Accurate Load Monitoring for Cluster-based Web Data-Centers over RDMA-enabled Networks

KARTHIKEYAN VAIDYANATHAN, HYUN-WOOK JIN, SUNDEEP NARRAVULA AND DHABALESWAR K. PANDA

Technical Report
OSU-CISRC-7/05-TR49

Accurate Load Monitoring for Cluster-based Web Data-Centers over RDMA-enabled Networks[*]

Karthikeyan Vaidyanathan Hyun-Wook Jin Sundeep Narravula Dhabaleswar K. Panda

Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{vaidyana, jinhy, narravul, panda}@cse.ohio-state.edu

Abstract

Monitoring a pool of resources in a cluster-based web data-center environment can be critical for successful deployment of applications such as web servers, database servers, etc. In particular, the monitored information assists system-level services like load balancing in enabling the data-center environment to efficiently adapt to the changing system load and traffic pattern. This information is not only critical in terms of accuracy and content, but it must also be gathered without impacting performance or affecting other applications. In this paper, we propose two accurate load monitoring schemes, namely, user-level load monitoring (ULM) and kernel-level load monitoring (KLM) in a web data-center environment and evaluate its benefits with respect to overall system load balancing. In our approach, we use the Remote Direct Memory Access (RDMA) operation (in user space or kernel space) provided by RDMA-enabled interconnects like InfiniBand. We further leverage the information provided by certain kernel data structures in designing these schemes without requiring any modifications to the existing data-center applications. Our experimental results show that the KLM and ULM schemes achieve an improvement of 22% and 12% in a single data-center and an improvement of 25% and 11% per web-site in shared data-centers, respectively. More importantly, our schemes take advantage of RDMA operations in accessing portions of kernel memory that is not exposed to user space for accurate load monitoring. Further, our design is resilient and well-conditioned to the load on the servers as compared to two-sided communication protocols such as TCP/IP.

Keywords: *Web Data-Centers, Load Monitoring, Remote Direct Memory Access (RDMA), InfiniBand, and Clusters*

[*]This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #CCR-0204429, and #CCR-0311542.

1 Introduction

With the increasing adoption of Internet as the primary means of interaction and communication, highly scalable and available web servers have become a critical requirement. A cluster-based web data-center has been proposed as a cluster architecture to provide scalable web services, which in turn is distinguished from high performance computing systems in several aspects. In particular, web data-centers are comprised of several software components such as proxy servers and web servers which have vastly different system resource requirements. Usually front-end servers of the data-center such as the proxy servers perform communication services between the network and the back-end server farm for providing edge services such as load balancing, security, caching, and others. The back-end servers consist of application servers that handle transaction processing and implement data-center business logic and database servers that hold a persistent state of the databases.

Since the front-end servers forward requests to back-end servers, problem of load balancing the requests among the nodes in back-end is critical for providing scalable web services. Request patterns seen inside the data-center over a period of time, may vary significantly in terms of the popularity of the content. In addition, requests themselves have varying computation time. Small documents get served quickly while large documents take more time to get transferred. Similarly dynamic web-pages take varying amount of computation time depending on the type of the client request. Due to these complex issues, balancing the requests in order to efficiently utilize all the nodes in a data-center becomes a challenging problem to solve. Moreover, in the past few years several researchers have proposed data-centers providing multiple independent web services, popularly known as shared web data-centers [6, 7]. For example, ISPs and other web service providers host multiple unrelated web-sites on their data-centers. The requests for these different web-sites compete with each other for the resources available in data-center, thus further complicating this problem.

Several researchers have proposed and evaluated various load balancing policies for cluster-based web servers [8, 11, 14, 23, 24]. As mentioned in [12], lot of them are valuable in general, but not all of them can be applicable to current or next generation web data-centers. Many of the policies focus on coarse-grained services and ignore the fine-grained services. Fine-grained services introduce many challenging issues of which most important one is how to provide *accurate* load monitoring.

The traditional load monitoring mechanisms are based on send/recv communication and cannot provide an accurate detection of fluctuating system load on the back-end. The send/recv communication based design requires separate processes running on front-end and back-end to monitor the load of back-end nodes in which the processes on the back-end nodes send load information to the processes on the front-end for every load sampling time interval. Thus if these processes on the back-end nodes are delayed to be scheduled because of heavy load, they cannot provide an accurate picture of the load information periodically. While capturing highly fluctuating load information requires smaller load sampling intervals, these load monitoring processes would increasingly compete with the back-end server applications for CPU resources. A previous research [16] shows that usage of such two-sided communication for sharing of information in highly loaded data-centers has significant adverse impact on achievable performance. More importantly, as the server applications running on the back-end nodes become diverse, simple load information (e.g., CPU usage) is not sufficient to accurately represent load status of the back-end nodes. For example, although CPU usage is very low there could be many pending I/O requests, which can result in very high CPU usage in near future. So we need a mechanism to obtain more detailed system information.

To address these issues, in this paper, we suggest the use of accurate load monitoring schemes in the web data-center environment by leveraging the Remote Direct Memory Access (RDMA) operations of modern high speed interconnects such as InfiniBand. The RDMA operations allow the network interface to transfer data between local and remote memory buffers without any interaction with the operating system or processor intervention. Moreover, we suggest a kernel-level support to allow the load monitor to access detailed system information of back-end nodes. We propose two schemes, User-level Load Monitoring (ULM) and Kernel-level Load Monitoring (KLM), and evaluate their benefits in providing accurate load monitoring for load balancing. Especially, the KLM scheme exploits certain kernel data structures that are not exposed to the user space in performing efficient load balancing. Further, such an implementation removes the need for a separate process on server nodes for load monitoring, thereby avoiding wastage of CPU cycles. This work contains several research contributions:

1. We propose novel schemes for accurate load monitoring in web data-centers. These schemes require minimal changes to the legacy data-center applications. We further utilize the advanced features provided by modern interconnects such as InfiniBand.

2. It takes the first step toward using RDMA operations in the kernel space for remote load monitoring in web data-centers.

3. Our schemes can be used to assist load balancing by monitoring the load between any two tiers of the web data-center. Hence our schemes are applicable to multi-tier data-centers and can result in significant benefits.

4. RDMA operations in kernel space extends the idea of reading the entire physical memory from remote nodes in monitoring the status of the machine. This idea opens up a new research direction in fault tolerance, monitoring machine status remotely even if the CPU is down.

We implement our schemes on Apache based web data-center. The experimental results show that both KLM and ULM schemes achieve an improvement of 22% and 15% in a single data-center and an improvement of 35% and 25% in shared data-centers, respectively.

The rest of the paper is organized as follows: Section 2 describes the background. In Section 3, we discuss the design issues and implementation details of our schemes. Section 4 presents the potential benefits that we can expect from our schemes. The experimental results are presented in Section 5 and related work in Section 6. We draw our conclusions and discuss possible future work in Section 7.

2 Background

2.1 InfiniBand Architecture and RDMA

InfiniBand Architecture (IBA) is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. A typical IBA cluster consists of switched serial links for interconnecting both processing nodes and I/O nodes. The IBA specification defines a communication and management infrastructure for both inter-processor communication as well as inter and intra node I/O. IBA defines also built-in QoS mechanisms which provide virtual lanes on each link and define service levels for individual packets. In an IBA network, processing nodes are connected to the fabric by Host Channel Adapters (HCA). Their semantic interface to consumers is specified in the form of IBA Verbs.

IBA mainly aims at reducing the communication processing overhead by decreasing the number of copies associated with a message transfer and removing the kernel from the critical message passing path. The InfiniBand communication stack consists of different layers. The interface presented by Channel Adapters to consumers belongs to the transport layer. A queue-based model is used in this interface.

Each Queue Pair (QP) is a communication endpoint. A Queue Pair consists of a send queue and a receive queue. Two QPs on different nodes can be connected to each other to form a logical bi-directional communication channel. An application can have multiple QPs. Communication requests are initiated by posting Work Queue Requests (WQRs) to these queues. Each WQR is associated with one or more pre-registered buffers from which data is either transferred (for a send WQR) or received (receive WQR).

IBA supports two types of communication semantics: channel semantics (send/recv communication model) and memory semantics (RDMA communication model).

In channel semantics, every send request has a corresponding receive request at the remote end. Thus there is a one-to-one correspondence between every send and receive operation.

In memory semantics, RDMA operations are used. These operations are transparent at the remote end since they do not require the remote end to involve in the communication. Therefore, an RDMA operation has to specify both the memory address for the local buffer as well as that for the remote buffer. There are two kinds of RDMA operations: RDMA Write and RDMA Read. In an RDMA write operation, the initiator directly writes data into the remote node's memory. Similarly, in an RDMA Read operation, the initiator reads data from the remote node's memory.

2.2 Cluster-Based Shared Web Data-Centers

A clustered data-center environment essentially relies on the benefits of a cluster environment to provide the services requested in a data-center environment (e.g., web hosting, transaction processing). As mentioned earlier, researchers have proposed data-centers that provide multiple independent services, such as hosting multiple web-sites, forming what is known as shared web data-centers.

Figure 1 shows a logical higher level layout of a shared web data-center architecture hosting multiple web-sites. External clients request documents or services from the data-center over the WAN/Internet through load-balancers using higher level protocols such as *HTTP*. The load-balancers on the other hand serve the purpose of exposing a single IP address to all the clients while maintaining a list of several internal IP addresses to which they forward the incoming requests based on a pre-defined algorithm (e.g., round-robin). Typically, the front-end tiers of the data-center form the load balancers of the inner tiers.

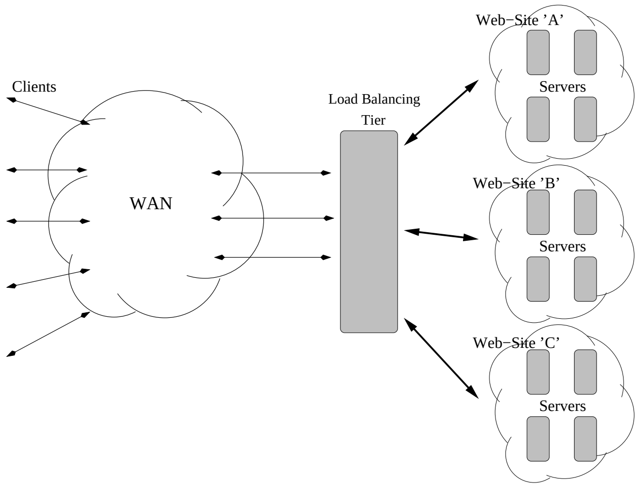


Figure 1. A Shared Cluster-Based Web Data-Center Environment

While hardware load-balancers are commonly available today, they suffer from being based on pre-defined algorithms and are difficult to be tuned based on the requirements of the data-center. On the other hand, though software load-balancers are easy to modify and tune based on the data-center requirements, they can potentially form bottlenecks themselves for highly loaded data-centers. Requests can be forwarded to this cluster of software load-balancers either by the clients themselves by using techniques such as DNS aliasing, or by using an additional hardware load-balancer.

The back-end servers of the clustered data-center provide the actual data-center services such as web-hosting, transaction processing, etc. Several of these services such as CGI scripts, Java servlets and database query operations can be computationally intensive. This makes the processing on the server nodes more CPU intensive in nature.

3 Design and Implementation of Accurate Load Monitoring

In this section, we describe the basic design issues involved in monitoring the server's load and the details about the implementation of the schemes we propose: User-level

Load Monitoring (ULM) and Kernel-level Load Monitoring (KLM).

3.1 Basic Load-Monitoring Support

In this section, we describe the basic design for our RDMA based load monitoring, which is applied to both ULM and KLM schemes.

Support for existing applications: A number of applications exist that allow highly efficient user request processing. These have been developed over a span of several years and modifying them to allow dynamic load-balancing is impractical. To avoid making these cumbersome changes to the applications, our design makes use of *external helper modules* which work along with the applications to provide effective dynamic load-balancing. Tasks related to system load monitoring, maintaining global state information, load-balancing, etc. are handled by these helper modules in an application transparent manner. The modules running on each node in the front-end balance the requests in the data-center depending on current load patterns. The applications on the other hand, continue with the request processing, unmindful of the changes made by the modules.

Front-end based load monitoring: Two different approaches could be taken for load-monitoring: back-end based load monitoring and front-end based load monitoring. In back-end based load monitoring, when a back-end node detects a significant load on itself, it informs the front-end (i.e., load balancer) about its high load. Though intuitively the loaded node itself is the best node to determine its load (based on its closeness to the required data and the number of messages required), performing load-monitoring on this node adds some amount of load to an already loaded server. Due to this reason, load-balancing does not happen in a timely manner and the overall performance is affected adversely. On the other hand, in front-end based load monitoring, the front-end servers detect the load on the back-end, appropriately choose the k least loaded servers, and perform the load balancing accordingly. Choosing k servers instead of just one least loaded server is a simple method to avoid load implosion. Hence all future requests get well distributed among a set of k nodes rather than implode the least loaded node. In this paper, our design is based on front-end load monitoring approach.

RDMA read based design: As mentioned earlier, by their very nature, the back-end nodes are compute intensive. Execution of CGI-Scripts, business-logic, servlets, database processing, etc. are typically very taxing on the server CPUs. In such environment, though in theory the back-end nodes can share the load information through explicit two-sided (i.e., send/recv) communication, in practice, such communication does not perform well [16]. InfiniBand, on the other hand, provides one-sided remote memory operations (i.e., RDMA) that allow access to remote memory without interrupting the remote CPU. In our design, we use RDMA read to perform efficient front-end based load monitoring. Since the load balancer is performing the load monitoring with no interruptions to the back-end nodes CPUs, this RDMA read based design is highly resilient to server load.

With these basic supports, we propose two novel schemes which can support accurate load monitoring in a data-center environment: User-level Load Monitoring (ULM) and Kernel-level Load Monitoring (ULM). We present these two schemes in the following sub-sections.

3.2 User-level Load Monitoring (ULM)

In order to monitor the load in a data-center, we need to collect system statistics and propagate this information to the front-end nodes. In the ULM scheme, we use two different kinds of helper modules running on the front-end nodes and the back-end nodes, respectively. The helper module in the back-end takes care of connection management by creating connections to all the front-end nodes. Further, the module also does exchange of queue pairs and memory handles so that the front-end helper module can perform the RDMA read operation on specific memory areas registered by the back-end in the user space. The front-end helper module after getting the memory handles, periodically performs RDMA read operation and gets the updated load information, which is used by the load balancer. The back-end helper module constantly calculates the CPU cycles spent for every time interval from */proc* and copies this information onto a user registered memory.

We use the */proc* file system for gathering several system statistics. The */proc* file system provides several information about processes running currently, CPU statistics, memory and I/O statistics, etc. However, for load monitoring we focus on */proc/stat* file which gives us information about kernel statistics like CPU cycles spent, total number of I/O interrupts received, context switches, etc. For implementing the ULM scheme, we capture the CPU cycles spent by the back-end node for every time interval from this file and copy the load information to the registered memory.

The sequence of steps in load monitoring is shown in Figure 2. In the first step, the back-end helper module reads */proc*. To access */proc*, a trap occurs because of file I/O in Step 2. In Step 3, the kernel reads certain kernel data structures and in Step 4, the kernel returns these information to the helper module. Finally, the helper module parses this information, calculates the load, and copies to the registered memory. This load information obtained from all front-end nodes (Step 5) is further being utilized by load-balancing algorithms in determining a set of least loaded nodes (Step 6) so that future requests get forwarded to those nodes.

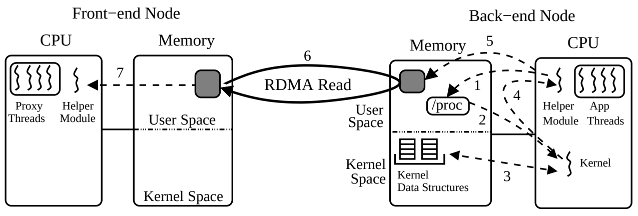


Figure 2. User-level Load Balancing Mechanism

3.3 Kernel-level Load Monitoring (KLM)

While the ULM scheme retains some of the benefits of an RDMA based design, since we are using a user process to implement the helper module, the information we can observe is restricted at the user-level. An alternative to avoid this limitation is we can make a kernel module read the necessary information directly and return this load information to the helper module instead of using */proc*. However, we still need a separate helper module in the back-end to constantly update the load information and this module competes with other processes in the system in the back-end system. Furthermore, in a data-center that is heavily loaded with multiple processes competing for the processor(s), the module which updates the load information may not get the CPU for a long time. This results in inaccurate load information.

To resolve these problems, we suggest a kernel-level load monitoring scheme that registers the necessary kernel data structures and let the front-end node directly read the kernel memory and get the load information. Such a design, avoids all the drawbacks mentioned above and more importantly it completely removes the need for an extra process in the server node. In this approach, we use a Linux kernel module which takes care of connection management, exchange of queue pairs and registers specific memory regions. After this initialization phase, the kernel is no longer disturbed. As shown in Figure 3, the load information is directly got from the kernel space, reading the kernel data structures using RDMA read (Step 1). Once the load information is obtained from all server nodes, a similar approach is taken by load monitor as mentioned in ULM scheme to perform load balancing (Step 2).

As mentioned earlier, we use a kernel module to register memory area for the kernel data structures. Fortunately we have access to almost all the data-structures via the kernel modules. For example, the information provided in */proc/stat* is a combination of two data structures namely *kstat and context switches*. Apart from these, there are several other data-structures that can be accessed at the kernel-level only. These data structures can provide detail load information. For example, a kernel data structure that KLM uses is the *irq_stat* data structure, which maintains the total number of software interrupts pending that each of the CPUs need to handle in future. This information, in particular, gives us an indication of number of pending software interrupts that the kernel needs to handle and helps in predicting the load for the next time interval. Usually only the kernel data structures which have been explicitly exported are available through a kernel module. But, if we have access to the *System.map* file which is generated while building the kernel, the kernel module can access all the global data structures of the kernel even if they are not explicitly exported. As a consequence, we now have the flexibility of accessing certain kernel data structures which are otherwise not exposed to user space. Since we allow only RDMA read operations on the registered kernel data structures, we are secure from overwriting the kernel data structures by other node.

An issue with this scheme is to provide a way to efficiently manage the history of load information. If the polling interval on the load information from front-end is very large, the resolution by which it captures the load information from server nodes is also large. Thus although KLM can still observe the accurate current load, it can miss the load information for last time interval. However, in a ULM scheme, since we have a separate module on the back-end, it can manage the load history. Interestingly, the kernel also maintains load averages and history of load for the past 5 seconds, 1 minute and 5 minutes. We can utilize this information in KLM to manage correct load history.

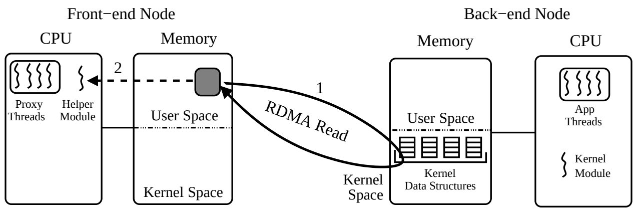


Figure 3. Kernel-level Load Balancing Mechanism

4 Potential Benefits of KLM

Using RDMA operations and kernel data structures to design and implement load monitoring in data-center has several potential benefits: (i) getting correct load information, (ii) utilizing detailed system information, (iii) no process requirement on the back-end side, (iv) enhanced robustness to load, and (v) no kernel modification. In this section, we describe them in detail.

Getting correct load information: As mentioned earlier, in the ULM scheme, the helper module on the back-end constantly calculates the load for every time interval t and stores this information in its registered memory. The front-end helper module gets this load information using RDMA read operation. Due to this interval t, there is always a delay between the time at which the back-end module updates the load information and the time at which the front-end reads the load information from the back-end. For example, if we assume that the load information is updated every 500ms at the back-end node, then the front-end node gets the information of load updated before 0 500ms and mostly not the current load in the back-end node. However, regardless of the t value, KLM can get accurate current load information. It is because KLM directly reads the load information from kernel data structures.

Utilizing detailed system information: While the ULM scheme operates at the user space, KLM scheme operates at the kernel space providing several opportunities to access most of the kernel data structures which may be useful for providing system-level services like load-balancing or dynamic reconfigurability in a data-center. Some of them are exposed via */proc* interface while some data structures like *irq_stat, dq_stat, and avenrun[]* are not. However, we can access these data structures through a kernel module. In addition, we can also register these memory regions and allow a remote node to access these data structures through an RDMA read operation. Accessing a kernel data structure has two main benefits. (i) The kernel space is the first to get the updated information and hence performing an RDMA read operation on kernel memory would provide the most accurate load information. (ii) Since we can get detail system load information we can enhance existing load balancing algorithms.

No process requirement on the back-end side: The ULM scheme requires a separate process on the back-end side to calculate the load of the back-end node periodically and place this information onto its registered memory. While this operation may not occupy considerable CPU, in a highly loaded data-center, it certainly competes for processor(s). If we have multiple processes in the data-center, it increases the time at which this load information is updated. Furthermore, if the traffic inside the data-center is extremely bursty such delays in capturing the load information might lead to poor load balancing. However, with the KLM scheme in-place, after the kernel module registers the necessary data structures and sends the memory handle and read access rights to the front-end, the kernel is free to execute its code and there is no requirement of any involvement of the module for load monitoring on the back-end. Moreover, under high loaded conditions, there is no extra thread competing for CPU and memory.

Enhanced robustness to load: It is a well known fact that the load on data-centers that support dynamic web services is very bursty and unpredictable [21, 22]. Performance of load balancing schemes over traditional network protocols can be degraded significantly especially if there is a high load in the back-end. This is because both sides should get involved in communication and it is possible that the module capturing the load on the back-end may never get the CPU for a long time. However, for protocols based on RDMA operations, the peer side is totally transparent to the communication procedure. Thus, the performance of both ULM and KLM based on RDMA operations is resilient and well-conditioned to load.

No kernel modification: It follows quite trivially that the ULM scheme needs no kernel modification. In addition, KLM scheme too does not need any kernel modification. We implement the basic connection management, exchange of queue pairs and memory handles for communication between the back-end and front-end using a kernel module. This kernel module can be loaded dynamically and does not require any kernel modification.

5 Experimental Results

For all our experiments we used a 24-nodes cluster in which each node is equipped with dual Intel Xeon 2.66 GHz processors, 512KB L2 cache and 2GB of main memory. The nodes are connected to InfiniBand fabric with 64-bit, 133 MHz PCI-X interface. The InfiniBand network connected with Mellanox InfiniHost MT23108 Host Channel Adapters (HCAs) through a Mellanox MTS 14400 144 port switch. The Linux kernel version used is 2.4.20-8smp. The InfiniHost SDK used is 3.2 and the HCA firmware version was 3.3.

The 8 nodes of the cluster are used as the back-end nodes in the data-center environment and the other 16 nodes are used as the front-end nodes. We use Apache version 2.0.48, PHP 4.3.1 and MySQL 4.0.12 in all our experiments. We have used a polling time of 50ms for load monitoring and, in the case of ULM, the load updating time on back-end nodes is also 50ms in all the experiments unless otherwise explicitly specified.

To measure the data-center throughput in TPS, we use the Zipf trace varying α value. In Zipf law, the relative probability of a request for the ith most popular content is proportional to $1/i^{\alpha}$, where α determines the randomness of file accesses. For base performance to compare with ULM and KLM, we measure the throughput of round-robin and random based load balancing, which perform the load balancing without any load information of the back-end nodes but simple forward requests to the back-end nodes in round-robin and random manner, respectively.

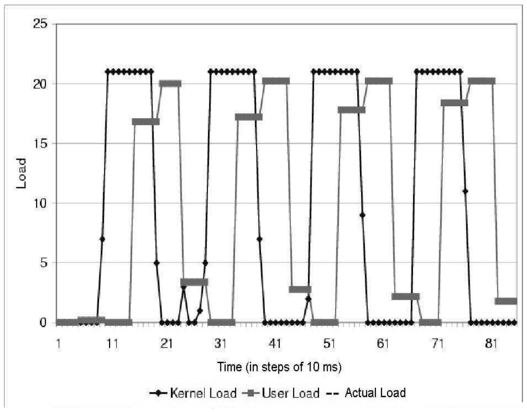


Figure 4. Accuracy of Load Information from ULM and KLM

5.1 Accuracy of Load Information

Figure 4 shows the CPU load captured by the ULM and KLM schemes and also reports the actual load seen at the back-end side. To measure all these numbers at the same time, we have designed the experiment in the following way. We create a process in the back-end node which alternatively repeats to load the CPU for sometime and go to sleep at other times. In parallel, we run both ULM and KLM together and compared the values reported by these schemes with the actual load values. To measure the actual load, we have a separate process in the back-end node which constantly calculates the load (CPU cycles spent) for every time interval 10 ms and stores this load information along with a timestamp. To generate timestamps, we use the kernel time, *xtime*, in identifying the time at which the front-end node captured the load information from the back-end node. As we can see in the figure, the KLM scheme reports almost the same value with actual load values. However, ULM scheme takes a while to get the updated load information mainly due to the fact that the server thread in the ULM scheme updates the load only for every interval 50ms. On the other hand, since the KLM scheme directly reads the actual load information from the kernel, the KLM scheme reports accurate load values.

5.2 Benefits in Single Data-Center

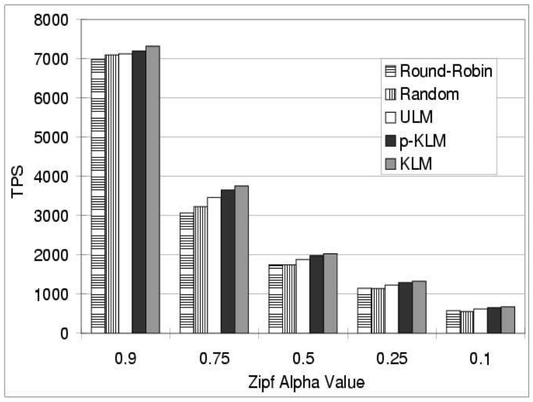
In this section, we present the data-center throughput with ULM and KLM schemes in a single data-center environment hosting only one website. To show the impact of load balancing, we run one Zipf trace with varying α values in our evaluation. Figure 5a shows the throughput achieved by five schemes. In order to understand the benefits of extra kernel-level information, we added the pKLB scheme, which only uses CPU load information as the metric to load balance whereas the KLB scheme uses both CPU load and interrupts pending in the queue while load balancing the requests. We observe that, in terms of the actual throughput values, ULM and KLM does better than round-robin or random load balancing. Figure 5b shows the benefits in terms of improvement percentage. We see that as the α value decreases in the Zipf trace, the performance improvement of all three suggested schemes increases. This is because, for lower α values in the Zipf trace, the temporal locality of files is low and the requests are well distributed in a given set of files giving more opportunity for the load balancing scheme to balance the requests. As a result, ULM and KLM can improve the throughput by 12% and 22%, respectively. An interesting result is that even though pKLM and ULM uses the same information for load balancing, we see that pKLM performs consistently better than the ULM scheme, due to accuracy of load information obtained by the pKLM scheme as as shown in Section 5.1. Another point to note here is that the KLM scheme consistently does better than the pKLB scheme for all traces validating the fact that detailed system information helps load balancer in balancing the requests in a better way in comparison with other schemes.

5.3 Impact of Heavy Load on the Back-end Nodes

In this experiment, we emulate the loaded conditions in the single data-center environment by performing background computation and communication operations on the back-end nodes. We run 16 processes in the background performing a simple ping-pong latency experiment exchanging message sizes of 4KB for every iteration. Figure 6 shows the performance benefits of ULM and KLM schemes with varying number of loaded nodes in the back-end side running Zipf trace. As the load in the back-end nodes increase, both ULM and KLM schemes inform the back-end nodes' state to the front-end nodes and these take the appropriate action of avoiding these loaded nodes. On the other hand, static load balancing schemes like round-robin and random are not aware of load in the back-end side and hence forward requests to all the back-end nodes even if these are heavily loaded. As a result, we can observe that ULM and KLM can achieve significantly better throughput as shown in Figure 6. It is also to be noted that KLM can achieve higher throughput than ULM, which is again due to the fact that KLM can obtain more detailed and timely accurate information.

5.4 Benefits in Shared Data-Centers

In this section, we present the throughput improvement of ULM and KLM schemes in comparison with random



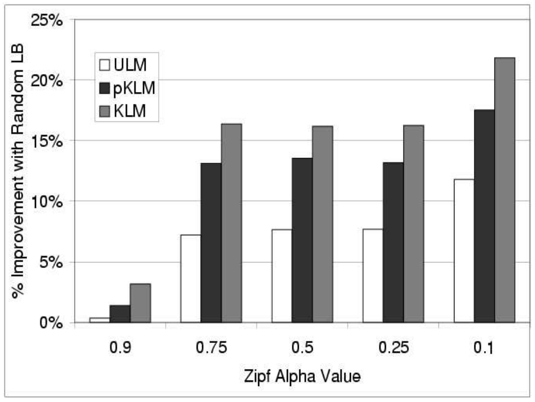


Figure 5. (a) Throughput of ULM and KLM in a Single Data-Center with Varying Zipf α Values (b) Improvement Percentage with Three Suggested Schemes

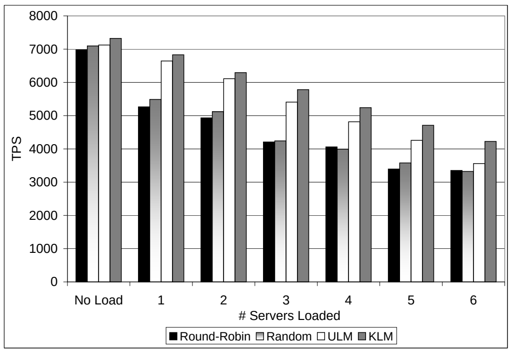


Figure 6. Throughput of ULM and KLM with Increasing Loaded Back-end Nodes Running Zipf Trace with α Value 0.9

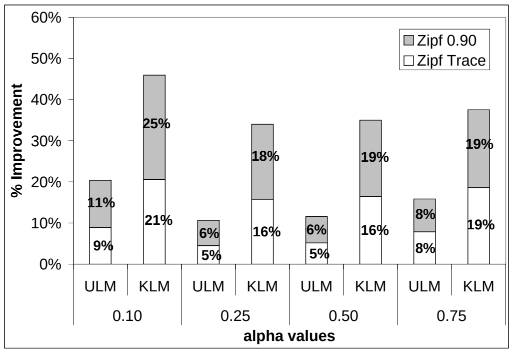


Figure 7. Performance Improvement of ULM and KLM in Shared Data-Centers: Zipf with α Value 0.9 with Varying Zipf Traces

load balancing in a shared data-center scenario. The experiment is designed in the following manner. We run two Zipf traces simultaneously with random load balancing and measure the throughput realized for each trace separately, where we fix the α value to 0.9 for a Zipf trace and vary it for the other trace from 0.1 to 0.75. Then, we run the same test with ULM scheme and KLM scheme and measured the throughput achieved in the same way as mentioned above. We report the improvement percentage seen in comparison with the random load balancing for each of these traces separately as shown in Figure 7. We can observe that in the case of Zipf traces with α value 0.9 and 0.1, the KLM scheme achieves a performance improvement of up to 25% and 21% per web-site, respectively. On the other hand, the ULM scheme achieves only 11% and 9% improvements for the same Zipf traces. In addition, we observe that the benefits in shared data-centers are more in comparison with a single data-center scenario. This is because there is more opportunity for performing fine-grained load balancing in shared data-centers since we have multiple websites with different resource requirements which change dynamically from time to time. However, as Zipf α value increases, the temporal locality of contents increases giving less opportunity to perform fine-grained load balancing but still we can see significant improvements with KLM.

5.5 Load Distribution

In this section, we show the main benefit of our KLM scheme with other schemes that we have compared in this paper. In order to show this, we use the shared data-center scenario hosting two Zipf traces with α value 0.9 and 0.1

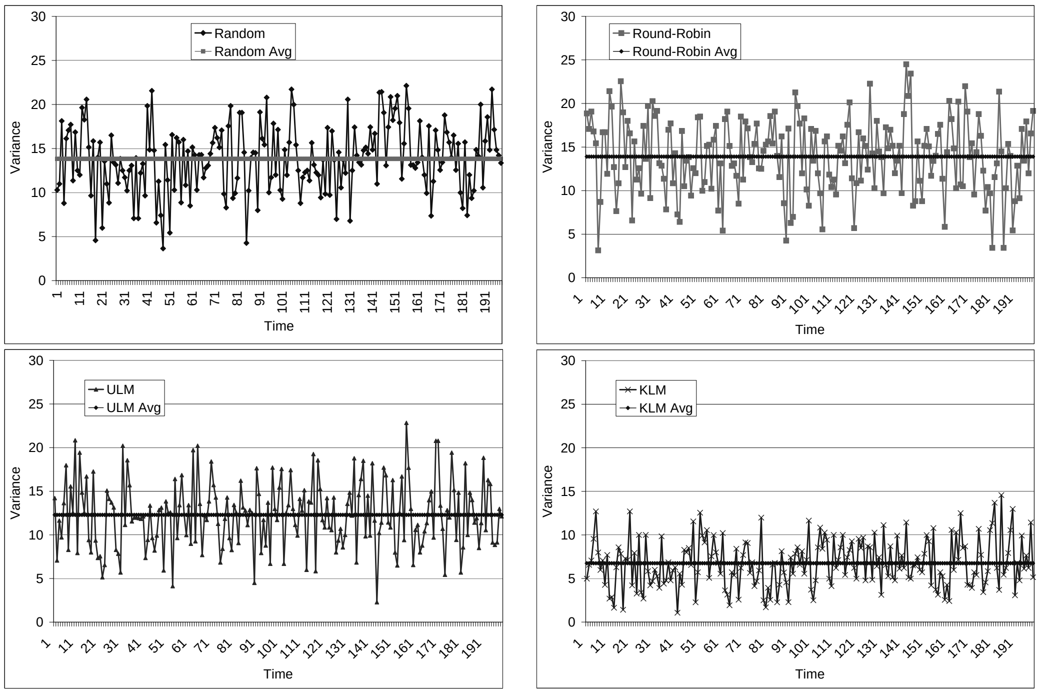


Figure 8. Load Distribution in Shared Data-Center Scenario Hosting Zipf Traces with α Values of 0.9 and 0.1 (a) Random (b) Round-Robin (c) ULM (d) KLM

as used in Section 5.4. We run this experiment, measure the CPU cycles spent among all back-end nodes for every time interval of 50ms, and calculate its variance. We report this variance value over a period of time in Figure 8. We know that variance on a set of values indicates the degree of fluctuation on those set of values. A lower value indicates that the set of values are very close to each other, while high values indicate large differences between these values. In our case, it is a measure of how well the load among the back-end nodes was distributed for a period of time in the data-center. As seen in Figure 8, the variance of load values reported for random and round-robin load balancing schemes fluctuate from very high values to low values, indicating that the load balancing algorithms do not distribute the load equally most of the time. We see similar trends even for ULM except that the fluctuation is little lesser and the average of this variance over time is a bit less than random and round-robin schemes. On the other hand, the variance reported by the KLM scheme is significantly less than all other three schemes indicating that the KLM scheme resulted in better distribution of load (almost by a factor of 2).

5.6 Sensitivity on Polling Interval

Figure 9 shows the performance sensitivity with different polling interval at which the front-end node gathers load information of back-end nodes by using RDMA read. For a static scheme like random load balancing we do not have any polling interval and hence the performance is not affected. However, we have an interesting observation for ULM and KLM schemes. We see that, in both these two schemes, higher polling intervals lead to poor performance. This is because the front-end node uses a snapshot of load information for long time, which cannot reflect the fluctuation of load on the back-end side. Also, choosing very small thresholds for ULM scheme yields to poor performance. This suggests that the responsiveness of the front-end node to load fluctuations is largely dependent on the polling interval and this needs to be carefully chosen in order to achieve maximum performance.

6 Related Work

Several researchers have proposed and evaluated various load balancing policies for cluster-based network services [8, 11, 14, 23, 24]. As mentioned in [12], a lot of them

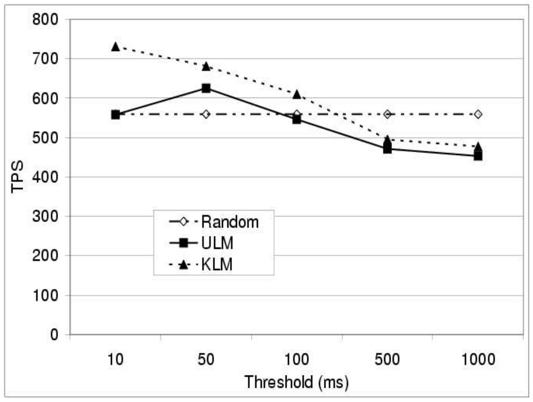


Figure 9. Impact of Polling Interval in Load Monitoring

are valuable in general, but not all of them can be applicable to current or next-generation data-centers. Many of the policies focus on coarse-grained services In addition, many of the load balancing policies proposed in [1, 19, 5, 9] rely on the assumption that all network packets go through a single front-end dispatcher so that all connection-based statistics can be accurately maintained. However, in a data-center environment with several nodes in each tier and interactions between tiers (e.g., proxy servers and application servers or application servers and database servers), such an approach may not be optimal or sometimes give worse performance. This constraint calls for a distributed and complex load information dissemination schemes to provide fine-grained services.

Several others have focused on the design of adaptive systems that can react to changing workloads in the context of web servers [15, 10, 20, 4, 18]. There has been some previous research which focus on dynamism in the data-center environment by HP labs and IBM Research [13, 17]. These are notable in the sense that they were the first to show the capabilities of a dynamic allocation of system resources in the data-center environment. However, some of the solutions focus on lower level architectural requirements mainly for storage related issues, rely on specific hardware and are hard to look at as commodity component based solutions. On the other hand, in our approach, we try to propose a solution that is not geared toward any specific hardware and try to give a generic solution at the application level without requiring any changes to existing applications. Further, some of these approaches rely on the servers to intelligently load balance the requests to other nodes. While these approaches are quite intuitive, in a real data-center scenario, the high server loads can make them inefficient and potentially unusable. Our approach of placing the onus of load balancing coupled with load monitoring on the relatively lightly loaded edge servers by utilizing the remote memory operations offered by InfiniBand tries to tackle these challenges in an efficient manner.

7 Conclusions and Future Work

In this paper, we proposed two schemes (ULM and KLM) for accurate load monitoring in a data-center environment and evaluated its benefits in providing load-balancing. In our approach, we used the advanced features of InfiniBand such as Remote Direct Memory Access (RDMA) operations without requiring any modifications to the existing data-center applications. We also proposed the use of certain kernel data structures which are not exposed to the user space for performing efficient load balancing. Our experimental results show that both KLM and ULM schemes achieve better performance in comparison with random and round-robin load balancing schemes that are widely used in current data-centers. Our results also validate that the load distribution performed by the KLM scheme is the best in comparison with the other schemes. Both KLM and ULM schemes achieved an improvement of 22% and 12% in a single data-center and an improvement of 25% and 11% per web-site in shared data-centers, respectively. More importantly, our schemes take advantage of RDMA read operation in accessing certain portion of kernel memory for accurate load monitoring and making it resilient and well-conditioned to the load on the back-end nodes. This feature becomes more important because of the unpredictability of load in a typical data-center environment which supports large-scale dynamic services.

Dynamic reconfiguration of resources has been studied in the context of nodes [3, 2] and storage environments [17]. Monitoring the load for such services is also critical for efficiently utilizing the resources in the data-center. We plan to extend the knowledge gained in this study to implement a full-fledged dynamic reconfiguration module coupled with accurate load monitoring.

8 Acknowledgments

We would like to thank Pavan Balaji for the detailed technical discussions and helping us with the initial implementation during the course of this work.

References

[1] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers.

In *In Proc of the 2000 USENIX Annual Techincal Conf.*, San Deigo, CA, June 2000.

[2] P. Balaji, S. Narravula, K. Vaidyanathan, H. W. Jin, and Dhabaleswar K. Panda. On the Provision of Prioritization and Soft QoS in Dynamically Reconfigurable Shared Data-Centers over InfiniBand. In *the Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2005.

[3] P. Balaji, K. Vaidyanathan, S. Narravula, K. Savitha, H. W. Jin, and D. K. Panda. Exploiting Remote Memory Operations to Design Efficient Reconfiguration for Shared Data-Centers. In *Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT)*, San Diego, CA, Sep 20 2004.

[4] C. Lu and T. Abdelzaher and J. Stankovic and S. Son. A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, June 2001.

[5] E. Carrera and R. Bianchini. Efficiency vs. portability in cluster-based network servers. In *Proceedings of the 8th Symposium on Principles and Practice of Parallel Programming, Snowbird, UT*, 2001.

[6] A. Chandra, W. Gong, and P. Shenoy. Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. In *Proceedings of ACM Sigmetrics 2003, San Diego, CA*, June 2003.

[7] L. Cherkasova and S. R. Ponnekanti. Optimizing a content-aware load balancing strategy for shared Web hosting service. In *8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 492 – 499, 29 Aug - 1 Sep 2000.

[8] E. D. Lazowska D. L. Eager and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. In *IEEE Transactions on Software Engineering*, 1986.

[9] R. P. King G. D. H. Hunt, G. S. Goldszmidt and R. Mukherjee. Network dispatcher: A connection router for scalable internet services. In *In Proceedings of the 7th International World Wide Web Conference, April*, 1998.

[10] J. Carlstrom and R. Rom. Application-Aware Admission Control and Scheduling in Web Servers. In *Proceedings of the IEEE Infocom 2002*, June 2002.

[11] M. Devarakonda K. K. Goswami and R. K. Iyer. Prediction-based dynamic load-sharing heuristics. In *IEEE Transactions on Parallel and Distributed Systems*, pages 638–648, 1993.

[12] T. Yang K. Shen and L. Chu. Cluster load balancing for fine-grain network services. In *Proc. of International Parallel and Distributed Processing Symposium. FL. April*, 2002.

[13] HP Labs. HP virtualization solutions: IT supply meets business demand: White Paper. In *http://h30046.www3.hp.com/uploads/infoworks/*, July 2003.

[14] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, 1997.

[15] N. Bhatti and R. Friedrich. Web server support for tiered services. In *IEEE Network*, September 1999.

[16] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting Strong Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. In *the Proceedings of the IEEE International Workshop on System Area Networks (SAN)*, 2004.

[17] D. O'Hare, P. Tandon, H. Kalluri, and P. Mills. SNIA SSF Virtualization Demonstration. In *IBM Systems Group - TotalStorage Software: White Paper*, October 2003.

[18] P. Pradhan and R. Tewari and S. Sahu and A. Chandra and P. Shenoy. An Observation-based Approach Towards Self-Managing Web Servers. In *Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002)*, May 2002.

[19] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. M. Nahum. Locality-aware request distribution in cluster-based network servers. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 205–216, 1998.

[20] S. Lee and J. Lui and D. Yau. Admission control and dynamic adaptation for a proportionaldelay diffserv-enabled web server. In *Proceedings of SIGMETRICS*, 2002.

[21] W. Shi, E. Collins, and V. Karamcheti. Modeling Object Characteristics of Dynamic Web Content. *Special Issue on scalable Internet services and architecture of Journal of Parallel and Distributed Computing (JPDC)*, Sept. 2003.

[22] M. Welsh, D. Culler, and E. Brewer. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In *the Eighteenth Symposium on Operating Systems Principles (SOSP-18)*, Oct. 2001.

[23] S. Zhou. An experimental assessment of resource queue lengths as load indices. In *In Proc. of the Winter USENIX Technical Conf*, pages 73–82, 1987.

[24] S. Zhou. A trace-driven simulation study of dynamic load balancing. In *IEEE Transactions on Software Engineering*, pages 1327–1341, 1988.