Improving the Scalability of a Multi-core Web Server

Raoufehsadat Hashemian
ECE Department, University of Calgary, Canada
rhashem@ucalgary.ca

Diwakar Krishnamurthy
ECE Department, University of Calgary, Canada
dkrishna@ucalgary.ca

Martin Arlitt
Sustainable Ecosystems Research Group
HP Labs, Palo Alto, USA
martin.arlitt@hp.com

Niklas Carlsson
CIS Department, Linköping University, Sweden
niklas.carlsson@liu.se

ABSTRACT

Improving the performance and scalability of Web servers enhances user experiences and reduces the costs of providing Web-based services. 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 2 socket, 4-cores per socket server. Our study show that default, "out-of-the-box" Web server configurations can cause the system to scale poorly with increasing core counts. We study two different types of workloads, namely a workload that imposes intense TCP/IP related OS activity and the SPECweb2009 Support workload, which incurs more application-level processing. We observe that the scaling behaviour is markedly different for these two types of workloads, mainly due to the difference in the performance characteristics of static and dynamic requests. The results of our experiments reveal that with workload-specific Web server configuration strategies a modern 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.

Keywords

Performance characterization, Web server scalability, Multi-core servers

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, service providers such as Google are interested in improving the effective utilization of their infrastructure, as this improves the economic sustainability of their business. 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 2000s [19] [3] [5] [18], 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.

In this paper, we present a detailed measurement-based study of the performance behaviour of the Lighttpd [16] Web server deployed on a 2 socket, 4-cores per socket system based on the Intel Nehalem [2] 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 [24] [10] 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 OS 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 99.9^{th} percentile response time target of 4 ms, request throughput increases by a factor of 7.2 from 1 to 8 cores. For a mean response time target of 0.5 ms, throughput only increases by a factor of 6.3 from 1 to 8 cores. In contrast, throughput increases nearly linearly with core count for the SPECweb2009 Support workload for various mean and 99.9^{th} 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 af-

ICPE'13, April 21–24, 2013, Prague, Czech Republic.

fected 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 migrate across the two sockets in the system. For the more memory intensive dynamic requests, the adverse impact of process data migration is offset by an overall reduction in the amount 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.* [10] and Hashemian [13] 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 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. Our experiments show that the multiple replica approach significantly reduces inter-socket data migrations and improves scalability for the TCP/IP intensive workload. However, the new configuration decreases scalability relative to the single replica configuration for the SPECWeb Support workload. 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.

In summary, our paper provides insights on configuring Web servers to efficiently leverage 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 environment and a University environment 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 [6] in an attempt to ensure good response time performance. We believe such a strategy is inefficient and may need to be revisited in light of evidence presented in this paper.

The remainder of the paper is organized as follows. Section 2 reviews previous work. Section 3 describes the experimental setup, methodology, workloads and the initial Web server configuration tunings done to better exploit the parallelism of Multi-core servers. Section 4 studies the performance and scalability of the tuned server under both workloads. Section 5 explains the multiple Web replica solution and experimentally evaluates the performance gains from employing it for each of the workloads. Section 6 summarizes our work and offers directions for future research.

2. RELATED WORK

Several researchers have in recent years proposed novel OS-level enhancements to improve the performance of Multi-core systems [7] [15]. For example, Boyd *et al.* [7] modified the Linux kernel to remove resource bottlenecks that prevent applications from fully exploiting Multi-core hardware. Their study shows how the Apache Web server benefits from these kernel modifications. Kumar *et al.* [14] characterized the performance of a TCP/IP intensive workload generated on an Intel Core 2, 2-socket, 4 cores per socket system whose network stack is modified to support Direct Cache Access (DCA), which aims 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.

Veal and Foong [24] 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 1 to 8 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.* [12] compare the performance of various Web server software deployed on a 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 6,000 Mbps using 4 cores. 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.

Gaud *et al.* [10] evaluated the performance of the Apache Web server on a 4-socket, quad core AMD Shanghai system using the SPECweb2005 benchmark. As mentioned in Section 1, the authors propose a multiple Web replica solution that minimizes migration of Web server data across sockets and show that the solution improves scalability. Our work differs from this study in many aspects. Firstly, our study is more detailed in that we compare scalability for two workloads with very different characteristics and we consider the entire response time distribution. Secondly, we present a fine-grained analysis of the response time distributions observed at various load levels for these two workloads to identify trends specific to static and dynamic requests. Thirdly, we conduct controlled experiments and report more comprehensive hardware monitoring data to identify low-level processor interactions that impact scalability. Finally, we show that for our system the multiple Web replica solution is *not* effective for workloads containing a significant fraction of dynamic requests and hence cannot be prescribed as a general scalability remedy to Web site administrators.

3. EXPERIMENTAL METHODOLOGY

3.1 System Under Study

The Multi-core server under 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 cache. 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) and multiple Receive Side Scaling (RSS) queues that enable the distribution of network interrupts between the cores.

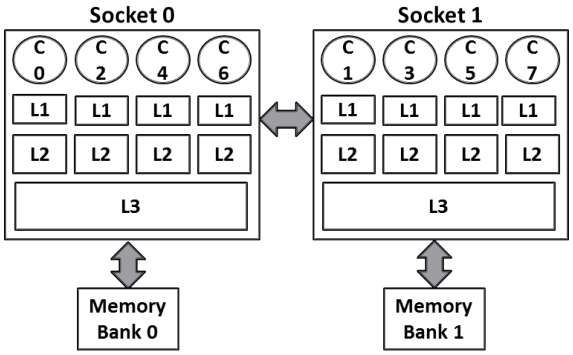

Figure 1: Server Architecture

In this study, we focus on CPU bottlenecks and their impact on scalability. Consequently, we intentionally prevent other resources (e.g., disks, 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 [16]. Lighttpd was chosen as initial tests showed it scaled better than other open source Web servers such as Apache [4]. 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 allowing 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 discussed in Section 3.4, our experiments used 1 worker process per core.

For the multi-tier SPECweb workload, we use the PHP language runtime as the application tier and the FastCGI [8] message exchange protocol as the means of communication between the Web and application server. In this configuration, a PHP-FastCGI process with a set of children are 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 Core2 processors and 2 GB of memory for this purpose, as shown in Figure 2. Each BeSim machine executes an instance of Apache Web server with the FastCGI module. The BeSim systems were carefully monitored to confirm that they are not the bottleneck in our experiments.

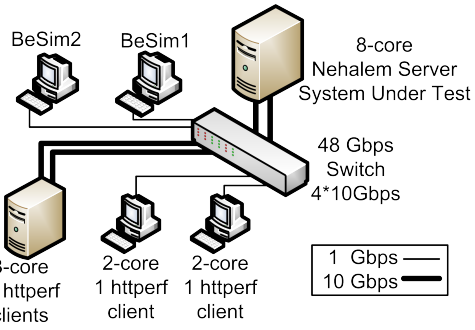

Figure 2: Testbed for Support workload

3.2 Workloads

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 OS 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* [20] Web workload generator. We used 4 instances of *httperf*, each running on a separate dual core client machine to submit this workload to the server. The client machines and the server are connected to each other through a non-blocking 10 Gbps Ethernet switch.

We also consider the SPECweb2009 Support workload. We use this workload to observe system behaviour when there is more application-level activity relative to OS 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 (including the OS 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 [13] *httperf* to submit a workload into multiple server addresses simultaneously. This enhancement is used in the experiments to evaluate multiple replica configuration, that is described in Section 5.4. Eight instances of *httperf* are deployed across two dual core and one eight-core machines, as shown in Fig-

ure 2. 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.

3.3 Performance Metrics

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 99.9^{th} 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 [9]. The response time thresholds differ depending on the Service Level Agreement (SLA) for a particular Web service. For instance, Amazon Dynamo Services uses an SLA where the 99.9^{th} percentile threshold is 300 msec [9]. 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.

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* [22] tool is used, which we confirmed incurred a negligible CPU overhead of around 1%. Low-level hardware monitoring data was collected using Performance Counter Monitor (PCM) [1] 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 our experiments, 95% confidence intervals were less than 1% of their corresponding mean values.

3.4 Initial Configuration Tuning

In this section, we describe our initial configuration tuning exercise aimed at preventing software or OS bottlenecks from limiting the scalability of the server. Table 1 shows the quantitative values of performance improvements achieved through three configuration tunings for both TCP/IP intensive and 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 msec and 5.0 msec, respectively for the TCP/IP and SPECweb2009 Support workloads. We confirmed that the tuning strategy's effect on the higher percentiles is also positive. The three tuning steps are as follows:

Number of Web server processes: The Lighttpd documentation suggests a setting of two HTTP worker processes per core for CPU bound workloads. As the first configuration tuning step, we set this parameter to one. From Table 1, the MAT value was 14% and 1.5% *higher* with this setting 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. Careful workload-specific tuning of this parameter is therefore critical for obtaining good performance.

OS scheduling: As the next tuning strategy, we examined the Linux process affinity mechanism [17] as an alternative to the default Linux scheduler. The default scheduler dynamically migrates Web application processes among cores. We used the affinity mechanism to statically assign each of the 8 Lighttpd worker processes to its own distinct core. From Table 1, core affinity significantly outperforms the default Linux scheduler. The MAT increased 10% and 4.0% 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.

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 was low. Therefore, the network interrupt handling load was distributed between the cores by creating 8 RSS queues on the NIC and affining each of the queues to a distinct core through the Linux *IRQ Affinity Setting* mechanism. From Table 1, 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.

Overall, these changes improved the MAT by 69% and 31% for the TCP/IP Intensive and Support workloads, respectively, relative to the default OS settings. The experiments presented in the next section use these tuned settings.

4. SCALABILITY EVALUATION

In this section, we describe the results of scalability evaluation experiments for the TCP/IP intensive and the SPECweb Support workloads. For each workload we change the load intensity to utilize the server to its capacity. We study the performance of the 8 core server when it has 1, 2, 4, or 8 active cores.

Table 1: Effect of configuration tunings on the server's scalability

Tuning Method	Scalability Improvement	
	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%
Balancing network interrupt handling load	45.0%	20.5%

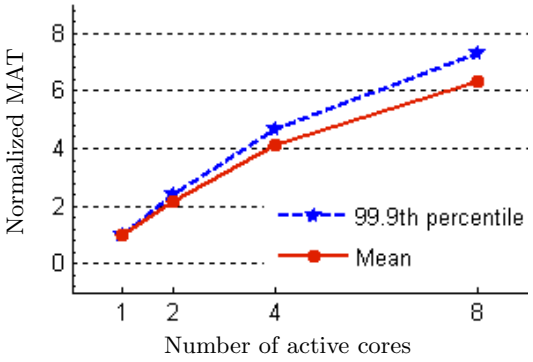

Figure 3: Normalized Maximum Achievable Throughput (MAT) with 1, 2, 4 and 8 active cores for the TCP/IP intensive workload

4.1 TCP/IP Intensive Workload

Figure 3 shows the normalized MAT values with respect to mean and 99.9th percentile response times for 1, 2, 4 and 8 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 msec, a 4 msec threshold is considered for 99.9th percentile of response time for this workload. In the experiments with 2 and 4 active cores, all cores are selected from the same socket. We normalize MAT values with respect to the value observed with 1core. From the figures, the server scales close to proportionally from 1 to 2 cores with respect to both mean and 99.9th percentile of response times. However, for more than 2 cores, the scalability factors starts to degrade. The degradation is more severe for the mean response times. The overall scalability from 1 to 8 core is 6.3 for mean and 7.2 for 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 4 to 8 cores. Moreover, the problem is not confined to the tail, but observed throughout the body of distribution.

Analysis of the server's response time at different load levels reveals that scalability problems become severe when moving from 4 cores, i.e., using 1 socket in the system, to 8 cores, i.e., using both sockets on the system. The measured response time values as a function of CPU utilization are depicted in Figures 4(a) and 4(b) for the mean and 99.9th percentile, respectively. Figure 4(a) show that mean response times only increase gradually till a mean per-core utilization of 80% for experiments with 1, 2 and 4 active cores. However, in the experiment with 8 active cores, the mean response times increase with a larger slope from 60% utilization. A similar trend is observable for the 99.9th percentile of response times in the experiments with 2, 4 and 8 active cores. Interestingly, the 99.9th percentiles are significantly higher with 1 active core. This demonstrates that when only 1 core is active in the system, a subset of requests have significantly high response rimes. We have not yet been able to determine the reason for this behaviour inspite of the large amount of hardware event data that we collected.

In summary, Figure 4(a) confirms the presence of a scalability problem in the system when all 8 cores are active while Figure 4(b) demonstrates that the problem manifests itself throughout the response time distribution. This problem prevents the Web server from fully utilizing available processor resources. In Section 5, we evaluate a Web server configuration strategy which can reduce the impact of this problem.

4.2 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 2 and 4 active cores all cores are selected from the same socket.

Figure 5 shows the MAT values with respect to mean and 99.9th percentile response times for 1, 2, 4 and 8 active cores. We consider thresholds of 5 msec and 500 msec for 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 sublinearly from 1 to 2 cores with respect to the mean response time, the scalability factors improve as we activate more cores. The overall scalability from 1 to 8 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 considerable effect on the scalability factors. However, selecting a lower threshold (e.g. 4 msec and 400 msec) results in a slightly better scalability from 4 to 8 cores.

Figures 6(a) and 6(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 1, 2, 4 and 8 active cores. They demonstrate four main observations related to the server's scalability. First, from Figure 6(a), with all 8 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 2 and 4 active cores have higher mean response times compared to the experiments with 1 and 8 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 2 and 4 core curves relative to the 8 core curve are less for the 99.9th percentile of response times compared to the mean. Finally, while the experiments with 1 active core have the

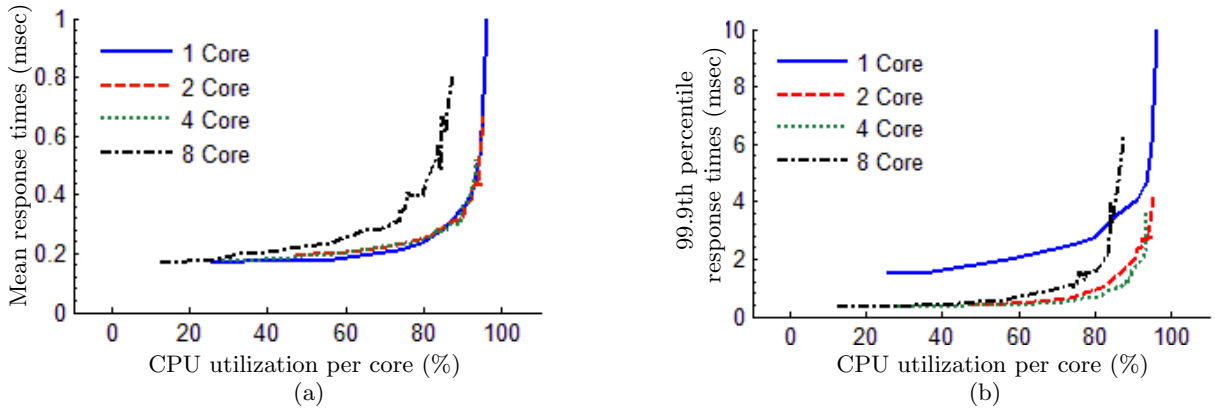

Figure 4: Web server response time vs. CPU utilization for the TCP/IP intensive workload: (a) Mean (b) 99.9th percentile

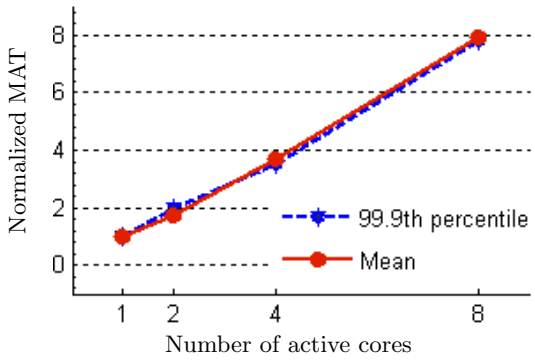

Figure 5: Normalized Maximum Achievable Throughput (MAT) with 1, 2, 4 and 8 active cores for the Support workload

best performance considering the mean response time, the best 99.9th percentile response times is measured with 8 active cores, especially at high load. These observations reveal that activating more cores 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. Moreover, activating another socket considerably improved the scalability.

We next analyze the reasons for the differences in scalability behaviour of the body and tail of the response time distribution. We 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 1, 2, 4 and 8 cores. In each experiment the selected test is the one which causes a CPU utilization of 80% per core. This way we could ensure that the comparison is done at a constant load level relative to the available resources (number of cores). From Figure 7, the response time distributions are almost identical for 1, 2, and 4 cores cases for the first 95 percentiles. Response times are significantly higher when 8 cores are active. The conditions are reversed for the last 5 percentiles. For this portion of requests, the response times in the tests with 2 and 4 core are considerably higher than the response times in the test with 1 and 8 cores. The CDF plots demonstrate that activating 8 cores causes a performance degradation for requests with smaller response times while requests with larger response time benefited from having 8 active cores.

We now focus on the identity of the requests with large response times, which benefit from having 8 active cores, i.e., both the sockets. We use the detailed response time log file output by *httperf* to plot 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 8 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 have the highest response times. Using the *httperf* log files, these response sizes correspond to content that is dynamically generated by the application tier. Therefore, dynamic requests scale well with activating both sockets.

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 4 to 8 cores, i.e., from 1 socket to 2 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 1 socket to both sockets.

Towards understanding the response time behaviour, we now discuss important low-level differences in system usage across the 1 socket and 2 sockets scenarios. When only one socket in the system is active, there is unlikely to be significant migration of data over the inter-socket QPI links since the Linux memory manager tries to allocate memory locally. In contrast, as discussed shortly, in the 2 socket scenario there can be significant data flow across the two sockets through QPI links. Contention for the QPI links can potentially hurt the performance of the 2 sockets case. How-

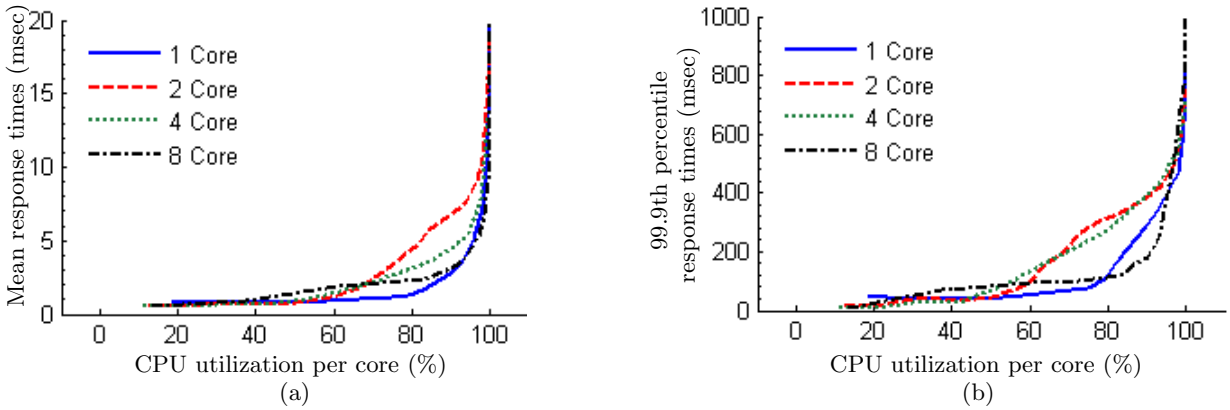

Figure 6: Web server response time vs. CPU utilization for the Support workload: (a) Mean (b) 99.9th percentile

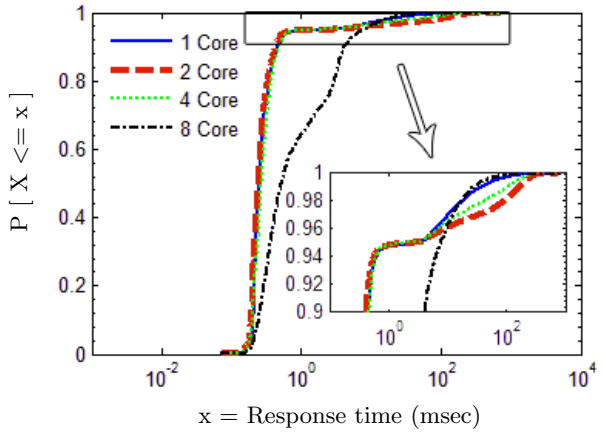

Figure 7: Cumulative Distribution Function (CDF) of response times in 80% CPU utilization for SPECweb Support workload

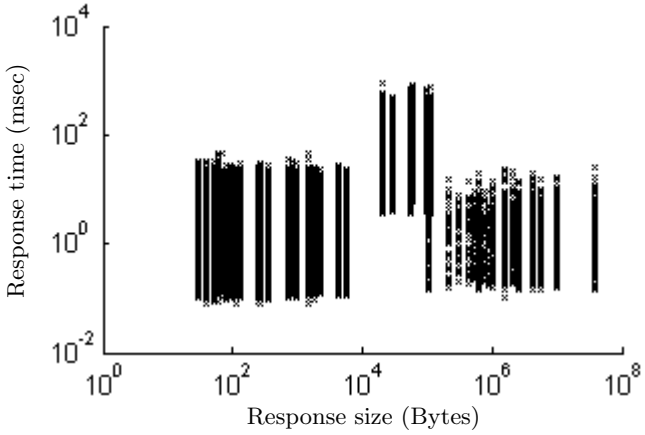

Figure 8: Response time vs. response size for a test with 8 active cores and 80% CPU utilization for the Support workload

ever, the 2 socket case exposes twice the L3 cache capacity to the Web application processes. This can afford performance gains for the PHP-FastCGI processes, which others have shown to be memory-intensive [23][1]. In the next section, we study the relative effects of such interactions in the context of a previously proposed scalability enhancement technique that aims to reduce inter-socket communications.

5. SCALABILITY ENHANCEMENT

In this section, we evaluate the multiple Web site replicas approach suggested recently [10] [13] to enhance the scalability of Multi-core Web servers. To facilitate subsequent discussions, we first provide some background on how network packet processing and application data shared among Web application processes can increase the traffic in the socket inter-connect. We then describe the multiple replicas solution and present experiments to characterize its effectiveness under various circumstances.

5.1 Effect of Network Packet Processing

When the TCP packets carrying an HTTP request arrive at the server, the NIC sends an interrupt to 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 is 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 has processed the packet interrupt. In this case, the data already

[1]To confirm the memory-intensiveness of PHP-FastCGI processes, we compared the memory access traffic for the Support workload with 8,000 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.

exists 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 L3 cache 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 [21]. This causes an increase in the QPI links' traffic which can result in contention in the QPI links at high request rates.

As discussed previously, the server's MSI-X enabled NICs can divide interrupts among a set of hardware RSS queues with packet distribution among the queues performed using a hashing technique. Furthermore, interrupts assigned to each queue can then be bound to a specific core through the *IRQ Affinity Setting* supported by Linux. We show how this feature can be exploited to reduce inter-socket communications in Section 5.3.2.

5.2 Effect of Shared Application Data

Both Lighttpd and PHP-FastCGI processes can share data, e.g., 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 L3 cache of the other socket due to a previous access by another process residing on that socket. This can have a positive impact on performance since it eliminates costly accesses to main memory. However, it triggers migration of shared data across sockets that may cause heightened contention for the QPI links. Hence, the overall performance impact of shared application data depends on which of these two effects dominates.

5.3 Multiple Web site Replicas

The multiple Web site 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 reduce 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.

5.3.1 *Original Configuration*

Figure 9(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 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[2].

The original Web server configuration used in the experiments with 8 active cores involves running a single Web server software instance with 8 Lighttpd processes. Each of these processes is bound to one of the 8 active cores as shown in Figure 9(a). When a request arrives at the server, it will first be processed by one of the queues of the NIC and next 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[3]. As mentioned previously, there are 128 PHP-FastCGI processes per core. These are not shown in the figure for the sake of clarity.

[2]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.

5.3.2 2-replica Direct *Mapping*

An alternative configuration is to run two instances of the Web server software, each having 4 Lighttpd processes and 4 x 128 FastCGI processes bound to the cores of one socket. Each instance is denoted as a Web site "*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 9(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, since both replicas are independent, a significant sharing of data among Web application processes will not trigger QPI traffic. However, as mentioned previously, the absence of sharing among sockets can represent a lost caching opportunity that might result in more main memory accesses. It follows from these discussions that the multiple replicas approach tries to leverage properties of the single socket scenarios discussed in the previous sections while still using both sockets in the system.

5.3.3 2-replica Indirect *Mapping*

To better understand the selective importance of eliminating network packet related QPI traffic and shared application data related QPI traffic, a third configuration is considered. This is done 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 NIC and replica shown in Figure 9(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 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 QPI traffic.

5.4 Evaluation

This section presents experiments performed to evaluate the effectiveness of the multiple replica approach. First, the

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

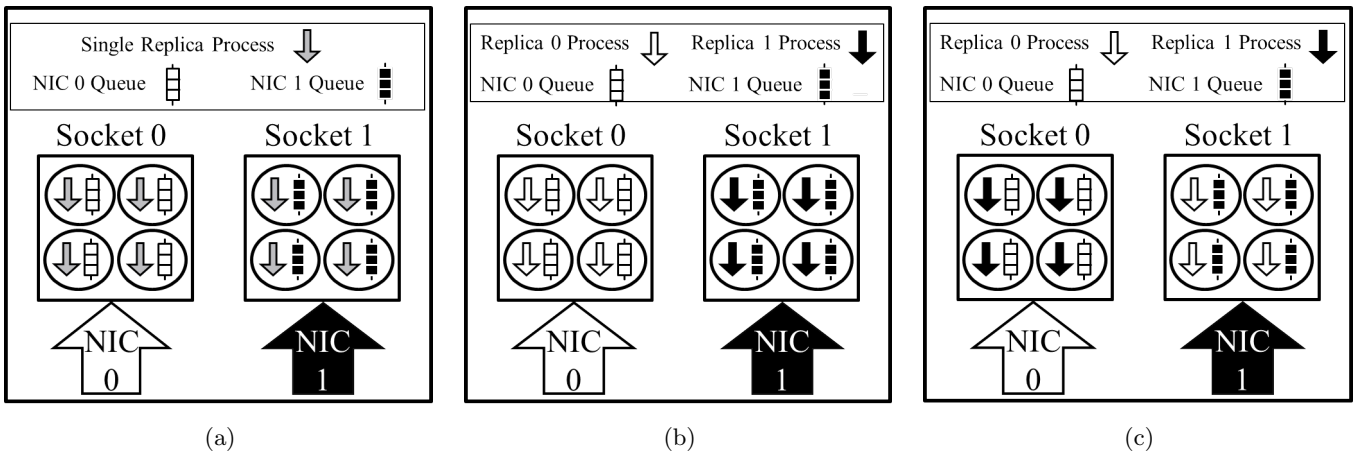

Figure 9: NIC queue and Web server process mappings: (a) Original (b) Direct mapping (c) Indirect mapping

results of experiments conducted using the new configurations under the TCP/IP intensive workload are presented. We then examine the server under the Support workload. Finally, the hardware event data collected during the experiments with both workloads is presented to identify the causes for the observed performance behaviour.

5.4.1 TCP/IP Intensive Workload

Results from the original configuration with 8 active cores are considered as the baseline. Figure 10 shows the mean response times as a function of request rates for the baseline (*1 replica*), *2-replica Direct* mapping and *2-replica Indirect* mappings. From the figure, the MAT (considering a mean response time threshold of 0.5 msec) 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 baseline. 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 500 ms (not shown due to space constraints) 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.

The improved scalability of the Web servers with the *2-replica Indirect* mapping compared to the baseline 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. Since 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, since Lighttpd processes are not memory-intensive [23], this workload does not seem to be impacted by the inability of a replica to benefit from the L3 cache of the socket hosting the other replica.

5.4.2 Support Workload

Figure 11 compares the mean response times for the experiments for the baseline, *2-replica Direct* and *2-replica Indirect* mappings. The figure demonstrates 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 are statistically similar (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 baseline for the *2-replica Direct* and *2-replica Indirect* mappings, respectively. This result contradicts previous studies [10] by indicating that the multiple replica configuration does not always improve a server's scalability.

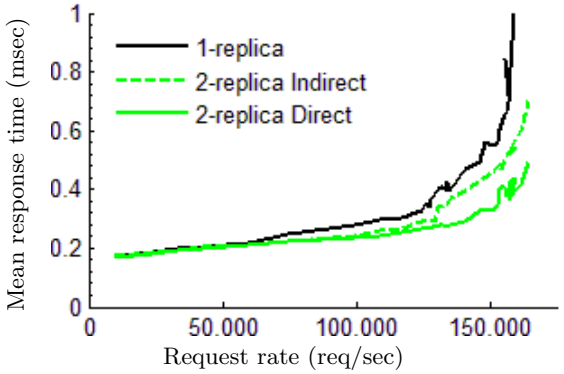

Figure 10: Web server response time vs. request rate for 8 cores with baseline and 2 replica configurations under TCP/IP intensive workload

We now investigate the effects of the multiple replica solution on the response time of the static and dynamic requests in the Support workload. The CDF plot of response times for the baseline, *2-replica Direct* and *2-replica Indirect* mappings for the tests with 80% CPU utilization is depicted in Figure 12. The plot was generated from the output of the tests with 7,000 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 2-replica configurations compared to the baseline. This is analogous to the results obtained from the experiments under TCP/IP intensive workload. On the other hand, the last 5 percentiles of the responses times, mainly dynamic requests, are lower for tests with 2 replicas. The results reveal that the multiple replica configuration degraded performance for the dynamic requests. Since 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.

The results of experiments with Support workload suggest that dynamic requests do not benefit from reducing QPI traffic. Due to the memory-intensive nature of such requests, the inability of PHP-FastCGI processes of one replica to use the cache hierarchy of the socket containing the other replica seems to hurt overall performance. In the next section, we provide more evidence for these claims, using hardware event data collected during the experiments.

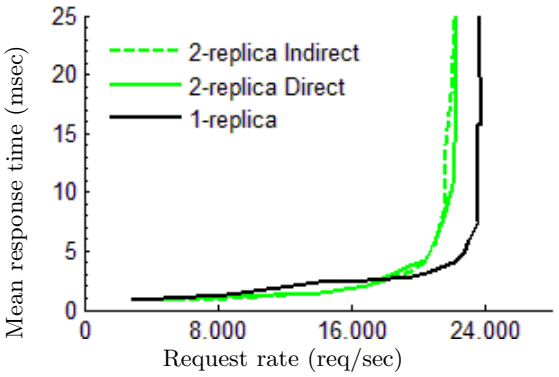

Figure 11: Web server response time vs. request rate for 8 cores with baseline and 2 replica configurations under Support workload

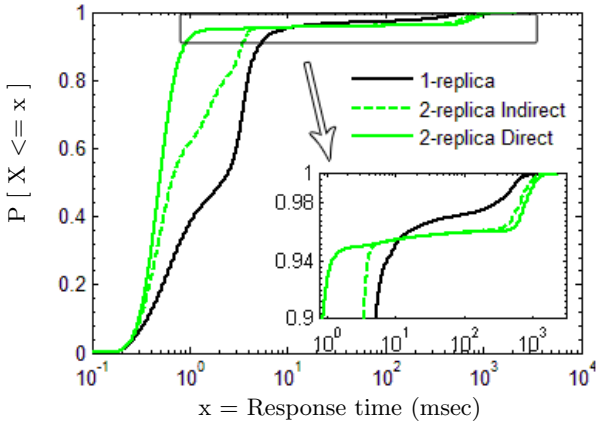

Figure 12: Cumulative Distribution Function (CDF) of response times in 80% CPU utilization with baseline and 2 replica configurations under the Support workload

5.4.3 *Hardware Events Data*

The results presented in Sections 5.4.1 and 5.4.2 revealed that the multiple replica approach improves performance for the TCP/IP intensive workload while degrades it for the Support workload. In this section, we identify possible causes of this performance behaviour. As mentioned is Section 3.3, we collected hardware event data to analyze the root causes of the observed performance characteristics in different stages of this study. To evaluate the effect of using multiple replicas, the socket-inter-connect traffic, L3 cache hit rates, and memory access statistics were collected.

Figure 13(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 the baseline to *2-replica Direct* mapping. Moreover, the QPI traffic is higher than the baseline 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 13(b), the average QPI traffic decreased from the baseline to *2-replica Direct* mapping for the Support workload. Moreover the QPI traffic for the *2-replica Indirect* mapping is higher than the baseline. This confirms that the multiple replica solutions were able to decrease the inter-socket migration for the Support workload as well. However, the amount of reduction is much less than the TCP/IP intensive workload. This is likely due to the fewer TCP/IP activities (lower request rate) for the Support workload which results in fewer inter-socket data migration of network processing data.

We now present measurements to discuss the reasons for the different effects of the multiple replica approach on the response times of static and dynamic requests. From Figure 14, the L3 hit ratio decreases at higher loads for the multiple replica configurations. Figure 15 shows that accesses to the main memory increase in the *2-replica Direct* and *2-replica Indirect* mappings due to a reduction the number of L3 cache hits. This confirms our previous observation that workloads with dynamic requests benefit from a PHP-FastCGI process having access to the cache hierarchies of both sockets in the single replica scenario. The results presented in this Section, confirmed that using multiple Web site replicas does not always improve the servers scalability. However, it is still effective for certain types of workloads.

6. CONCLUSIONS

In this paper, we characterized the behaviour of a Web server deployed on a Multi-core architecture under two different workloads. The non-intrusive configuration changes we explored resulted in significant scalability improvements. Investing time in server tuning allowed us to utilize all of the server's cores up to 80% without causing significant queuing delays for users; even at 90%, delays may be tolerable for peak usage. We believe that such increased utilization would enable popular service providers to deploy fewer servers, thereby reducing both their capital and operating expenses [11]. For example, service providers such as Google and Facebook operate hundreds of thousands of servers; even a 10% reduction in the number of servers required could save these companies non-negligible amounts of money, as well as reducing their environmental footprints.

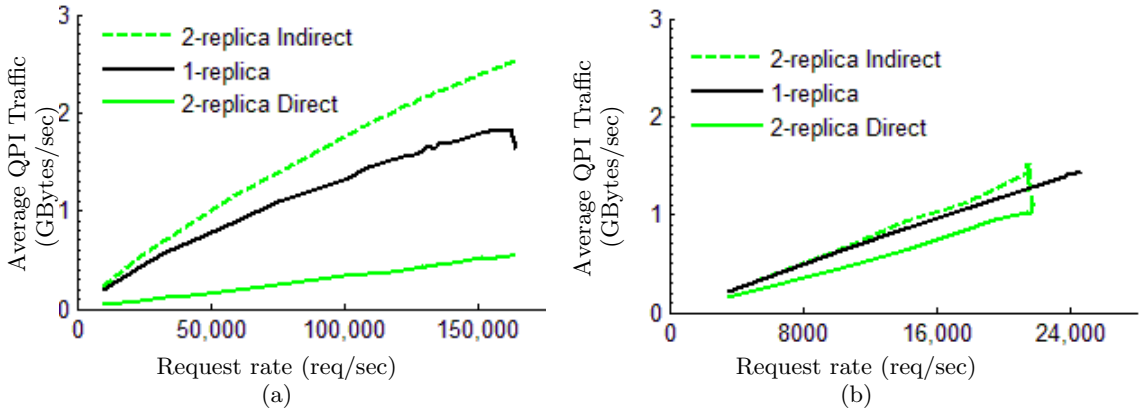

Figure 13: Average QPI traffic vs. request rate for 8 cores with baseline and 2 replica configurations: (a) TCP/IP intensive workload (b) SPECweb Support workload

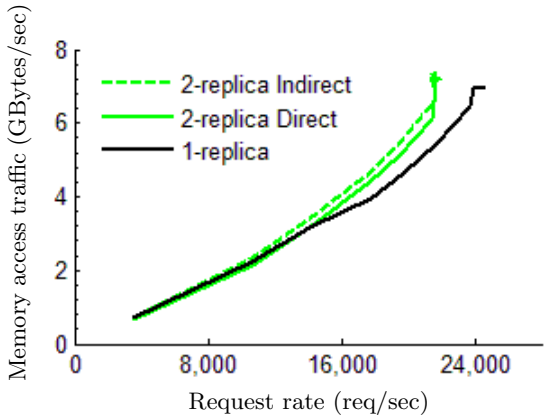

Figure 15: Total memory access vs. request rate for 8 cores with baseline and 2 replica configurations under Support workload

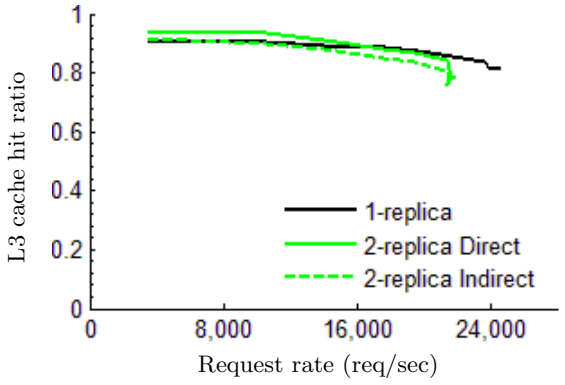

Figure 14: L3 cache hit ratio vs. request rate for 8 cores with baseline and 2 replica configurations under Support workload

Reducing the number of servers required to support a workload could also enable other savings, such as reduced cooling demand in data centers.

Our work studied in detail the relationship between workload characteristics, low level processor usage, and performance. Specifically, detailed experimentation involving two different workloads showed that the server exhibits different scaling characteristics for static and dynamic requests. The performance of static requests handled degrades as the core count increases and is especially poor when both sockets are fully utilized. In contrast, the performance of dynamic requests increases from 1 socket to 2 sockets. We showed that static requests suffer from inter-socket communication triggered while using both sockets in the system. In contrast, dynamic requests handled by the significantly more memory-intensive application tier benefit from the ability to use the cache hierarchy of both sockets.

Due to these findings, our paper establishes that a previously proposed approach to enhance Web server scalability by reducing inter-socket communications [10] [13] may not be effective for workloads with dynamic requests. Our work suggests that scalability enhancement strategies need to be arrived at after careful consideration of workload characteristics. In particular, a Web server configuration can be adapted at runtime to fluctuations in the mix of static to dynamic requests. As an example, consider a popular news Web site. Most news Web sites include dynamic content. However, in the event of an important alert or breaking news that triggers a significant increase in the user traffic, the Web site administrators may disable dynamic content to reduce the load on the server. From our results, the system can in this situation benefit from having multiple Web replicas. The Web site can resume normal operation by disabling those replicas when the user traffic returns to normal.

Our study focused on a widely used processor microarchitecture (Nehalem) and a well-known combination of Web and application servers (Lighttpd, FastCGI-PHP). However, we believe that the scalability behaviour can be different for other systems with different hardware architectures or software configurations. For instance, the performance degradation of dynamic requests due to the use of multiple replicas may not appear in a system with a very large last level cache. On the other hand, a Web server with larger memory footprint, such as Apache, may even suffer performance degradation for static requests. This motivates the need for systematic experimentation to ascertain the best scalability enhancement strategy for a given system. While this time and budget investment may not be of large profit for a small organization, the achieved gain for enterprise Web service

providers accumulates over their hundreds of thousands of servers. As a results, they will be willing to study the scalability behaviour of their workloads on various architectures and adapt servers' configurations accordingly.

We collected detailed hardware level statistics and used this large amount of data to analyze the performance behaviours observed in various experiments. However, 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 99.9^{th} percentile response time in the experiments with 1 active cores 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.

Our future work will focus on other Web server software and hardware platforms and statistical analyses to identify the impact of hardware architectural differences such as cache size and socket inter-connect bandwidth on the behavior of servers. This can be of interest for researchers working on resource sharing problems, such as hosting multiple virtual machines per host. Moreover, we will work on developing an approach to dynamically enable or disable replicas based on the user traffic and the mix of requests experienced by a Web server.

7. REFERENCES

[1] Intel performance counter monitor - a better way to measure cpu utilization. http://software.intel.com/en-us/articles/intel-performance-counter-monitor/.

[2] First the tick, now the tock: Intel microarchitecture (nehalem). White Paper, 2009.

[3] J. Almeida, V. Almeida, and D. Yates. Measuring the behavior of a world-wide web server. In *Proc. HPN*, Apr. 1997.

[4] The apache http server project. http://httpd.apache.org/.

[5] M. Arlitt and C. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, 5:631–645, Oct. 1997.

[6] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, Dec. 2007.

[7] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, and N. Zeldovich. An analysis of linux scalability to many cores. In *Proc. USENIX OSDI*, 2010.

[8] M. R. Brown. Fastcgi: A high-performance gateway interface. In *Proc. WWW*, May 1996.

[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proc. SOSP*, Oct. 2007.

[10] F. Gaud, R. Lachaize, B. Lepers, G. Muller, and V. Quema. Application-level optimizations on numa multicore architectures: the apache case study. Research Report RR-LIG-011, LIG, Grenoble, France, 2011.

[11] P. Gill, M. Arlitt, N. Carlsson, A. Mahanti, and C. Williamson. Characterizing organizational use of web-based services: Methodology, challenges, observations, and insights. *ACM Trans. Web*, 5(4):19:1–19:23, Oct. 2011.

[12] A. S. Harji, P. A. Buhr, and T. Brecht. Comparing high-performance multi-core web-server architectures. In *Proc. SYSTOR*, Jun. 2012.

[13] R. Hashemian. Revisiting web server performance and scalability. Master's thesis, University of Calgary, Nov. 2011.

[14] A. Kumar, R. Huggahalli, and S. Makineni. Characterization of direct cache access on multi-core systems and 10gbe. In *Proc. IEEE HPCA*, Feb. 2009.

[15] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *Proc. ACM/IEEE SC*, Nov. 2007.

[16] Lighttpd web server. http://www.lighttpd.net/ .

[17] R. Love. Cpu affinity. http://www.linuxjournal.com/article/6799.

[18] D. A. Menasce and V. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[19] J. C. Mogul. Network behavior of a busy web server and its clients. Technical Report WRL-95-5, DEC Western Research Laboratory, Palo Alto, CA, Oct. 1995.

[20] D. Mosberger and T. Jin. httperf: a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26:31–37, Dec. 1998.

[21] T. Scogland, P. Balaji, W. Feng, and G. Narayanaswamy. Asymmetric interactions in symmetric multi-core systems: Analysis, enhancements and evaluation. In *Proc. ACM/IEEE SC*, Nov. 2008.

[22] M. Seger. collectl. http://collectl.sourceforge.net/Documentation.html , 2008.

[23] T. Suzumura, S. Trent, M. Tatsubori, A. Tozawa, and T. Onodera. Performance comparison of web service engines in php, java and c. In *Proc. IEEE ICWS*, Sept. 2008.

[24] B. Veal and A. Foong. Performance scalability of a multi-core web server. In *Proc. ACM/IEEE ANCS*, Dec. 2007.