Dynamic Reconfigurability Support for providing Soft QoS Guarantees in Cluster-based Multi-Tier Data-Centers over InfiniBand

S. KRISHNAMOORTHY, P. BALAJI, K. VAIDYANATHAN, H. -W. JIN AND D. K. PANDA

Technical Report
OSU-CISRC-2/04-TR10

Dynamic Reconfigurability Support for providing Soft QoS Guarantees in Cluster-based Multi-Tier Data-Centers over InfiniBand*

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

Computer and Information Science,
The Ohio State University,
2015 Neil Avenue,
Columbus, OH43210
{savitha, balaji, vaidyana, jinhy, panda}@cis.ohio-state.edu

Abstract

Current cluster-based data-centers are configured as multiple tiers, each with specific functionalities and containing multiple nodes. Over-provisioning of nodes in these data-center tiers is an accepted approach to provide Quality of Service (QoS) guarantees due to the unpredictability of incoming requests. However, this approach is not cost-effective due to the wastage of resources it could potentially incur. Current high performance networks such as InfiniBand, Myrinet, Quadrics, etc., not only provide high performance in terms of latency and bandwidth, but also a number of advanced features such as one-sided communication operations including remote memory operations (RDMA) and network based atomic operations. In this paper, we present a novel architecture to provide dynamic reconfigurability of the nodes in a cluster-based data-center which enables them to adapt their functionality based on the system load and the QoS guarantees provided by the data-center; this avoids the need for over-provisioning of nodes. Dynamic reconfigurability in this scheme is achieved with the help of the one-sided communication operations offered by InfiniBand without requiring any modifications to the existing data-center applications. We evaluate this scheme with different patterns of static and dynamic content requests using three kinds of traces: (i) Single file traces, (ii) Zipf based traces and (iii) a real life world-cup trace. Our experimental results show that the dynamic reconfigurability scheme can be used to provide better QoS guarantees (up to 20% better), meet the same guarantees with lesser resources (up to 25% lesser nodes), or even both in some cases.

Keywords: QoS, Clusters, High Performance Networking, Data-Center, InfiniBand, RDMA, Atomic Operations

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #EIA-9986052, #CCR-0204429, and #CCR-0311542

1 Introduction

Commodity clusters are now becoming popular with the high performance community with the growth of advanced and high-performance networks. Some of leading products in the network interconnects market include Myrinet [12], Ethernet [20, 1, 23, 19], GigaNet cLAN [17], Quadrics [36, 35, 37, 4, 32]. Some of these interconnects provide very low latencies (even less than 10 μs) and very high bandwidth (of the order of Gbps). These interconnects provide memory-protected user-level access to the network interface, thereby allowing data transfer operations to be performed without kernel intervention. Thus the interconnect no longer is the bottleneck in the critical path of the communication. Some of the interconnects like InfiniBand Architecture (IBA) [5, 2] provide hardware-based support for Quality of Service(QoS) and for multicast operations. These provisions at the hardware level open avenues to develop higher-level layers using a novel approach. Being built from commodity hardware, Network of Workstations (NOWs) are becoming a cost-effective and viable alternatives to mainstream supercomputers for a broad range of applications and environments. Out of the current Top 500 Supercomputers, 149 systems are clusters [22].

During the last few years, the research and industry communities have been proposing and implementing user-level communication systems to address some of the problems associated with the traditional networking protocols for cluster-based systems. These user-level protocols such as U-Net [44], BIP [33, 34], FM [31], GM [18], EMP [42, 43] Virtual Interface Architecture (VIA) [13, 3, 10, 16, 9] was proposed earlier to standardize these efforts. InfiniBand Architecture (IBA) [5, 2] has been recently standardized by the industry to design next generation high-end clusters.

IBA is envisioned as the default interconnect for several

environments in the near future. 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 processor intervention by using DMA engines. It also provides features for performing network based atomic operations on the remote memory regions which can be leveraged in providing efficient support for multiple environments [27, 45].

On the other hand, the increasing adoption of Internet as the primary means of electronic interaction and communication has made highly scalable, highly available and high performance web servers a critical requirement for companies to reach, attract, and keep customers. Over the past few years, several researchers have been looking at the feasibility and the potential of using clusters in the data-center environment to form cluster-based multi-tier data-centers [6, 28, 40].

Figure 1 represents a typical cluster-based multi-tier data-center. The front tiers consist of front-end servers such as proxy servers that provide web, messaging and various other services to clients. The middle tiers usually comprise of application servers that handle transaction processing and implement the data-center business logic. The back-end tiers consist of database servers that hold a persistent state of the databases and other data repositories. As mentioned in [40], a fourth tier emerges in today's data-center environments: a communication service tier between the network and the front-end server farm for providing edge services such as load balancing, security, caching, and others.

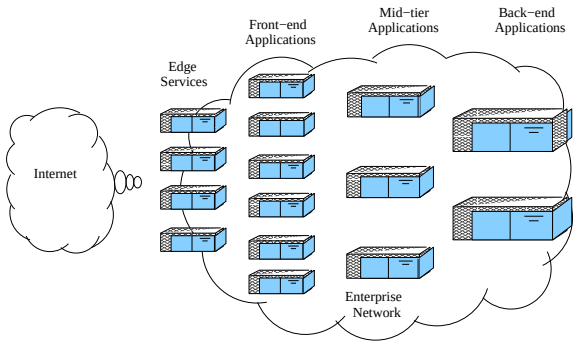


Figure 1. A Typical Multi-Tier Data-Center (Courtesy CSP Architecture design [40])

With ever-increasing online businesses and services and the growing popularity of personalized Internet services in the recent past, differentiation in the service provided to the end users is becoming critical in such data-center environments. Such differentiation becomes essential in several scenarios. For example, a data-center may try to provide some kind of guarantees in the response time perceived by frequent customers. So, requests from such customers need to be given a higher priority as compared to those coming from a new customer. Similarly, if a data-center is hosting web-sites for multiple companies, it may want to give a higher priority to all requests pertaining to company #1 (a high paying customer) as compared to company #2 (a low paying customer). Further, a data-center may try to give a better performance to all financial transactions such as web requests involving purchasing something from an online market as compared to web requests involving normal site browsing. These scenarios emphasize the need to have a provision for the data-centers to provide Quality of Service (QoS) guarantees to users.

There have been several studies on the relative frequencies of accesses between documents. However, there has not been much study on the actual arrival patterns of "similar" requests with respect to the resources they demand. Traditional data-centers base the configuration of their systems on assumptions on resource requirement for the client requests. For example, a data-center which expects requests for static documents might require more nodes in the proxy tier due to the ability of such documents to be cached. On the other hand, for a data-center which expects dynamic requests requiring some kind of computation to be performed, more nodes in the application tier would be beneficial to the compute intensive nature of these requests. However, due to interesting documents or dynamic web-pages becoming available and unavailable, there might be bursty traffic for certain kinds of documents at some times and for some other kinds of documents at a different time. Such scenarios cause request patterns to be unknown apriori. In this scenario, a burst of one kind of data requests would leave one tier in the data-center environment over-loaded while the nodes in the other tiers would be idle. Such dynamism in the incoming requests becomes especially concerning for data-centers which host multiple web-sites or services.

Over-provisioning of nodes in the data-center tiers is an accepted and much used approach to provide Quality of Service (QoS) guarantees due to the unpredictability of incoming requests. In this approach, nodes are allocated to the multiple tiers in the data-center environment based on the worst case requirements of the guaranteed QoS. This essentially means that each tier in the data-center is *Provisioned* with as many nodes as required by that tier alone to satisfy the QoS guarantees. Though this approach avoids the problem of the nodes in one tier being overloaded during bursty traffic, it is not cost-effective due to the potential wastage of resources incurred in the scheme. Also, with this increased resources the data-center can now intuitively provide much higher throughput than it actually guarantees.

However, it cannot exploit its resources in terms of QoS guarantees due to the unpredictability of incoming request patterns which makes the complete utilization of the data-center nodes rather indeterminate.

In this paper, we present a novel architecture to provide dynamic reconfigurability of nodes in the data-center which enables them to adapt their functionality based on the system load and QoS guarantees provided by the data-center. Dynamic reconfigurability in this scheme is achieved with the help of the one-sided communication operations (both remote memory access and network based atomic) offered by InfiniBand without requiring any modifications to the existing data-center applications. We evaluate this scheme with different patterns of static and dynamic content requests using three kinds of traces: (i) Single file traces, (ii) Zipf based traces and (iii) a real life world-cup trace. Our experimental results show that the dynamic reconfigurability scheme can be used to provide better QoS guarantees (up to 20% better), meet the same guarantees with lesser resources (up to 25% lesser nodes), or even both in some cases.

The remaining part of the paper is organized as follows: Section 2 provides a brief background in two aspects, (i) the tiered architecture of the data-center environment and details about the functionalities of each tier and (ii) the InfiniBand architecture. In Section 3, we describe the design and implementation details of the dynamic reconfigurability approach. In Section 4, we discuss more details into the issues related in providing Soft QoS guarantees to web-sites and the existing approaches used in this domain. We describe our experimental results in Section 5 and present some concluding remarks and possible future work in Section 7.

2 Background

In this section, we provide a brief background in two aspects, (i) the tiered architecture of the data-center environment and details about the functionalities of each tier and (ii) the InfiniBand architecture.

2.1 Data-Center Tiered Architecture

A typical data-center architecture consists of multiple tightly interacting layers known as tiers. Each tier can contain multiple physical nodes. Requests from clients are load-balanced on to the nodes in the proxy tier. This tier mainly does caching of content generated by the other back-end tiers. The other functionalities of this tier include embedding inputs from various application servers into a single HTML document (for framed documents for example), balancing the requests sent to the back-end based on certain pre-defined algorithms such as the load on the different nodes and other such services.

The second tier consists of two kinds of servers. First, those which host static content such as documents, images, music files and others which do not change with time. These servers are typically referred to as web-servers. Second, those which compute results based on the query itself and return the computed data in the form of a static document to the users. These servers, referred to as application servers, usually handle compute intensive queries which involve transaction processing and implement the data-center business logic.

The last tier consists of database servers. These servers hold a persistent state of the databases and other data repositories. These servers could either be compute intensive or I/O intensive based on the query format. For simple queries, such as search queries, etc., these servers tend to be more I/O intensive requiring a number of fields in the database to be fetched into memory for the search to be performed. For more complex queries, such as those which involve joins or sorting of tables, these servers tend to be more compute intensive.

Other than these three tiers, various data-center models specify multiple other tiers which either play a supporting role to these tiers or provide new functionalities to the data-center. For example, the CSP architecture [40] specifies an additional edge service tier which handles security, caching, SAN enclosure of packets for TCP termination and several others. In this paper, we only deal with the traditional 3-tier data-center architecture.

Since the kind of requests coming in to the data-center is not known apriori, the nodes present in the cluster are distributed to the various tiers based on a certain *expected workload*. Thus, as discussed earlier, in a data-center which expects more requests for static documents would have more nodes in the proxy tier, while a data-center which expects dynamic requests involving computing results for each request would have more nodes in the application tier and so on.

2.2 InfiniBand Architecture

The InfiniBand Architecture (IBA) is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. In a typical IBA cluster, switched serial links connect the processing nodes and the I/O nodes. The compute nodes are connected to the IBA fabric by means of Host Channel Adapters (HCAs). IBA defines a semantic interface called as Verbs for the consumer applications to communicate with the HCAs. VAPI is one such interface developed by Mellanox Technologies.

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. This is achieved by providing the consumer applications direct and protected access to the HCA. The specification for Verbs includes a queue-based interface, known as a Queue Pair (QP), to issue requests to the HCA. Figure 2 illustrates the InfiniBand Architecture model.

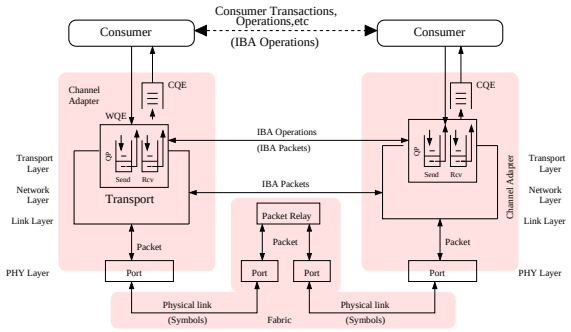


Figure 2. InfiniBand Architecture (Courtesy InfiniBand Specifications)

Each Queue Pair is a communication endpoint. A Queue Pair (QP) consists of the send queue and the 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). The application can either choose the request to be a Signaled (SG) request or an Un-Signaled request (USG). When the HCA completes the processing of a signaled request, it places an entry called as the Completion Queue Entry (CQE) in the Completion Queue (CQ). 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 kind of 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.

2.2.1 RDMA Communication Model

IBA supports two types of communication semantics: channel semantics (send-receive 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 one-to-one correspondence between every send and receive operation. Failure to post a receive 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.

Most 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 which has to be specified for RDMA operations. 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 user buffer. Similarly, in an RDMA Read operation, the initiator reads data from the remote node's user buffer.

2.2.2 Atomic Operations Over IBA

In addition to RDMA, the reliable communication classes also optionally atomic operations directly against the memory at the end node. Atomic operations are posted as descriptors at the sender side as in any other type of communication. However, the operation is completely handled by the NIC and involves very little host intervention and resource consumption.

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 use to compare two 64-bit values and swap the remote value with the data provided if the comparison succeeds.

Atomics are effectively a variation on RDMA: a combined write and read RDMA, carrying the data involved as immediate data. Two different levels of atomicity are optionally supported: atomic with respect to other operations on a target CA; and atomic with respect to all memory operation of the target host and all CAs on that host.

3 Design and Implementation

In this section, we describe the basic design issues in the dynamic reconfigurability scheme and some details about the implementation of this scheme using the native Verbs layer over InfiniBand (VAPI).

3.1 Reconfigurability Support

Requests to a data-center may vary in terms of resources such as the amount of computation (dynamic requests) or in terms of the memory (static requests serviced from cache) they demand. The basic idea of reconfigurability is to utilize the idle nodes of the system to share the dynamically varying resource requirements (memory or CPU) of requests in the data-center. Dynamic reconfigurability requires some extent of functional equivalence between the nodes of the data-center. We provide this equivalence by enabling software homogeneity such that each node is capable of belonging to any tier of a multi-tier data-center. Depending on the resource that is currently in demand (e.g., due to a burst of a certain kind of requests), nodes reconfigure themselves to support these requests.

Designing such dynamic adaptability and reconfigurability in the system involves various design issues and challenges. Some of the major ones are listed below.

- Support for Existing Applications
- Providing a System Wide Shared State
- Concurrency Control to avoid Live-locks and Starvation
- Equating CPU and I/O loads for different tiers
- Avoiding server thrashing through history aware reconfiguration
- Tuning the reconfigurability module sensitivity

We present some of these challenges in the following few subsections.

3.1.1 Support for Existing Applications

A number of applications have been developed to allow a highly efficient and concurrent file access to the various web requests. These applications have been developed over a span of several years and modifying them to allow additional features such as dynamic reconfigurability support is cumbersome and impractical. In light of this, we try to design our algorithms in such a way that there are no changes required to the existing applications.

The basic idea in our design is to achieve reconfigurability by using *external helper modules* which work along with the traditional data-center applications such as Apache, but in a transparent manner. All issues related to load verification, maintaining a logical global view, reconfigurability, etc., are handled by these modules and are obscured from the data-center servers, i.e., using these modules, dynamic reconfigurability is achieved with absolutely no changes to the data-center applications.

These modules work independently on the nodes and determine the best configuration for the data-center at any point in time. Based on this decision, they automatically start or stop the various servers on the physical nodes and modify the appropriate run-time configuration files used by the data-center servers to reflect this decision. The data-center servers on the other hand, just continue with their request processing based on the run-time configuration files, unaware of the modifications the modules have made.

3.1.2 System Wide Shared State

As discussed in Section 3.1.1, the external helper modules present in the system, handle various issues related to reconfigurability. However, the decision each module needs to make is pertinent to the global state of the system and can not be made based on the view of a single node. So, these modules need to communicate with each other to share such information regarding the system load, current configuration of the system, etc. Further, these communications tend to be asynchronous in nature. For example, the nodes in the front tiers are not aware about when the nodes in the back-end tiers require their state information, etc.

An interesting point to note in this communication pattern is the amount of replication in the information exchanged between the nodes. For example, let us consider a case where the information is being shared between the proxy tier and the application tier in the data-center. Here, each proxy provides its state information to each one of the nodes in the application tier every time they need it, i.e., the same information needs to be communicated with every node in the application tier.

Based on these communication patterns, intuitively a global shared state (as presented in Figure 3) seems to be the ideal environment for efficient distribution of data amongst all the nodes. In the architecture described by this figure, the nodes in the proxy tier can write their relevant information into the shared state and the nodes in the application tier can asynchronously read this information without disturbing the nodes in the proxy tier. This architecture essentially depicts a producer-consumer scenario for non-consumable resources.

One approach for implementing such a shared state, is by distributing the data across the physical memories of various nodes and allowing the nodes in the data-center to read or write into these memory locations. While an implementation of such a logical shared state is possible using the traditional TCP/IP based sockets interface (with the modules explicitly reading and communicating the data upon request from other nodes), such an implementation would lose out on all the benefits a shared state could provide. In particular: (i) All communication needs to be explicitly performed by the nodes in the proxy tier by sending (replicated) infor-

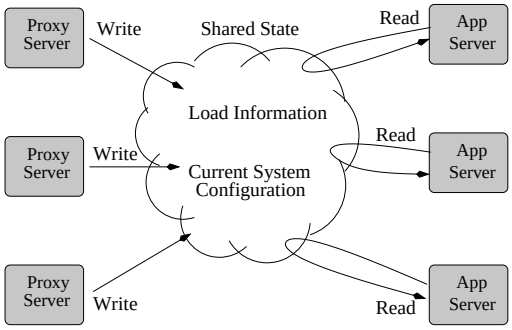


Figure 3. Shared State Architecture

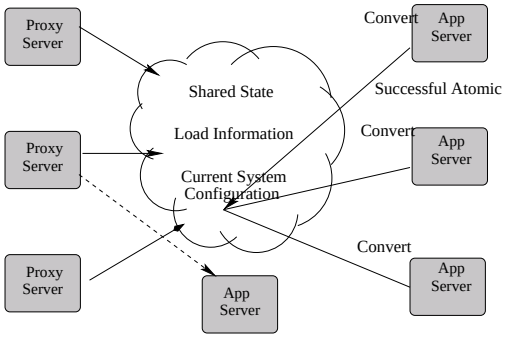


Figure 4. Shared State Architecture With Coherency Requirement

mation to each of the nodes in the application tier and (ii) Asynchronous requests from the nodes need to be handled by either using a signal based mechanism (using the SIGIO signal handler) or by having a separate thread block for incoming requests, both of which require the proxy node host intervention.

Further, as we had observed in our previous work [28], due to various factors such as the skew and the load on the remote nodes, which are very common in the data-center environment, even a simple two sided communication operation might lead to a significant degradation in the performance.

On the other hand, InfiniBand provides several advanced features such as one-sided communication operations, including RDMA operations. In our implementation, each node in the proxy tier writes information related to itself on its local memory. The nodes in the application tier can directly read this information using an RDMA read operation without disturbing the proxy node at all. This implementation of a logical shared state retains the efficiencies of the initially proposed shared state architecture, i.e., the nodes in the application tier can read data asynchronously from the shared state and the nodes in the proxy tier do not explicitly need to send the data to each of the nodes in the application tier.

The reconfigurability modules occasionally check the state of the system in the logical shared state and update their information based on this. However, the frequency in which each module checks the state is not the same. Modules on the proxy tiers need to check the information for every request so that they can send it to the appropriate back-end server on a cache miss. However, the modules on the application tier only need to read the shared state occasionally to check the system load and decide if a reconfiguration is required. Based on this, we decided to go ahead with a shared state with non-uniform access for the different nodes, i.e., the shared data is closer to the nodes in the proxy tier (in their local physical memory) compared to the nodes in the application tier.

3.1.3 Shared State with Concurrency Control

The logical shared state presented in Section 3.1.2 provides a simplistic view of the shared state architecture where one set of nodes write information pertinent to them to distinct variables in the shared state while a different set of nodes read these variables. Once a module reads this information and makes a decision about the best configuration for the data-center, it can modify the system configuration information in the shared state. At this point, one basic issue with the simple shared state architecture needs to be revisited.

As described earlier, the simple shared state architec-

ture described in Section 3.1.2 forms a producer-consumer scenario for non-consumable resources, i.e., the nodes in the front tiers only write data to the shared state and the nodes in the back-end tiers only read data from the shared state. However, since the modules on the application tier are the ones which are reading the shared state information and making the decision about the best configuration for the data-center, they will need to modify the system configuration information in the shared state. Also, for this modified system configuration to be reflected at the nodes in the front tier, the corresponding modules will also need to read information from the shared state. This means that all the nodes in the data-center will need to concurrently read and write from the shared state.

As shown in figure 4 multiple servers can simultaneously try to reconfigure the data-center under loaded conditions. This shared-state should be modifiable but only by one node while all other nodes are concurrently notified of this change to avoid thrashing. In order to solve this problem, it is imperative to sequentialize any change in configuration and keep all modules aware of these changes as they occur. Otherwise a stable configuration may never be reached even after the requirements of the current system load are met. In order to achieve the required concurrency between the multiple accesses of the data by the nodes, in our implementation, we use the network based remote atomic operations provided by InfiniBand.

In our implementation of the shared state with concurrency control, a node makes an atomic compare-and-swap operation on the status of the remote node, when a configuration change is required. If the node's view of the system configuration is accurate, the atomic operation is successful and the swap operation is performed. However, if the node's view of the system configuration is not accurate, the atomic operation fails and returns the current system configuration to the node trying to perform the operation. Also, the node succeeding in making the change is responsible to communicate the reconfiguration information to the other modules in the system, so that they are aware of the current configuration of the system. Figure 5 illustrates the approach used to achieve concurrency control in this architecture.

3.1.4 I/O and CPU Load Equivalence

Though load is a factor greatly affecting the current configuration of the data-center, the perceived load in different tiers is different. While the load at the front end is mostly due to the number of requests serviced from the apache cache (file system I/O), the load at the back-end servers is mainly perceived as CPU utilization due to the computational demands of the incoming requests. Hence it is important to determine the load conditions at the node depending on the tier in which it currently belongs to. Other factors such as

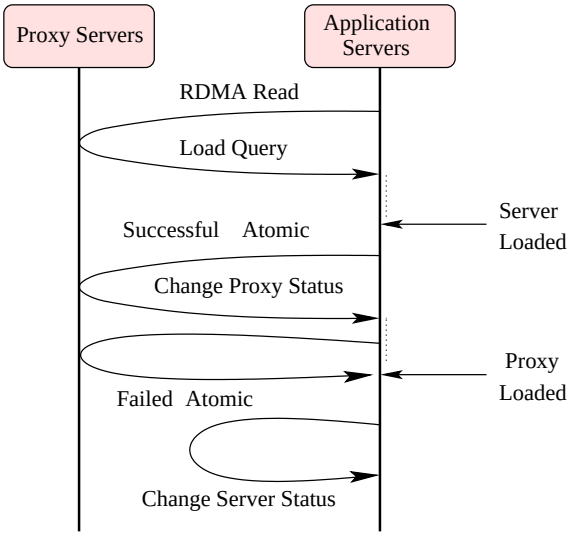


Figure 5. Concurrency Control

number of active and passive connections also determine the load on the system.

At the application tier, we use the load on the CPU as well as the fraction of the total number of incoming requests currently being serviced at the back-end to determine the load at the back-end application tier. The fraction of requests that were serviced from cache determines the front-end tier load.

3.1.5 History Aware Reconfiguration

Due to the irregular nature of incoming requests, a small burst of similar requests might potentially trigger a reconfiguration in the data-center tiers. Because of this, small bursts of similar requests can cause nodes in the data-center tiers to be moved between the various tiers to satisfy the instantaneous load, resulting in *thrashing* in the data-center configuration.

To avoid such *thrashing*, in our scheme, we allow a history aware reconfiguration of the nodes, i.e., the nodes in one tier are re-allocated to a different tier only if the load in the remote tier stays high for a pre-defined period of time *T*. This approach has its own trade-offs. A small value for *T* could result in *thrashing* in the data-center environment. On the other hand, a large value of *T* could make the approach less respondent to bursty traffic providing a similar performance as that of the non-reconfigurable or rigid system. The optimal value of *T* depends on the kind of workload and request pattern. While we recognize the importance of the value of *T*, in this paper, we do not concentrate on the effect of its variation and fix it to a pre-defined value for all the experiments.

3.1.6 Reconfigurability Module Sensitivity

As mentioned earlier, the modules on the nodes in the application tier occasionally read the system information from the shared state in order to decide the best configuration at that instant of time. The time interval between two consecutive checks is a system parameter *S* referring to the sensitivity of the reconfigurability modules. A small value of *S* allows a high degree of sensitivity, i.e., the system is better respondent to a variation in the workload characteristics. However, it would increase the overhead on the system due to the frequent monitoring of the state. On the other hand, a large value of *S* allows a low degree of sensitivity, i.e., the system is less respondent to variation in the workload characteristics. At the same time, it would also result in a lower overhead on the system to monitor the state. We have performed experiments to study this variation and the results for the same are presented in Section 5.

4 Soft Quality of Service Guarantees

In this section, we discuss the current trend in the data-center environments to provide some kind of a differentiated service to different requests. We also discuss the currently used over-provisioning approach for providing such differentiated service and describe the use of the reconfigurability approach to facilitate similar QoS guarantees using a smaller number of nodes. Finally, we describe the potential impacts of the reconfigurability approach for data-centers which host multiple web-sites.

4.1 Quality of Service Considerations

As mentioned earlier, with the increasing online businesses and services and the growing popularity of personalized Internet services, differentiation of service provided to the end users is becoming critical in the data-center environment. There are multiple kinds of QoS guarantees that have been studied in the literature.

The first kind of distinction in the various kinds of QoS guarantees is between response-time guarantees and throughput guarantees. In response time guarantees (or end-to-end QoS), a guarantee on the response time perceived by the end user is provided, i.e., the end users are the final customers for the QoS provider. On the other hand, for throughput guarantees, assuming a sufficiently high incoming rate of requests, a certain throughput (transactions per second) of servicing for these requests would be guaranteed. In this case, the web-site owner is the end customer for the QoS provider.

Throughput based QoS guarantees become extremely relevant for several web-hosting services and ISPs which have multiple different websites hosted on the same data-center nodes. In this paper, we concentrate on this kind of QoS guarantees.

4.2 Over-Provisioning approach to provide QoS

Currently, the most common approach used to provide throughput guarantees in the data-center environment is by using over-provisioning of nodes. In this approach, nodes are allocated to the multiple tiers in the data-center environment based on the worst case requirements of the QoS guarantees. For example, suppose the data-center providing a throughput guarantee of *N* transactions per second, is expecting either static or dynamic content. Say, in the case where there are only static requests, suppose the data-center needs *P1* nodes in the proxy tier and *A1* nodes in the application tier. Similarly in the case where there are only dynamic requests, suppose the data-center needs *P2* nodes in the proxy tier and *A2* nodes in the application tier. With over-provisioning, max(*P1, P2*) nodes are provided in the proxy tier and max(*A1*, *A2*) nodes are provided in the application tier. It can be seen that this approach can easily deliver the requested throughput.

However, during a burst of static content, most of the nodes in the application tier remain idle since the requests are cached in the proxy tier. Similarly, during a burst of the dynamic content, most of the nodes in the proxy tier remain idle since the requests can not be cached and need to be forwarded to the application tier. This essentially points to the fact that, though this approach is effective in terms of performance, it is not cost-effective due to the wastage of resources it could potentially incur.

4.3 Reconfigurability approach to provide QoS

In this paper, we propose the reconfigurability approach to efficiently support throughput based QoS guarantees. As described in Section 3, reconfigurability allows dynamically shifting nodes from one tier in the data-center to another tier based on the system load as well as the QoS guarantees provided by the data-center. Considering the same example as the one in Section 4.2, in this case we would only need max(*P1*, *P2*, *A1*, *A2*) + 1 nodes to provide nearly the same QoS guarantees as the over-provisioning case[1]. In this approach, when there's a burst of static or dynamic requests, the scheme automatically shifts nodes from the lesser utilized tiers to the more utilized tiers. This would create a more balanced load amongst the various nodes in the data-center improving the overall performance provided. It is

[1] Ideally, the reconfigurability scheme should only use max(*P1, P2, A1, A2*) number of nodes. However, the current implementation of the reconfigurability modules do not support sharing the same physical node for multiple servers. This puts a restriction to have at least one node in each tier, requiring max(*P1, P2, A1, A2*) + 1 nodes in the above mentioned example

to be noted that the reconfigurability approach is the most useful when the traffic is very bursty, which is quite common in typical data-center environments. When the traffic is not very bursty, even a rigid configuration of nodes would utilize all the nodes in the data-center. In this case, the reconfigurability scheme would perform comparably with the normal rigid configuration based scheme.

4.4 Reconfigurability in Multi-Website scenarios

As mentioned earlier, throughput based QoS guarantees become extremely relevant for several web-hosting services and ISPs which have multiple different websites hosted on the same data-center nodes. In this case, the data-center might want to provide a higher priority to all requests pertaining to website #1 (a high paying customer) as compared to website #2 (a low paying customer). In this scenario, the over-provisioning based scheme would give more number of nodes to the high priority website and a lesser number of nodes to the low priority website. In this case, having a burst of requests for the lower priority requests would overload the nodes allocated to the low priority requests while the nodes allocated to the high priority requests would remain idle. However, the reconfigurability based scheme can dynamically reassign nodes for the low priority requests, improving their performance as well.

5 Experimental Results

In this section, we present four sets of results. First, we present the ideal case performance achievable by the native Verbs API (VAPI) over InfiniBand in the form of micro-benchmark results. Second, in Section 5.2, we present the improvement in the basic performance achievable through dynamic reconfigurability. Third, we demonstrate the enhanced ability to provide soft Quality of Service guarantees using dynamic reconfiguration in Section 5.3 and show the results for multi-website hosting data-centers in Section 5.4.

For all our experiments we used 2 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 with the kernel.org SMP kernel version 2.4.22smp.

Cluster2: A cluster system consisting of 16 nodes which include 64-bit 66 MHz PCI interfaces. Each node has four Intel Pentium-III 700 MHz processors with 1 MB L2 cache and a 400 MHz front side bus and 1 GB of main memory. We used the RedHat 7.1 Linux distribution with the kernel.org SMP kernel version 2.4.18.

All nodes in Cluster1 and Cluster2 are interconnected with multiple networks:

Interconnect1: 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-x86-3.0.1-build-003. The adapter firmware version is fw-23108-rel-3_00_0001-rc4-build-001. This interconnect connects all nodes in Cluster1.

Interconnect2: Myrinet network with 133 MHz LANai 9.1 processors, connected through a Myrinet 2000 network. The GM version used is 1.6.3. This interconnect connects all nodes in both Cluster1 and Cluster2.

For all the experiments, we used Cluster 1 to represent the data-center environment and Cluster 2 to represent the clients. We used Apache-2.0.48 for the data-center related experiments. The experiments used 6 client nodes with 12 threads for each client with traces containing nearly 20 million requests distributed among the client threads.

Interconnect 1 was used to perform the Remote DMA operations and NIC-based atomic operations. The implementation of the design was carried out using the VERBS interface API over Interconnect 1. Interconnect 1 and Interconnect 2 were used as the inter-node networks, for communication within the data-center and for communication between the external clients and the data-center.

The use of high-speed interconnects for external clients serves as a means to generate high traffic and load on the data-center nodes.

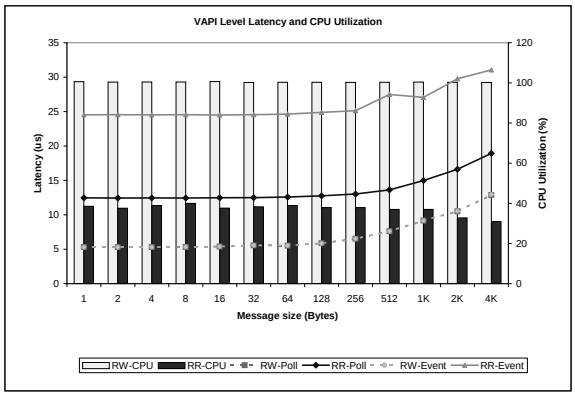
5.1 Micro-Benchmark Results

In this section, we present the ideal case performance achievable by the Verbs API (VAPI) over InfiniBand using micro-benchmark tests.

As mentioned earlier, InfiniBand provides two mechanisms for completion notification. The first approach requires the host application to continuously poll on the completion queue and check for the completion of the message transmission or reception. The second approach allows the host application to request a interrupt based notification from the network adapter. The notification based approach incurs the additional cost of an interrupt. The polling based approach does not incur this additional cost, but results in a high CPU utilization due to the continuous polling of the completion queue. In this section, we present results for both the polling based approach as well as the notification based approach for Cluster 1.

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.

Figure 6a shows the one-way latency achieved by the



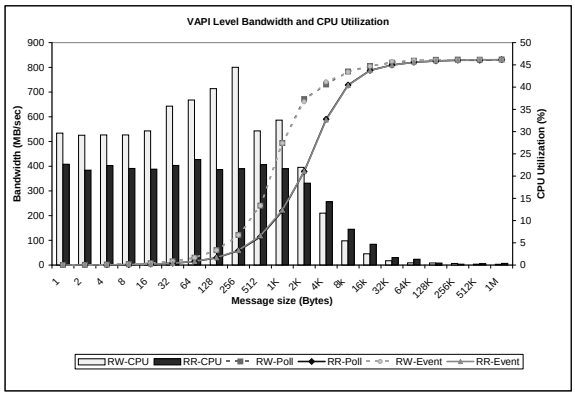
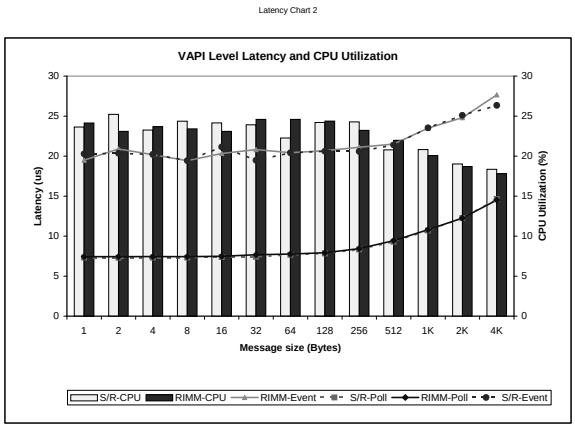


Figure 6. Micro-Benchmarks for RDMA write and read: (a) Latency and (b) Bandwidth



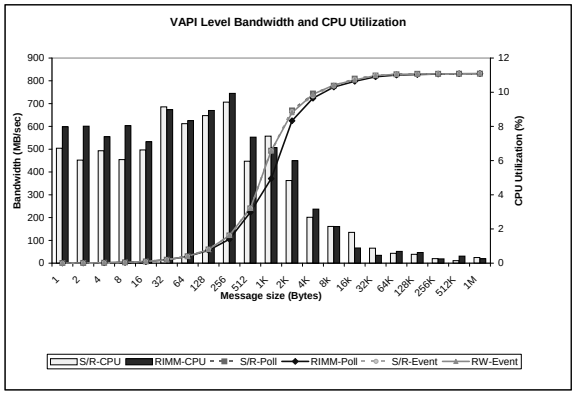


Figure 7. Micro-Benchmarks for Send/Recv and RDMA with Immediate: (a) Latency and (b) Bandwidth

VAPI RDMA write and RDMA read communication models for various message sizes. RDMA write achieves a latency of around 5.5μs for 4 byte messages compared to the 10.5μs achieved by RDMA read in the polling based approach. For the notification based approach, RDMA write continues to achieve a latency of around 5.5μs while that of the RDMA read approach goes to 30μs. This indifference in the latency for the RDMA write communication model toward the completion approach is attributed to the way the micro-benchmark test was written. RDMA write is receiver transparent. So, the only way the receiver can know about the arrival of an incoming message is by polling on the last byte of the message. This results in the notification based approach for RDMA write to be equivalent to the polling based approach, resulting in a high utilization of the CPU and a latency similar to that of the polling based approach.

Figure 6b shows the uni-directional bandwidth achieved by the two models. Again, we see that for both the communication models, the polling based approach and the notification based approach perform comparably. Further, the CPU for both the approaches is around 20% for small messages and negligible for large messages.

Figure 7a shows the uni-directional latency achieved using the Send-Receive and the RDMA write with immediate data schemes for various message sizes. While RDMA write with immediate achieves up to 7.9μs, the Send-Receive scheme achieves up to 6.7μs for 4 byte messages. However, RDMA with immediate data shows a maximum CPU utilization of 13% and Send-Receive shows a maximum of 30% CPU utilization.

Figure 7b shows the uni-directional bandwidth performance using these two schemes. Both Send-Receive and RDMA with immediate data achieve high bandwidths comparable to the Remote DMA schemes, with lower CPU utilizations.

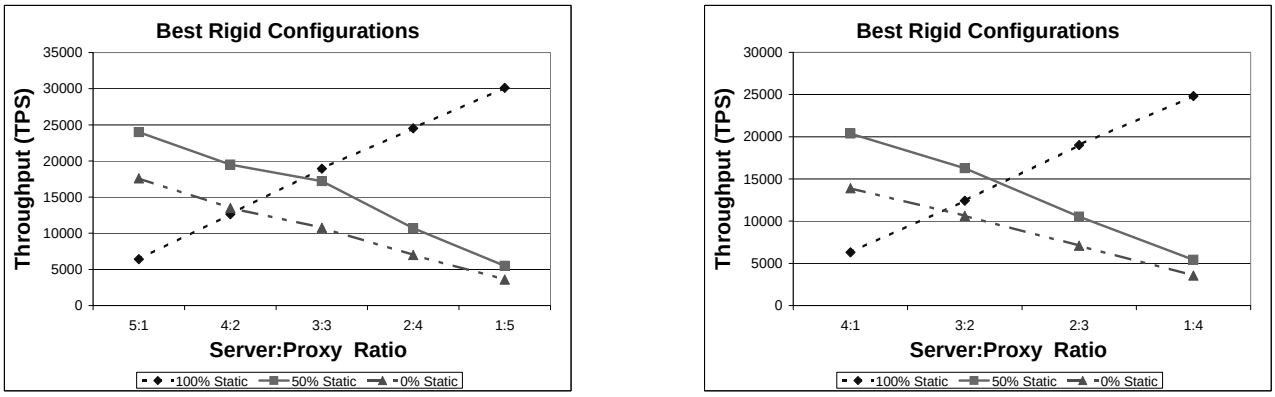


Figure 9. Best Rigid Configurations for (a) A 6-Node Data-Center, (b) A 5-Node Data-Center

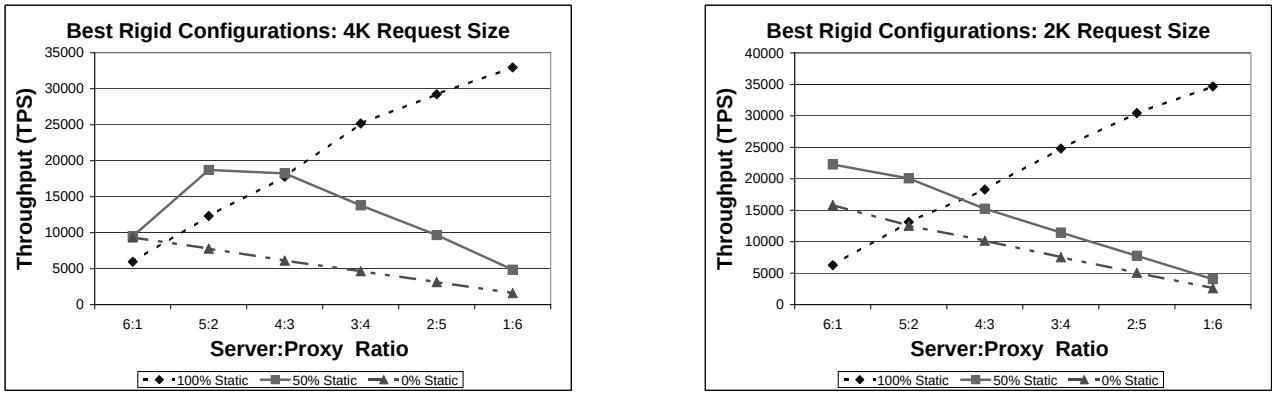


Figure 10. Best Rigid Configurations: Different Average Request Sizes (a) 4KB , (b) 2KB

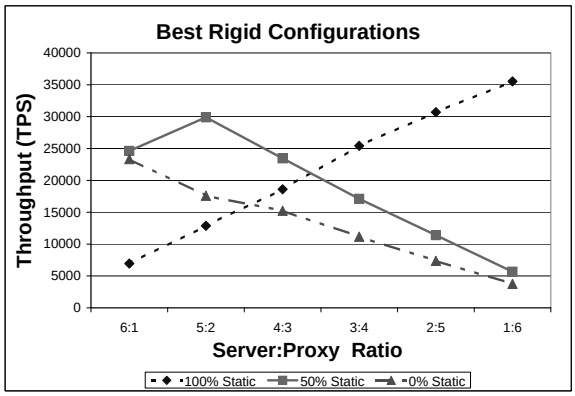


Figure 8. Best Rigid Configurations: 7-Node Data-Center

5.2 Basic Performance of Dynamic Reconfigurability

In this section, we present the basic performance benefits achieved by the reconfigurability based scheme as compared to a standard data-center which does not have any such support.

5.2.1 Best Rigid Configurations

In this section, we study the performance of the rigid configuration scheme for different data-center configurations. In the subsequent sections, we choose the best static configuration possible for a given workload based on these results.

Figure 8 shows the performance of the rigid configuration based scheme for different workloads and data-center configurations for a seven-node Data-Center. This is the data-center size that has been considered for all future experiments. As discussed earlier, static content is capable of being cached and would benefit from a large number of nodes being present in the proxy tier. On the other hand, dynamic content require large amounts of back-end computation and would benefit from a large number of nodes being present in the application tier. This can be seen in Figure 8 where a trace with only static requests (100% static) would perform the best for 6 nodes in the proxy tier and 1 node in the application tier. On the other hand, a trace with only dynamic requests (0% static) would perform the best for 1 node in the proxy tier and 6 nodes in the application tier. Configurations for intermediate workloads such as 50% static and 50% dynamic content depend on the amount of computation required for the requests. In our trace, a combination of 2 nodes in the proxy tier and 5 nodes in the application tier performed the best.

We have also evaluated the performance of traces in three different dimensions: (a) The number of nodes present in the data-center, (b) the average file sizes of the requested data and (c) the percentage of dynamic and static content present in the trace. Figure 9 shows the best configuration for various patterns of requests for data-centers consisting of 6 Nodes and 5 Nodes, for various mix of static and dynamic requests. Figure 10 shows the best configuration for various percentages of static content when the average request size in the data-center is 4KB and 2KB.

5.2.2 Performance for bursty traffic

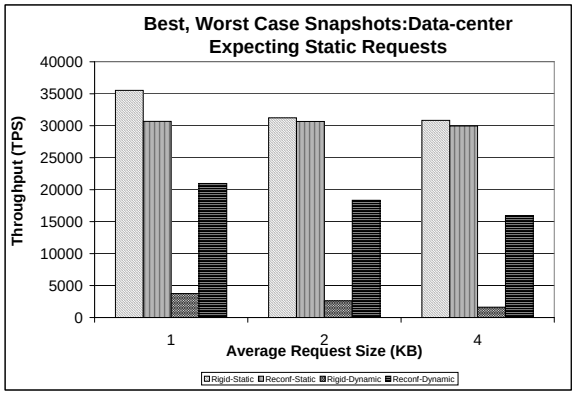
In this section, we present the performance of the dynamic reconfigurability scheme as compared to the rigid configuration scheme in several scenarios varying the burstiness of the traffic.

Figure 11a shows the performance of the data-center that is initially configured expecting purely static data. We show the performance of the data-center for snapshot of request conforming to this expectation and a snapshot of requests that is totally different from expected pattern. Figure 11 shows the performance for data-center when it expects purely dynamic requests. In both cases we see that a re-configurable data-center is able to provide sustained performance owing to increased utilization of resources at all times as compared to a rigidly configured data-center.

Figure 12a shows the snap-shot performance of a data-center that is configured for 50% static and 50% dynamic mixed requests during periods incurring one type of request. Based on the results from Section 5.2.1, we chose the best configuration for the rigid scheme for this workload with 5 nodes in the application tier and 2 nodes in the proxy tier. In this scenario, when there's a burst of dynamic requests, only 5 of the nodes are fully utilized (in the application tier) while the 2 nodes in the proxy tier are relatively idle. Similarly, when there's a burst of static requests, only 2 of the nodes are fully utilized (in the proxy tier) while the 5 nodes in the application tier are idle. On the other hand, the reconfigurability based scheme is dynamically able to reconfigure itself to give the best performance during each burst of traffic.

This shows that reconfigurability can take advantage of idle nodes during such bursts to give better throughput by improving utilization. On the whole, reconfigurability achieves an improvement of about 20% during a burst of dynamic content and up to 150% during a burst of static content. It is to be noted that the actual amount of benefit achieved varies depending on the amount of computation performed by the requests.

Figure 12b shows the overall performance of the data-center for different traces with bursts of similar files (the burst length is depicted on the x-axis of the graph), for different compute requirements. We can see that for small burst lengths, both the rigid based approach and the recon-



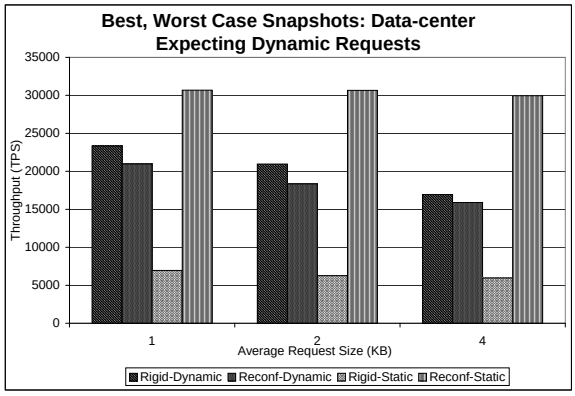
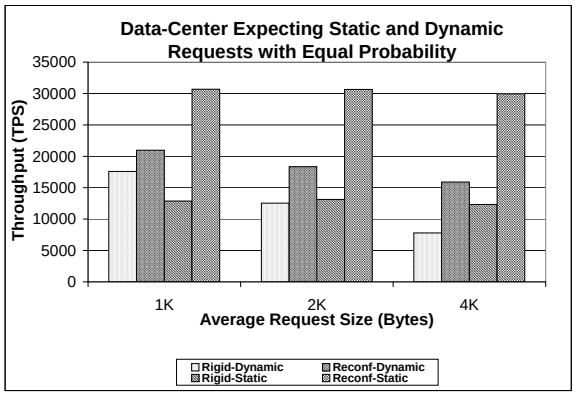


Figure 11. Snapshot performance for Data-Center: (a) Expecting Static Requests (b) Expecting Dynamic Requests



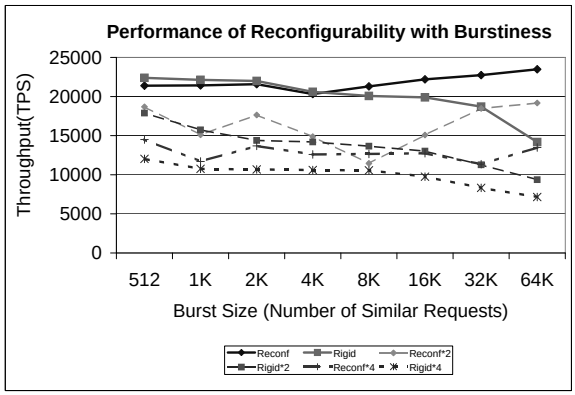
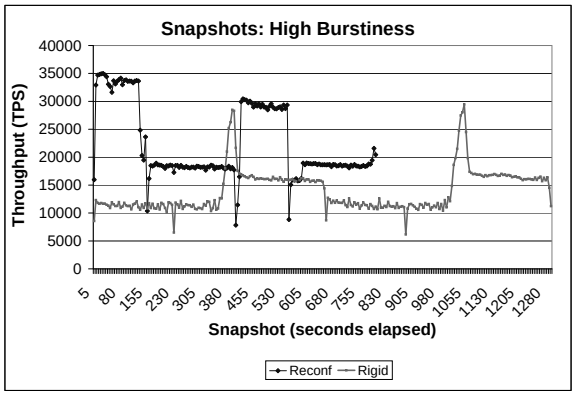


Figure 12. Performance for bursty traffic: (a) Snapshot performance, (b) Overall performance: Various Compute requirements



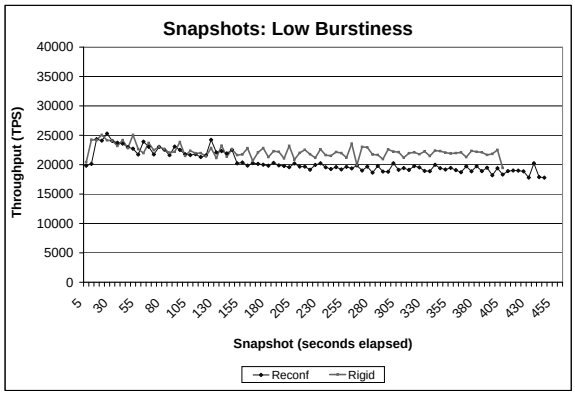


Figure 13. Throughput at Various Snapshots

figurability based approach perform equally for small compute requirements. This is because, when the burst length is very small, even a rigid scheme can make sure that all the nodes are utilized as requests do not contend for a single type of resource. However, when the burst length gets larger, the rigid scheme drops in performance while the reconfigurability scheme continues to give a high performance. For a burst length of 64K requests, we see an improvement of around 80% for the reconfigurability scheme. However, We can see from this result that as the compute requirement increases an adaptable data-center can outperform a rigid one for all sizes of burstiness.

Figures 13a and 13b show the throughput given by the system at different snap-shots for high and low burst lengths respectively. For high burst lengths, the dynamic reconfigurability scheme provides a much higher performance as compared to the rigid scheme. Again, for low burst lengths, both the schemes give comparable performances.

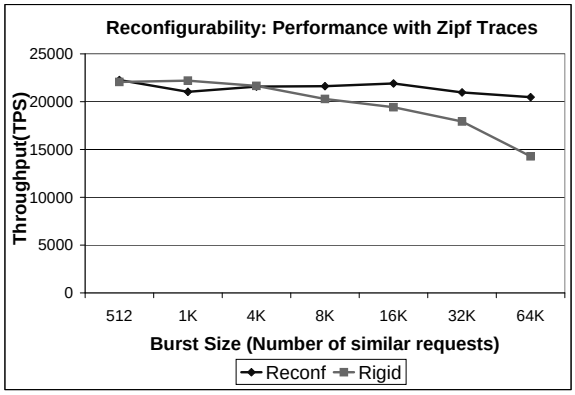


Figure 14. Performance of a Zipf Trace

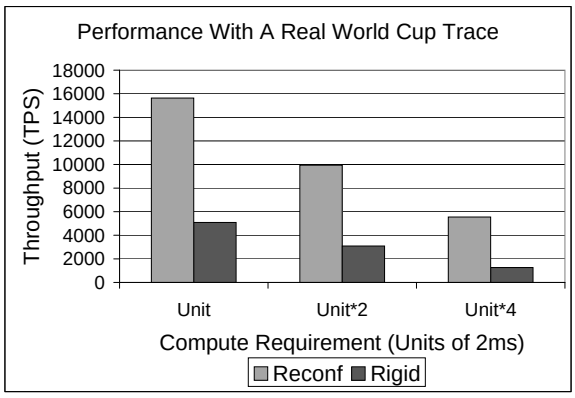


Figure 15. Performance of a Real Trace

Performance of a Zipf trace: Figure 14 shows the performance of the reconfigurability scheme for traces following a Zipf pattern. We see that the performance of the scheme is similar to that of the single file based trace. We have also performed experiments with real life workloads.

Figure 15 shows the performance of reconfigurability with a real life world cup trace, consisting two-thirds of dynamic requests in bursts, having some static file requests at intervals. The rigid data-center was configured to accept an equal mix of both type of requests. This performance was evaluated for different assumed requirements of computation for these dynamic requests. In this case, we are able to see up to three times improvement with an adaptable data-center. We clearly see that in scenarios like work cup traces, such extensive bursts of requests requiring a single type of resource, namely computation can take advantage of the adaptability to provide performance to the end clients.

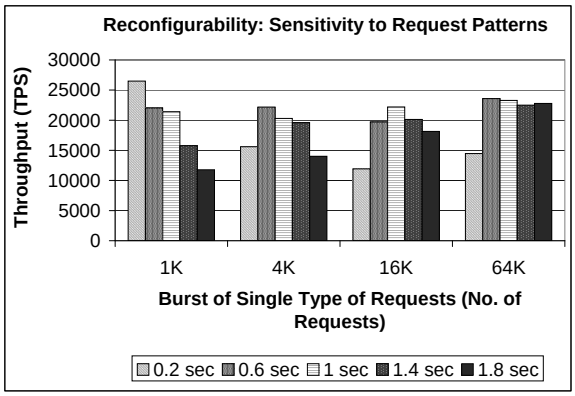
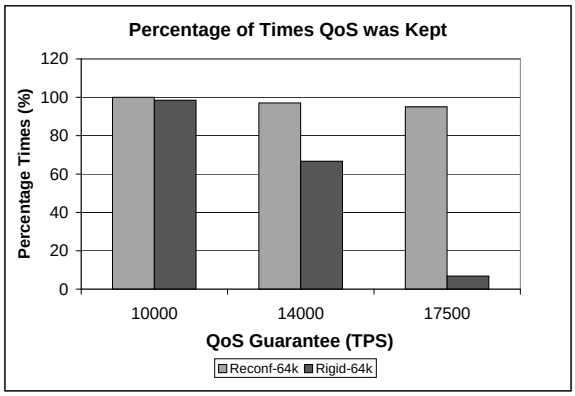


Figure 16. Impact of Sensitivity on the reconfigurability module

Impact of sensitivity on the reconfi gurability module: Figure 16 shows the impact of sensitivity of the module to reconfigure over the throughput achievable for different types of request patterns. As mentioned earlier, a small value for the sensitivity factor makes the system more respondent to a change in the workload pattern (more sensitive), but adds more overhead. On the other hand, a large value for the sensitivity factor makes the system less respondent to a change in the workload pattern (less sensitive), but adds a lesser overhead.

As shown in the figure, for traces with small burst lengths, we see a drop in performance when the sensitivity factor is high (1.8 sec). This is because, when the burst length is low, reconfigurability is only beneficial if it is able to respond to the small bursts quickly. A high sensitivity factor reduces its capability to do so. On the other hand, for traces with high burst lengths, we see a drop in performance when the sensitivity factor is low (0.2 sec). This is because, when the burst length is high, additional probes of the system state are only adding a high overhead for the system without any benefit.



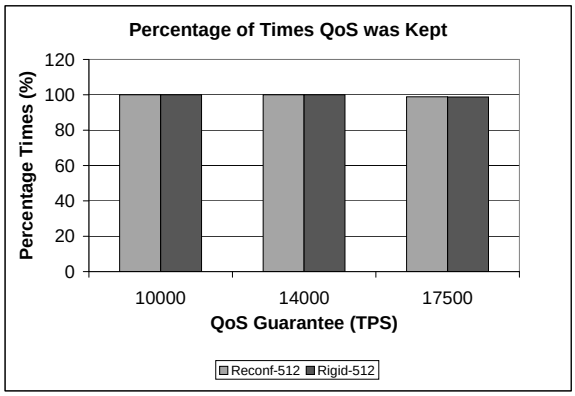


Figure 17. QoS guarantee keeping capability: (a) High Burstiness, (b) Low Burstiness

5.3 Quality of Service Guarantees

Figures 17a and 17b show the percentage of times the two schemes (rigid and reconfigurability) keep the QoS guarantees promised for traces with high (burst length = 64K) and low (burst length = 512) burst lengths respectively. For the trace with low burstiness, we can see that both the schemes are able to meet the promised QoS nearly 100% of the times. For large burst lengths (64K), when the QoS guarantee provided is low, the rigid scheme is still able to meet the QoS requirement nearly 100% of the times. However, in this case, when the QoS guarantee provided is increased, we can see that the QoS meeting capability of the rigid scheme drops drastically to about 5%. On the other hand, with the reconfigurability scheme, we can keep QoS guarantees close to 100% of the times irrespective of the burstiness of incoming request patterns even for high QoS requirements.

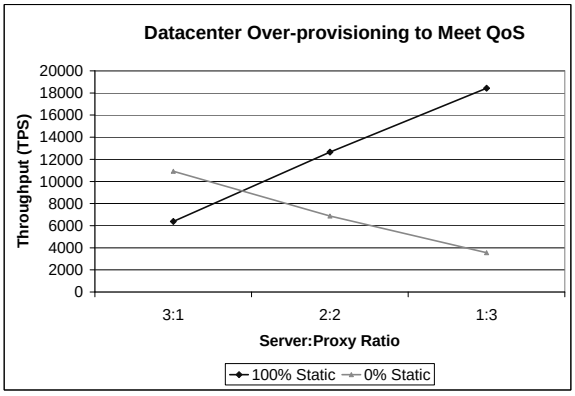


Figure 18. Best Rigid Configuration for 4 nodes

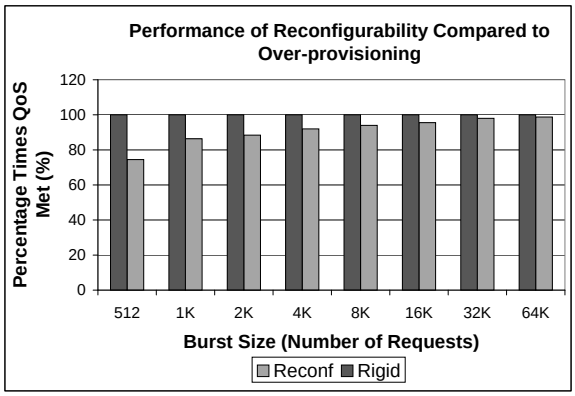
Over-Provisioning To Meet QoS Guarantees: Figure 18 shows the performance of the rigid scheme for different kinds of workloads and data-center configurations. As we can see in the figure, the peak throughput guarantee we can provide is bounded by the performance achieved for dynamic content. This is because dynamic content need to be processed at the back-end before returning the data to the user unlike static content which can be cached at the proxy tier. Based on this, we picked a QoS guarantee of 9,000 TPS, which the over-provisioning based scheme can provide using 3 nodes in the application tier and 2 nodes in the proxy tier (note that we don't need 3 nodes in the proxy tier since the same QoS for static content can be achieved with just two nodes in the proxy tier).

Figure 19a shows the performance of the reconfigurability scheme compared to the over-provisioning based scheme. We see that when the burst length is low, reconfigurability is able to meet the QoS guarantee only about 75% of the times while the over-provisioning based scheme meets it almost 100% of the times. However, when the burst length becomes larger, both schemes meet the QoS guarantees nearly always. This again points to the efficient utilization of all the nodes in the system by the reconfigurability based scheme.

Figure 19b shows the normalized performance of the reconfigurability scheme (normalized with respect to the number of nodes used) compared to the over-provisioning based scheme. We can see that both schemes are able to meet the QoS requirements nearly 100% of the times.

5.4 Multi-Website Hosting Servers

In this section, we present a scenario where the data-center hosts multiple websites. Though the scheme is generic and can allow any number of different websites, for ease of understanding, we have only considered two websites where one of them is a high priority website for which a QoS guarantee is given. The second website is a low priority website for which only the best effort service is guaranteed.



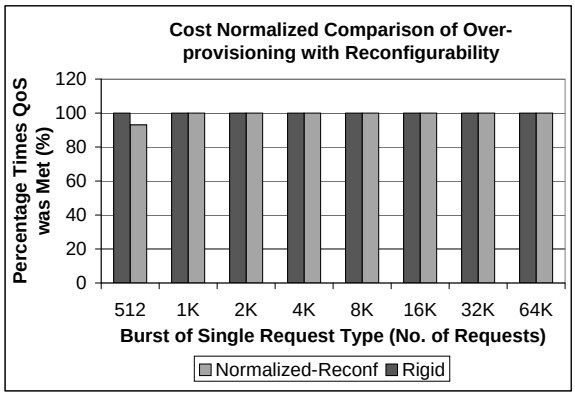


Figure 19. QoS guarantee keeping capability: (a) Against Over-Provisioning, (b) Normalized value with number of nodes used

For the rigid approach, we allow over-provisioning for the high priority requests by providing 5 nodes (3 nodes in the application tier and 2 nodes in the proxy tier) - similar to the configuration in Section 5.3 for over-provisioning. The remaining nodes (2 nodes) are provided to the low priority requests (1 node in the application tier and 1 node in the proxy tier). However, it is to be noted that the concept of over-provisioning is not strictly the best-case in this scenario because of the presence of multiple websites, i.e., the reconfigurability scheme can always shift nodes from the low priority website to the high priority website to allow a higher performance.

Figure 20a shows the QoS meeting capabilities for the rigid scheme (with over-provisioning) as well as the reconfigurability scheme. On guaranteeing the same QoS as the single website case (9,000 TPS) or lower, we observe that both the schemes are able to meet the guarantee 100% of the times. On the other hand, when we increase the guaranteed throughput to 12,000 TPS, for low burst lengths (512), the rigid scheme is still able to meet the QoS requirements 100% of the times. However, on increasing the burst length to 64K, the rigid scheme is only able to meet the QoS guarantee 40% of the times. The reconfigurability based scheme, however, meets the QoS guarantee 100% of the times in all cases.

Figure 21a shows the QoS meeting capability of the over-provisioning scheme and the reconfigurability scheme for the high priority requests with varying burst lengths.

Figure 21b shows the performance of the low priority requests. Since only a minimal number of nodes are allocated for the low priority requests, during a burst of low priority requests, the rigid scheme would provide a very low throughput. However, the reconfigurability scheme can reassign nodes to service the low priority requests, thus achieving a significantly higher throughput, especially for large burst lengths.

Based on these results we can observe that the reconfigurability scheme can: (i) achieve a much higher performance for low priority requests in a multi-website scenario and (ii) give better QoS guarantees than even the over-provisioning scheme, which was traditionally considered the best-case scenario with respect to QoS guarantee meeting capabilities.

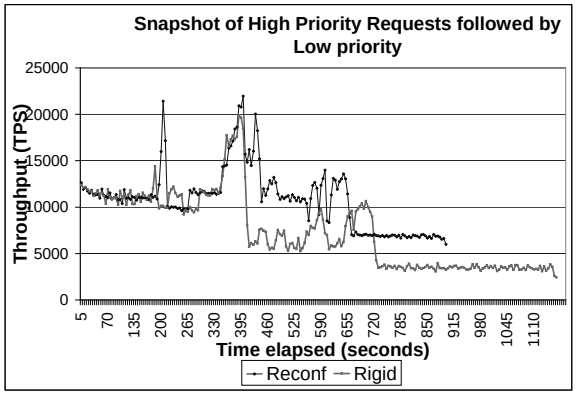
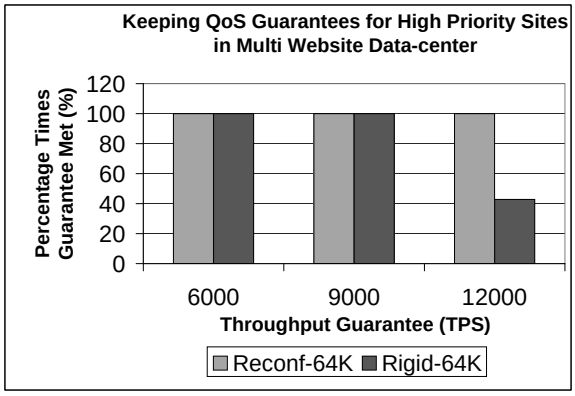


Figure 22. Behavior of Reconfigurability with Requests for Multiple sites

Figure 22 shows the snapshot performance for the data-center with a trace having bursts of high priority and low priority requests alternatively. We can see that the reconfigurability scheme performs better than the rigid scheme in nearly every case.

6 Related Work

There has been some previous research which focus on dynamism in the data-center environment by HP labs and IBM Research [26, 29]. These are notable in the sense that



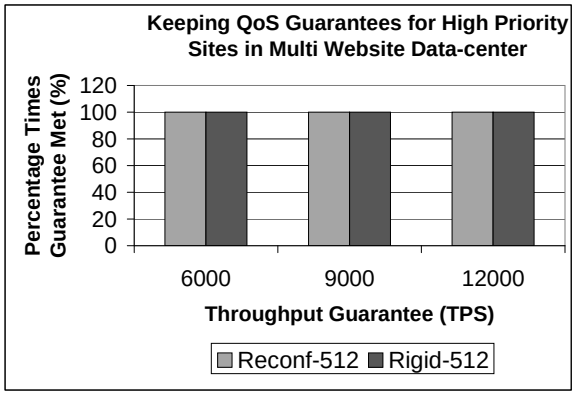
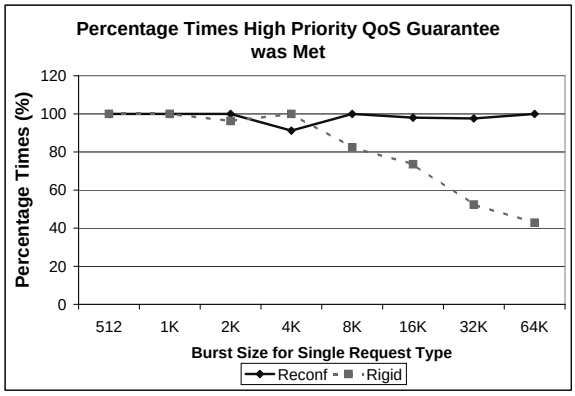


Figure 20. QoS guarantees with multiple sites hosted in the data-center: (a) High burstiness, (b) Low burstiness



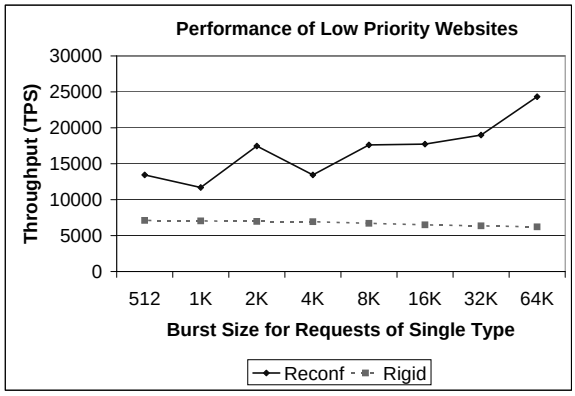


Figure 21. Performance with multiple sites hosted in the data-center: (a) QoS meeting capability for high priority requests, (b) Performance for low priority requests

they were the first to show the capabilities of a dynamic allocation of system resources in the data-center environment. However, these solutions focused on lower level architectural requirements mainly for storage related issues and are not aware of the guarantees or requirements of the application. Further, these rely on specific hardware to provide these solutions and are hard to look at as commodity component based solutions. 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.

Shah, Kim, Balaji, et. al., have done significant research in User Level High Performance Sockets implementations [41, 24, 25, 7, 8, 6]. In one of our previous works [6], we had evaluated the capabilities of such a pseudo-sockets layer over InfiniBand in the data-center environment. However, as we had observed in [28], 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 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.

There has been previous research in QoS guarantees. Senapathi, Gulati, et. al., have looked at providing differentiated service in terms of network bandwidth for scientific computing environments [39, 38, 21]. Bhatti et. al., have worked on architectures for supporting end-to-end response time guarantees [11]. Most of the research focus has been on improving QoS guarantees using novel methods of caching and load-balancing [15, 30]. The focus has also been on providing guarantees for real time applications [14]. We believe that these approaches can be used in a complementary manner with our re-reconfigurability technique to provide better QoS guarantees.

7 Concluding Remarks and Future Work

Current data-centers are configured as multiple tiers with each tier having a number of physical nodes. Each tier has specific functionalities and services which they provide to the users. With ever-increasing online businesses and the growing popularity of personalized Internet services, differentiation in the service provided to the end users in the form of Quality of Service (QoS) guarantees is becoming critical. Over-provisioning of nodes in the data-center tiers is an accepted and widely used approach to provide Quality of Service (QoS) guarantees due to the unpredictable nature of incoming requests. However, this approach is not cost-effective due to the wastage of resources it could potentially incur. On the other hand, current high performance networks such as InfiniBand, Myrinet, Quadrics, etc., not only provide high performance in terms of latency and bandwidth, but also a number of advanced features such as one-sided communication operations including remote memory operations (RDMA) and network based atomic operations.

In this paper, we presented a novel architecture to provide dynamic reconfigurability of nodes in the data-center which enables them to adapt their functionality based on the system load and QoS guarantees provided by the data-center; this avoids the need for over-provisioning of nodes. Dynamic reconfigurability in this scheme is achieved with the help of the one-sided communication operations offered by InfiniBand without requiring any modifications to the existing data-center applications. We evaluated this scheme with different patterns of static and dynamic content requests using three kinds of traces: (i) Single file traces, (ii) Zipf based traces and (iii) a real life world-cup trace. Our experimental results show that the dynamic reconfigurability scheme can be used to provide better QoS guarantees (up to 20% better), meet the same guarantees with lesser resources (up to 25% lesser nodes), or even both 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-level 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 Sundeep Narravula for all the help he provided in understanding the details and capabilities of the apache server. We would also like to thank Jiesheng Wu and several other members of the Network Based Computing group at the Ohio State University for all the valuable discussions we had during the course of the project.

References

[1] 10 Gigabit Ethernet Alliance. http://www.10gea.org/.

[2] InfiniBand Trade Association Specifications. http://www.infinibandta.org/estore.html.

[3] M-VIA: A High Performance Modular VIA for Linux. http://www.nersc.gov/research/FTG/via.

[4] Quadrics Supercomputers World Ltd. http://www.quadrics.com/.

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

[6] Pavan Balaji, Sundeep Narravula, Karthikeyan Vaidyanathan, Savitha Krishnamoorthy, Jiesheng Wu, and Dhabaleswar K. Panda. Sockets Direct Protocol over InfiniBand in Clusters: Is it Beneficial? In *the Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, Texas, March 10-12 2004.

[7] Pavan Balaji, Piyush Shivam, Pete Wyckoff, and Dhabaleswar K. Panda. High Performance User Level Sockets over Gigabit Ethernet. In *the Proceedings of the IEEE International Conference on Cluster Computing*, pages 179–186, Chicago, Illinois, September 23-26 2002.

[8] Pavan Balaji, Jiesheng Wu, Tahsin Kurc, Umit Catalyurek, Dhabaleswar K. Panda, and Joel Saltz. Impact of High Performance Sockets on Data Intensive Applications. In *the Proceedings of the IEEE International Conference on High Performance Distributed Computing (HPDC)*, pages 24–33, Seattle, Washington, June 22-24 2003.

[9] M. Banikazemi, B. Abali, L. Herger, and D. K. Panda. Design Alternatives for VIA and an Implementation on IBM Netfinity NT Cluster. *Special Issue of the Journal of Parallel and Distributed Computing (JPDC), Vol. 61, No. 11, pp. 1512-1545*, November 2001.

[10] M. Banikazemi, V. Moorthy, L. Hereger, D. K. Panda, and B. Abali. Efficient Virtual Interface Architecture Support for IBM SP switch-connected NT clusters. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, 2000.

[11] Nina Bhatti and Rich Friedrich. Web Server Support for Tiered Service. In *the Proceedings of the IEEE Network, September/October*, 1999.

[12] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su. Myrinet: A Gigabit-per-Second Local Area Network. http://www.myricom.com.

[13] P. Buonadonna, A. Geweke, and D. E. Culler. BVIA: An Implementation and Analysis of Virtual Interface Architecture. In *Proceedings of Supercomputing*, 1998.

[14] Surendar Chandra, Carla Schlatter Ellis, and Amin Vahdat. Application-level differentiated multimedia web services using quality aware transcoding. *IEEE Special Issue on QOS in the Internet*, 18(12):2544–2565, December 2000.

[15] G. Chen, C.L. Wang, and F.C.M. Lau. A scalable cluster-based web server with cooperative caching support.

[16] Compaq, Intel Corporation, and Microsoft Corporation. Virtual Interface Architecture (VIA) Specifications.

[17] GigaNet Corporations. cLAN for Linux: Software Users' Guide.

[18] Myricom Corporations. The GM Message Passing System.

[19] W. Feng, J. Hurwitz, H. Newman, S. Ravot, L. Cottrell, O. Martin, F. Coccetti, C. Jin, D. Wei, and S. Low. Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study. In *Proceedings of the IEEE International Conference on Supercomputing*, Phoenix, Arizona, November 2003.

[20] H. Frazier and H. Johnson. Gigabit Ethernet: From 100 to 1000Mbps.

[21] A. Gulati, D. K. Panda, P. Sadayappan, and P. Wyckoff. NIC-based Rate Control for Proportional Bandwidth Allocation in Myrinet Clusters. In *International Conference on Parallel Processing*, September.

[22] http://www.top500.org. Top 500 supercomputer sites.

[23] J. Hurwitz and W. Feng. End-to-End Performance of 10-Gigabit Ethernet on Commodity Systems. *IEEE Micro*, January 2004.

[24] Jin-Soo Kim, Kangho Kim, and Sung-In Jung. Building a High-Performance Communication Layer over Virtual Interface Architecture on Linux Clusters. In *the Proceedings of the IEEE International Conference on Supercomputing (ICS)*, pages 335–347, Naples, Italy, June 16-21 2001.

[25] Jin-Soo Kim, Kangho Kim, and Sung-In Jung. SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture. In *the Proceedings of the IEEE International Conference on Cluster Computing*, pages 399–408, California, USA, October 8-11 2001.

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

[27] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *SC*, June 2003.

[28] Sundeep Narravula, Pavan Balaji, Karthikeyan Vaidyanathan, Savitha Krishnamoorthy, Jiesheng Wu, and Dhabaleswar 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.

[29] Daniel OHare, Pankaj Tandon, Hemanth Kalluri, and Phil Mills. SNIA SSF Virtualization Demonstration. In *IBM Systems Group - TotalStorage Software: White Paper*, October.

[30] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, , and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *the proceedings of the ACM eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, October.

[31] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of Supercomputing*, 1995.

[32] F. Petrini, W. C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network (QsNet): High-Performance Clustering Technology. In *the Proceedings of the IEEE International Conference on Hot Interconnects*, August 2001.

[33] L. Prylli and B. Tourancheau. BIP: A New Protocol designed for High Performance Networking on Myrinet. In *Proceedings of the International Parallel Processing Symposium Workshop on Personal Computer Based Network of Workstations*, 1998.

[34] L. Prylli, R. Westerlin, and B. Tourancheau. Modeling of a High Speed Network to Maximize Throughput Performance: the Experience of BIP over Myrinet. In *Proceedings of Parallel and Distributed Processing Techniques and Applications (PDPTA '98)*, 1998.

[35] Quadrics Supercomputers World Ltd. Elan Programming Manual. 1999.

[36] Quadrics Supercomputers World Ltd. Elan Reference Manual. 1999.

[37] Quadrics Supercomputers World Ltd. Elite Reference Manual. 1999.

[38] S. Senapathi, B. Chandrasekharan, D. Stredney, H.-W. Shen, and D.K. Panda. QoS-aware Middleware for Cluster-based Servers to Support Interactive and Resource-Adaptive Applications. In *The Twelfth IEEE International Symposium on High Performance Distributed Computing*, June.

[39] S. Senapathi, D. K. Panda, D. Stredney, and H.-W. Shen. A QoS Framework for Clusters to support Applications with Resource Adaptivity and Predictable Performance. In *the Proceedings of the IEEE International Workshop on Quality of Service (IWQoS)*, May.

[40] Hemal V. Shah, Dave B. Minturn, Annie Foong, Gary L. McAlpine, Rajesh S. Madukkarumukumana, and Greg J. Regnier. CSP: A Novel System Architecture for Scalable Internet and Communication Services. In *the Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, pages pages 61–72, San Francisco, CA, March 2001.

[41] Hemal V. Shah, Calton Pu, and Rajesh S. Madukkarumukumana. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In *the Proceedings of the CANPC workshop (held in conjunction with HPCA Conference)*, pages 91–107, 1999.

[42] Piyush Shivam, Pete Wyckoff, and Dhabaleswar K. Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. In *the Proceedings of the IEEE International Conference on Supercomputing*, pages 57–64, Denver, Colorado, November 10-16 2001.

[43] Piyush Shivam, Pete Wyckoff, and Dhabaleswar K. Panda. Can User-Level Protocols Take Advantage of Multi-CPU NICs? In *the Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, April 15-19 2002.

[44] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A user-level network interface for Parallel and Distributed Computing. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, December 1995.

[45] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *ICPP*, 2003.