5.1 The Preliminary Benchmarking Experiments

The preliminary benchmarking experiments performed for the shortlisted web servers revealed a number of significant insights. The aim of these experiments was to further shortlist these web servers based on their performance so that more detailed performance intensive and scalability driven experiments can be performed more effectively. These experiments were divided into categories based on the type of the workload invoked and the underlying operating system platform used. First set of experiments were performed for static workloads for Windows and Linux platforms while the second set involved dynamic workloads for Windows and Linux platforms. The comparisons were thus made between the various web servers on a particular platform, invoking a particular type of workload. The result of the first set of experiments involving static workloads for Windows environment highlighted that Apache web server exhibited considerably higher performance than all other web servers. The Microsoft IIS web server came close to Apache followed by the other web servers. As listed in Table 4.1, the difference in the performance between Apache and IIS web server for static workloads was less than 14% whereas the Apache web server was found to be more than twice as fast as the Oracle iPlanet web server. Next, the results of same experiments involving static workloads suggested that TUX web server did not saturate at all and lead the performance results among all web servers. The reason being that TUX is a kernel-mode web server and avoids data copies, scheduling and context switching overhead (Aaqib et al., 2011). The use of kernel-mode TUX web server for static content thus reduces the overall communication overhead in socket layer (Joubert et al, 2001). The performance of Apache web server also increased by about 11% for the same experiment
on Linux platform. The *Nginx* web server also gained about 15% performance while running on Linux platform. All other web servers were also found to perform better on Linux platform with *LightTPD* gaining about 85% performance on Linux environment than on Windows. Ramana et al., (2005) also came to same conclusion in their work and had found that *Apache* on Linux yields better performance than *Apache* on Windows.

For *CGI* based dynamic workloads, *Apache* web server exhibited better performance than all other web server on both Windows and Linux environments. The performance of *Apache* on Linux was still better than *Apache* on Windows. But the web servers did not perform as good for dynamic workload as they performed for static workloads. The reason for the reduction in the performance of web servers for dynamic requests is that the scripts generated using *CGI* are more computational and processor intensive (Iyengar et al., 1997; Almeida et al., 1998; Arlitt et al, 2001 & Aaqib et al., 2011). The reason for the decrease in the performance of *TUX* web server using dynamic workloads is that it could not take advantage of the network and memory optimizations as it did for the experiments involving static-only workloads. Another reason for the decrease in the performance of *TUX* web server is its inherent techniques of accepting connections. The *TUX* web server accepts new connections more insistently, instead of making forward progress on already accepted connections (Arjan, 1999). Mendes et al., (1998) too obtained similar results in their work by concluding the fact that large dynamic requests degrade the performance of a web server and have strong impact on its performance.

At the end of these preliminary benchmarking experiments, the web servers which exhibited better performance included *Apache* web server, Microsoft *IIS* and kernel-mode
TUX web server. These web servers were thus selected for the more detailed experimental evaluation stressing computational intensive, retrieve intensive and scalability factors.

### 5.2 Kernel-Mode and User-Mode Web Servers

After shortlisting the web servers, results of the detailed performance evaluation between kernel-mode TUX and user-mode Apache web server revealed that the kernel-mode web servers have a significant performance improvement over user-mode web servers on Linux platform. The kernel-mode web server for static workloads on Linux was up to ten times faster than user-mode web server for small-workload file. As the size of the workload file was increased this difference decreased by up to 32%. The reason for the big improvement in achieved rate and throughput is the various generic design advantages of kernel-mode web servers (Arjan, 1999 & Joubert et al, 2001). Throughput was improved the most using kernel-mode web servers but this difference decreased as medium and large workload files were used. The main reason behind the higher throughput from the TUX is it avoids data copies, scheduling overhead and context switching overhead. Also, it reduces the overall communication overhead in the socket layer. Thus, it resulted in handling large number of requests per second and also helped in increasing the throughput. By comparing the absolute performance between these two types of web servers, it was revealed that kernel-mode web servers are faster than user-mode web servers for static workloads. Thus, there is a need to further improve the user-mode web servers, interfaces and mechanisms to match with the performance obtained using kernel-mode web servers for static workloads. For dynamic workloads, the performance difference between kernel-mode and user-mode web server became
negligible. The *TUX* web server in the experiments crashed a number of times resulting in a whole system failure. For user-mode web servers, this problem did not arise as the web server even after crashing could be easily restarted without affecting any other running applications. Thus, kernel-mode web servers should be worth considering from a performance point of view even though kernel-mode web servers do not provide add-on security and reliability benefits as in user-mode web servers.

### 5.3 Compute and Retrieve Intensive Workloads

An exhaustive experimental evaluation for the analysis of computational and retrieve intensive workloads for the performance of web servers on Windows and Linux environments yielded many interesting results. The results of this work compared and highlighted the affinity of dynamic web technologies like *PERL*, *PHP* and *Java Servlets* with *Apache* web server running on Windows and Linux environments. Parallel comparisons performed for computational and retrieve intensive workloads running on windows and Linux environments also provided insights into the affinity of web technologies with a particular operating system platform. The first experiment in which a non-intensive workload was invoked for both Linux and Windows environment revealed that performance on Linux was considerably higher than the performance on Windows. Affinity comparison revealed that *PERL* on Linux has upto 13% higher performance than *PHP* and upto 21% higher performance than *Java Servlets* for these non-intensive workloads. Similar performance and affinity improvements were observed on Windows platforms. For the compute intensive applications it was observed that performance of *PERL* decreased whereas the significant performance gains were achieved using *PHP*. The reason is that *PERL* handles small dynamic content requests very efficiently, but
struggles while processing computational and retrieve intensive workloads. The same result has also been established by Kothari et al., (2001). On the other hand, PHP performed considerably well for high computational workloads and with even those involving heavy database retrievals. The tradeoffs in terms of high performance for personalization of web content and the overhead for database access come at a price. It was also observed that the web server performance under high computational and retrieve intensive workloads can be very unpredictable as many a times during the evaluation misleading results were obtained for some experiments which only after validation were established to hold true. Ramana et al., (2001) and Titchkosky et al, (2005) in their work had reached to similar conclusion that the affinity of PHP is better compared to other dynamic web technologies. The results of these experiments were fitted to a multiple linear regression model to predict the performance and validate the results. Results showed that the model predicted the performance of compute and retrieve intensive application for all three technologies i.e. PHP, PERL and JAVA within 8-10% of the measured values. Work by various researchers (Kothari et al., 2001; Checchet et al., 2002; Swales et al., 2003; Checchet et al., 2003 and Trent et al., 2008) have also found similar performance and affinity results which also validates these results.

5.4 Scalability Analysis of Web Servers

Table 4.6 lists the results obtained from the experiments using the nine different cluster configurations for the Apache and μserver web servers. It also lists the maximum throughput and CPU utilization achieved. For evaluation of vertical scalability, measurements of throughput were performed for $p$ number of processor cores ($p = 1, 2$ and $4$). For horizontal scalability, measurements of throughput were performed for $n$
number of server nodes \(n=1, 2\) and 4). Table 4.6 shows the results of these measurements for *Apache* and *μserver* web server. The determination of the cause of server saturation while trying to scale-up or scale-out multi-threaded *Apache* and event-driven *μserver* web server using a web server cluster provided many insights. The results of the analysis indicated a negligible difference in the behavior of the system for vertical versus horizontal scaling. Measurements revealed that almost same throughput was recorded when the server were scaled-up or when the server were scaled-out. The impact of adding more server nodes as in horizontal scaling did not improve performance to a greater degree. As can be seen in Table 4.7, the measured throughput values for *Apache* web server running on a single server having 2 active processor cores is approximately equal to the same *Apache* web server running on a cluster server comprising of two nodes having one processor core active. The factor that was used for the comparison was the number of active processor cores \(p\). The scalability evaluation results have thus showed comparable results for the experiments where the number of active processor cores \(p\) are equal across the cluster server regardless of the number of nodes \(n\). The flattening curve in Fig 4.35 comparing the relative capacity \(C(p, q)\) values indicated the thinning returns achieved while adding more processors. This implied that servers eventually had reached a point at which it was no longer justifiable to add more resources. The analysis of the relative capacity values showed that the measurements for the small number of processor cores appeared to be ‘near linear’ for \(p = (1, 2, 4)\) and for \(p = (8, 16)\), the relative capacity values revealed sub-linear results. Also, while comparing the performance of multi-threaded *Apache* and event-driven *μserver* web server, the difference in the performance is due to the internal architecture of the web server itself. The most likely explanation for
this difference is if the underlying web server architecture has not been developed keeping in consideration the principles of scalability, the presence of the multiple processors would not help in scaling the performance of the web server. Although, it may be tempting to generalize this result and conclude that vertical scalability can be used in all situations to get comparable performance and discredit the horizontal scaling as a costly overhead for all cases. But as the number of processor cores increase in the system, more significant amount of inter processor communication increases which eventually results in the loss of computing power (Gunther, 1993). This phenomenon is called the MP Effect, which results in the loss of computing capacity that occurs when more processors to a single platform are added. This loss of capacity is due to additional overhead and/or contention between processors for shared system resources, for example, the system bus (Artis, 1991 & Gunther, 1996). The results presented here have also been validated by comparing them with the similar conclusions drawn by researcher for vertical scalability (Guitart et al., 2005) and by Veal et al, (2007) and Haddad et al., (2004) for general scalability evaluations. The scalability of a web server system thus depends on the characteristics of both the web server architecture and the execution environment. If the web server architecture is not structured into units that could execute independently in parallel, regardless of the number of processor cores or server nodes added, the results will not be scalable. Thus, there is a need to improve the operating system interfaces and the web server architectures to make them work in parallel to exploit the benefits of resources that are added to it vertically or horizontally. The identification of bottlenecks was also found to be important and its impact on the
scalability of a web server system was found to be crucial. All these considerations should never be overlooked before planning to undertake expensive system upgrades.

5.4 Design, Implementation and Evaluation of Java-based hybrid web server

The evaluation and comparison of performance characteristics of Java-based hybrid experimental web server, multi-threaded Apache and event-driven μserver web server was performed. The two experimental scenarios comprising of static and dynamic workloads were thoroughly investigated. Achieved response rate was measured for all three types of web server architectures and as per the graph in Figure 4.46, the multi-threaded Apache web server exhibited lower performance than the other two web servers for static workloads. The performance of μserver and the experimental hybrid web server was nearly equal. The recorded peak saturation point of μserver, Java-based hybrid and Apache web server is 1711 reqs/sec, 1723 reqs/sec and 1619 reqs/sec respectively. Thus, the experimental hybrid web server gained about 0.7013% and 6.423% performance in terms of achieved response rate than μserver web server and Apache web server respectively. It was observed that the same amount of performance as obtained by the experimental hybrid web server with thread pool size of 10 thread whereas for Apache web server about 400 threads were utilized for the same experimental result. For this reason, the implementation of experimental hybrid web server offered better performance than the multi-threaded Apache web server. It was also observed that when the system was not saturated (upto the target request rate of 1500 requests per second), the achieved response rate for the multi-threaded Apache, event-
driven μserver and the experimental web server were almost same. The reason for the lesser performance of Apache web server in this case is that it had spawned about 400 threads already and now the overhead of extra operations was costing the system for keep up with its performance. Another observation was the number of errors recorded during the tests for all the three web servers. These errors included the closing of the connection due to elapsing of client inactivity period, outrunning of the file descriptors etc. The client timeout in case of the experimental web server was determined by setting \textit{CONN\_TMT} variable. The results showed that while the multi-threaded Apache web server produced a lot such errors while reaching their saturation phase, the experimental web server did not produce any such results. The reason for this is that multi-threaded Apache web server does not free worker threads more quickly. In case of the experimental web server, the incoming requests are assigned to the worker threads as work units rather than to full client connections. This technique is also implemented in the event-driven μserver which allows it to avoid the problems of reestablishing the disconnected client connections due to timeout. For the next experiment involving dynamic workloads, it was noticed that the performance of Apache web server started degrading just after the target request rate of 500 requests per second. The multi-threaded Apache web server, exhibited lower performance than other two web servers and attained saturation level at the target request rate of 2000 reqs/sec. As per the results of the experiments, it was found that the performance of the experimental hybrid web server was better than multi-threaded Apache and slightly better than event-driven μserver. The reason behind the degrading of the performance of multi-threaded web server can be explained by the fact that a pure multi-threaded design does not free the worker threads
till the full client connection to which it has been assigned is closed. And in case of systems where the number of client connections increases enormously due to high loads, to satiate the incoming client connections, some of the worker threads waiting on idle connections are terminated due to the client timeout, so that they can be assigned to newer requests. But this situation, leads to a scenario where those connections which were earlier terminated by the server require to be reconnected again. But a server goes through problems for reconnection because the number of clients trying to establish or reestablish a connection is extraordinarily higher than the number of worker threads available in the server pool. This problem thus results in total saturation of the system, as the probability of successfully completing a client connection reduces approximately to zero. The disadvantage of the experimental web server is that for situations where the load on the web server is low, the extra web server operation that the hybrid based server must perform in order to register socket channel selector and to switch them between blocking and non-blocking mode incurs heavy overhead compared to multi threaded or event-driven web servers where there is no such overhead for lesser server loads.

5.5 Study of Security Mechanisms In-Place in Web Servers

A case study presented in Section 4.6.3 compared the security mechanism in various web server systems. This case study was used to check the number of best security mechanism in place in these web server systems. A number of significant insights were gained from this study. One of the interesting observations was that the two web servers of different version from a same vendor, showed different results in this study. Such different results were obtained for a same web server while comparing their installations on different platforms. The reason being the security of a web server is not dependent only on the web
server software only but it is also characterized by the underlying operating system architecture, the network management and its configuration. For example, while comparing the same *Apache HTTPd* server on Scientific Linux *CERN* and Windows XP 2000, it was found that *Apache* on SLC *CERN* system passed more tests and thus was more secure. Another aspect used in this case study was the comparison of different web server system, of different versions and operating system support. While comparing the security mechanism in *Apache Tomcat 6.0.13* and *Apache Tomcat 6.0.16* on Windows Server 2003 platform, it was revealed that *Apache Tomcat 6.0.16*. Here also the explanation is the support provided by the underlying operating system platform and its security configuration. Among all the web servers under study, it was found that Microsoft *IIS* passed more number of tests than any other web server and thus implements higher number of security mechanisms. The only limitation of this approach is that the execution of these tests requires immense computer expertise as the approach requires verification of mechanisms in-place for different web servers systems.

...