ICNA 2012 1569661225

A Split Protocol Technique for Web Server Migration

B. S. Rawal
Department of Computer & Information Sciences
Shaw University
Raleigh, NC

R. K. Karne and A. L. Wijesinha
Department of Computer & Information Sciences
Towson University
Towson, MD

H. Ramcharan
Department of Computer & Information Sciences
Shaw University
Raleigh, NC

S. Liang
Department of Computer & Information Sciences
Towson University
Towson, MD

***Abstract*—Reliability, availability and survivability are important characteristics of Web servers that enable them to provide continual service to clients. Server migration is one of the many approaches used to improve the reliability of Web servers. We propose a novel technique for migrating Web servers that is based on protocol splitting, wherein two servers, a connection server and a data server, function as a single logical server. The technique extends splitting by having the connection server dynamically transfer a TCP connection to another connection server with no client involvement. We describe an implementation of the migration technique and present preliminary performance results using bare PC Web servers in a LAN environment with Linux routers. The connection transfer is achieved by means of an additional inter-server packet and a script to modify a routing table. The results indicate the feasibility of adapting this approach in the future to work with conventional (non-bare) servers on the Internet.**

Keywords—protocol; performance; Web server migration; checkpoint/restart; bare machine computing

I. INTRODUCTION

Protocol splitting enables TCP to be split into its connection and data phases so that these phases are executed on different machines during a single HTTP request [1]. In the basic form of splitting, the state of the TCP connection on the original server is transferred to a data server after receiving the HTTP Get request with no client involvement. The data server then transfers the data to the client, and connection closing can be handled by either the original server or the data server. Many variations on basic TCP/HTTP splitting are possible and have been used to improve Web server performance with delegation [1], split mini-clusters [2], and split architectures [3].

In this paper, we adapt TCP/HTTP splitting to devise a novel technique for Web server migration. It enables an alternate connection server to dynamically take over active TCP connections and pending HTTP requests from the original connection server. Migration is achieved by transferring the complete state of a TCP connection to the alternate server. Migration based on splitting can be used to improve Web server reliability with only a small penalty in performance. Additional benefits of splitting such as data server anonymity and load sharing can also be achieved with this approach to migration.

We implement Web server migration using split bare PC Web servers [1] that run the server applications with no operating system or kernel support. We also conduct preliminary tests to evaluate performance with migration in a test LAN where the split bare PC servers are located on different subnets. Protocol splitting is especially convenient to implement on bare machine computing systems [4] due to their intertwining of protocols and tasks. However, the migration technique based on splitting is general, and can be implemented using conventional servers that require an operating system or kernel to run.

The security and addressing issues that arise due to protocol splitting and migration can be solved in a variety of ways. The simplest solution is to deploy the servers in the same subnet or in the same LAN if host-specific routes are supported. The latter is used for testing migration performance with splitting in this paper. Some alternatives to host-specific routes are also indicated this paper.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes Web server migration, and its design and implementation. Section 4 presents performance measurements. Section 5 discusses the limitations of this work and possible ways to address them. Section 6 contains the conclusion.

II. Related Work

Checkpoint/restart (C/R) techniques [5, 6] are widely employed in practice to achieve fault tolerance. A snapshot of all current processes in an application is captured and used to recover after a failure by reverting to the previous checkpoint. In this case, the entire application needs to be aborted and resubmitted to the job scheduler. The main disadvantages of the approach are the overhead and large queuing delay. Techniques such as job/process migration [7, 8] are used to reduce full checkpoint [9]. Several popular MPI (Message Passing Interface) implementations support process migration and tradeoff its benefits versus the loss in performance. Process migration schemes such as local file system-based migration, PPMR, shared file system-based migration and RDMA-local file system-based migration are well-known. Sample completion times for one migration with PPMR, local approach and RDMA-local are reported to be 3.1 seconds, 33.3 seconds, and 13.5 seconds respectively [10]. In contrast, migration using bare PC servers with splitting in a LAN environment takes less than 20 milliseconds.

In conventional Web server migration, the TCP connection is not split, and client involvement is necessary. For example in Migratory TCP (M-TCP) [11], a client connection is migrated to a new server by adapting TCP on the client and servers. Protocol splitting does not require a load balancer or dispatcher since servers handle the splitting themselves. Splitting also allows a server to handle some of its HTTP requests completely, while delegating the other requests to a data server [1]. Splitting is different from TCP splicing since there is only a single TCP connection that is split between the connection and data servers. Migration based on splitting is not the same as process migration [12] or migrating mobile Web applications [13] since there is no process context involved when the state of a TCP connection is migrated.

The proposed migration technique is implemented using split bare PC Web servers [1]. In a bare PC (or bare machine computing) system, the operating system (OS) or kernel is completely eliminated and application objects (AO) [4] incorporate their own lean network protocol stack and the necessary drivers. Bare PC applications and systems are similar in principle to the many approaches that attempt to reduce operating system overhead in applications such as [14], [15]. However, while these approaches require some form of an operating system or kernel, bare PC applications include the necessary functionality to manage and control system resources and the hardware. Numerous bare PC client and server applications have been developed previously. For example, the design and implementation of bare PC Web and email servers are discussed in [16] and [17] respectively.

III. Migration With Protocol Splitting

A. Overview

Split protocols require a minimum of two servers: a connection server (CS) and a data server (DS). The CS establishes the connection via SYNs and ACKs. When the HTTP Get request is received by the CS, it sends an ACK to the client, and uses an inter-server packet (called a delegate message) DM1 to transfer the TCP state to the DS, which sends the data to the client. In bare PC servers, the TCP state and other attributes of a request are contained in an entry in the TCP table (known as a TCB entry). The CS also handles the TCP ACKs for the data and the connection closing via FINs and ACKs. Typically, the CS has information about the requested file (i.e., its name, size, and other attributes), and the DS has the actual file (the CS may or may not have a copy). When the DS gets DM1, it creates its own TCB entry and starts processing the request. When a DS sends data to the client, it uses the CS's IP. After the CS receives the FIN-ACK, it sends another inter-server packet DM2 to DS. The receipt of DM2 closes the state of the request in the DS. More details of protocol splitting are given in [1]. Each TCB entry is about 168 bytes in size. The essence of splitting is in the transfer of the TCB entry from CS to DS. In this case, communication between the DS and the client is only one-way (from the DS to the client) i.e., the DS does not receive any packets from the client. The DS uses the IP of CS as the source address when sending data to the client. It is assumed that DS is permitted to send packets with the prefix of CS if CS and DS are on different subnets.

For Web server migration, the TCB entry is moved from one CS to another CS (called CS* for convenience), enabling the latter to take over the connection. Migrating server content in this manner and requiring that CS and CS* use the same IP address for two-way communication poses a new challenge: now CS* must be able to send and receive packets with the IP of CS, which may have a different prefix. Furthermore, the client must remain unaware that migration or protocol splitting has occurred. The migration process can be initiated when the current connection server detects that it is going down or has to be taken down. The means by which a server might detect its imminent failure is beyond the scope of this paper.

For Web server migration, the TCB entry is moved from one CS to another CS (called CS* for convenience), enabling the latter to take over the connection. Migrating server content in this manner and requiring that CS and CS* use the same IP address for two-way communication poses a new challenge: now CS* must be able to send and receive packets with the IP of CS, which may have a different prefix. Furthermore, the client must remain unaware that migration or protocol splitting has occurred. The migration process can be initiated when the current connection server detects that it is going down or has to be taken down. The means by which a server might detect its imminent failure is beyond the scope of this paper.

B. Design and Implementation

Before the connection server CS shuts down, it must send all of its pending requests to the alternate server CS*. We assume that CS* is connected to the network, but that it will not process any requests (i.e., it is in stand-by mode). Also, CS and CS* are able to communicate with each other. Prior to the connection transfer, inter-server packets are being sent from CS to the data server DS according to the usual protocol splitting [1] when GET requests arrive. Under large load conditions, it is possible that CS could have many unprocessed requests in its TCP table. In addition to these pending requests, new requests

may still continue to be sent by the client during the time between when CS shuts down and CS* takes over. These requests will be lost unless they are processed later by CS* when the client retransmits them. Before CS shuts down, it also sends a final inter-server packet to CS* to confirm it is shutting down. Only minimal modifications had to be made to the current split server and inter-server packet format to implement migration.

Alternatively, the data server DS can also assume the role of CS* (instead of using a separate CS*) if CS sends its pending requests to DS. If DS has some of its previous data transfer requests still to be processed, it will complete them before it begins to act as CS*. Protocol splitting is designed so that the same server can provide service as a CS and/or a DS, so it is capable of assuming the role of CS* to implement migration.

In our implementation, DS and CS* were on different subnets as seen in Fig. 1, which also shows the IP addresses used. After migration, CS* at IP address 10.55.10.240 is allowed to send and receive packets with the IP address 10.55.12.241 of CS. An ARP broadcast by CS* with this address is used to ensure that Routers 1-3 update their forwarding tables to bind CS's IP address with CS*'s MAC address.

Re-routing of CS's packets to CS* was accomplished by activating a host-specific route to CS's IP after CS shuts down. The Linux script program shown in Fig. 2 was written to effect the route change when it detects that CS is unreachable. It works as follows. The eth0 interface (10.55.10.3) of Router3 constantly monitors the connectivity with CS (10.55.12.241) by sending an ICMP (ping) message till 100% packet loss is seen. Then Router3 deletes the old route to CS via next hop 10.55.10.10 and adds the new route to CS* via its own interface 10.55.10.3 (for the destination IP address 10.55.12.241 of CS now assumed by CS*). During this process, the client is not connected to either server and so its requests are lost until the new route is activated. In our experimental setup and implementation, CS* is on the same subnet 10.55.10/24 as Router3 (but CS, CS* and DS are on different subnets); however, in general, CS* can be located on a different subnet than Router3 provided all routers on the path to CS* run scripts that activate a host-specific route to CS* that will forward packets along this path to CS* (assuming this path is stable). Router3 could send a message to these routers when it detects that CS is unreachable.

Implementation of migration with splitting was done in C/C++. The code sizes are similar to those for splitting since only minimal changes were needed as noted above. A state transition diagram was used to implement lean intertwined versions of HTTP, TCP and other network protocols on the bare PC Web server. The split servers currently run on any Intel-based CPUs that are IA32 compatible. Since they do not use the hard disk, BIOS is used to boot the system. Also, we use a USB to boot, load, and store a bare PC file system (with raw files). No hardware interface code needed to be changed during to handle splitting or migration. The software is placed on the USB and includes the boot program, startup menu, AO executable, and the persistent raw file system (used for resource files). The bootable USB containing this information is generated by a tool (designed and run on MS-Windows), which creates the bootable bare PC application for deployment.

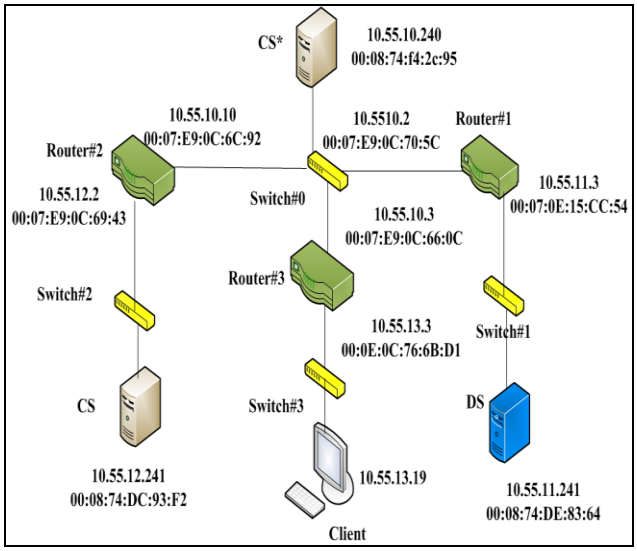

Figure 1. Test network.

IV. Performance Measurements

A. Experimental Setup

The experiments to evaluate server migration with protocol splitting were conducted using the test network shown in Fig. 1 with Linksys 16-port 1 Gbps switches and Linux (Fedora) routers. All machines were Dell Optiplex GX260 PCs each equipped with an Intel Pentium 4, 2.8 GHz Processor, 1GB RAM, and an Intel 1Gbps NIC card on the motherboard. HTTP traffic was generated using the Linux-based http_load stress tool [18] to send up to 1000 requests/sec, and a Web client running on a bare PC to send up to 5700 requests/sec. Although, the split-protocol servers were only implemented on bare PC systems due to the ease of adapting these systems for splitting, OS-based servers could have also been used for this purpose. To verify that the split servers functioned correctly, we also sent requests to these servers using the popular Internet Explorer and Firefox browsers on Windows and Linux respectively.

B. Results

To evaluate the performance of Web server migration, we measured the number of Get requests pending when CS is about to shut down, the time it takes for the CS to migrate those requests to its alternate CS*, the time it takes for these requests to be processed at CS*, the relation between the number of pending requests and the current CS load, and the time needed to complete the migration process. The results are discussed below.

Fig. 3 shows how the number of pending requests increases with the server load when using a 4 KB resource file. There are few pending requests before the server reaches

capacity since the delay in delegating a Get request to the DS is minimal (it only involves a single inter-server packet containing the state of the request stored in the TCB table). For example, it can be seen that at most 20 requests/sec are pending when the request rate is 5000 requests/sec. However, 138 requests/sec are pending when the load increases to 7000 requests/sec. The reason for the large increase is due to the high utilization of the server when the rate exceeds 5000 requests/sec.

Fig. 4 shows that the time spent in migrating the pending requests from one CS to another (in this case CS*) for 4 KB files is between 1-13 milliseconds when the request rate is increased from 1000-7000 requests/sec. This suggests that migration can be achieved within a very small window of time provided the server is not close to saturation.

Figs. 5 and 6 show how the number of pending requests and the time for migration from CS to CS* increase when the requested file size varies from 4K to 128K using a constant request rate of 500 requests/sec. The 128K file size results in a large increase in the number of pending requests. This is because the larger file size requires more processing time resulting in a larger backlog as would be expected. Likewise, although the time for small files sizes is negligible, it increases abruptly to 37 milliseconds for the 128K file size. This is to be expected, since there are now more unprocessed requests and more time is needed to transfer all of them. Since network delays in the testing environment are minimal, the main factors affecting migration time within a LAN are the request rate and the average file size.

```
# monitor routes
while [ ${_unl} -gt 0 ]
do
 if [ $( ping -c 1 ${_ip1} |grep -ic "100% packet loss" ) -gt 0 ];then
   if [ "${_interf}" = "10.55.10.10" ]; then
     if [ ${_flg0} -eq 0 ]; then
      ip r d ${_ip1} via ${_interf}
      if [ "$?" -ne 0 ]; then
        ip r d ${_ip1} via 10.55.10.3
      fi
      ip r a ${_ip1} via 10.55.10.3
      echo "New route added 10.55.10.3"
      _flg0=1
      _flg3=0
      _interf="10.55.10.3"
    fi
     elseif [ "${_interf}" = "10.55.10.3" ]; then
      if [ ${_flg3} -eq 0 ]; then
      ip r d ${_ip1} via ${_interf}
      if [ "$?" -ne 0 ]; then
        ip r d ${_ip1} via 10.55.10.10
      fi
        ip r a ${_ip1} via 10.55.10.10
      echo "New route added to 10.55.10.10"
       _flg3=1
       _flg0=0
       _interf="10.55.10.10"
   fi
  fi
 # avoid the dead loop
  let i=i+1
  if [ $i -eq 3 ]; then
   exit 1
  fi
 else
  sleep 2
 fi
 if [ $j -eq 10 ]; then
  j=0
  echo $j
 else
  echo $j
  let j=j+1
 fi
done
```

Figure 2. Linux script for the IP route change.

For the remaining experiments, 4 KB files were used. Fig. 7 shows that the time to process migrated requests at CS* increases approximately linearly as the number of pending requests is increased. For this experiment, the processing of requests was intentionally delayed in order to determine the effect of increasing the number of pending requests. More studies are needed to verify if such a linear relationship also holds with conventional (non-bare) split servers. Fig. 8 shows the connection time (CT), response time (RT), connection time for migration (CT-M), and response time for migration (RT-M) using split protocol servers (CS and DS) with varying request rates.

Here, CT and RT are the times for normal operation and CT-M and RT-M show the times when migration is performed (the connection time is the total time to process a request, which includes the response time). The results show that connection and response time degradation due to migration is very small for loads of up to 3000 requests/sec. However, the overhead due to migration is larger for higher request rates primarily due to the larger time to migrate requests when more requests are pending and the CPU utilization is higher.

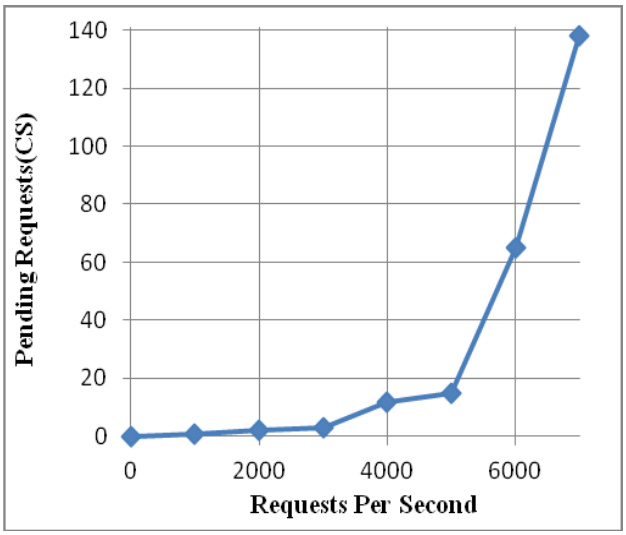

Figure 3. Pending requests (4 KB files).

Fig. 9 shows the number of requests that were lost due to server (CS) migration (because the CS was not able to process all incoming requests from the client during the migration period). It can be seen that between 14-56 requests are lost when the request rate is varied from 500-5000 requests/sec. It is possible to reduce the number of lost requests by splitting requests prior to receiving the Get. Then, once the migration process is complete, such requests can also be handled by CS*. This has not been done in the present implementation.

V. DISCUSSION

Web server migration using HTTP protocol splitting increases server reliability without any client involvement. Furthermore, it provides server anonymity since the data servers and alternate connection servers use only the IP address of the original connection server (not their actual addresses). We only conducted tests to illustrate the basic

migration technique with Web servers, but this approach could also be used for improving the reliability of database servers and file servers. The connection server could also periodically send the state of its connections to an alternate server enabling it to serve as a backup if needed.

The experimental results show that migration using split protocols has only a small delay to transfer requests, a low loss request rate, and low latency to complete requests. More comprehensive tests using conventional (non-bare) servers (with kernel modifications) in a real LAN environment are needed to fully evaluate the performance of migration with splitting. We have tested migration with splitting when the servers are within the same LAN. We used a host-specific route to divert packets to the new connection server. An alternative method is to have Router3 translate the address of packets sent to the IP of CS to the actual IP address of CS* in a manner similar to NAT. While this approach avoids security issues due to incorrect network prefixes, it would have additional overhead due to the need to translate the address of each packet (and to distinguish between translated packets for CS* and packets that are actually CS*'s own packets). Finally, it is possible to assign a second temporary IP address to CS* (that is a valid address in CS*'s network) for it to receive CS's packets, and make router R3 aware of this address.

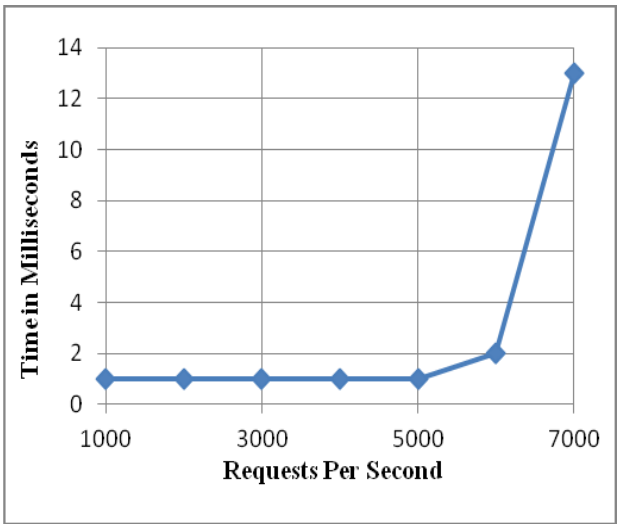

Figure 4. Time to migrate pending requests at CS with 4KB resource file.

VI. CONCLUSION

In this paper, we devised a technique for migrating Web servers based on protocol splitting that does not require client involvement. The technique was implemented using bare PC Web servers in a test LAN environment. The preliminary performance studies demonstrate the feasibility of using this approach for Web server migration, and show that it has only a small performance penalty compared to values reported elsewhere for conventional migration techniques. The migration technique based on protocol splitting was demonstrated using bare PC Web servers, and in a LAN environment. Further studies are needed to explore the alternatives to host-specific routes that could be used with this approach to server migration, and to investigate the security issues associated with migration and splitting protocols.

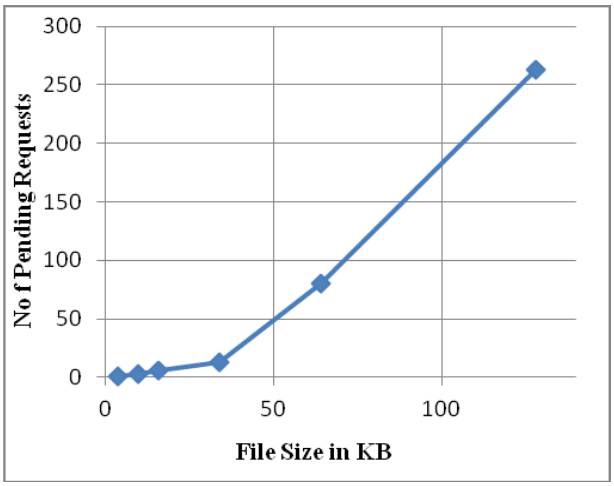

Figure 5. Pending requests at CS with 500 requests/sec load.

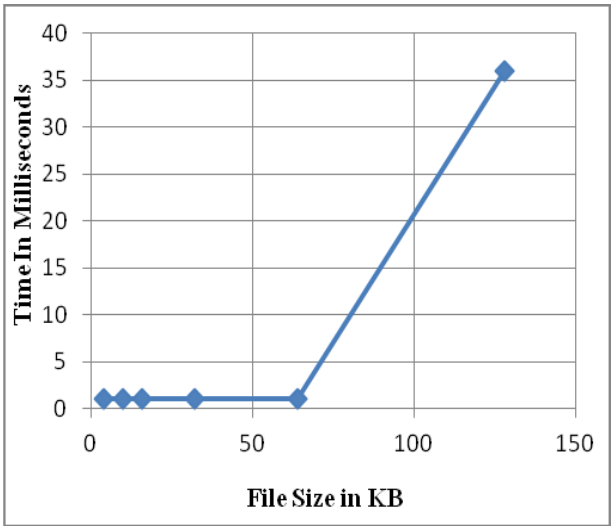

Figure 6. Time to migrate requests (rate: 500 requests/sec).

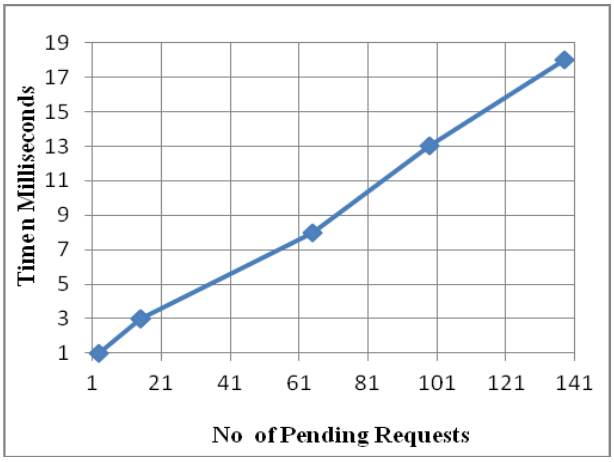

Figure 7. Time taken to process requests at new CS.

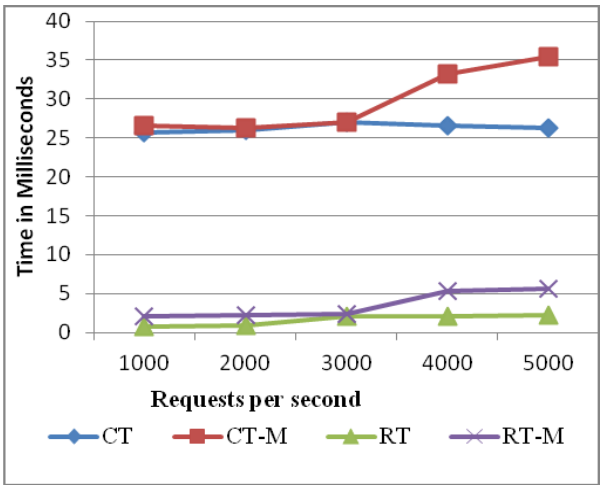

Figure 8. Connection/response times (w/wout migration).

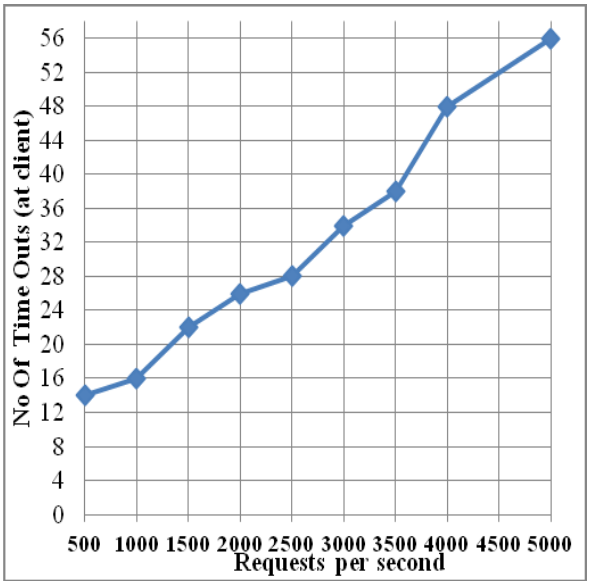

Figure 9. Timeouts relative to total requests.

REFERENCES

[1] B. Rawal, R. Karne, and A. L. Wijesinha, "Splitting HTTP Requests on Two Servers", 3rd International Conference on Communication Systems and Networks (COMSNETS), 2011.

[2] B. Rawal, R. Karne, and A. L. Wijesinha, "Mini Web Server Clusters for HTTP Request Splitting", 13th International Conference on High Performance Computing and Communication (HPCC), 2011.

[3] B. Rawal, R. Karne, and A. L. Wijesinha, "Split Protocol Client/Server Architecture", 17th IEEE Symposium on Computers and Communications (ISCC), 2012.

[4] R. K. Karne, K. V. Jaganathan, T. Ahmed, and N. Rosa, "DOSC: Dispersed Operating System Computing", 20th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA), Onward Track, pp. 55-61, 2005.

[5] E. N. Elnozahy and J. S. Plank, "Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery", IEEE Transactions on Dependable and Secure Computing, 2004.

[6] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent checkpointing under Unix", Usenix Winter Technical Conference, 1995.

[7] S. Chakravorty, C. Mendes, and L. Kale, "Proactive fault tolerance in MPI applications via task migration", International Conference on High Performance Computing, 2006.

[8] J. Hursey, "A Transparent Process Migration Framework for Open MPI", http://www.open-mpi.org/papers/sc-2009/jjhursey-ciscobooth.pdf, Accessed: January 20 2012.

[9] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in HPC environments", ACM/IEEE Conference on Supercomputing, 2008.

[10] X. Ouyang, R. Rajachandrasekar, X. Besseron, D. K. Panda, "High Performance Pipelined Process Migration with RDMA", 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2011.

[11] K. Sultan, D. Srnivasan, D. Iyer and L. lftod. "Migratory TCP: Highly Available Internet Services using Connection Migration", 22nd International Conference on Distributed Computing Systems, 2002.

[12] D.S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler and S. Zhou. "Process Migration", ACM Computation Surveys, vol. 32 (3), pp. 241-299, 2000.

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

[14] J. Lange et al., "Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing", 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2010.

[15] 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), pp. 37-66, 2000.

[16] L. He, R. K. Karne, and A. L. Wijesinha, "Design and Performance of a bare PC Web Server", International Journal of Computer and Applications, vol. 15, pp. 100-112, 2008.

[17] G.H. Ford, R.K. Karne, A.L. Wijesinha, and P. Appiah-Kubi, "The Performance of a Bare Machine Email Server", 21st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 143-150, 2009.

[18] http_load, http://www.acme.com/software/http_load. Accessed: January 20, 2012.