On the Provision of Prioritization and Soft QoS in Dynamically Reconfigurable Shared Data-Centers over InfiniBand

P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, H. -W. Jin and D. K. Panda

Technical Report
OSU-CISRC-10/04-TR55

On the Provision of Prioritization and Soft QoS in Dynamically Reconfigurable Shared Data-Centers over InfiniBand*

P. Balaji S. Narravula K. Vaidyanathan H. -W. Jin D. K. Panda
Department of Computer Science and Engineering
The Ohio State University
{balaji, narravul, vaidyana, jinhy, panda}@cse.ohio-state.edu

Abstract

In the past few years several researchers have proposed and configured data-centers providing multiple independent services, known as shared data-centers. For example, several ISPs and other web service providers host multiple unrelated web-sites on their data-centers allowing potential differentiation in the service provided to each of them. Such differentiation becomes essential in several scenarios in a shared data-center environment. In this paper, we extend our previously proposed scheme on dynamic reconfigurability to allow service differentiation in the shared data-center environment. In particular, we point out the issues associated with the basic dynamic reconfigurability scheme and propose two extensions to it, namely (i) Dynamic Reconfiguration with Prioritization *and (ii)* Dynamic Reconfiguration with Prioritization and QoS. *Our experimental results show that our extensions can allow the dynamic reconfigurability scheme to attain a performance improvement of up to five times for high priority websites irrespective of any background low priority requests. Also, these extensions are able to significantly improve the performance of low priority requests when there are minimal or no high priority requests in the system. Further, they can achieve a similar performance as a static scheme with up to 43% lesser nodes in some cases.*

1 Introduction

Cluster computing systems are becoming increasingly popular for providing *cost-effective* and *affordable* computing environments for a wide range of applications. These systems are typically built by interconnecting a set of commodity PCs or workstations using *high-speed interconnect architectures.* InfiniBand Architecture (IBA) [8] is one such architecture that has been recently standardized by the industry to design next generation high-end clusters. IBA relies on two key features, namely *User-level Networking* and *One-Sided Communication Operations.* User-level Networking allows applications to directly and safely access the network interface without going through the operating system. One-sided communication allows the network interface to transfer data between local and remote memory buffers without any interaction with the operating system or the host processor. It also provides features for performing network based atomic operations on the remote memory regions.

On the other hand, 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. Based on these two trends, researchers and industries have proposed the feasibility and potential of cluster-based data-centers [29, 17, 9, 24].

The various nodes in a traditional cluster-based data-center (Figure 1) are logically partitioned to provide various related services including web and messaging services, transaction processing, business logic, databases, etc. These nodes interact with each other depending on the query to provide the service requested by the end user. In the past few years several researchers have proposed and configured data-centers providing multiple independent services, known as shared data-centers [14, 15]. For example, several ISPs and other web service providers host multiple unrelated web-sites on their data-centers allowing potential differentiation in the service provided to each of them. Such differentiation becomes essential in several scenarios in a shared data-center environment.

For example, a data-center may want to give a higher priority to all requests pertaining to website A (a high paying customer) as compared to website B (a low paying customer). Similarly, a data-center might guarantee a certain Quality of Service (QoS) guar-

*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

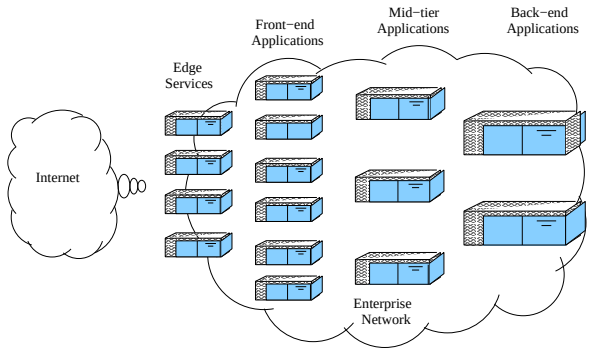

Figure 1: A traditional cluster-based Data-Center

antees in the resources provided to each service or website it is hosting. In the context of data-centers, we classify QoS guarantees into two classes: *Hard QoS guarantees* and *Soft QoS guarantees*.

Hard QoS guarantees require that the resources guaranteed be available to the website at all times, i.e., there will not be any time instance that the website is provided lesser resources than what it was initially guaranteed. On the other hand, Soft QoS guarantees rely on the average load on the website and client workload pattern studies. For example, though traffic to a website could be bursty at times requiring a large number of nodes, the average load on the website could be much lesser requiring fewer nodes. In this case, if a website is provided a soft QoS guarantee of N resources, it is guaranteed these resources if they are not already allocated to other higher priority websites.

Statically assigning the nodes in the shared data-center for each service provided depending on the priority or the QoS guarantees given to the website is a widely used approach. Though this approach is easily capable of meeting the priority or QoS requirements given by the data-center, it might incur severe under utilization of resources especially when the traffic is bursty and directed to a single web-site.

In our previous work [11], we presented a novel design to provide dynamic reconfigurability of the nodes in the data-center environment. This technique enables the nodes in the data-center environment to efficiently adapt their functionality based on the system load and traffic pattern. However, as we will see in Section 4, the basic dynamic reconfigurability scheme does not have any concept of service differentiation *per se*. Thus, it cannot be *directly* used in a shared data-center environment having different service requirements for different websites. In this paper, we extend the basic dynamic reconfigurability scheme to allow service differentiation in the shared data-center environment. In particular, we point out the issues associated with the basic dynamic reconfigurability scheme and propose two extensions to it, namely (i) *Dynamic Reconfiguration with Prioritization* and (ii) *Dynamic Reconfiguration with Prioritization and QoS*.

We evaluated our proposed schemes on an InfiniBand based cluster and attempted to bring out the benefits and issues associated with these schemes. Our experimental results show that our extensions can allow the dynamic reconfigurability scheme to attain a performance improvement of up to five times for high priority websites irrespective of any background low priority requests. Also, these extensions are able to significantly improve the performance of low priority requests when there are minimal or no high priority requests in the system. Further, they can achieve a similar performance as a static scheme with up to 43% lesser nodes in some cases.

The remaining part of the paper is organized as follows: We provide a brief background about InfiniBand and Shared Data-Centers in Section 2. In Section 3, we describe the details about the design and implementation of our basic dynamic reconfiguration scheme. In Section 4 we describe the provisioning of prioritization and soft QoS guarantees in shared data-centers and our extensions to the dynamic reconfigurability approach to provide these. We describe our experimental results in Section 5, present some previous related work in Section 6 and draw our conclusions and possible future work in Section 7.

2 Background

In this section, we provide a brief background on (i) the InfiniBand architecture and its advanced one-sided operations like RDMA and atomic operations and (ii) the Clustered data-center environment.

2.1 InfiniBand

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 the processing nodes and the 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 also defines built-in QoS mechanisms which provide virtual lanes on each link and define service levels for individual packets.

In an InfiniBand network, processing nodes and the I/O nodes are connected to the fabric by Host Channel Adapters (HCA) and Target Channel Adapters (TCA). HCAs are associated with processing nodes

and their semantic interface to consumers is specified in the form of InfiniBand Verbs. TCAs connect I/O nodes to the fabric and have interfaces to consumers that are implementation specific and are not defined in the IBA specifications. Channel Adapters usually have programmable DMA engines with protection features.

IBA mainly aims at reducing the system 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 Pair (QP) based model is used in this interface. Figure 2 illustrates the InfiniBand Architecture.

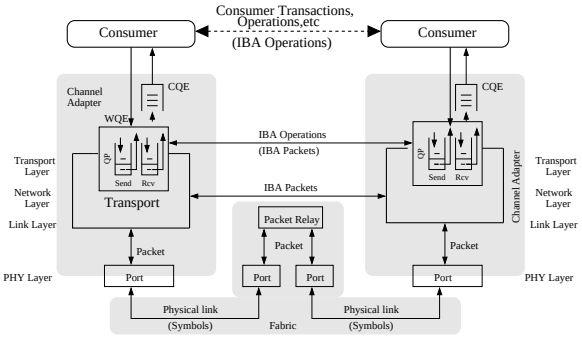

Figure 2: The InfiniBand Architectural Model

Each Queue Pair 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). Further, the application can either choose to be signaled on the completion of a WQR using the Signaled (SG) request or alternatively choose an Unsignaled (USG) request. When the HCA completes the processing of a signaled request, it places an entry in the Completion Queue (CQ) called as the Completion Queue Entry (CQE).

The consumer application can poll on the CQ associated with the work request to check for completion. There is also the feature of triggering event handlers whenever a completion occurs. For Unsignaled requests, no completion event is returned to the user. However, depending on the implementation, the driver cleans up the Work Queue Request from the appropriate Queue Pair on completion.

IBA Communication Models: IBA supports two types of communication semantics: Channel Semantics (Send-Receive communication model) and Memory Semantics (RDMA communication model). In channel semantics, each 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. Failure to post a descriptor on the remote node results in the message being dropped and if the connection is reliable, it might even result in the breaking of the connection. In memory semantics, Remote Direct Memory Access (RDMA) operations are used. These operations are transparent at the remote end since they do not require a receive descriptor to be posted. In this semantics, the send request itself contains both the virtual address for the local transmit buffer as well as that for the receive buffer on the remote end. The RDMA operations are available with the Reliable Connection (RC) service type. The IBA specifications does not provide different primitives for the channel semantics and the memory semantics. It is the "opcode" entry in the WQR, which distinguishes between the channel semantics and the memory semantics. Most other entries in the WQR are common for both the Send-Receive model as well as the RDMA model, except an additional remote buffer virtual address (and other related entries) which has to be specified for RDMA operations. There are two kinds of RDMA operations: RDMA Write and RDMA Read. In the RDMA Write model, the initiator directly writes data into the remote node's memory. Similarly, in the RDMA Read model, the initiator directly reads data from the remote node's memory.

In addition to RDMA, the reliable communication classes also optionally include atomic operations directly against the memory at the end node. Atomic operations are posted as descriptors as in any other type of communication. However, the operation is completely handled by the HCA. The atomic operations supported are Fetch-and-Add and Compare-and-Swap, both on 64-bit data. The Fetch-and-Add operation performs an atomic addition at the remote end. The Compare-and-Swap is used to compare two 64-bit values and swap the remote value with the data provided if the comparison succeeds.

IBA Communication Protocols: There are two main types of traffic over IBA fabrics. The first type of traffic is IBA native protocols such as VAPI [4], IBAL (InfiniBand Access Layer) [1], uDAPL (User-Level Direct Access Transport APIs) [5] and kDAPL (Kernel-Level Direct Access Transport APIs) [6]. These interfaces are low-level APIs for IBA. Applications can be programmed using these APIs to take full advantage of IBA user

level networking and RDMA features. IP is the other type of traffic (and a very important one) that could use this interconnect. InfiniBand would benefit greatly from a standardized method of handling IP traffic. IPoIB provides standardized IP encapsulation over IBA fabrics as defined by the IETF Internet Area IPoIB working group [2]. The IPoIB project [3] implements this proposed standard as a layer-2 Linux network driver. The primary responsibilities of the driver are performing address resolution to map IPv4 and IPv6 addresses to InfiniBand Unreliable Datagram (UD) address vectors, the management of multicast membership, and the transmission and reception of IPoIB protocol frames.

2.2 Shared Data-Center Environment

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

Figure 3 shows a higher level layout of a shared 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).

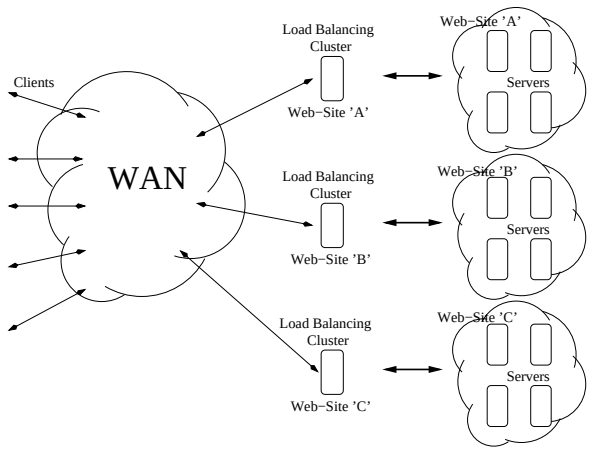

Figure 3: A Shared Cluster-Based Data-Center Environment

While hardware load-balancers are commonly available today, they suffer from being based on a pre-defined algorithm 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 tuned based on the data-center requirements, they can potentially form bottlenecks themselves for highly loaded data-centers. In the past, several researchers have proposed the use of an additional cluster of nodes (known as the edge tier) [29] to perform certain services such as intelligent load-balancing, caching, etc [15]. 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 servers inside the clustered data-center provide the actual services such as web-hosting, transaction processing, etc. Several of these services require computationally intensive processing such as CGI scripts, Java servlets and database query operations (table joins, etc). This makes the processing on the server nodes CPU intensive in nature.

3 Dynamic Reconfiguration Overview

In our previous work [11], we presented a novel design to provide dynamic reconfigurability of the nodes in the data-center environment. This technique enables the nodes in the data-center environment to efficiently adapt their functionality based on the system load and traffic pattern. Dynamic reconfigurability attempts to provide benefits in several directions: (i) cutting down the time needed for configuring and assigning the resources available by dynamically transferring the traffic load to the best available node/server, (ii) improving the performance achievable by the data-center by reassigning under-utilized nodes to loaded services, (iii) cutting down the cost of the data-center by reducing over-provisioning of nodes and improving the utilization of the resources available inside the data-center and several others.

While reconfigurability is a widely used technique for clusters, the data-center environment poses several interesting challenges for the design and implementation of such a scheme. In this section, we describe some of the challenges involved in implementing dynamic reconfigurability in the data-center environment and the details about the implementation of this scheme using the native Verbs layer over InfiniBand (VAPI). In Section 4, we propose several extensions to the basic dynamic reconfigurability scheme to allow service differentiation in the shared data-center environment.

3.1 Dynamic Reconfigurability Support

Request patterns seen over a period of time, by a shared data-center, may vary significantly in terms of the ratio of requests for each co-hosted web-site. For example, interesting documents or dynamic webpages becoming available and unavailable might trigger bursty traffic for some web-site at some time and for some other web-site at a different time. This naturally changes the resource requirements of a particular co-hosted web site from time to time.

Statically assigning the nodes in the shared data-center for each service provided depending on the priority or the QoS guarantees given to the website is a widely used approach. In this approach, nodes are alloted to each service depending either on the worst case estimates of the load expected or on the QoS guarantees provided together with the nodes available in the data-center. It is easy to see that this approach is easily capable of meeting the priority of QoS requirements given by the data-center. However, this approach has several disadvantages. First, it might incur severe under utilization of resources especially when the traffic is bursty and directed to a single web-site. Second, this approach restricts the QoS guarantees provided by the data-center based on the number of nodes present. Third, this approach can result in a poor performance for the low priority requests even when there are no high priority requests available in the system.

Dynamic Reconfigurability attempts to tackle these issues with the static assignment scheme. The basic idea of reconfigurability is to utilize the idle nodes of the system to satisfy the dynamically varying resource requirements of each of the individual co-hosted web-sites in the shared data-center. Depending on the current demands (e.g., due to a burst of requests to one web-site), nodes reconfigure themselves to support these demands.

Support for Existing Applications: A number of applications have been developed in the data-center environment over the span of several years to process requests and provide services to the end user. To avoid making cumbersome changes to such existing applications, our design makes use of *external helper modules* which work alongside the applications to provide effective dynamic reconfiguration. Tasks related to system load monitoring, maintaining a global state information, reconfiguration, etc. are handled by these helper modules in an application transparent manner. These modules, running on each node in the shared data-center, reconfigure the data-center depending on current request and load patterns. They use the run-time configuration files of the data-center applications to reflect these changes.

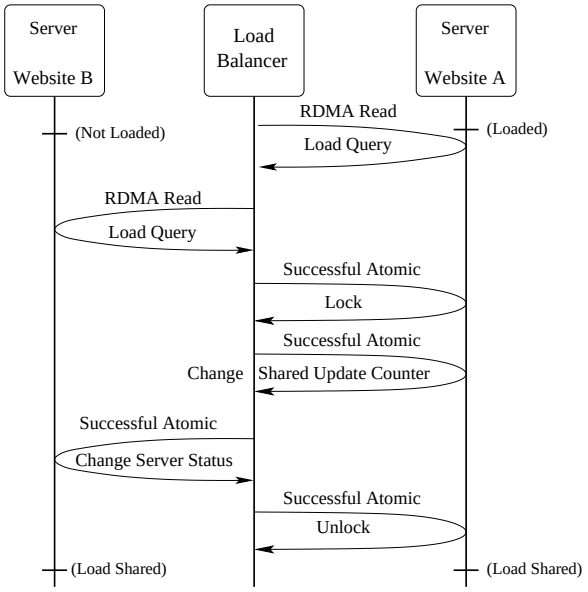

Figure 4: RDMA based Protocol for Dynamic Reconfigurability

Load-Balancer Based Reconfiguration: Two different approaches could be taken for reconfiguring the nodes: Server-based reconfiguration and Load-balancer based reconfiguration. In server-based reconfiguration, when a particular server detects a significant load on itself, it tries to reconfigure a relatively free node that is currently serving some other web-site content. Though intuitively the loaded server itself is the best node to perform the reconfiguration (based on its closeness to the required load information), performing reconfiguration on this node adds a significant amount of load to an already loaded server. Due to this reason, reconfiguration does not happen in a timely manner and the overall performance is affected adversely. On the other hand, in a load-balancer based reconfiguration, the edge servers (functioning as load-balancers) detect the load on the servers, find a free server to alleviate the load on the loaded server and perform the reconfiguration themselves. Since the shared information like load, server state, etc. is closer to the servers, this approach incurs the cost of requiring more network transactions for its operations.

Remote Memory Operations Based Design: As mentioned earlier, by their very nature, the server nodes are compute intensive. Execution of CGI-Scripts, business-logic, servlets, database processing, etc. are typically very taxing on the server CPUs. So, the helper modules can potentially be starved for CPU on these servers. Though in theory the helper modules on the servers can be used to share the

load information through explicit two-sided communication, in practice, such communication does not perform well [24]. InfiniBand, on the other hand, provides one-sided remote memory operations (like RDMA and Remote Atomics) that allow access to remote memory without interrupting the remote node. In our design, we use these operations to perform load-balancer based server reconfiguration in a server transparent manner. Since the load-balancer is performing the reconfiguration with no interruptions to the server CPUs, this RDMA based design is highly resilient to server load. Figure 4 shows the RDMA based protocol used by Dynamic Reconfigurability. As shown in the figure, the entire cluster management and dynamic reconfiguration is performed by the lightly loaded load-balancer nodes without disturbing the server nodes using the RDMA and remote atomic operations provided by InfiniBand.

Some of the other major design challenges and issues involved in dynamic adaptability and reconfigurability of the system are: (i) Providing a System Wide Shared State, (ii) Concurrency Control to avoid Live-locks and Starvation, (iii) Avoiding server thrashing through history aware reconfiguration and (iv) Tuning the reconfigurability module sensitivity. Further details about the other design issues can be found in [11].

4 Service Differentiation with Reconfigurability

In this section, we point out the issues associated with the dynamic reconfigurability scheme and propose extensions to this scheme to allow service differentiation in a shared data-center environment.

4.1 Service Differentiation

As mentioned earlier, differentiation in service becomes essential in several scenarios in a shared data-center environment. For example, a data-center may want to give a higher priority to all requests pertaining to website A (a high paying customer) as compared to website B (a low paying customer). Similarly, a data-center might guarantee a certain Quality of Service (QoS) guarantees in the resources provided to each service or website it is hosting.

Issues with Dynamic Reconfigurability: In the dynamic reconfigurability approach the nodes in the data-center environment adapt their functionality based on the system load and traffic pattern. This allows a higher utilization of the system resources and essentially improves the overall performance of the system. However, this scheme does not have concept of service differentiation *per se*. Thus, it cannot be *directly* used in a shared data-center environment having different service requirements for different websites. The scheme performs reconfiguration only based on the load on the physical nodes and does not consider any prioritization and QoS guarantees the website might have to meet.

This means that a burst of requests to one website would result in all the nodes in the data-center to be re-assigned to this website. Now, while this website is still loaded, if there is a burst of requests to another higher priority website, the reconfigurability scheme will not be able to find a lightly loaded physical node and thus will not perform any additional reconfigurations to handle the new burst of high priority requests. Figure 5 depicts this issue with the basic dynamic reconfigurability mechanism in a shared data-center environment hosting two websites where one of the websites is a high priority website and the other is a lower priority website.

Figure 5a shows the initial state of the system where both the websites have a small amount of requests coming in. Both websites are initially configured with four nodes each. Figure 5b shows a state where there is a heavy load of requests to website B. Since website A is not heavily loaded, the nodes servicing this website are reconfigured to serve website B. Figure 5c shows a state where there is a heavy load of requests for website A, while website B is still heavily loaded. In this scenario, since website A cannot find any lightly loaded servers, it does not perform any reconfiguration of nodes, causing a degradation of performance for the high priority website. However, ideally we would like this burst of high priority requests to reconfigure the nodes from the lower priority website to handle the requests on the high priority website.

4.2 Schemes to provide Service Differentiation

In this section, we briefly describe two approaches to provide service differentiation in shared data-centers, namely (i) *Rigid* and (ii) *Dynamic Reconfiguration based Service Differentiation.*

4.2.1 Rigid Approach

In the rigid approach, the data-center statically assigns the available nodes in the data-center as requested by the websites. For example, if the data-center has eight nodes and hosts two websites (website A and website B), the number of nodes it can

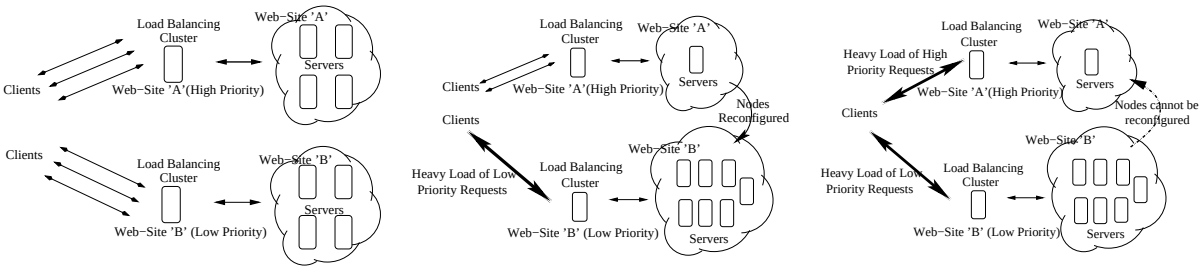

Figure 5: Basic Dynamic Reconfigurability: (a) Step 1, (b) Step 2 and (c) Step 3

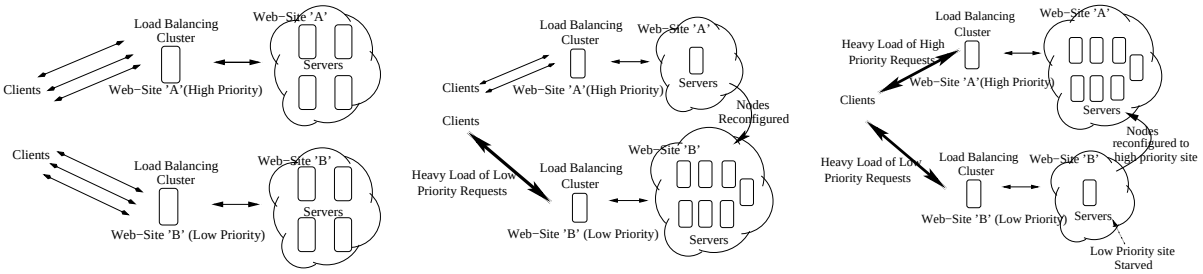

Figure 6: Dynamic Reconfigurability with Prioritization: (a) Step 1, (b) Step 2 and (c) Step 3

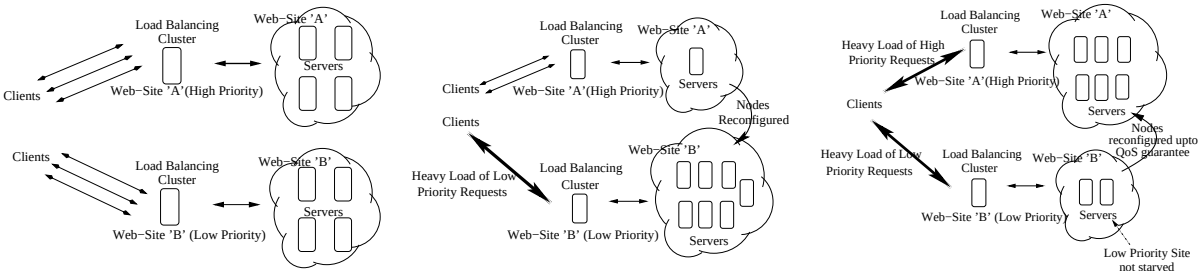

Figure 7: Dynamic Reconfigurability with Prioritization and QoS: (a) Step 1, (b) Step 2 and (c) Step 3

commit to both the websites together cannot exceed eight nodes. For example, six nodes to website A and two nodes to website B is one possible node guarantee the data-center can give in this model.

We use two instances of this scheme in our evaluation: *Rigid-Small* and *Rigid-Large*. Both instances are similar except that Rigid-Large uses a larger number of physical nodes as compared to Rigid-Small.

4.2.2 Reconfiguration based Service Differentiation

We propose two extensions to the basic dynamic reconfigurability scheme, namely (i) *Dynamic Reconfiguration with Prioritization* and (ii) *Dynamic Reconfiguration with Prioritization and QoS* to allow dynamic reconfigurability to do re-allocation of nodes while considering the service differentiation requirements provided by the data-center.

Dynamic Reconfiguration with Prioritization: In the prioritization scheme, the reconfigurability modules keep track of the priority-level to which each website belongs. On detecting a high load, the scheme allows reconfiguring another node if either it is lightly loaded or it belongs to a lower priority level. Figure 6 depicts the previous example with this scheme.

Figure 6a shows the initial state of the system where both the websites have a small amount of requests coming in. Again, both websites are initially configured with four nodes each. Figure 6b shows a state where there is a heavy load of requests to website B. Similar to the basic reconfigurability scheme, since website A is not that heavily loaded at this time, the nodes servicing this website are reconfigured to serve website B. Figure 6c shows a state where there is a heavy load of requests for website A, while website B is still heavily loaded. Unlike the basic reconfigurability scheme, in this scenario since website B is a low priority website, with the dynamic reconfiguration with prioritization scheme the nodes can be reconfigured to serve website A. So, as the figure depicts, the nodes are reconfigured to serve website A thus improving the performance of the high priority website. Now however, website B only has one node left to serve its pertinent incoming requests; this could lead to starvation for requests for website B. In summary, though the dynamic reconfigurability with prioritization scheme can give a higher performance to the high priority websites as compared to the basic reconfigurability scheme, it might result in starvation for the low priority requests.

Dynamic Reconfiguration with Prioritization and QoS: The dynamic reconfiguration with prioritization scheme allows the higher priority requests to reconfigure nodes from the lower priority sites and thus increase the performance of these requests. However, just shifting the nodes based on priorities might cause lower priority requests to be starved. For example, in Figure 6c, the lower priority requests only have one node to send requests to.

In the dynamic reconfiguration with prioritization and QoS scheme, each website has two kinds of QoS guarantees: (i) A Hard QoS guarantee which specifies the minimum number of resources allotted to the website at any point of time (this guarantee represents the amount of resources allotted to the website in a rigid manner and that cannot be reconfigured) and (ii) A Soft QoS guarantee which specifies the number of resources allotted to the website when the resources are available and have not been already allotted to a higher priority website (this guarantee represents the amount of resources that are allotted to the website in an optimistic manner; these resources can be reconfigured to other websites, but when the load on this website increases the resources that had been reconfigured to other lower priority websites are taken back as long as the Hard QoS guarantees of the other websites are maintained).

This scheme converts nodes from lower priority websites only if the soft QoS guarantee of the higher priority website is not met. If the soft QoS guarantee of the higher priority website is already met, no further reconfiguration is done. Figure 7 depicts the behavior of this scheme in the previous example with each website having a hard QoS guarantee of two nodes and a soft QoS guarantee of six nodes.

Figure 7a shows the initial state of the system where both the websites have a small amount of requests coming in. Again, both websites are initially configured with four nodes each; each website is provided a soft QoS guarantee of six nodes. Figure 7b shows a state where there is a heavy load of requests to website B. Similar to the previous schemes, since website A is not that heavily loaded at this time, the nodes servicing this website are reconfigured to serve website B. Figure 7c shows a state where there is a heavy load of requests for website A, while website B is still heavily loaded. This scheme is similar to the previous scheme in the sense that when the high priority requests arrive, the scheme reconfigures nodes serving the lower priority website B to serve the higher priority website A. However, the scheme reconfigures nodes only till the soft QoS guarantee of website A is met, i.e., it reconfigures nodes till website A has six nodes serving it. This ensures that the lower priority website (website B) is ensured of having at least its Hard QoS guarantee of two nodes (in this example). It is to be noted that, by chang-

ing the soft QoS guarantees of the website A, the amount of starvation of website B can be controlled in this scheme (e.g., if website A was only provided with a QoS guarantee of five nodes, website B could utilize three nodes in such a scenario).

5 Experimental Results

In this section, we present various performance results. First, in Section 5.1, we present the ideal case raw performance achievable by the native Verbs API (VAPI) over InfiniBand and TCP/IP over InfiniBand (IPoIB) using micro-benchmark results. In Section 5.2, we present the impact of the load conditions in the data-center environment on the performance achievable by VAPI and IPoIB. In Section 5.3 we present the evaluation of our VAPI based reconfiguration schemes in a shared data-center environment.

For all our experiments we used two clusters whose descriptions are as follows:

Cluster1: A cluster system consisting of 8 nodes built around SuperMicro SUPER P4DL6 motherboards and GC chipsets which include 64-bit 133 MHz PCI-X interfaces. Each node has two Intel Xeon 2.4 GHz processors with a 512 kB L2 cache and a 400 MHz front side bus and 512 MB of main memory. We used the RedHat 9.0 Linux distribution and Linux-2.4.22smp kernel.org kernel.

Cluster2: A cluster system consisting of 8 nodes built around SuperMicro SUPER X5DL8-GG motherboards with ServerWorks GC LE chipsets which include 64-bit 133 MHz PCI-X interfaces. Each node has two Intel Xeon 3.0 GHz processors with a 512 kB L2 cache and a 533 MHz front side bus and 512 MB of main memory. We used the RedHat 9.0 Linux distribution and Linux-2.4.22smp kernel.org kernel.

The following interconnect was used to connect all the nodes in Clusters 1 and 2.

Interconnect: InfiniBand network with Mellanox InfiniHost MT23108 DualPort 4x HCA adapter through an InfiniScale MT43132 twenty-four 4x Port completely non-blocking InfiniBand Switch. The Mellanox InfiniHost HCA SDK version is thca-x86-3.2. The adapter firmware version is fw-23108-rel-3_2_0-rc4-build-001. The IPoIB driver for the InfiniBand adapters was provided by Mellanox Corporation. The driver was obtained from Mellanox Golden CD version 0.5.0.

Cluster 2 was used to represent the software load-balancers and Cluster 1 was used to represent the server nodes in the data-center environment. We used Apache version 2.0.50 in all our data-center experiments. For dynamic content requests, the application server was configured using PHP version 4.3.7 and the database used was MySQL version 4.1 with Master-Slave clustering for INNODB table type. Requests from the software load-balancers were generated using sixteen threads on each load-balancer.

5.1 Basic Micro-benchmarks

VAPI provides multiple communication models for transferring data, namely: (a) Send-Receive, (b) RDMA write, (c) RDMA write with immediate data and (d) RDMA Read. In this paper, we are particularly interested in the performance of the RDMA Read model and will be presenting the same.

RDMA Read achieves a latency of 11.89μs for 1 byte messages compared to the round trip latency of 53.8μs achieved by IPoIB. Also, RDMA Read achieves a peak bandwidth of 839.1 MBps as compared to a 231 MBps achieved by IPoIB.

5.2 Impact of Background Threads

In this section, we present performance results showing the impact of the loaded conditions in the data-center environment on the performance of RDMA Read and IPoIB on Cluster 1.

We emulate the loaded conditions in the data-center environment by performing background computation and communication operations on the server while the load-balancer performs the test with a separate thread on the server. This environment emulates a typical cluster-based shared data-center environment where multiple server nodes communicate periodically and exchange messages, while the load balancer, which is not as heavily loaded, attempts to get the load information from the heavily loaded machines.

The performance comparison of RDMA Read and IPoIB for this experiment is shown in Figure 8. We observe that the performance of IPoIB degrades significantly with the increase in the background load. On the other hand, one-sided communication operations such as RDMA show absolutely no degradation in the performance. These results show the capability of one-sided communication primitives in the data-center environment.

5.3 Evaluation of Dynamic Reconfigurability

In this section, we evaluate the basic Dynamic Reconfigurability scheme as well as our extensions to the scheme, namely *Dynamic Reconfigurability with Prioritization* and *Dynamic Reconfigurability with Prioritization and QoS* in several different scenarios. We

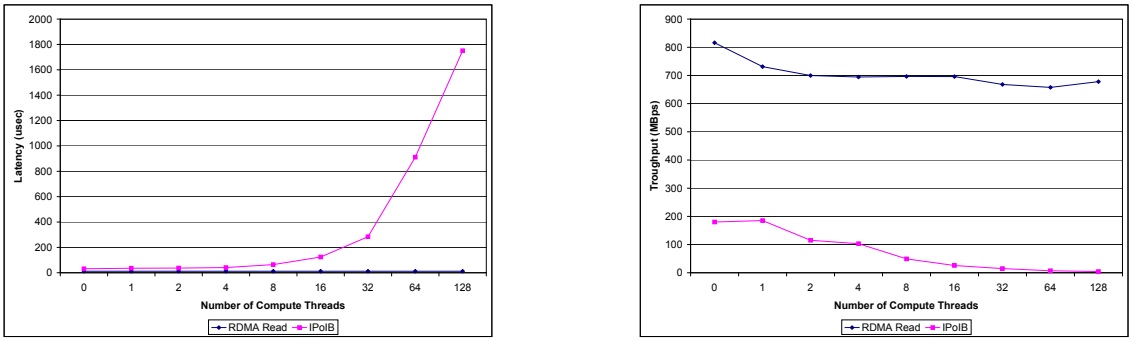

Figure 8: Performance of IPoIB and RDMA Read with background load: (a) Latency, (b) Bandwidth

first show the basic performance achieved by the dynamic reconfigurability scheme in Section 5.3.1. In Section 5.3.2, we evaluate the capabilities of the basic dynamic reconfigurability scheme in a shared data-center hosting multiple websites of different priority levels and compare it with the performance of the two extensions we proposed in this paper.

For our evaluations, we used three different traces. We used a single file trace with a request file which is 1KB, 4KB or 16KB in size in order to understand the impact of various schemes without being diluted by other system parameters. We also show the applicability of our scheme in a real environment using the Zipf workload benchmark trace [32] and a real-world WorldCup trace [7]. In this paper, we show the evaluation results with a single file trace of 1KB file size and some selected results with the Zipf and WorldCup traces.

5.3.1 Basic Dynamic Reconfigurability

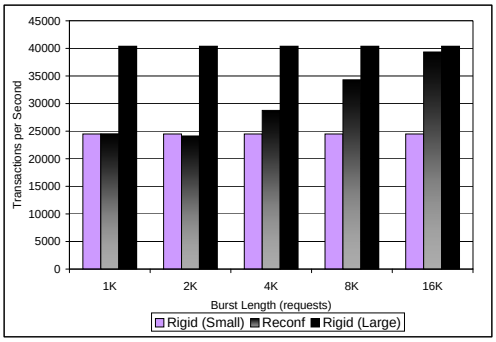

Figure 9: Reconfigurability Performance

In this section, we present the basic performance achieved by the dynamic reconfigurability scheme in a shared data-center hosting two websites with equal priorities and no QoS guarantees. We evaluate two instances of rigid or static node allocation: *Rigid-Small* and *Rigid-Large*. Rigid-Small considers a data-center with eight nodes and allots four nodes to each website (both websites are of equal priority). Rigid-Large considers a data-center with fourteen nodes and allots seven nodes to each website. We compare these schemes with the basic dynamic reconfigurability scheme; this scheme considers a data-center with only eight nodes and tries to emulate the capabilities of a larger data-center (Rigid-Large) by dynamically moving around nodes based on the request pattern.

The extensions for dynamic reconfigurability proposed in this paper make the scheme QoS and prioritization capable. However, in this experiment we are only considering equal priority websites with no QoS guarantees, thus these extensions would have no additional impact as compared to the basic dynamic reconfigurability scheme. Due to this reason, we present results only for the basic dynamic reconfigurability scheme in this section.

Figure 9 shows the performance achieved by these schemes. The x-axis depicts the length of requests to each website. For example, a burst length of 2K means that the data-center receives a burst of 2K requests for the first website followed by a burst of 2K requests for the second website and so on in a round-robin manner. We see that for small burst lengths the dynamic reconfigurability scheme performs comparably with the Rigid-Small scheme. As the burst length increases, its performance increases and converges with that of the Rigid-Large scheme. This is because, for small burst lengths the dynamic reconfigurability scheme does not have enough time to shift the nodes from the lightly loaded website to the heavily loaded website. As the burst length increases, the scheme has more time to move the nodes according to the load on the websites. Further, for burst lengths of close to 16K requests, the time taken to shift the nodes is negligible compared to the time for which the requests arrive. This reflects in a better perfor-

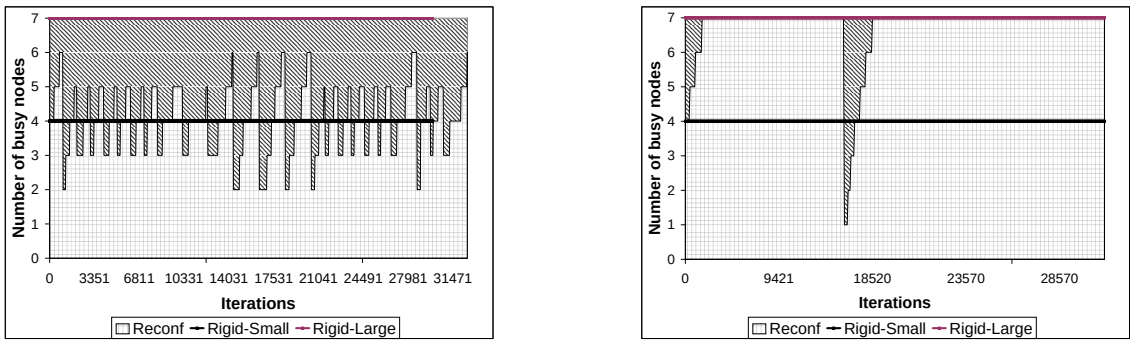

Figure 10: Node Utilization for Reconf (a) Burst Length = 1K and (b) Burst Length = 16K

mance for the dynamic reconfigurability schemes for large burst lengths.

To further understand the behavior of dynamic reconfigurability with burst length, we show the node utilization in the data-center for burst lengths of 1K and 16K in Figure 10. As seen in the figure, for a small burst length of 1K requests, the number of nodes used fluctuates rapidly and stays close to four nodes which is the initial number of nodes provided for the dynamic reconfigurability scheme as well as the Rigid-Small scheme. However, for a large burst length of 16K, the node utilization is high and close to the maximum.

5.3.2 Service Differentiation with Reconfiguration

In this section, we evaluate the *Dynamic Reconfigurability with Prioritization* and *Dynamic Reconfigurability with Prioritization and QoS* schemes and compare the performance achieved with that of the basic Dynamic Reconfigurability scheme. We carry out these evaluations in a shared data-center hosting two websites where one of the websites is a higher priority website compared to the other; each website has a hard QoS guarantee of two nodes and a soft QoS guarantee of six nodes. We compare the performance attainable for both the high priority as well as the low priority requests based on two metrics: (i) the number of transactions the data-center can support per second and (ii) the percentage of times it is able to meet the provided hard QoS guarantee.

In order to evaluate different aspects of the three schemes mentioned, we created three test case scenarios. Though several other scenarios are possible in a production data-center, we believe that these three scenarios would capture the bulk of the characteristic differences between the three schemes.

1. *Case 1:* A load of high priority requests arrives when a load of low priority requests already exists.
2. *Case 2:* A load of low priority requests arrives when a load of high priority requests already exists.
3. *Case 3:* Both the high priority requests and low priority requests arrive simultaneously.

Figure 11 shows the basic performance achieved by the three schemes for the high priority as well as low priority website. In the figure, legend "Reconf" refers to the basic reconfigurability scheme, "Reconf-P" refers to the reconfigurability with prioritization scheme and "Reconf-PQ" refers to the reconfigurability with prioritization and QoS scheme. The analysis of these results for the different cases are as follows:

Case1: As mentioned above, in this case, the data-center first experiences a burst of requests for the low-priority website. While, this burst of low-priority requests is still on, we study the performance of a new burst of requests for the high-priority website. Figure 12 shows the variation in the number of nodes allocated to the low-priority website for the basic reconfigurability, prioritization and prioritization with QoS schemes respectively.

Initially, when there is a burst of low-priority requests, all three schemes allocate more nodes to serve these low priority requests since the load on the high priority server nodes is initially low. However, the basic reconfigurability scheme and the prioritization scheme allocate the maximum number of nodes possible to the low-priority website, while the prioritization with QoS scheme allocates nodes up to the soft QoS guaranteed to that website. In our experiment, the prioritization with QoS scheme allocates up to six nodes to each website, while the other two schemes can allocate up to seven nodes.

Now, when there is a burst of high priority requests, each scheme takes a different path. The basic reconfigurability scheme does not have any concept of differentiated service and cannot find any lightly loaded server in the system; thus it does not do any additional reconfiguration. This would result in a

degradation of performance of the high priority requests. The prioritization and prioritization with QoS schemes, however, allow service differentiation and would reconfigure the nodes from the lower priority website to the higher priority website. Again, the prioritization scheme allocates the maximum number of nodes possible to the high priority website; the prioritization with QoS scheme on the other hand allocates nodes only till the soft QoS requirement of the website is met, i.e., it allocates a maximum of only six nodes in our experiment. We see that our extension schemes can achieve up to a factor of *five* improvement in performance as compared to the basic reconfigurability scheme.

Case2: In this case, the data-center first experiences a burst of requests for the high-priority website. While these requests are on, we study the impact of a burst of low-priority requests on the performance of the high-priority requests. On a burst of high-priority requests, all three schemes initially allocate more nodes to service the high priority website (prioritization with QoS only allocates up to the QoS guarantee, while the other schemes allocate up to the maximum possible number of nodes).

Now, when a burst of low-priority requests arrive, the basic reconfigurability scheme does not perform any additional reconfigurations; thus the performance of the high priority requests will not be affected. Similarly, for the prioritization and prioritization with QoS schemes, since the incoming requests are of a lower priority, there is no additional re-allocation of nodes. Thus, we expect all three schemes to perform in a similar manner in this case. However, it is to be noted that for the prioritization scheme, the incoming low-priority requests have only one node remaining to service them, while the prioritization with QoS scheme has two nodes to service the low priority requests.

Case 3: In this case, the data-center experiences a burst of both high priority and low priority requests at the same time. For the basic dynamic reconfigurability scheme, since all the nodes in the data-center are heavily loaded, there is no re-allocation of nodes. For the prioritization scheme, the maximum possible nodes are re-allocated to the high-priority website. Similarly, for the prioritization with QoS scheme, nodes are re-allocated to the high-priority website till the soft QoS guarantee for the high priority website is met.

QoS Meeting Capability: As mentioned earlier, the prioritization with QoS scheme reconfigures the available nodes to the loaded websites only till their soft QoS requirements are fulfilled. Thus, it might incur some amount of degradation in the peak performance it can provide to the high priority websites. However, this allows it to maintain the hard QoS requirements guaranteed to the high priority as well as the low priority websites.

Figure 13 compares the QoS meeting capabilities of each of the schemes for the three cases. We see that the basic reconfigurability and the prioritization schemes perform well in some cases for the high priority requests and in some other cases for the low priority requests. However, these schemes lack the consistency in providing the guaranteed QoS requirements to both the websites. The prioritization with QoS scheme on the other hand meets the guaranteed QoS requirements in all cases for both the websites. We also evaluated the three models discussed above for the Zipf workload benchmark trace and a real-world WorldCup trace. We see similar trends in performance for the three models as shown in Figure 14.

6 Related Work

Several researchers have focused on the design of adaptive systems that can manage clusters and/or react to changing workloads in the context of web servers [23, 18, 28, 13, 26, 16, 19, 31]. There has been some previous research which focus on dynamism in the data-center environment by HP labs and IBM Research [22, 25]. 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, 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 reconfigurability 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.

Ranjan, et. al., have also previously looked at the problem of providing QoS guarantees in data-center environments [27]. However, this work does not deal with shared data-centers. Also, while the approach suggested in this paper are suitable and efficient for several data-centers, it might not be potentially usable for highly loaded or compute intensive data-centers due to the two sided nature of communication used.

Shah, Kim, Balaji, et. al., have done significant research in User Level High Performance Sockets implementations [30, 20, 21, 10, 12]. In our previous work [9], we had evaluated the capabilities of such a pseudo-sockets layer over InfiniBand in the

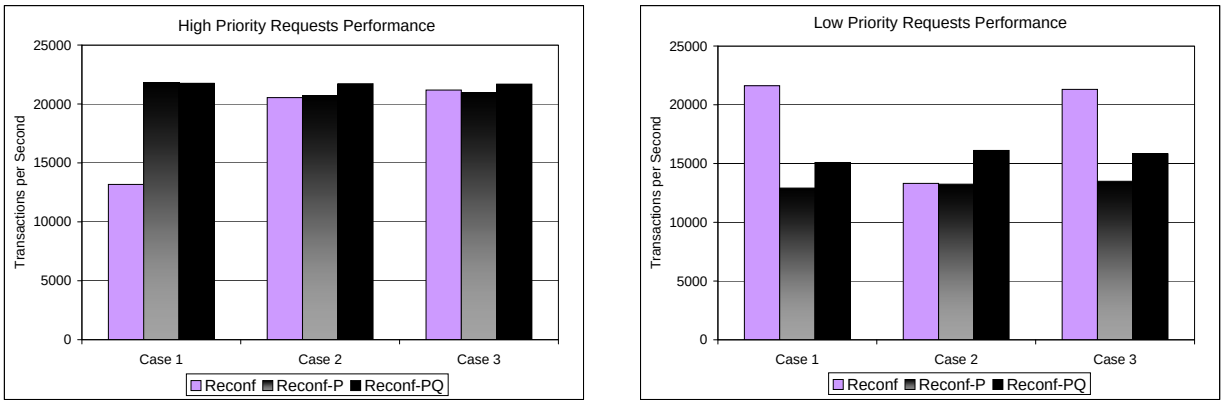

Figure 11: Reconfigurability Performance: (a) High Priority Requests and (b) Low Priority Requests

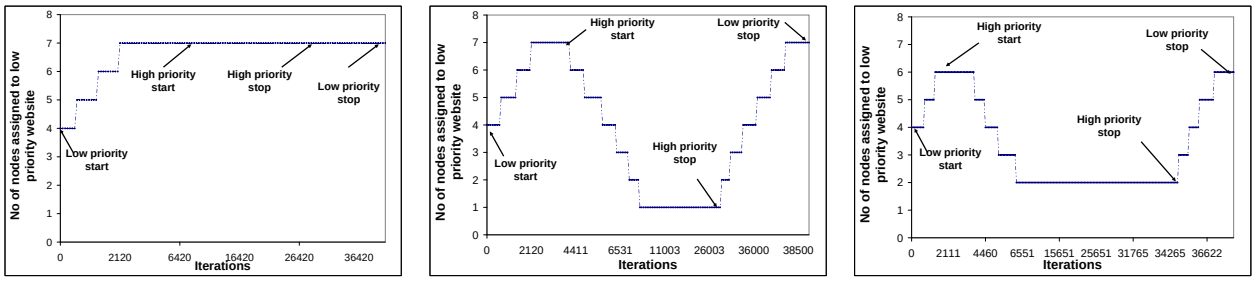

Figure 12: Node Usage: (a) Basic Reconfigurability, (b) Reconfigurability with Prioritization, (c) Reconfigurability with Prioritization and QoS

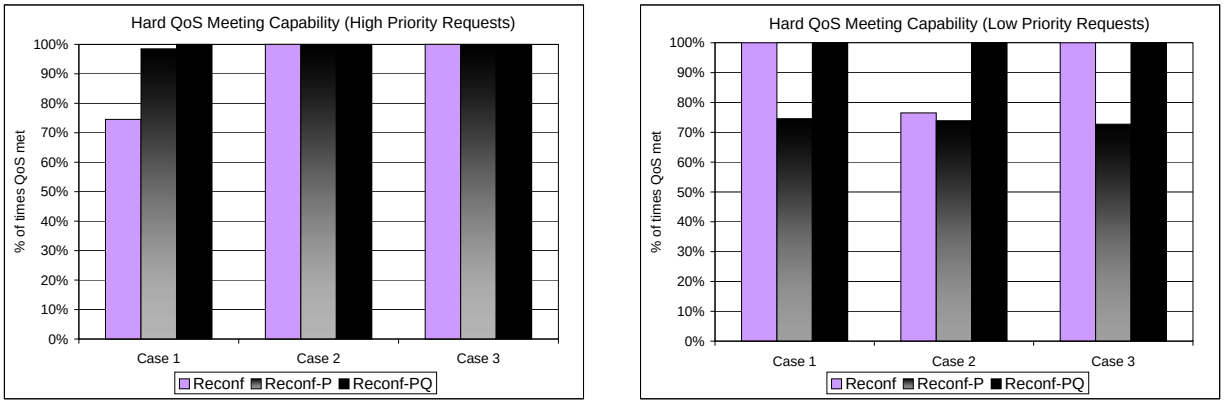

Figure 13: QoS Meeting capability: (a) High Priority Requests, (b) Low Priority Requests

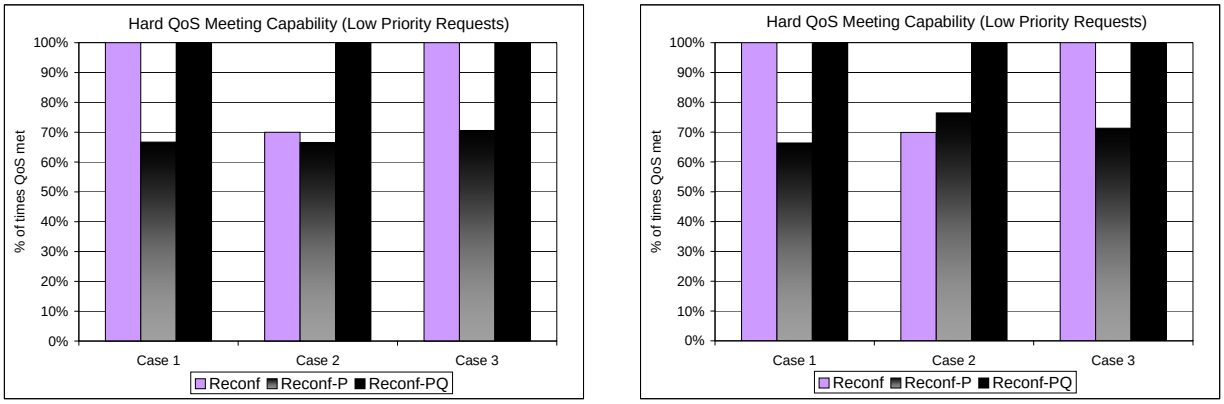

Figure 14: QoS Meeting capability for Low Priority Requests: (a) Zipf and (b) WorldCup Traces

data-center environment. However, as we had observed in [24, 11], the two-sided nature of Sockets API becomes an inherent bottleneck due to the high load conditions common in data-center environments. Due to this, we have focused on the one-sided nature of InfiniBand to develop our external modules. Further, the existing data-center framework (Apache, PHP, etc.,) is still based on the sockets API and can benefit from such high-performance sockets implementations. Thus, these approaches can be used in a complementary manner with our reconfigurability technique to make better utilization of system resources and provide high performance in a data-center environment.

7 Concluding Remarks

In this paper, we have extended our previously proposed work on dynamic reconfigurability to allow service differentiation in the shared data-center environment. In particular, we have pointed out the issues associated with the basic dynamic reconfigurability scheme and propose two extensions to it, namely (i) *Dynamic Reconfiguration with Prioritization* and (ii) *Dynamic Reconfiguration with Prioritization and QoS*. We have evaluated the benefits and issues associated with these schemes on an InfiniBand based cluster. Our experimental results show that our extensions can allow the dynamic reconfigurability scheme to attain a performance improvement of up to five times for high priority websites irrespective of any background low priority requests. Also, we have shown a significant improvement in the performance of low priority requests when there are minimal or no high priority requests in the system. Further, our extensions show a similar performance as a static scheme with up to 43% lesser nodes in some cases.

We are currently working on multi-stage reconfigurations. In the scheme presented, the least loaded node reconfigures itself to belong to the highest loaded tier in an attempt to share the load. However, due to the heterogeneity (hardware components available) in the cluster, this might not be the optimal solution. On the other hand, a multi-stage reconfiguration, where a sequence of changes in the different tiers allowing the most appropriate node be reconfigured to the high load tier, could be more beneficial.

8 Acknowledgments

We would like to thank Savitha Krishnamoorthy (Microsoft Corporation) and Dr. Jiesheng Wu (Ask Jeeves Corporation) for their contributions in the initial version of this paper.

References

[1] IBAL: InfiniBand Linux SourceForge Project. `http://infiniband.sourceforge.net/IAL/Access/IBAL`.

[2] IP over InfiniBand Working Group. `http://www.ietf.org/html.charters/ipoib-charter.html`.

[3] IPoIB: InfiniBand Linux SourceForge Project. `http://infiniband.sourceforge.net/NW/IPoIB/overview.htm`.

[4] Mellanox Technologies. http://www.mellanox.com.

[5] The DAT Collaborative. `http://www.datcollaborative.org/udapl.html`.

[6] The DAT Collaborative. `http://www.datcollaborative.org/kdapl.html`.

[7] The Internet Traffic Archive. http://ita.ee.lbl.gov/html/traces.html.

[8] InfiniBand Trade Association. http://www.infinibandta.org.

[9] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Sockets Direct Protocol over InfiniBand in Clusters: Is it Beneficial? In *ISPASS '04*.

[10] P. Balaji, P. Shivam, P. Wyckoff, and D. K. Panda. High Performance User Level Sockets over Gigabit Ethernet. In *Cluster Computing '02*.

[11] 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 *RAIT '04*.

[12] P. Balaji, J. Wu, T. Kurc, U. Catalyurek, D. K. Panda, and J. Saltz. Impact of High Performance Sockets on Data Intensive Applications. In *HPDC '03*.

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

[14] A. Chandra, W. Gong, and P. Shenoy. Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. In *Sigmetrics '03*.

[15] L. Cherkasova and S. R. Ponnekanti. Optimizing a content-aware load balancing strategy for shared Web hosting service. In *the Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems '00*.

[16] I. Chung and J. K. Hollingsworth. Automated Cluster-Based Web Service Performance Tuning. In *HPDC '04*, June 2004.

[17] Voltaire Inc. http://www.voltaire.com/.

[18] J. Carlstrom and R. Rom. Application-Aware Admission Control and Scheduling in Web Servers. In *Infocom '02*, June 2002.

[19] A. Jacob, I. Troxel, and A. George. Distributed Configuration Management for Reconfigurable Cluster Computing. In *ERSA '04*.

[20] J. S. Kim, K. Kim, and S. I. Jung. Building a High-Performance Communication Layer over Virtual Interface Architecture on Linux Clusters. In *ICS '01*.

[21] J. S. Kim, K. Kim, and S. I. Jung. SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture. In *Cluster Computing '01*.

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

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

[24] 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 *SAN '04*.

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

[26] P. Pradhan and R. Tewari and S. Sahu and A. Chandra and P. Shenoy. An Observation-based Approach Towards Self-Managing Web Servers. In *IWQoS '02*, 2002.

[27] S. Ranjan, J. Rolia, E. Knightly, and H. Fu. Qos-driven server migration for internet data centers, 2002.

[28] 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.

[29] H. V. Shah, D. B. Minturn, A. Foong, G. L. McAlpine, R. S. Madukkarumukumana, and G. J. Regnier. CSP: A Novel System Architecture for Scalable Internet and Communication Services. In *USITS '01*.

[30] H. V. Shah, C. Pu, and R. S. Madukkarumukumana. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In *CANPC '99*.

[31] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated Resource Management for Cluster-based Internet Services. In *OSDI '02*.

[32] George Kingsley Zipf. Human Behavior and the Principle of Least Effort. Addison-Wesley Press, 1949.