masking of failures of TCP-based servers," ACM Transactions on Computer Systems, Volume 27, Issue 2, Article 4, May 2009.

[17] http://www.acme.com/software/http_load.

[18] http://www.tinyos.net/.

[19] A. Cohen, S. Rangarajan, and H. Slye, "On the performance of TCP splicing for URL-Aware redirection," Proceedings of USITS'99, The 2nd USENIX Symposium on Internet Technologies & Systems, October 1999.

[20] Y. Jiao and W. Wang, "Design and implementation of load balancing of a distributed-system-based Web server," 3rd International Symposium on Electronic Commerce and Security (ISECS), pp. 337-342, July 2010.

[21] Ciardo, G., A. Riska and E. Smirni. EquiLoad: A Load Balancing Policy for Clustered Web Servers". *Performance Evaluation*, 46(2-3):101-124, 2001.

[22] Sujit Vaidya and Kenneth J.Chritensen , "A Single System Image Server Cluster using Duplicated MAC and IP Addresses," Proceedings of the 26th Annual IEEE conference on Local Computer Network (LCN'01)

[23] B. Rawal, R. Karne, and A. L. Wijesinha. Splitting HTTP Requests on Two Servers, The Third International Conference on Communication Systems and Networks: COMPSNETS 2011, January 2011, Bangalore, India.