CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2014)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3239

SPECIAL ISSUE PAPER

Characterizing the scalability of a Web application on a multi-core server

Raoufehsadat Hashemian[1], Diwakar Krishnamurthy[1,*,†], Martin Arlitt[2] and Niklas Carlsson[3]

[1]*Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB, Canada*
[2]*HP Labs, Palo alto, CA, USA*
[3]*Department of Computer and Information Science, Linköping University, Linköping, Sweden*

SUMMARY

The advent of multi-core technology motivates new studies to understand how efficiently Web servers utilize such hardware. This paper presents a detailed performance study of a Web server application deployed on a modern eight-core server. Our study shows that default Web server configurations result in poor scalability with increasing core counts. We study two different types of workloads, namely, a workload with intense TCP/IP related OS activity and the SPECweb2009 Support workload with more application-level processing. We observe that the scaling behaviour is markedly different for these workloads, mainly because of the difference in the performance of static and dynamic requests. While static requests perform poorly when moving from using one socket to both sockets in the system, the converse is true for dynamic requests. We show that, contrary to what was suggested by previous work, Web server scalability improvement policies need to be adapted based on the type of workload experienced by the server. The results of our experiments reveal that with workload-specific Web server configuration strategies, a multi-core server can be utilized up to 80% while still serving requests without significant queuing delays; utilizations beyond 90% are also possible, while still serving requests with 'acceptable' response times. Copyright © 2014 John Wiley & Sons, Ltd.

Received 10 January 2014; Accepted 22 January 2014

KEY WORDS: experimental performance characterization; Web server scalability; multi-core architecture

1. INTRODUCTION

As organizations increasingly use Web-based services to support their customers and employees, Quality of Service (QoS), typically measured by the response times that the users experience, becomes an important design consideration. At the same time, large-scale service providers are interested in improving the effective utilization of their infrastructure, as this improves the economic sustainability of their business. For such companies, even a 10% increase in the effective utilization of their hardware results in 10% reduction in the cost of expanding hardware. Therefore, systematic techniques are required to help organizations meet the QoS objectives of their Web customers while effectively utilizing underlying resources. One such technique is performance evaluation. While this technique was frequently used in the 1990s and early 2000s [1–4], such studies predate the emergence of multi-core server architectures. Thus, revisiting Web server performance and scalability is crucial to see whether previously investigated behaviours exist on modern hardware.

*Correspondence to: Diwakar Krishnamurthy, Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB, Canada.
†E-mail: dkrishna@ucalgary.ca

In this paper, we present a detailed measurement-based study of the performance behaviour of the Lighttpd [5] Web server coupled with a PHP-FastCGI module deployed on a two socket, four cores per socket system based on the Intel Nehalem [6] microarchitecture. We observe system behaviour under two synthetic yet realistic workloads, namely, a TCP/IP intensive workload and the SPECweb2009 Support workload. The TCP/IP intensive workload emulates a server that handles a large volume of user requests to cached static content. Conversely, the SPECweb2009 Support workload contains a significant fraction of dynamic requests, which trigger application-level CPU activity. In contrast to previous work [7, 8] that consider only mean response time measures, we focus on characterizing the response time distributions for these workloads while enabling progressively more cores on the system. Our experiments with these workloads show that the task of leveraging the performance benefits of multi-core systems to be non-trivial. After deploying the Web server on this system and with the default configurations, the servers scaled poorly under both workloads. To eliminate bottlenecks other than the CPU and allow the Web server to scale, we first perform three initial configuration tunings related to the Web server software, network interrupt processing and Operating System scheduling. Together, we find that these tunings allow the system to scale up to 69% relative to the default settings.

Tests on the tuned system reveal that the two workloads we consider scale very differently. Specifically, the TCP/IP intensive workload scales sub-linearly with increasing core count. For example, considering a 99.9th percentile response time target of 4 ms, request throughput increases by a factor of 7.2 from one to eight cores. For a mean response time target of 0.5 ms, throughput only increases by a factor of 6.3 from one to eight cores. In contrast, throughput increases nearly linearly with core count for the SPECweb2009 Support workload for various mean and 99.9th response time percentile targets.

Deeper analysis of response time distributions shows that differences in performance behaviour across the two workloads stem from the way static and dynamic requests exploit multi-core systems. Response times of static requests are affected adversely by overheads related to migration of data belonging to the Web application's processes across cores. In particular, response time degradations are significant when data migrates across the two sockets in the system. For the more memory intensive dynamic requests, the adverse impact of process data migration across sockets due to the use of the cache hierarchy of both sockets is offset by an overall reduction in the number of accesses to main memory.

Due to this workload-dependent behaviour, our results show that a Web server configuration strategy previously proposed by Gaud *et al.* [8] and Hashemian [9] to improve scalability is not likely to be effective in systems where the mix of static to dynamic requests fluctuates over time. Specifically, this strategy relies on deploying multiple Web server replicas on a multi-core host. Web server process and network interface card (NIC) affinity settings are configured to distribute requests to replicas such that application-level processing of a request is carried out on the same socket that handles the network interrupt processing of that request. Through experiments that use a modified SPECweb Support workload to control the fraction of dynamic requests, we show that the multiple replica approach significantly reduces inter-socket data migrations and improves scalability for workloads where static requests dominate. However, this configuration decreases scalability relative to the single replica configuration for workloads with more dynamic content. These results suggest that significant scalability improvements can be achieved by dynamically adapting Web server configuration policies at runtime based on the type of workload experienced by the system.

The root causes of the performance degradation for dynamic requests with the multiple replica approach are identified through detailed analysis of hardware event statistics. The analysis reveals that the contention in the last level cache (LLC), caused by the memory intensive PHP-FastCGI processes leads to increased memory accesses with the multiple replica configuration. The increase in memory accesses inflates the response time for dynamic requests.

Preliminary version of our work appeared as a 12-page conference paper [10]. This work significantly extends our earlier work. In addition to expanding our experimental description (for reproducibility purposes), expanding our discussions to include deeper insights and system observations as well as provide a more thorough investigation of the related literature in which scalability of multi-core servers was improved using more than one application instance, we added two

DOI: 10.1002/cpe

significant experimental sections to strengthen the work. First, we present a comprehensive set of new experiments to evaluate the impact of workload mix on the scalability effect of the multiple replica approach (Section 6.4). Second, using detailed hardware metrics, we present a root cause analysis of the bottlenecks and investigate how these bottlenecks impact the scalability (Section 7).

Summarizing, the main contributions of this paper are as follows:

- Our paper provides insights on how configuration tuning of a Web server leads to efficiently leveraging the capacity of multi-core hardware. With a carefully selected, workload-specific configuration strategy, our results show that a modern multi-core server can be kept 80% busy while still serving requests without significant queuing delays. Furthermore, utilizations beyond 90% are also possible, while still serving requests with 'acceptable' response times. A recent analysis of traces collected from an enterprise and a university suggests that the top 10 domains in the traces account for a vast majority of observed traffic [11]. Yet, evidence suggests that production servers in these large-scale service provider domains are intentionally very lightly utilized [12] in an attempt to ensure good response time performance. We believe such a strategy is inefficient and needs to be revisited in light of the evidence presented in this paper.
- We show that the previously proposed multiple replica scalability improvement approach is not always effective. Through controlled experimentation, we show that this approach can *degrade* performance for workloads that have a significant fraction of dynamic requests. These results motivate the need for workload-specific scalability improvement techniques.
- We present a detailed analysis of hardware performance counter data that sheds light on how low-level processor bottlenecks impact performance and how they shift with changing workload characteristics. In particular, we show that LLC misses in the multiple replica configuration increase as the fraction of dynamic requests in a workload increases. The increased cache misses trigger expensive memory accesses that cause degradation in end user response times.

The remainder of the paper is organized as follows. Section 2 reviews previous work. Section 3 describes the experimental setup, methodology, workloads and our servers configuration with multiple replicas. Section 4 describes the initial Web server configuration tunings carried out to better exploit the parallelism of multi-core servers. Section 5 studies the performance and scalability of the tuned server under both workloads. Section 6 presents our scalability evaluation with multiple replica configurations. Section 7 identifies the root causes of the observed performance behaviour. Section 8 summarizes our work and while Section 9 concludes the paper.

2. BACKGROUND AND RELATED WORK

The increased availability of multi-core hardware prompted researchers to devise novel Operating System and application-level enhancements to improve the performance of systems deploying this technology. For example, Boyd *et al.* [13] modified the Linux kernel to remove resource bottlenecks that prevent applications from fully exploiting multi-core hardware. Their study shows how various applications, including the Apache Web server, benefit from these kernel modifications. Kumar *et al.* [14] characterized the performance of a TCP/IP intensive workload generated on an Intel Core 2, two sockets, four cores per socket system whose network stack is modified to support Direct Cache Access (DCA). They aimed to improve response times by allowing a NIC to directly place data into processor caches. The authors show that a DCA-enhanced Linux kernel can perform 32% faster than a stock kernel. In contrast to these studies, we focus on simple performance improvement strategies that do not require application and kernel modifications.

Several studies have focused on bottlenecks arising out of multiple cores sharing common resources such as caches, system buses and memory controllers [7, 15]. Scheduling and resource management techniques have been proposed to alleviate such bottlenecks thereby enhancing system scalability [16–20]. For example, recent work by Jaleel *et al.* [21] studies the performance of the cache hierarchy when multiple applications execute on multi-core processors. The authors propose a scheduling policy called CRUISE that leverages knowledge about processor cache replacement

DOI: 10.1002/cpe

policies to determine how best to co-schedule these applications. Using an x86 trace driven simulator and the SPEC CPU benchmark, the authors find that their approach results in a near-optimal performance for the majority of application scenarios studied. While our work considers the impact of shared bottlenecks, we focus on Web servers and employ a measurement-based approach.

Performance issues of Web servers running on traditional single core architectures have been studied comprehensively in the past [22, 23]. However, only a few previous studies have focused on Web server performance on multi-core systems. Among them is a study by Veal and Foong [7]. They conducted a performance evaluation of the Apache Web server deployed on a centralized memory Intel Clovertown system. Their work reports that the scalability of the Web server for a SPEC Web workload increases by only a factor of 4.8 from one to eight cores. The authors establish the system's address bus as the main scalability bottleneck. We did not encounter such a bottleneck in this work as we used newer hardware that employs on-chip memory controllers, a NUMA architecture and faster inter-socket communication mediums.

Harji *et al.* [24] compare the performance of various Web server software deployed on a system with a single quad-core socket. Using two different static workloads, the authors show that their μserver and WatPipe implementations specifically optimized for multi-core systems outperform well-known software such as Apache and Lighttpd. They show that with careful tuning, the new implementations can sustain up to 6000 Mbps using four cores in the socket. In contrast to this work, our paper considers both static and dynamic workloads and investigates deployments spanning multiple sockets. Furthermore, we focus on response time measures in addition to throughput and provide a detailed characterization of low-level hardware usage triggered by Web request processing.

You and Zhao [25] studied a group of request scheduling algorithms for a Web server executing on multi-core hardware. They propose a scheduling approach that considers the distribution of service times of dynamic requests. The service time distribution is used to implement a Weighted Fair Queuing system to schedule requests. Similar to our work, this work explores the impact of affinity based scheduling policies where a Web process is constrained to execute only on a specific subset of processor cores. However, in contrast to our measurement-based approach the authors use simulations to verify the effectiveness of their scheduling mechanisms.

The idea of running multiple instances of applications or middleware on multi-core systems to improve performance has been previously examined for various applications including Web servers. For instance, running multiple JVM instances on multi-socket multi-core systems was shown to result in better performance [26, 27] than running a single instance. Salomie *et al.* [28] showed that on multi-core hardware, executing multiple replicated database instances yields better performance than executing a single instance. For Web applications, Boyd *et al.* [13] conclude that executing one instance of Apache per core with each instance listening on a distinct accept socket can result in significant performance gains. They claim that a single instance of Apache scales very poorly on multi-core hardware because of contention on a mutex protecting the single accept socket. Gaud *et al.* [8] evaluated the performance of the Apache Web server on a four-socket, quad-core AMD Shanghai system using the SPECweb2005 benchmark. Similar to the other studies, the authors show that a multiple Web replica solution with one replica per socket improves performance by minimizing migration of Web server data across sockets.

In contrast to these studies, we present a more fine-grained analysis of Web server performance that captures workload-specific behaviours. We run a set of controlled experiments and find that the multiple Web replica solution is *not* always effective. By submitting a modified version of SPECweb Support workload mix, we show that in our system, the approach is harmful for workloads containing a significant fraction of dynamic requests. Hence, running multiple instances of Web application cannot be prescribed as a general scalability remedy to Web site administrators. Furthermore, unlike the other multi-core Web server studies, our work focuses on the entire response time distribution, which is typically of more interest to Web service providers than mean performance measures. Finally, our paper employs a comprehensive analysis of hardware performance counters to provide new insights on how low-level processor bottlenecks shift in response to changes in Web workload characteristics.

DOI: 10.1002/cpe

3. EXPERIMENTAL SETUP

In this section, we first explain the multi-core server, which hosts our Web application. After having described the experimental methodology, scalability metrics, monitoring tools and our workloads, the remaining sections explain the multiple replica approach for scalability enhancements. We first describe how the server can benefit from this configuration in Section 3.6. Next, the details of multiple replica deployment of Web server in our system are provided in Section 3.7.

3.1. Server under study

The multi-core server we used for this study is a Dell PowerEdge R510 equipped with two quad-core Intel Xeon 5620 processors with the Intel Nehalem architecture. The processors operate at a frequency of 2.4 GHz. They have a three-level cache hierarchy as shown in Figure 1. Each core in a socket has private 64 KB L1 and 256 KB L2 caches. Cores in a socket share a common 12 MB L3 as the LLC. The machine is equipped with two 8 GB memory banks, each connected to one socket through three DDR3-1333MHz channels. The server uses Intel's QuickPath Inter-connect (QPI) with a capacity of 5.6 Giga Transfers per second (GT/s) between sockets. The server has a dual-port 10 Gbps Broadcom 57711 NIC and a dual-port 1 Gbps Intel 82576 NIC. The NICs support Message Signal Interrupts-eXtended (MSI-X) [29] and multiple Receive Side Scaling (RSS) [30] queues that enable the distribution of network interrupts for received packets between the cores.

In this study, we focus on CPU bottlenecks and their impact on scalability. Consequently, we intentionally prevent other resources (e.g. disks and network) from becoming the bottleneck. This influences our choice of the Web server and workloads used in our study. The Web server software used in our experiments is Lighttpd 1.4.28 [5]. Lighttpd was chosen as initial tests showed it scaled better than other open source Web servers such as Apache [31]. The event-driven, asynchronous architecture of Lighttpd avoids the need to manage a large number of Web server processes and decreases the size of its memory footprint. This enables the server to experience a CPU bottleneck and allows for system performance under such a condition to be examined. Lighttpd can exploit multi-core hardware by spawning multiple worker processes. The fork() mechanism is used to create these worker processes; therefore, they share the same address space and share some data structures in this space. As we discuss in Section 4, our experiments used 1 worker process per core.

For the multi-tier SPECweb Support workload [32], we use the PHP language runtime as the application tier and the FastCGI [33] message exchange protocol as the means of communication between the Web and application servers. In this configuration, a PHP-FastCGI process with a set of children is spawned by the Lighttpd Web server at startup. This creates a constant size pool of PHP-FastCGI processes that serve requests for dynamically generated content. We had to use 128 PHP-FastCGI processes per core in our experiments to avoid a software bottleneck.

The SPECweb benchmark emulates the behaviour of a database tier using another Web server referred to as the Backend Simulator (BeSim). We use two machines with Intel Core 2 processors

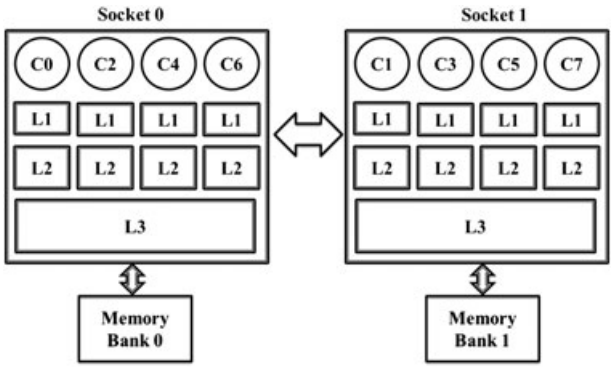


Figure 1. Servers architecture.

DOI: 10.1002/cpe

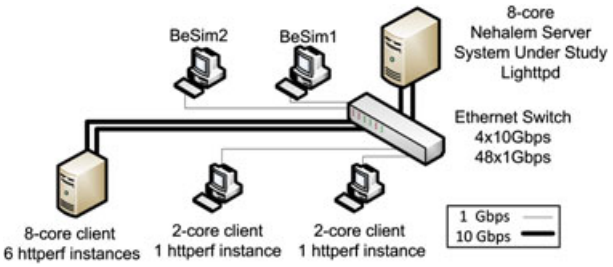


Figure 2. Testbed for Support workload.

and 2 GB of memory for this purpose, as shown in Figure 2. Each BeSim machine executes an instance of an Apache Web server with the FastCGI module. The BeSim systems were carefully monitored to confirm that they are not the bottleneck in our experiments.

3.2. Methodology

Our experimental methodology consists of the following four main steps:

1. *Tuning the system under study to eliminate software and Operating System bottlenecks*: In this step, we explore various configuration parameters related to the Web server and the Operating System. This allowed us to utilize the available processing resources close to their maximum capacity. Specifically, tuning the system allowed us to utilize the server to more than 90% and equally balance the load between the cores. We describe the tuning process in Section 4.
2. *Evaluating the scalability of the Web server with respect to core count*: The experiments in this step are performed to address three main goals. First, we aim to measure how scalability changes as we increase the number of cores. Second, we want to examine how scalability with respect to core count changes as we change the workload. Finally, we want to see how the response time distribution changes for different types of requests in the workload. These experiments are performed when one, two, four and eight cores are active in our eight-core server. We run the experiments for TCP/IP intensive and SPECweb Support workloads. The results is described in Section 5. For each workload, the workload intensity is varied to cover a range of CPU utilizations between 10% and 95%. For the SPECweb Support workload, we focus on response time distributions for static and dynamic requests. The results of experiments in this step are presented in Section 5.
3. *Evaluating the scalability of the Web server with two replicas*: In this step, the goal is to measure the effect of running multiple instances of the Web application on scalability. We reconfigure the system under study by deploying two instances of our Web application, as we explain in Section 3.6. These experiments are run when all the cores are active in the server. Similar to step 2, we run the experiment for both workloads and cover a range of CPU utilizations between 10% and 95%. We characterize scalability and evaluate the effect of the multiple replica approach on the response time distribution. We collect hardware event statistics during the experiments, as we describe in Section 3.4. The collected data are later used to identify low-level performance bottlenecks that affect performance. The results of the experiments in this step are presented in Section 6.
4. *Analysing the scalability behaviour of the Web server for static and dynamic requests*: The goal in this step is to identify the reason for the different scalability behaviour observed for static and dynamic requests in steps 2 and 3. We perform a new set of experiments with a third workload generated by altering the SPECweb Support workload. This workload allows us to vary the fraction of dynamic requests in the Support workload as we explain in Section 3.5. Hardware event statistics are also collected in this step to understand differences in the way these two types of requests exercise processor resources. The results of these experiments are presented in Section 7.

DOI: 10.1002/cpe

3.3. Scalability metric

The primary scalability metric that we use in this study is the *Maximum Achievable Throughput* (MAT). We define this parameter as the maximum throughput at which a selected statistical property of the response time distribution is less than a 'threshold'. MAT is measured for two typical statistical metrics used in practice by performance analysts, namely, the mean and the 99.9th percentile of the response time distribution. The mean (average) response time is often used by queuing theorists, whereas practitioners and Internet service providers often are interested in percentiles; particularly, the upper percentiles that capture the tail effects [34]. The response time thresholds differ depending on the Service Level Agreement for a particular Web service. For instance, Amazon Dynamo Services use an Service Level Agreement where the 99.9th percentile threshold is 300 ms [34]. In our experiments, the mean response time thresholds are set to about four to five times the mean response time under low load. This represents a scenario where there is contention among users for server resources yet users experience an acceptable interactive response. The threshold values are constant for all the experiments; however, the effect of selecting other thresholds is also discussed in Section 5.

3.4. Monitoring tools

To quantify the server's behaviour, a variety of metrics are collected. To monitor the CPU, disk and network utilizations of the server, the *collectl* [35] tool is used, which we confirmed incurred a negligible CPU overhead of around 1%. Low-level hardware monitoring data were collected using the Performance Counter Monitor (PCM) [36] tool (specific to Intel architectures). To eliminate the effect of the hardware monitoring overhead on the scalability measurements, all the experiments were run twice, first without PCM to collect the scalability metrics, and then with PCM to collect the hardware event statistics. The experiments without PCM were repeated multiple times to achieve tight 95% confidence intervals for the reported mean response times. In all of our experiments, 95% confidence intervals were less than 1% of their corresponding mean values.

3.5. Workloads

In this section, we describe the three different workloads we used in this experimental study, namely, TCP/IP intensive, SPECweb Support and Modified Support workloads.

3.5.1. TCP/IP intensive workload. As mentioned previously, the TCP/IP intensive workload is selected to expose CPU bottlenecks when a server is subjected to very high traffic volumes that induce significant Operating System activity related to processing of incoming and outgoing network packets. In each experiment, HTTP requests are sent to the server at a rate of λ requests per second. The value of λ is varied to study the behaviour of the server under a wide range of traffic intensities. To isolate the impact of TCP/IP activity, we limit the amount of application-level processing required by each request. Accordingly, all user requests are for a single static 1 KB file, which is highly likely to be served from the processors' caches. With this workload, we were able to carry out experiments up to a maximum rate of 155 000 requests per second, which resulted in a 90% average utilization of the server's cores. To generate such high request rates, we selected the *httperf* [37] Web workload generator. We used four instances of *httperf* to generate this workload.

3.5.2. SPECweb Support workload. We use SPECweb2009 Support workload to observe system behaviour when there is more application-level activity relative to operating system level activity. The increased application-level activity is caused by a more diverse working set of static files, which may trigger processor cache misses. Furthermore, around 5% of requests are dynamically generated and rely on the services of the application tier and the BeSim tier. To prevent the server's disk from becoming a bottleneck, we change the "DIRSCALING" parameter of the Support workload from 0.25 to 0.00625. This reduces the size of the file-set to fit in the server's main memory; however, the probability distributions of file sizes and file popularity remain the same. To ensure that all files are accessed in memory and prevent disk activity, before each experiment, the memory cache

DOI: 10.1002/cpe

(including the Operating System file cache) is cleared and then warmed by accessing all files in a 'cache filling' run.

To submit the Support workload, we use *httperf* instead of the Java-based SPECweb load generator client bundled with the benchmark suite. We previously enhanced *httperf* to submit a workload into multiple server addresses simultaneously [9]. This enhancement is used in the experiments to evaluate the multiple replica configuration, which is described in Section 6. We also modified *httperf* to obtain more detailed statistics about the request response times [38]. Eight instances of *httperf* are deployed across two dual core and one eight-core machines, as shown in Figure 2. The eight-core server is connected to the server through 2×10 Gbps links and through an Ethernet switch with four, 10 Gbps ports. The dual core machines are also connected to the server each with 2×1 Gbps links and through the same Ethernet switch. We first instrumented the SPECweb client to record all request URLs sent to the server along with the send time stamp in an output file. A set of tests were run using multiple Java-based SPECweb clients, and the lists of request URLs were recorded. In each experiment, depending on the number of concurrent users for that experiment, a subset of the recorded sessions is submitted to the server using *httperf*. We verified that the characteristics of the load generated by *httperf* is very close to that generated by the SPECweb client.

3.5.3. Modified Support workload. We also generate a set of synthetic workloads by modifying the SPECweb Support workload to investigate the individual effect of the multiple replica approach on the response time of static and dynamic requests. For each workload, we replace a subset of dynamic requests in the Support workload with a static request for a file that has the same size as the reply to the original dynamic request. We refer to this set of mixes as the Modified Support workload. The fraction of dynamic requests that are replaced by the static one is considered as the variable factor in our experiments. We refer to this factor as 'Replaced Fraction' (RF). The values assigned to this variable are 0.2, 0.4 and 0.6. As the value of RF increases, the number of dynamic requests decreases, and as a result, the load on the PHP-FastCGI processes decreases. These characteristics of the Modified Support workload are exploited in Section 7 to identify the reason for the different behaviour of static and dynamic requests.

For all the experiments, we closely monitored the clients and the network to verify that they were not the bottleneck.

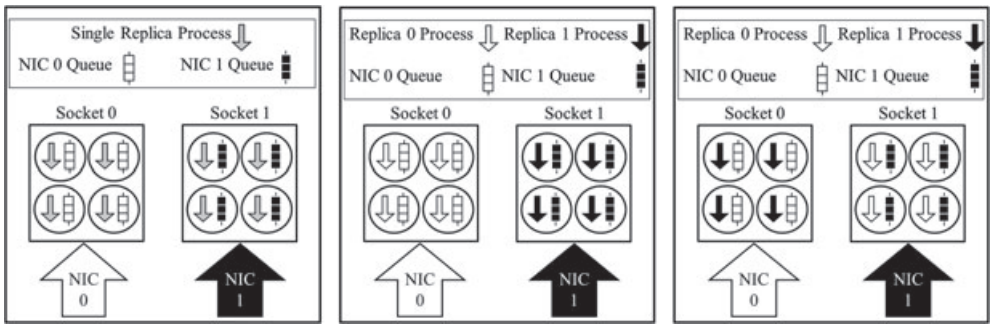
3.6. Multiple Website replicas

As mentioned earlier, one way to improve the scalability of applications running on multi-core hardware with more than one socket is to deploy multiple instances of that application, in particular, one instance per socket. This approach is expected to improve the performance of the application by reducing contention for shared resources such as the LLC, memory controller and socket inter-connect. In this study, we focus on Web servers running on multi-core hardware and how replication can affect the performance of Web applications. We refer to this approach as 'multiple Website replicas'. This section discusses the potential effect of the multiple Website replica approach on a system's performance and explains the two multiple Website replica deployments we used in our experimental setup.

One of the potential benefits of deploying multiple Website replicas is reducing contention in the socket inter-connect. When there is only one instance of the Web server application deployed on the multi-core hardware, data shared between processes residing on different sockets should migrate through the socket inter-connect media. Higher traffic volumes in the socket inter-connect media may result in contention and poor performance at higher loads. However, with multiple instances, each running on the cores of one socket, data sharing will be eliminated, leading to reduced traffic in the socket inter-connect. The type of data shared between the processes and the amount of traffic it generates in the socket inter-connect can potentially affect the performance of deploying multiple Website replicas. The following are the two different types of data that can be shared between the processes in a Web server:

Network packet processing: Consider a multi-core Web server responding to HTTP requests. When the TCP packets carrying an HTTP request arrive at the server, the NIC sends an interrupt to

DOI: 10.1002/cpe



(a) (b) (c)

Figure 3. Network interface card (NIC) queue and Web server process mappings: (a) original, (b) direct mapping and (c) indirect mapping.

one of the processor cores. The packet is then copied to the private cache of the core that is going to handle the interrupt. After the packet is processed by the core (e.g. TCP/IP protocol processing such as checksum calculation), a Web server process is chosen to serve the HTTP request. The data within the packet are then copied to the process's buffer space. There are three possibilities in this stage. First, the core on which the Web server process resides is the same core that processed the packet interrupt. In this case, the data already exist in the private cache of that core. Second, the core on which the Web server process resides and the packet processing core are not the same but are from the same socket. In this case, the data may exist in the shared LLC of that socket. Third, the Web server process residing core and the packet processing cores are from different sockets. In this case, the data should be fetched from the cache hierarchy of the other socket [15]. This causes an increase in the socket inter-connect traffic, which can result in contention in the inter-connect links at high request rates.

Shared application data: Both Web and application tier processes can share data, for example, kernel libraries and associated data structures, between themselves. A process residing on a given socket may request a given piece of data cached at the LLC of the other socket due to a previous access by another process residing on that socket. This can have a positive impact on performance because it reduces costly accesses to main memory and reduces the traffic in the memory controller. However, it triggers migration of shared data across sockets that may cause heightened contention for the socket inter-connect links. Hence, the overall performance impact of shared application data depends on which of these two effects dominates.

3.7. Multiple replicas deployment

In our setup, the multiple Website replicas solution takes advantage of the Linux *taskset* and *IRQ Affinity* mechanisms, multiple network interfaces available on modern servers, and their support for MSI-X and RSS to eliminate inter-socket communications stemming from the migration of network packets and data shared among Web application processes residing on different sockets. In the following subsections, we describe how we setup the non-replica approach and two variants of the multiple replica approach on our server.

3.7.1. Original configuration. Figure 3(a) shows the schematic of the network and Web server configuration for the server used in our setup. The server has a dual-port NIC that can operate as two individual NICs. Each NIC has its own IP address and is configured to use four RSS queues. Queues 0 to 3 of NIC 0 are assigned to cores 0 to 3 of socket 0 and queues 0 to 3 of NIC 1 are assigned to cores 0 to 3 of socket 1[‡].

[‡]There are other options for configuring the two NICs, such as assigning the same IP address through NIC teaming and enabling eight RSS queues for each NIC. We experimentally evaluated these configurations and did not observe a considerable performance difference for our workloads.

DOI: 10.1002/cpe

The original Web server configuration used in the experiments with eight active cores involves running a single Web server software instance with eight Lighttpd processes. Each of these processes is bound to one of the eight active cores as shown in Figure 3(a). When a request arrives at the server, it will first be processed by one of the queues of the NIC, and then, the HTTP request is handled by one of the Web server processes. The NIC and the Web server process to handle the request are each chosen with equal probabilities. Therefore, when the TCP packets of a request are processed by the cores of one socket, there is a 50% chance that the HTTP request is processed by a process residing in the other socket[§]. As mentioned previously, there are 128 FastCGI processes per core. These are not shown in the figure for the sake of clarity.

3.7.2. 2-replica direct mapping. An alternative configuration is to run two instances of the Web server software, each having four Lighttpd processes and 4×128 FastCGI processes bound to the cores of one socket. Each instance is denoted as a Website '*replica*'. The processes of replica 0 are bound to the IP address of NIC 0. Consequently, the requests coming through NIC 0 are processed by the processes of replica 0. The same setting is performed on replica 1 and NIC 1. Figure 3(b) shows the schematic of the alternative configuration with two instances of the Web server. From the figure, the processes of replica 0 reside in socket 0. The queues of NIC 0 are also assigned to the cores of socket 0. This ensures that for all the requests sent through NIC 0, both the TCP packets and the HTTP requests are processed by cores of socket 0. This means the TCP packets and HTTP request will be processed on the same socket with 100% probability. We refer to this mapping of NIC queues and replica processes as *2-replica direct* mapping.

Employing the *2-replica direct* mapping eliminates costly inter-socket communications involving migration of network packets. Furthermore, because both replicas are independent, significant sharing of data among Web application processes will not trigger inter-connect traffic. However, in contrast to the *Original* configuration, any given Web application process only has access to one of the two 12 MB L3 caches. It follows from these discussions that the multiple replicas approach tries to leverage properties of the single socket scenarios while still using both sockets in the system.

3.7.3. 2-replica indirect mapping. Recalling from Section 3.6, the multiple replica approach can reduce traffic in the socket inter-connect. The two main sources of reduced traffic are local network packet processing and reduced sharing of application data between the two sockets. To better understand the selective importance of the two sources of inter-connect traffic reduction, a third configuration is considered. This is carried out by configuring replica 1 on socket 1 to process the requests sent through NIC 0 (on socket 0). Thus, when the TCP packets of a request are processed by the cores of one socket, the HTTP request is processed by a Web server process residing in the opposite socket. This mapping of NICs and replicas, shown in Figure 3(c), is referred to as *2-replica indirect* mapping. Employing the *2-replica indirect* mapping forces the occurrence of costly inter-socket data migrations due to the network (TCP/IP) processing. However, as Web application processes within a replica still share the same socket, this mapping is similar to the *2-replica direct* mapping in terms of avoiding shared application data related socket inter-connect traffic. While we considered this configuration to help us quantify the impact of reduced inter-connect traffic, it is not a configuration we would use in practice.

4. INITIAL CONFIGURATION TUNING

In this section, we describe our initial configuration tuning exercise aimed at preventing software or Operating System related bottlenecks from limiting the scalability of the server. Table I shows the quantitative values of performance improvements achieved through each of the three configuration tunings for both the TCP/IP intensive and the SPECweb Support workloads. The first column describes the configuration tuning, while the other two columns show the improvement in the MAT for mean response time thresholds of 0.5 and 5.0 ms, respectively, for the TCP/IP intensive and

[§]Our workload generator sends requests to the IP addresses of both NICs of the server with equal probabilities.

DOI: 10.1002/cpe

Table I. Effect of configuration tunings on the server's scalability.

	Scalability improvement	
Tuning method	TCP/IP intensive workload (%)	Support workload (%)
Decreasing number of Lighttpd processes from 2 to 1	14.0	1.5
Employing core affinity	10.0	9.0
Distributing interrupt handling load	45.0	20.5

the SPECweb Support workloads. We confirmed that the tuning strategy's effect on the higher percentiles is also positive. The three tuning steps are performed at the Web application level by setting up the number of Web server processes and in the Operating System level for process scheduling options and network interrupt handling, as described later. Overall, these changes improved the MAT by 69% and 31% for the TCP/IP Intensive and Support workloads, respectively, relative to the default Operating System settings. The experiments presented in the next section use these tuned settings.

4.1. Number of web server processes

The Lighttpd documentation suggests a setting of two HTTP worker processes per core for CPU bound workloads [39]. The main advantage of running two worker processes per core is that while one worker is waiting for resources other than CPU (e.g. disk, network or memory) to process a request, the second worker can use the core to serve another request. However, for less memory or disk intensive workloads, the waiting time for other resources can be small. For these workloads, the overhead of context switching between the two processes is greater than the gain achieved through duplicating workers. As a result, the single worker per core may outperform the two workers per core setting for these types of workloads. As the first configuration tuning step, we set this parameter to one and compare the results with the default two process per core configuration. From Table I, the MAT value was 14% and 1.5% *higher* with one process for the TCP/IP intensive and the Support workloads, respectively, when compared to the default setting. This result indicates that for our workloads, the overhead of context switching between two worker processes is greater than the gains achieved through duplicating workers. Furthermore, the performance gain from this tuning is greater for the TCP/IP workload than for the Support workload. The small size of transactions for TCP/IP intensive workload causes the waiting time for other resources to be very small. The insignificant (only 1.5% improvement in MAT) gain achieved for the SPECweb Support workload is likely because of the memory intensive characteristic of a subset of requests in this workload. Careful workload-specific tuning of this parameter is therefore critical for obtaining good performance.

4.2. Operating System scheduling

As the next tuning strategy, we examined the Linux Process Affinity mechanism [40] as an alternative to the default Linux scheduler. The default scheduler dynamically migrates worker processes among cores. We used the affinity mechanism to statically assign each of the eight Lighttpd worker processes to its own distinct core. From Table I, core affinity significantly outperforms the default Linux scheduler. The MAT increased 10% and 4% for the TCP/IP Intensive and Support workloads, respectively, compared to the MAT obtained after the previous tuning step. This result shows that for a TCP/IP intensive workload, migrating cached data pertaining to a worker process across cores can adversely impact performance. For the Support workload, we ran an additional experiment where the affinity setting was also applied to the application tier in a way that the PHP-FastCGI processes were equally divided between the cores and statically assigned to them. This further improves the MAT value by around 5% with respect to the mean response time. Therefore, the two affinity settings can improve the server's scalability up to 9% for the Support workload.

DOI: 10.1002/cpe

4.3. Network interrupt handling

The default NIC configuration uses a single core on the system to process interrupts from the NIC. As a result, the interrupt handling core became saturated under high loads, even though the utilization of the other cores were low. Therefore, the network interrupt handling load was distributed between the cores by creating eight RSS queues on the NIC and affining each of the queues to a distinct core through the Linux *IRQ Affinity Setting* mechanism. From Table I, distributing the interrupts improved the MAT value up to 45% and 20.5% for the TCP/IP Intensive and Support workloads, respectively relative to the MAT obtained after the previous tuning step.

5. SCALABILITY EVALUATION

As mentioned in Section 3.2, in the second step of our experimentation, we evaluate the scalability of our Web application as the core count increases under two different workloads. This section explains the results of these experiments in detail. The experiments are performed both with the TCP/IP intensive and the SPECweb Support workloads. Recalling from Section 3.2, for each workload, we change the load intensity to utilize the server to its capacity. We study the performance of the eight-core server when it has one, two, four or eight active cores.

5.1. TCP/IP intensive workload

Figure 4(a) shows the normalized MAT values with respect to mean and 99.9th percentile response times for one, two, four and eight active cores. Different threshold values are used for the two different statistical properties of response time. While the acceptable mean response time is set to 0.5 ms, a 4 ms threshold is considered for 99.9th percentile of response time for this workload. In the experiments with two and four active cores, all cores are selected from the same socket. We normalize MAT values with respect to the value observed with one core. From the figures, the server scales close to proportionally from one to two cores with respect to both the mean and 99.9th percentile of response times. However, for more than two cores, the scalability factors starts to degrade. The degradation is more severe for the mean response times. The overall scalability from one to eight core is 6.3 for the mean and 7.2 for the 99.9th percentile. Selecting larger response time thresholds results in similar numbers. However, lower thresholds slightly decrease the scalability factors. These numbers suggest that there is a scalability problem in the system from four to eight cores. Moreover, the problem is not confined to the tail but observed throughout the body of the response time distribution.

Analysis of the server's response time at different load levels reveals that scalability problems become severe when moving from four cores, that is, using one socket in the system, to eight

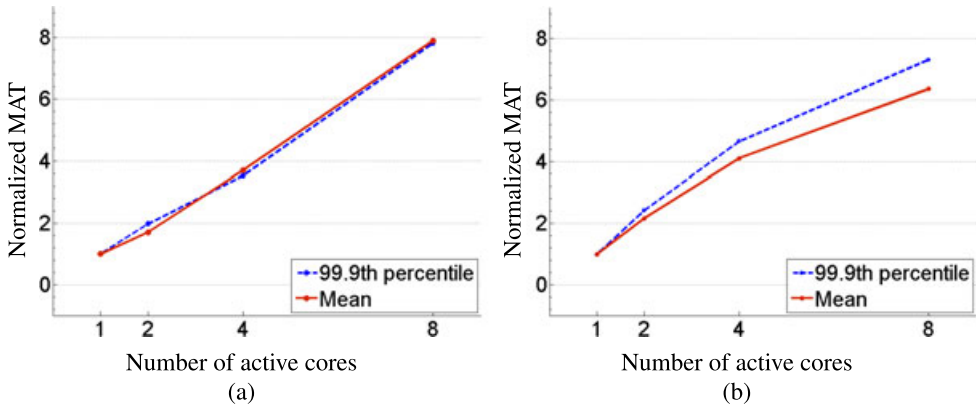


Figure 4. Normalized maximum achievable throughput (MAT) with one, two, four and eight active cores: (a) TCP/IP intensive workload and (b) Support workload.

DOI: 10.1002/cpe

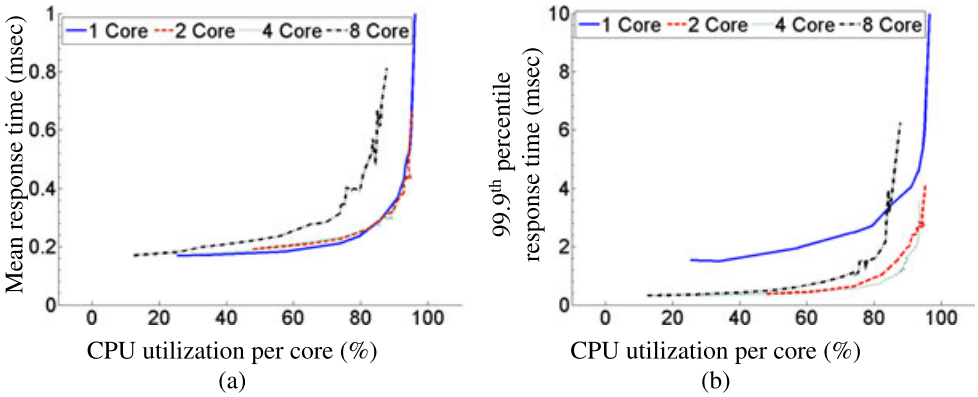


Figure 5. Web server response time versus CPU utilization for the TCP/IP intensive workload: (a) mean and (b) 99.9th percentile.

cores, that is, using both sockets on the system. The measured response time values as a function of CPU utilization are depicted in Figure 5(a) and (b) for the mean and 99.9th percentile, respectively. Figure 5(a) show that mean response times only increase gradually till a mean per core utilization of 80% for experiments with one, two and four active cores. However, in the experiment with eight active cores, the mean response times increase with a larger slope beyond a 60% utilization. A similar trend is observable for the 99.9th percentile of response times in the experiments with two, four and eight active cores. Interestingly, the 99.9th percentiles are significantly higher with one active core compared to higher numbers of active cores. This demonstrates that when only one core is active in the system, a subset of requests have significantly high response times. We have not yet been able to determine the reason for this behaviour in spite of the large amount of hardware event data that we collected and analysed.

In summary, Figure 5(a) confirms the presence of a scalability problem in the system when all eight cores are active, while Figure 5(b) demonstrates that the problem manifests itself throughout the response time distribution. This problem prevents the Web server from fully utilizing the available processor resources. In Section 6, we evaluate the multiple Website replica configuration to see how this approach can help reduce this problem.

5.2. SPECweb Support workload

For this workload, the load on the server is changed by varying the number of concurrent users in the system. Similar to the TCP/IP intensive workload, in the experiments with two and four active cores, all cores are selected from the same socket.

Figure 4(b) shows the MAT values with respect to the mean and 99.9th percentile response times for one, two, four and eight active cores. We consider thresholds of 5 and 500 ms for the mean and 99.9th percentile of response time, respectively. From the figure, the server shows a different scalability behaviour under the Support workload. While the server scales sub-linearly from one to two cores with respect to the mean response time, the scalability factors improve as we activate more cores. The overall scalability from one to eight core is 7.8 and 7.9 for the mean and 99.9th percentile of response time, respectively. The numbers suggest that the server scales well for the Support workload. Increasing the threshold did not have a considerable effect on the scalability factors. However, selecting a lower threshold (e.g. 4 and 400 ms) results in a slightly better scalability from four to eight cores.

Figure 6(a) and (b) further emphasize the differences between the TCP/IP workload and the Support workload. The figures show the mean and 99.9th percentile of response times as a function of server CPU utilizations for one, two, four and eight active cores. They demonstrate four main observations related to the server's scalability. First, from Figure 6(a), with all eight cores enabled, it is possible to utilize the cores to up to 80% without encountering significant response time increases. Second, at high loads and for a given utilization level, the experiments with two and four active

DOI: 10.1002/cpe

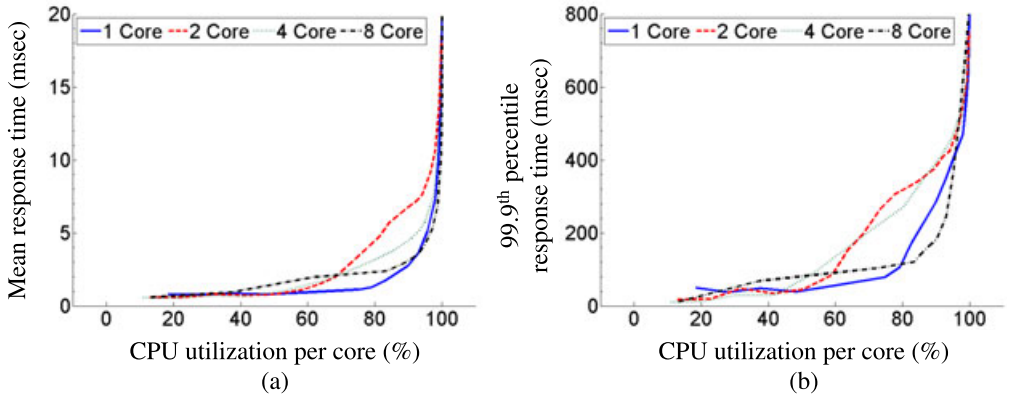


Figure 6. Web server response time versus CPU utilization for the Support workload: (a) mean and (b) 99.9th percentile.

cores have higher mean response times compared to the experiments with one and eight active cores. A similar trend is observable for the 99.9th percentile of response time as depicted in Figure 6(b). Third, the performance degradations of the two and four core curves relative to the eight core curve are less for the 99.9th percentile of response times compared to the mean. Finally, while the experiments with one active core have the best performance considering the mean response time, the best 99.9th percentile response times is measured with eight active cores, especially at high load. These observations reveal that activating more than one core in a single socket has a negative effect on the server's scalability for the Support workload. This effect is more in the lower percentiles and less on the tail of the response time distribution. However, activating another socket considerably improved the scalability.

5.3. Analysis of the response time distribution

We next plot the cumulative distribution function (CDF) of response times in Figure 7 to study how the response time of different types of requests differ when we activate more cores in the server. A single test was selected from the experiments with one, two, four and eight cores. In each experiment, the selected test is the one that causes a CPU utilization of 80% per core. This allows us to ensure that the comparison is carried out at a constant load level relative to the available resources (number of cores). From Figure 7, the response time distributions are almost identical for one, two and four cores cases for the first 95 percentiles. Response times are significantly higher when eight

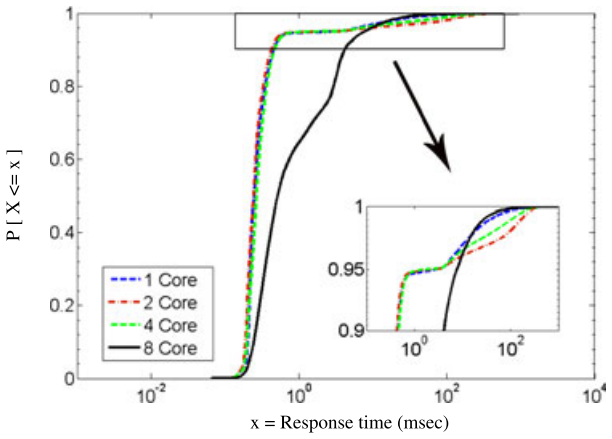


Figure 7. Cumulative distribution function (CDF) of response times at 80% CPU utilization for SPECweb Support workload.

DOI: 10.1002/cpe

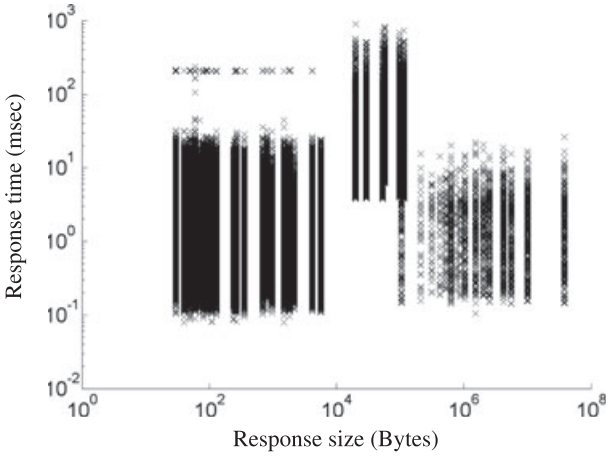


Figure 8. Response time versus response size for a test with eight active cores at 80% CPU utilization for the Support workload.

cores are active. The conditions are reversed for the last 5 percentiles. For this portion of requests, the response times in the tests with two and four cores are considerably higher than the response times in the test with one and eight cores. The CDF plots demonstrate that activating eight cores causes a performance degradation for requests with smaller response times while requests with larger response time benefit from having eight active cores.

We now focus on the identity of the requests with large response times, which benefit from having eight active cores, that is, both sockets. We use the detailed response time log file output by *httperf* to plot the response times as a function of the size of responses to request as shown in Figure 8. The plot was generated from the output of a test with eight active cores at 80% per core utilization. Plots for other numbers of cores and load levels have similar trends. Each point in the graph represents a single request submitted to the server during the test. The plot reveals that requests with 'medium' response sizes (10 to 100 KB) have the highest response times. These response sizes correspond to content that is dynamically generated by the application tier. Therefore, high response time requests that scale well with activating both sockets as shown in the response time distribution in Figure 7 are dynamic requests. The low response time requests with 'low' (less than 10 K) and 'high' (more than 100 K) response sizes are static requests that scale poorly from activating both sockets according to the CDF in Figure 7.

Considering the observations from Figures 7 and 8, we can conclude that the server has a different scalability behaviour when handling the static and dynamic requests. While the server scales well for the dynamic requests, there are some scalability problems when handling static requests, especially when moving from four to eight cores, that is, from one socket to two sockets. The fraction of dynamic requests in the Support workload is significant enough that this workload sees an overall scalability improvement with increased core count. However, the TCP/IP intensive workload contains a solitary static file and hence scales comparatively poorer with core count, especially while moving from one socket to both sockets.

Towards understanding the response time behaviour, we now discuss important low-level differences in system usage across the one socket and two socket scenarios. When only one socket in the system is active, there is unlikely to be significant migration of data over the inter-socket the QPI links because the Linux memory manager tries to allocate memory locally. In contrast, in the two socket scenario, there can be significant data flow across the two sockets through QPI links. Contention for QPI links can potentially hurt the performance of the two sockets case. However, the two socket case exposes twice the LLC capacity to the Web application processes. This can afford performance gains for the PHP-FastCGI processes, which others have shown to be memory

DOI: 10.1002/cpe

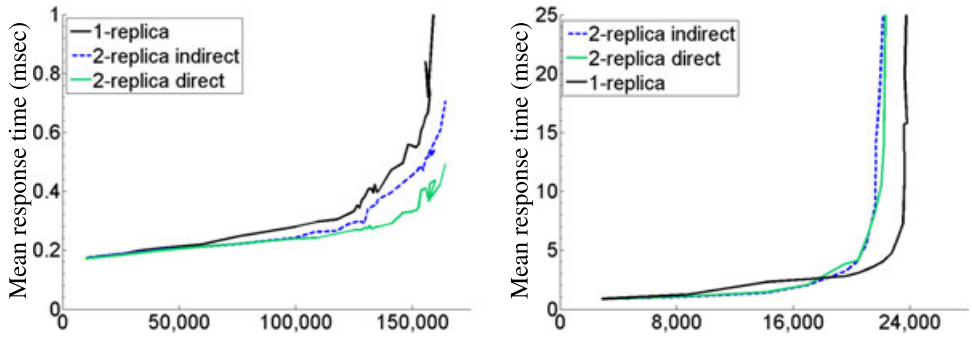


Figure 9. Web server response time versus request rate for eight cores with *1-replica* and *2-replica* configurations: (a) TCP/IP intensive workload and (b) Support workload.

intensive [41][¶]. In the next section, we study the relative effects of such interactions in the system's performance behaviour when using multiple replicas. We then provide hardware event statistics in Section 7 to elaborate on the effect of LLC capacity on the response time of dynamic and static requests.

6. SCALABILITY ENHANCEMENT

The third step of the experimental methodology (Section 3.2) is to evaluate the effect of the multiple replica approach on the scalability of our multi-core Web server. This section presents experiments performed for step 3. First, the results of experiments conducted using the new configurations under the TCP/IP intensive and Support workloads are presented. Next, we discuss the variations observed in the performance effect of the multiple replica approach under the two workloads and for the two types of requests.

6.1. TCP/IP intensive workload

Figure 9(a) shows the mean response times as a function of the request rates for the *1-replica* (results of the experiments with eight cores are explained in Section 5), *2-replica direct* mapping and *2-replica indirect* mapping. From the figure, the MAT (considering a mean response time threshold of 0.5 ms) is the highest for the *2-replica direct* mapping. Furthermore, the MAT for the *2-replica indirect* mapping, while less than that of *2-replica direct*, is *higher* than the *1-replica* configuration. Specifically, the MAT values improve around 12.3% and 6.5% with the *2-replica direct* and *2-replica indirect* mappings, respectively. The improvements in the MAT with respect to a 99.9th percentile threshold of 4 ms (not shown) are 5.2% and 3.1% for the *2-replica direct* and *2-replica indirect* mappings, respectively. While selecting a lower threshold reduces the scalability improvements, choosing a thresholds larger than our selected thresholds does not affect the results.

6.2. Support workload

Figure 9(b) compares the mean response times for the experiments with the Support workload for the *1-replica*, *2-replica direct* and *2-replica indirect* mappings. The figure reveals that the server's scalability degrades significantly with the new configurations under the Support workload. From the figure, the MAT for *2-replica direct* and *2-replica indirect* mappings is statistically similar

[¶]To confirm the memory intensiveness of PHP-FastCGI processes, we compared the memory access traffic for the Support workload with 8000 concurrent users to a statistically similar workload where the dynamic requests were replaced by static requests of the same response size. We noticed that the workload with dynamic requests caused 3.6 times more traffic to the main memory than the workload with static requests.

DOI: 10.1002/cpe

(considering the 95% confidence interval) and around 9.9% lower than the baseline. The graph for the 99.9th percentile of the response times (not shown) showed a similar performance degradation for the multiple replica cases. The MAT value with respect to the 99.9th percentile of the response time was 9.9% and 14.8% lower than the *1-replica* for the *2-replica direct* and *2-replica indirect* mappings, respectively. This result contradicts previous studies [8] by indicating that the multiple replica configuration does not always improve a server's scalability.

6.3. Comparison between workloads

After presenting the effect of the multiple replica approach on server scalability, we now explore the reason for the different performance effects observed for the two workloads. For the TCP/IP intensive workload, the improved scalability of the Web servers with the *2-replica indirect* mapping compared to the *1-replica* despite the 50% additional inter-socket migration of data due to network processing suggests that reducing QPI traffic triggered by processes accessing shared data is most responsible for observed scalability improvements. Because the TCP/IP workload has no dynamic requests and does not use the PHP-FastCGI processes, this result suggests that there is significant sharing of data among Lighttpd processes. Furthermore, because Lighttpd processes are not memory intensive [41], this workload does not seem to be impacted by the inability of a replica to benefit from the LLC of the socket hosting the other replica. However, for the Support workload, the observed effect is the resultant of the performance effect for both Lighttpd and PHP-FastCGI processes. This suggests that running multiple replicas has an overall negative effect. We will elaborate on this effect in Section 6.4 by comparing the response time of static and dynamic requests.

The average traffic in the QPI link collected during the experiments confirms the aforementioned hypothesis. Figure 10(a) shows the average QPI traffic at various load levels for the TCP/IP intensive workload. From the figure, a considerable decrease in QPI traffic is observable from *1-replica* to *2-replica direct* mapping. Moreover, the QPI traffic is higher than the *1-replica* for *2-replica indirect* mapping (because for 50% of requests, HTTP request and TCP packet are processed on the same socket). These measurements confirm that applying the multiple replica approach can decrease inter-socket migration under a TCP/IP intensive workload.

From Figure 10(b), the average QPI traffic decreased from *1-replica* to *2-replica direct* mapping for the Support workload. Moreover the QPI traffic for the *2-replica indirect* mapping is higher than the *1-replica* configuration. This confirms that a multiple replica solution was able to decrease the inter-socket data migration for the Support workload as well. However, the amount of reduction is much less than for the TCP/IP intensive workload. This is likely because of less TCP/IP activity (lower request rate) relative to the application-level processing for the Support workload. This results in fewer inter-socket data migrations of the network processing data. Section 7 discusses the reason for the performance degradation despite the reduction in QPI traffic under the Support workload.

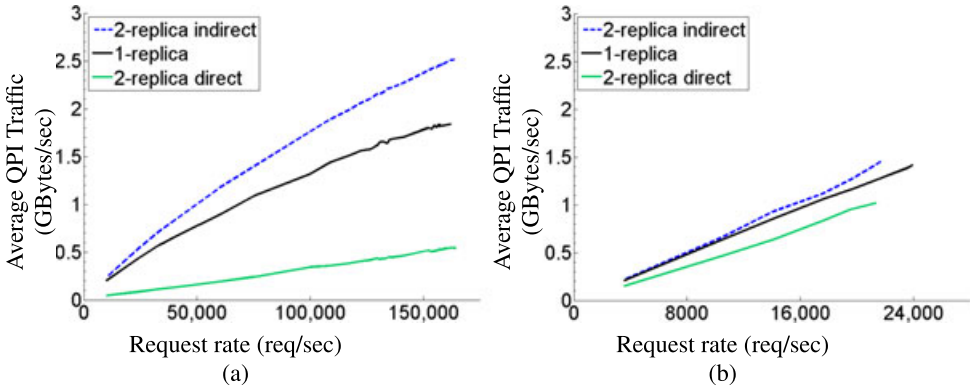


Figure 10. Average QPI traffic versus request rate for *1-replica* and *2-replica* configurations: (a) TCP/IP intensive workload and (b) SPECweb Support workload.

DOI: 10.1002/cpe

6.4. Effect on dynamic and static requests

The results presented in the previous section suggest that using multiple Web site replicas do not improve the average response time for the Support workload. We now quantify the separate effect of the multiple replica approach on the response time of static and dynamic requests. The CDF plot of response times for the *1-replica*, *2-replica direct* and *2-replica indirect* mappings for the tests with 80% CPU utilization is depicted in Figure 11. The plot was generated from the output of the tests with 7000 concurrent users. Plots for other high load tests have similar trends. From the graph, the first 95 percentiles of response times, mainly static requests, improved significantly with the *2-replica* configurations compared to the *1-replica*. This is analogous to the results obtained from the experiments under TCP/IP intensive workload. On the other hand, in the last 5 percentiles of the responses times, which are mainly dynamic requests, the curves are lower (higher response times) for tests with *2-replica* configurations. The results indicate that the multiple replica configuration degraded performance for the dynamic requests. Because these requests consume most of the processing resources and their response times are an order of magnitude higher than the static requests, their performance decides the overall scalability of the server under the Support workload. To further analyse the workload-dependent effect of the multiple replica approach on the response time of dynamic requests, we conducted another set of experiments using the Modified Support workload. The experiments are designed to help us quantify the effect of workload mix on the overall performance of the server with two replicas. Recall from Section 3.5 that the set of workloads at this stage are a modified version of the SPECweb Support mixes, in which we replace a fraction of the dynamic requests (RF) with a request for a static file. The size of this file is similar to the reply size of the server for that dynamic request. Three different values of 0.2, 0.4 and 0.6 are examined for this variable. Due to a network bottleneck (tracked to the switch) that prevented us from fully utilizing the 20 Gbps bandwidth capacity of the server's NIC, we could not conduct experiments for values beyond RF $= 0.6$.

The results of the experiments with the three RF values are presented in Figure 12. Figures 12(a), (b) and (c) show the mean response time values as a function of the request rate for RF $= 0.2$, RF $= 0.4$ and RF $= 0.6$, respectively. In each graph, the last rate represents a test with 85% CPU utilization. The plots show that as we decrease the fraction of dynamic requests (increase RF), the difference between the *1-replica* and *2-replica direct* curves decreases. This confirms that as the load on the application tier (PHP-FastCGI processes) decreases, the server can benefit more from the *2-replica* configuration. Moreover, this plot suggests that for values beyond RF $= 0.6$, the server will benefit from the multiple replica configuration.

We now focus on the response time distribution for dynamic and static requests. Figure 13(a) and (b) show the CDF of response time for RF $= 0.6$ at 80% CPU utilization with *1-replica*,

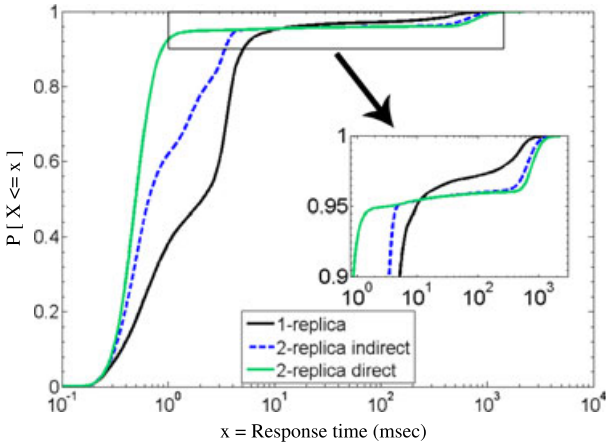


Figure 11. Cumulative distribution function (CDF) of response times in 80% CPU utilization with baseline and *2-replica* configurations under the Support workload.

DOI: 10.1002/cpe

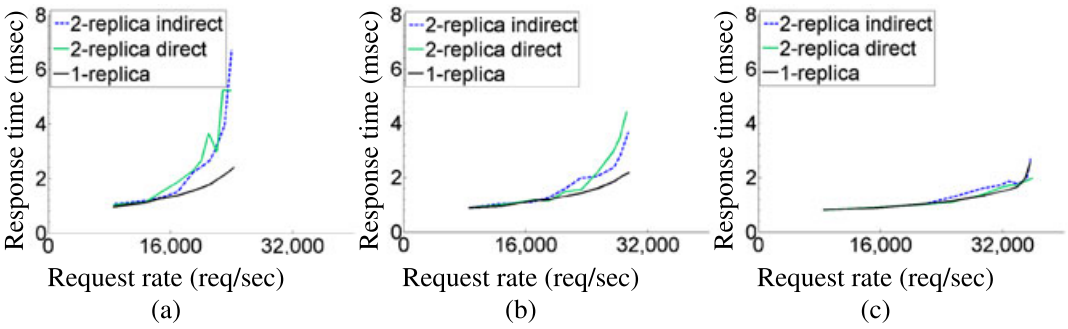


Figure 12. Web server response time versus request rate for Modified Support workload: (a) RF = 0.2, (b) RF = 0.4 and (c) RF = 0.6.

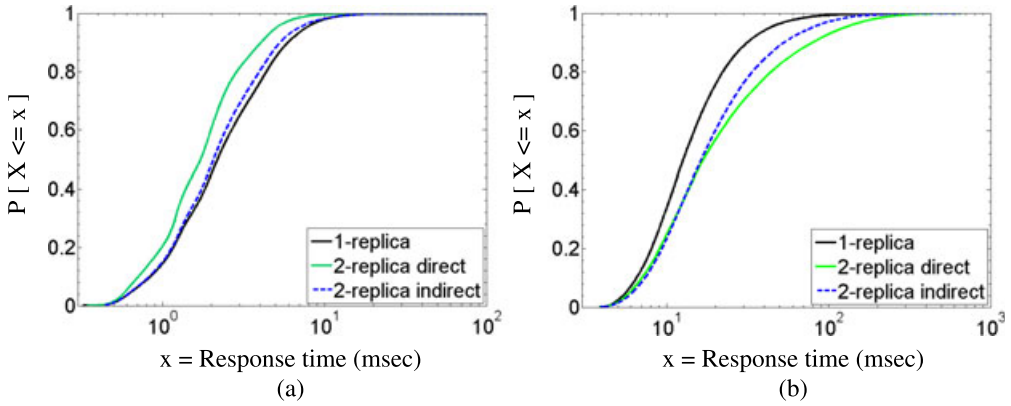


Figure 13. Cumulative distribution function (CDF) of response times in 80% CPU utilization for the synthetic mix with RF = 0.6 (a) static requests and (b) dynamic requests.

2-replica configurations for static and dynamic requests, respectively. From Figure 13(a) for the static requests, the response times of static requests decrease considerably throughout the distribution with the *2-replica direct* mapping compared to the *1-replica* configuration. The response times for the *2-replica indirect* configuration also slightly decreased throughout the distribution compared to the*1-replica* configuration. This confirms that the multiple replica approach improves the response time of static requests throughout the distribution. In particular, the distribution plots reject the hypothesis that the improvement is only related to a subset of static requests (e.g. requests for files in a specific size range). From Figure 13(b) for the dynamic requests, *2-replica direct* degraded the response time of dynamic requests. The degradation is smaller for the *2-replica indirect* mapping. The trends suggest that the response time degradation with *2-replica* configurations is observable throughout the distribution for the dynamic requests. Similar to the static requests, the distribution plots reject the hypothesis that the response time degradation is only related to a subset of dynamic requests (e.g. requests that require a particular amount of PHP processing time). The order of the curves for dynamic requests also suggests that the magnitude of the problem that causes the performance degradation is higher for the *2-replica direct* compared to the *2-replica indirect* mapping. The results of experiments with the Support and Modified Support workloads indicate that dynamic requests do not benefit from reducing QPI traffic. In other words, a problem related to the PHP-FastCGI processes and with a considerable negative effect on the response time arises with the *2-replica* configurations. Considering the observation from the scalability behaviour of dynamic and static requests explained in Section 5.3 and the memory intensive nature of dynamic requests, the problem is likely related to the inability of PHP-FastCGI processes of one replica to use the cache

DOI: 10.1002/cpe

hierarchy of the socket containing the other replica. This seems to hurt overall performance. In the next section, we examine this hypothesis by presenting the hardware event data collected during the experiments.

7. SCALABILITY BOTTLENECKS

As we showed in the previous section, the effect of the multiple replica approach was different on the response time distribution of dynamic and static requests. In this section, we perform a root cause analysis using hardware event data to identify possible bottlenecks responsible for this performance behaviour. The main metrics that we consider are the average number of Instructions per Cycle, the LLC hit ratios, the average QPI traffic and the memory access statistics. We compare the mean values of these metrics for the *1-replica* and *2-replica* mappings for the Modified Support workload with RF $= 0$ (Original Support workload) and RF $= 0.6$. As explained in Section 3.5, the difference between these two workloads is that in the latter, 60% of the dynamic requests are replaced by static ones. Therefore, a comparison between the values of a hardware metrics for these two workloads for different configurations can reveal the effect of the multiple replica approach on the values that hardware metric for static and dynamic requests.

7.1. Instructions per clock cycles

To ensure that the higher response times of the dynamic requests with the *2-replica* configurations is a result of a server bottleneck and not related to other problems such as a saturated network, we first compare the number of Instructions per CPU clock cycles (IPC) for the two workloads. The higher IPC values can be a sign of better performance of the servers processing unit. Figure 14(a) and (b) show the IPC value at four different load levels for workloads with RF $= 0$ (Support workload) and RF $= 0.6$, respectively. The figures show that the IPC is higher for the RF $= 0$. This suggests that the presence of a higher number of dynamic requests triggers more CPU activities. From the figures, for the higher fractions of dynamic requests (RF $= 0$), the IPC for the *2-replica* mappings decreases as the load increases. However, the trend is reversed for the *1-replica* configuration. In particular, at the same high load of around 21 000 req/s, the IPC is around 10% lower for the *2-replica* mappings. In contrast, for the lower fractions of dynamic requests (RF $= 0.6$), the IPCs are almost similar for the *2-replica direct* and *1-replica* configuration at this highest load (21 000 req/s). This observation is analogous to the trend we observed for the response times in Figures 9(b) and 12(c) and suggests that the increased response time for the dynamic requests with *2-replica* mappings is related to an internal CPU bottleneck in the server and is not caused by an external bottleneck in the network and workload generators.

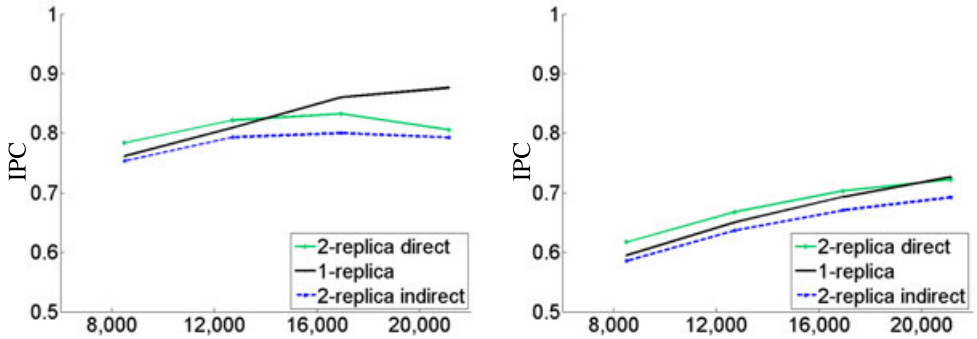


Figure 14. Instructions per cycle (IPC) versus request rate (req/s) with *1-replica*, *2-replica direct* and *2-replica indirect* configurations under (a) Support workload (RF $= 0$) and (b) Modified Support workload (RF $= 0.6$).

DOI: 10.1002/cpe

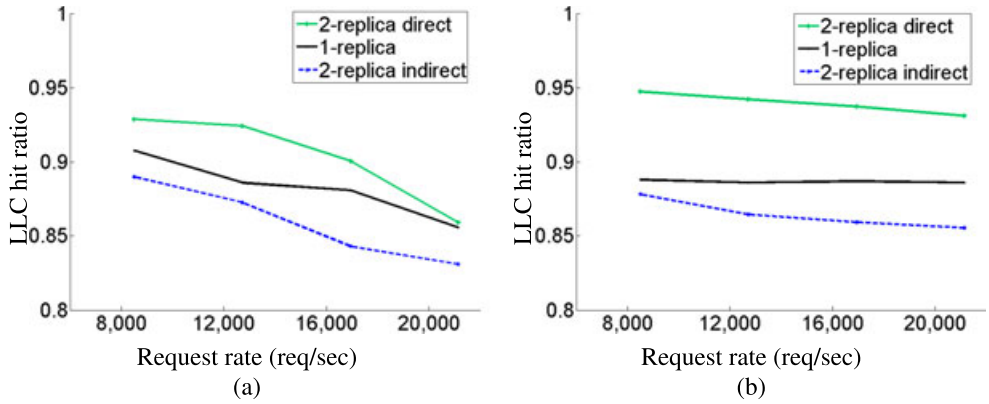


Figure 15. Last level cache (LLC) hit ratio versus request rate (req/s) with *1-replica*, *2-replica direct* and *2-replica indirect* configurations under (a) Support workload (RF = 0) and (b) Modified Support workload (RF = 0.6).

We next investigate other metrics, in particular those related to the memory hierarchy to identify the bottleneck that causes the increased response time of dynamic requests with the *2-replica* mappings.

7.2. *Last level cache*

As mentioned in Section 3.1, the multi-core processors of the server have a three-level cache hierarchy. The L1 and L2 caches are dedicated to one core, while the LLC (L3) is shared between the four cores of each processor. Competition among cores for this shared resource can create a potential bottleneck in the system. Therefore, we consider the LLC hit ratio as one of the metrics measured during the experiments. This metric is defined as the fraction of requests to the LLC that result in a hit.

Figure 15(a) and (b) shows the LLC hit ratio at four different load levels for workloads with RF = 0 (Support workload) and RF = 0.6, respectively. First consider the curves for *1-replica* and *2-replica direct* mapping. For the lower fractions of dynamic requests (RF = 0.6), the LLC hit ratio is consistently *higher* for the *2-replica direct* compared to *1-replica* configuration. This is a positive effect of using multiple replicas and is mainly because of the local processing of network and application data when using *2-replica direct* configuration. In particular, when there is one replica running per socket and the network and application data processing is performed locally, the cores on the same socket share data in the local LLC. This reduces the number of misses that are served by the remote LLC, therefore, increasing the LLC hit ratio. In contrast, for the higher fractions of dynamic requests (RF = 0), the LLC hit ratio decreases with a higher slope for the *2-replica direct* configuration compared to *1-replica*. Due to this trend, for the highest load (request rate of around 21 000 req/s), the LLC hit ratio is almost *similar* for the *2-replica direct* and *1-replica* configurations. This indicates that the dynamic requests and in particular PHP-FastCGI processes reduce the LLC hit ratio, and the reduction is more severe with *2-replica direct* configuration.

Next, we consider the curve for the *2-replica indirect* mapping. The LLC hit ratio has the lowest values with *2-replica indirect* mapping for both RF = 0 and RF = 0.6. This can be explained with the remote application and network processing imposed by the *Indirect* mapping. This causes more misses to be served afterward with an access to the LLC of the remote socket. Moreover, similar to the *2-replica direct* mapping, the LLC hit ratio degrades more severely for RF = 0, due to the presence of more dynamic requests. This indicates that both *2-replica* mappings suffer from LLC performance degradation caused by dynamic requests.

The LLC graphs show that the dynamic requests cause a performance degradation in the LLC with *2-replica* configurations. However, in view of the response time results shown in Figures 9(b) and 12(c), one can conclude that the LLC performance (hit ratio) is not a precise predictor metric for the mean request response times. As an example, for RF = 0 (Figure 9(b)), while the response

DOI: 10.1002/cpe

time for the load of around 21 000 req/s is higher with *2-replica direct* mappings compared to *1-replica*, the LLC hit ratio is almost identical for the *2-replica direct* and *1-replica* configurations. Another example is the higher values of LLC hit ratio for the *2-replica direct* mapping compared to *1-replica* in all load levels for the RF = 0.6 while the response times of the two configurations are quite similar for this workload. These dissimilarities can be explained by the variable delay penalties for the LLC misses. The LLC misses that are served by the remote cache hierarchy and the remote memory will have a higher delay compared to those served by the local memory. We next consider the average memory access rates as the next metric and examine whether this metric is a good indicator of the response time behaviour.

7.3. Memory access traffic

We now consider the average memory access rates and compare these value for two workloads with different fractions of dynamic requests. Figure 16(a) and (b) shows the average memory access rates at four different load levels for workloads with RF = 0 (Support workload) and RF = 0.6, respectively. We first compare the memory access rates between the two workloads. From the figures, reducing the fraction of dynamic requests (from RF = 0 to RF = 0.6) reduces the memory access rate. This confirms that the dynamic requests, in particular, the PHP-FastCGI processes that handle the dynamic requests, are more memory intensive compared to the static requests.

We now compare the average memory access rates between the *1-replica* and *2-replica* configurations. From the figures, the memory access rates are almost similar between *1-replica* and *2-replica* configurations for both workloads with RF = 0 and RF = 0.6. However, at higher loads, and with RF = 0, the average memory access is *higher* with the *2-replica* configurations compared to *1-replica*, as shown in Figure 16(a). For a lower fraction of dynamic requests (RF = 0.6), the memory access rate is almost similar for the three configurations as shown in Figure 16(b). These trends are analogous with the response time plots presented in Figures 9(b) and 12(c). This shows that unlike the LLC hit ratio, the memory access rates are a good predictor of response time in this scenario.

The reason for the observed behaviour in the memory hierarchy is explained next. We first review the differences between the *1-replica* and *2-replica* mappings at the cache hierarchy and memory levels. As explained in Section 3.6, when there are multiple instances of the Web application, each running on a socket, the dataset of Lighttpd and PHP-FastCGI processes is only shared by the cores of one socket. Therefore, each request that misses the local cache hierarchy results in a local or remote memory access. In contrast, when a single replica is running on both sockets, the requests that miss the local cache hierarchy can be served by the cache hierarchy of the other socket. This results in a lower memory access rate for the *1-replica* configuration. On the other hand, due to the memory intensive behaviour of PHP-FastCGI processes, as the fraction of dynamic

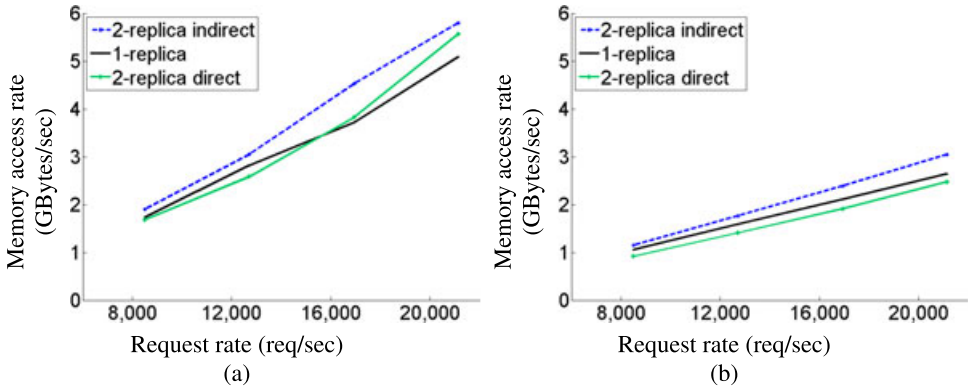


Figure 16. Memory access rate versus request rate (req/s) with *1-replica*, *2-replica direct* and *2-replica indirect* configurations under (a) Support workload (RF = 0) and (b) Modified Support workload (RF = 0.6).

DOI: 10.1002/cpe

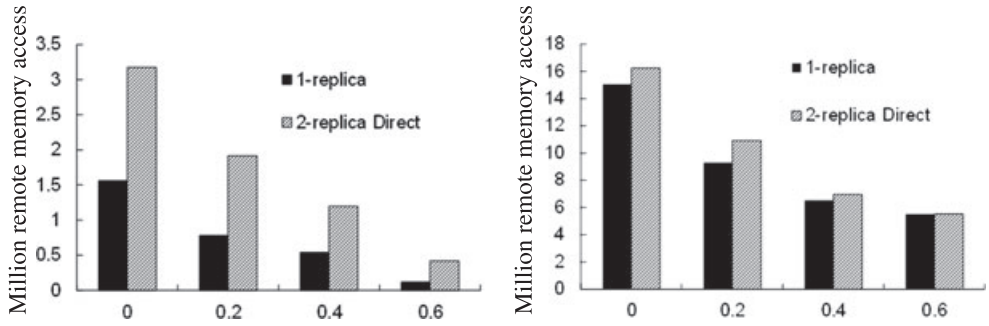


Figure 17. Total remote memory access per module for workloads with RF = 0, RF = 0.2, RF = 0.4 and RF = 0.6 with *1-replica*, *2-replica direct* and *2-replica indirect* configurations under (a) PHP-FCGI and (b) VMLinux.

requests increases the LLC hit ratio decreases, as we confirmed in Figure 15(a) and (b). This leads to more memory accesses with the *2-replica* configurations compared to *1-replica*. The latency for the local memory accesses is almost similar to the remote cache accesses in our server's architecture. However, the latency for the remote memory access is around 60 to 70 cycles (60%) more than that of a remote cache access [42]. This inflates the response times for dynamic requests in the *2-replica* configurations. Figure 17(a) and (b) shows the amount of remote memory accesses for Support workload (RF = 0) and Modified Support workloads (RF = 0.2, 0.4, 0.6) at the high load of 21 000 req/s for the PHP-FastCGI and Operating System module (vmlinux), respectively. From the figure, the number of remote memory accesses decreases as the percentage of dynamic requests decreases (RF increases). The trend is similar for both the Operating System and PHP-FastCGI functions. Moreover, the number of remote memory accesses is more with *2-replica direct* configuration compared to with *1-replica*. The trends in Figure 17(a) and (b) further confirm that the *2-replica* configuration increases the amount of remote memory accesses.

In summary, the requests that miss the local cache hierarchy are served by the remote cache hierarchy, local or remote memory. We have shown that with the *2-replica* configuration, an increase in the ratio of dynamic requests results in more memory accesses. This increase in remote memory accesses can inflate the response time of dynamic requests. It should be noted that a different LLC behaviour for other applications and architectures may create different results. These results therefore illustrate that for a particular system and application, the behaviour of the memory hierarchy should be examined prior to applying the multiple replica approach.

8. SUMMARY

The previous sections presented the results of a set of controlled experiments to evaluate the scalability of our multi-core Web server under the TCP/IP intensive, SPECweb Support and modified versions of the Support workload. Our results show that the server scales well from one to four cores under the TCP/IP intensive and Support workloads. When only one socket was active in the system, the multi-core server was utilized up to 80% while still serving requests without significant queuing delays, as shown in Figures 5 and 6 for both workloads. The same behaviour was observable for eight active cores with the SPECweb Support workload. However, for the TCP/IP intensive workload, significant scalability degradations appear from four to eight cores when both sockets on the system are active.

We evaluated the effect of running multiple instance of Web application on a single multi-core server for our three workloads. Running multiple instances of the Web server on the machine, in particular one per socket, could improve this scalability issue for the TCP/IP intensive workload. With the configuration that had one instance of our Web server software (Lighttpd) per core, the

DOI: 10.1002/cpe

MAT with respect to mean response time improved up to 12.3%. The multiple replica approach also improved the response time of static requests in the SPECweb Support workload. However, the approach was harmful to the response time of dynamic requests in the Support workload. This resulted in an overall degradation of 9.9% in the MAT with respect to mean response time for the Web server with multiple replica configuration and under the Support workload. As we decreased the fraction of dynamic requests in the Support workload, the server's scalability with the multiple replica configuration improved. Therefore, the overall benefit of deploying the multiple replica configuration was shown to be dependent on the fraction of static and dynamic requests in the workload mix.

We collected detailed hardware level statistics and used this large amount of data to analyse the performance behaviours observed in various experiments. The analysis results revealed that the higher response times of dynamic requests with multiple replica configuration is likely because of the contention in the cache hierarchy, particularly in the LLC, caused by the memory intensive PHP-FastCGI processes. This in turn leads to larger amounts of costly memory accesses that can inflate the response time. While the hardware event data could explain the response time behaviour of dynamic and static requests, in some cases, we were not able to correlate the hardware events with the measured response times. For instance, the hardware event statistics could not explain the reason for the high 99.9th percentile response time in the experiments with one active core and under the TCP/IP intensive workload. We believe that due to the complexity of hardware architecture of the multi-core server, the hardware event data are not sufficient for the scalability analysis of these architectures.

9. CONCLUSION

This paper provided a detailed characterization of the scalability of Web applications on multi-core hardware using three different workloads. We evaluated the effect of using multiple instances of Web application on improving the server's scalability. In our experiments, the server was utilized up to 80% under different workloads while still serving requests with acceptable response times. While we only investigated one multi-core architecture, the tuning strategies are applicable for other similar architectures with architecture specific revisions. We believe that being able to sustain such high utilization levels in peak loads would enable popular service providers to deploy fewer servers, thereby reducing both their capital and operating expenses [11]. Given previously reported utilization numbers in the 6% to 12% range [43], our results suggest that for service provider companies operating hundreds of thousands of servers, even a 10% reduction in the number of servers required could save them substantial amounts of money, as well as reducing their environmental footprints. Reducing the number of servers required to support a workload could also enable other savings, such as reduced cooling demand in data centres.

In contrast to previous studies that all have suggested that the multiple replica approach improves the performance of Web applications [8, 9], our performance evaluation results and root cause analysis reveal that the performance effects differ depending on the workload mix, hardware architecture and the behaviour of Web and application software. Future work will involve systematic testing of alternative architectures, workloads and system configurations. Building on the insights from this study, we are currently developing techniques that leverage a server's workload characteristics to automatically decide whether or not to deploy multiple replicas on the server.

REFERENCES

1. Mogul JC. Network behavior of a busy web server and its clients. *Technical Report WRL-95-5*, DEC Western Research Laboratory: Palo Alto, CA, 1995.
2. Almeida J, Almeida V, Yates D. Measuring the behavior of a world-wide web server. *Proceedings of the IFIP TC6 Seventh International Conference on High Performance Networks (HPN '97)*, White Plains, New York, USA, 1997; 57–72.
3. Arlitt M, Williamson C. Internet web servers: workload characterization and performance implications. *IEEE/ACM Transactions on Networking* 1997; **5**:631–645.

DOI: 10.1002/cpe

4. Menasce DA, Almeida V. *Capacity Planning for Web Services: Metrics, Models, and Methods* (1st edn). Prentice Hall PTR: Upper Saddle River, NJ, USA, 2001.
5. Lighttpd web server. (Available from: http://www.lighttpd.net/).
6. First the tick, now the tock: Intel microarchitecture (nehalem), 2009. White Paper.
7. Veal B, Foong A. Performance scalability of a multi-core web server. *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS '07),* ACM, New York, NY, USA, 2007; 57–66.
8. Gaud F, Lachaize R, Lepers B, Muller G, Quema V. Application-level optimizations on numa multicore architectures: the apache case study. *Research Report RR-LIG-011*, LIG: Grenoble, France, 2011.
9. Hashemian R. Revisiting Web server performance and scalability. *Master's Thesis*, University of Calgary, 2011.
10. Hashemian R, Krishnamurthy D, Arlitt M, Carlsson N. Improving the scalability of a multi-core web server. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13. ACM: New York, NY, USA, 2013; 161–172. (Available from: http://doi.acm.org/10.1145/2479871.2479894).
11. Gill P, Arlitt M, Carlsson N, Mahanti A, Williamson C. Characterizing organizational use of web-based services: methodology, challenges, observations, and insights. *ACM Transactions on the Web* 2011; **5**(4):19:1–19:23. DOI: 10.1145/2019643.2019646.
12. Barroso LA, Hölzle U. The case for energy-proportional computing. *Computer* 2007; **40**(12):33–37. DOI: 10.1109/MC.2007.443.
13. Boyd-Wickizer S, Clements AT, Mao Y, Pesterev A, Kaashoek MF, Morris R, Zeldovich N. An analysis of linux scalability to many cores. *Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation (OSDI'10),* USENIX Association, Berkeley, CA, USA, 2010; 1–8.
14. Kumar A, Huggahalli R, Makineni S. Characterization of direct cache access on multi-core systems and 10gbe. *Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture*, Raleigh, NC, 2009; 341–352.
15. Scogland T, Balaji P, Feng W, Narayanaswamy G. Asymmetric interactions in symmetric multi-core systems: analysis, enhancements and evaluation. *Proceedings of the ACM/IEEE conference on Supercomputing (SC '08),* IEEE Press, Piscataway, NJ, USA, 2008; Article 17, 12 pages.
16. Tam D, Azimi R, Soares L, Stumm M. Managing shared l2 caches on multicore systems in software. *Proceedings of the Workshop on the Interaction Between Operating Systems and Computer Architecture WIOSCA*, Florida, USA, 2007.
17. Lin J, Lu Q, Ding X, Zhang Z, Zhang X, Sadayappan P. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *Proceeding of the IEEE 14th International Symposium on High Performance Computer Architecture*, Salt Lake City, UT, USA, 2008; 367–378.
18. Hammoud M, Cho S, Melhem R. C-amte: a location mechanism for flexible cache management in chip multiprocessors. *Journal of Parallel and Distributed Computing* 2011; **71**(6):889–896. (Available from: http://www.sciencedirect.com/science/article/pii/S0743731510002418), Special Issue on Cloud Computing.
19. Zhang Y, Kandemir M, Yemliha T. Studying inter-core data reuse in multicores. In *Proceedings of the ACM Sigmetrics Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11. ACM: New York, NY, USA, 2011; 25–36. DOI: 10.1145/1993744.1993748.
20. Blagodurov S, Zhuravlev S, Fedorova A, Kamali A. A case for numa-aware contention management on multicore systems. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10. ACM: New York, NY, USA, 2010; 557–558. DOI: 10.1145/1854273.1854350.
21. Jaleel A, Najaf-abadi HH, Subramaniam S, Steely SC, Emer J. Cruise: cache replacement and utility-aware scheduling. In *Proceedings of the Seventeenth ACM International Conference on Architectural Support for Programming Languages and Operating Systems,* ASPLOS XVII, London, UK, 2012; 249–260. DOI: 10.1145/2150976.2151003.
22. Cherkasova L, Ciardo G. Characterizing temporal locality and its impact on web server performance. *Proceedings Ninth International Conference on Computer Communications and Networks*, Las Vegas, Nevada, USA, 2000; 434–441.
23. Elnozahy M, Kistler M, Rajamony R. Energy conservation policies for web servers. *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems,* Vol. 4, Seattle, WA, USA, 2003; 8–8.
24. Harji AS, Buhr PA, Brecht T. Comparing high-performance multi-core web-server architectures. *Proceedings of the 5th Annual International Systems and Storage Conference (SYSTOR '12),* ACM: New York, NY, USA, 2012; Article 1 , 12 pages.
25. You G, Zhao Y. A weighted-fair-queuing (wfq)-based dynamic request scheduling approach in a multi-core system. *Future Generation Computer Systems* 2012; **28**(7):1110–1120. DOI: 10.1016/j.future.2011.07.006. (Available from: http://www.sciencedirect.com/science/article/pii/S0167739X11002135), Special section: Quality of Service in Grid and Cloud Computing.
26. Ban K, Chow K, Lee Y-F, Butera R, Friberg S, Peers E. Java application server optimization for multi-core systems. *Intel Tech. Paper*, 2009.
27. Inoue H, Nakatani T. Performance of multi-process and multi-thread processing on multi-core smt processors. *2010 IEEE International Symposium on Workload Characterization (IISWC)*, Atlanta, GA, USA, 2010; 1–10.
28. Salomie T-I, Subasu IE, Giceva J, Alonso G. Database engines on multicores, why parallelize when you can distribute? In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11. ACM: New York, NY, USA, 2011; 17–30. DOI: 10.1145/1966445.1966448.

29. Pci local bus specification revision 2.3 - msi-x ecn, 2003. (Available from: http://www.pcisig.com/specifications/conventional/msi-x_ecn.pdf) [Accessed on September 2013].
30. Scalable networking: eliminating the receive processing bottleneck-introducing rss, 2004. (Available from: http://download.microsoft.com/download/5/d/6/5d6eaf2b-7ddf-476b-93dc-7cf0072878e6/ndis_rss.doc) [Accessed on September 2013].
31. The apache http server project. (Available from: http://httpd.apache.org/).
32. Specweb2009, 2009. (Available from: http://www.spec.org/web2009/) [Accessed on June 2013].
33. Brown MR. Fastcgi: a high-performance gateway interface. *Proceedings of the fifth International World Wide Web Conference*, Paris, France, 1996.
34. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazon's highly available key-value store. *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07),* ACM, New York, NY, USA, 2007; 205–220.
35. Seger M. collectl, 2008. (Available from: http://collectl.sourceforge.net/Documentation.html).
36. Intel performance counter monitor - a better way to measure cpu utilization. (Available from: http://software.intel.com/en-us/articles/intel-performance-counter-monitor/).
37. Mosberger D, Jin T. httperf: a tool for measuring web server performance. *SIGMETRICS Performance and Evaluation Review* December 1998; **26**:31–37.
38. Hashemian R, Krishnamurthy D, Arlitt M. Web workload generation challenges: an empirical investigation. *Software: Practice and Experience* 2012; **42**(5):629–647. DOI: 10.1002/spe.1093. (Available from: http://dx.doi.org/10.1002/spe.1093).
39. Lighttpd web server documentation: Smp and lighty. (Available from: http://redmine.lighttpd.net/projects/lighttpd/wiki/Docs-MultiProcessor).
40. Love R. *Cpu affinity*. (Available from: http://www.linuxjournal.com/article/6799).
41. Suzumura T, Trent S, Tatsubori M, Tozawa A, Onodera T. Performance comparison of web service engines in php, java and c. *Proceedings of the IEEE International Conference on Web Services (ICWS '08),* IEEE Computer Society: Washington, DC, USA, 2008; 385–392.
42. Hackenberg D, Molka D, Nagel WE. Comparing cache architectures and coherency protocols on x86-64 multicore smp systems. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42. ACM: New York, NY, USA, 2009; 413–422. (Available from: http://doi.acm.org/10.1145/1669112.1669165).
43. The cloud factories: power, pollution and the internet. (Available from: http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html) [Accessed on September 2013].

DOI: 10.1002/cpe