Exploiting Remote Memory Operations to Design Efficient Reconfiguration for Shared Data-Centers over InfiniBand

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

Technical Report
OSU-CISRC-7/04-TR44

Exploiting Remote Memory Operations to Design Efficient Reconfiguration for Shared Data-Centers over InfiniBand*

P. Balaji K. Vaidyanathan S. Narravula K. Savitha H. -W. Jin D. K. Panda
Computer Science and Engineering,
The Ohio State University,
2015 Neil Avenue,
Columbus, OH-43210
{balaji, vaidyana, narravul, savitha, jinhy, panda}@cse.ohio-state.edu

Abstract

In this paper, we present 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. 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 our approach, we use the advanced features of InfiniBand such as Remote Direct Memory Access (RDMA) operations and network based atomic operations to tackle these challenges in an efficient manner without requiring any modifications to the existing data-center applications. Our experimental results show that the reconfigurability scheme provides a significantly higher performance (up to a factor of 2.5 improvement in the throughput) with the same resources or provides a similar performance while using fewer resources (up to half the nodes) as compared to a rigidly configured data-center. More importantly, our scheme takes advantage of the one-sided communication primitives offered by InfiniBand making it resilient and well-conditioned to the load on the servers as compared to two-sided communication protocols such as TCP/IP (sockets).

Keywords: *Shared Data-Centers, InfiniBand, Reconfigurability, Clusters, Remote Memory Operations*

1 Introduction

Cluster systems have become the main system architecture for a number of environments mainly due to their high performance-to-cost ratio. In the past, they had replaced mainstream supercomputers as a cost-effective alternative in a number of scientific domains. During the last few years, research and industry communities have been proposing and implementing several high performance communication systems to address some of the problems associated with the traditional networking protocols for cluster-based systems. InfiniBand Architecture (IBA) [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. It also provides features for performing network based atomic operations on the remote memory regions. These can be leveraged in providing efficient support for multiple environments [14, 22].

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 [20, 8, 3, 16].

Figure 1 represents a typical cluster-based data-center. The various nodes in the traditional data-center 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.

*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

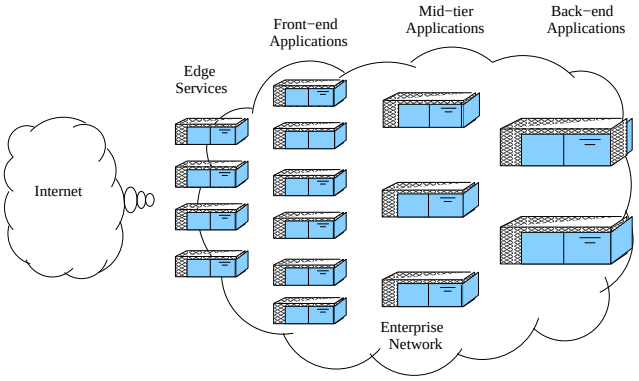


Figure 1. A Typical Cluster-Based Data-Center (courtesy CSP Architecture design [20])

In the past few years several researchers have proposed and configured data-centers providing multiple independent services, known as shared data-centers [7, 12]. 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. The increase in such services results in a growing fragmentation of the resources available and ultimately in the degradation of the performance provided by the data-center.

Over-provisioning of nodes in the data-center for each service provided is a widely used approach. In this approach, nodes are alloted to each service depending on the worst case estimates of the load expected and the nodes available in the data-center. For example, if a data-center hosts two web-sites, each web-site is provided with a fixed subset of nodes in the data-center based on the traffic expected for that web-site. It is easy to see that though this approach gives the best possible performance, it might incur severe under utilization of resources especially when the traffic is bursty and directed to a single web-site.

In this paper, we present 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 our approach, we use the advanced features of InfiniBand such as Remote Direct Memory Access (RDMA) operations and network based atomic operations to tackle these challenges in an efficient manner without requiring any modifications to the existing data-center applications.

This work has several research contributions. Primarily, we extend on our previous work [3, 16] in understanding the role of the InfiniBand architecture in next-generation data-centers. The other main contributions are:

1. We propose an architecture for dynamically reconfiguring the nodes present based on the load on the different services provided by the data-center. This architecture requires no changes to the legacy data-center applications.

2. As we will see in the next few sections, though a reconfigurability scheme can be implemented based on the standard sockets interface, the high computation load typical in the data-center environment can significantly hurt the performance achieved. With the focus on this observation, we design our approach to utilize the remote memory capabilities of the InfiniBand architecture to remotely monitor and manage the nodes in the data-center environment. Our results also show that this one-sided communication based architecture is mostly resilient and well-conditioned to the load on the servers as compared to two-sided protocols such as TCP/IP (sockets).

3. InfiniBand provides several opportunities to revise the design and implementation of many subsystems, protocols, and communication mechanisms in the data-center environment. The rich features of IBA offer a flexible design space and tremendous optimization potential.

Our experimental results show that the reconfigurability scheme provides a significantly higher performance (up to a factor of 2.5 improvement in the throughput) with the same resources or provides a similar performance while using fewer resources (up to half the nodes) as compared to a rigidly configured data-center.

The remaining part of the paper is organized as follows: Section 2 provides a brief background of the InfiniBand architecture and clustered data-center environments. In Section 3, we describe the design methodology and implementation details of the dynamic reconfigurability approach. We describe our experimental results in Section 4, present some previous related work in Section 5 and draw our conclusions and possible future work in Section 6.

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 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. 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.

2.1.1 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.

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

2.1.2 Atomic Operations Over IBA

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.

2.2 Shared Cluster-Based 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 2 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).

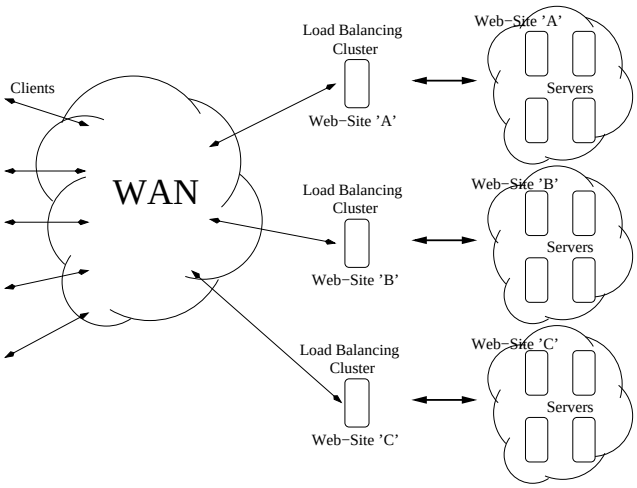


Figure 2. A Shared Cluster-Based Data-Center Environment

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) [20] to perform certain services such as intelligent load-balancing, caching, etc [12]. 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 Design of Reconfiguration Based on Remote Memory Operations

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

3.1 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 web-pages 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. 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. Dynamic reconfigurability of the system 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 web-site 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 requests.

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; modifying them to allow dynamic reconfigurability is impractical. To avoid making these cumbersome changes to the 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 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 nodes in 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. The servers on the other hand, just continue with the request processing, unmindful of the changes made by the modules.

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 data and the number of messages required), 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 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 messages 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 [16].

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.

The major design challenges and issues involved in dynamic adaptability and reconfigurability of the system ones are listed below.

- Providing a System Wide Shared State
- Concurrency Control to avoid Live-locks and Starvation
- Avoiding server thrashing through history aware reconfiguration
- Tuning the reconfigurability module sensitivity

We present these challenges in the following few subsections.

3.1.1 System Wide Shared State

As discussed earlier, 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 cannot 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 server nodes are not aware about when a particular load-balancer might require their state information.

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 web-site 'A' and the load-balancers in the shared data-center. Here, each node serving web-site 'A' provides its state information to each one of the load-balancing nodes every time they need it, i.e., the same information needs to be communicated with every node that needs it.

Based on these communication patterns, intuitively a global shared state seems to be the ideal environment for efficient distribution of data amongst all the nodes. In this architecture each node can write its relevant information into the shared state and the other nodes can asynchronously read this information without interrupting the source node. 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 server nodes by sending (replicated) information to each of the load-balancers 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 server node host intervention.

Further, as mentioned earlier and observed in our previous work [16], due to various factors such as the skew and the load on the server nodes, 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 writes information related to itself on its local memory. Other nodes needing this information can directly read this information using an *RDMA read* operation without disturbing this node at all. This implementation of a logical shared state retains the efficiencies of the initially proposed shared state architecture, i.e., each node can write data into its shared state and the other nodes can read data asynchronously from the shared state without interrupting the source node.

3.1.2 Shared State with Concurrency Control

The logical shared state described in Section 3.1.1 is a very simplistic view of the system. The nodes use the information available and change the system to the best possible configuration. However, for using this logical shared state, several issues need to be taken into consideration.

As shown in Figure 3, each load-balancer queries the load on each server at regular intervals. On detecting a high load on one of the servers, the load-balancer selects a lightly loaded node serving a different web-site, and configures it to ease the load on the loaded server. However, to avoid multiple simultaneous transitions and hot-spot effects during reconfiguration, additional logic is needed.

In our design, we propose an architecture using a two-level hierarchical locking with dual update counters to address these problems.

As shown in Figure 4, each web-site has an unique internal lock. This lock ensures that exactly one of the multiple load-balancers handling requests for the same web-site, can attempt a conversion of a server node, thus avoiding multiple simultaneous conversions. After acquiring this internal lock (through a remote atomic operation), the load-balancer selects a lightly loaded server and performs a second remote atomic operation to configure that server to serve the loaded web-site. This second atomic operation also acts as a mutually exclusive lock between load-balancers that are trying to configure the free server to serve their respective web-sites.

It is to be noted that after a reconfiguration is made, some amount of time is taken for the load to get balanced. However, during this period of time other load balancers can still detect a high load on the servers and can possibly attempt to reconfigure more free nodes. To avoid this unnecessary reconfiguration of multiple nodes, each relevant load-balancer needs to be notified about any recent reconfiguration done, so that it can wait for some amount of time before it checks the system load and attempts a reconfiguration. In our design, each load-balancer keeps a *local update counter* and a *shared update counter* to keep track of all reconfigura-

tions. Before making a reconfiguration, a check is made to see if the *local update counter* and the *shared update counter* are equal. In case they are equal, a reconfiguration is made and the *shared update counters* of all the other relevant load-balancers is incremented (using atomic fetch-and-add). Otherwise, if the *shared update counter* is more than the *local update counter*, it indicates a very recent reconfiguration, so no reconfiguration is made at this instance by this load-balancer. However, the *local update counter* is updated to the *shared update counter*. This ensures that each high load event is handled by only one load-balancer.

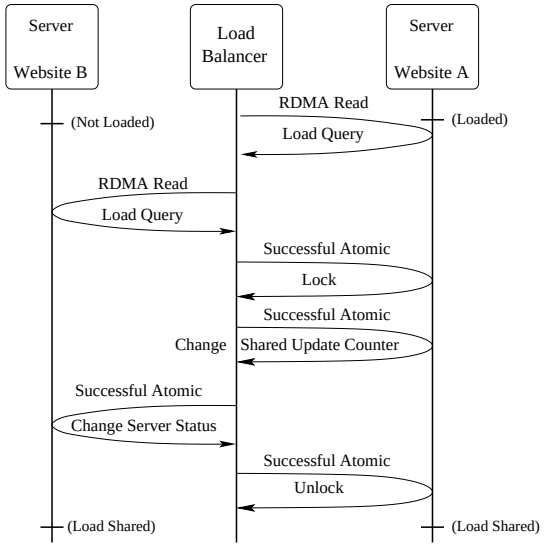


Figure 3. Concurrency Control for Shared State

3.1.3 History Aware Reconfiguration

Due to the irregular nature of the incoming requests, a small burst of similar requests might potentially trigger a re-configuration in the data-center. Because of this, small bursts of similar requests can cause nodes in the shared data-center to be moved between the various co-hosted web-sites 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 serving one web-site are re-allocated to a different web-site only if the load to the second web-site stays high for a pre-defined period of time *T*. However, 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

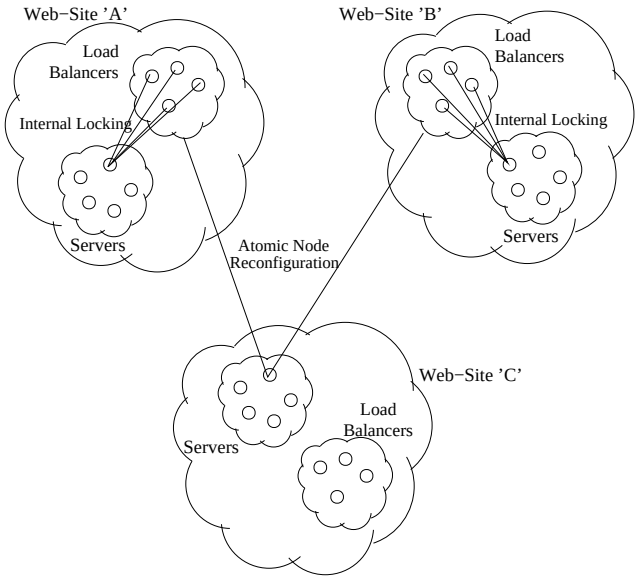


Figure 4. Hot-Spot Avoidance with Hierarchical Locking

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.

3.1.4 Reconfigurability Module Sensitivity

As mentioned earlier, the modules on the load-balancers 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 external helper 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.

4 Experimental Results

In this section, we present various performance results. First, in Section 4.1, 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 4.2, we present the impact

of the load conditions in the data-center environment on the performance achievable by VAPI and IPoIB. Finally, in Section 4.3 we present the performance of our VAPI based reconfiguration scheme 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.

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.

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-x86-3.1-build-003. The adapter firmware version is fw-23108-rel-3_00_0001-rc4-build-001. The IPoIB driver for the InfiniBand adapters was provided by Voltaire Incorporation [8]. The version of the driver used was 2.0.5_10.

Cluster 1 was used to represent the software load-balancers and Cluster 2 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. Requests from the software load-balancers were generated using sixteen threads on each load-balancer.

We have considered two shared data-center scenarios: (i) a data-center hosting three web-sites with nodes alloted in the ratio 3:3:2, (ii) a data-center hosting four web-sites with nodes alloted in the ratio 2:2:2:2.

4.1 Basic Microbenchmarks

The ideal case performance achievable by the Verbs API (VAPI) over InfiniBand and IPoIB using micro-benchmark tests are presented in this section. 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.

InfiniBand provides two mechanisms for completion notification. The first approach is polling-based which 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 is notification-based which allows the host application to request an 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 on Cluster 2.

The latency achieved by the VAPI RDMA Read communication model and IPoIB (round-trip latency) for various message sizes is shown in Figure 5a. RDMA Read, using the polling based approach, achieves a latency of 11.89μs for 1 byte messages compared to the 53.8μs achieved by IPoIB. The event based approach, however, achieves a latency of 23.97μs. Further, with increasing message sizes, the difference between the latency achieved by VAPI and IPoIB tends to increase. The figure also shows the CPU utilized by RDMA Read (notification based) and IPoIB. We can see that RDMA utilizes negligible CPU on the receiver side, i.e., with RDMA, the initiator can read or write data from the remote node without requiring any interaction with the remote host CPU.

Figure 5b shows the uni-directional bandwidth achieved by RDMA Read and IPoIB. RDMA Read (with polling as well as with notification) is able to achieve a peak bandwidth of 839.1 MBps as compared to a 231 MBps achieved by IPoIB. Again, the CPU utilization for RDMA is negligible on the receiver side.

The results for the other communication models (send/receive, RDMA write and RDMA write with Immediate data) are shown in Figures 6 through 8.

4.2 One-sided vs Two-sided Communication

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 2.

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 9. We observe that the performance of IPoIB degrades significantly with the

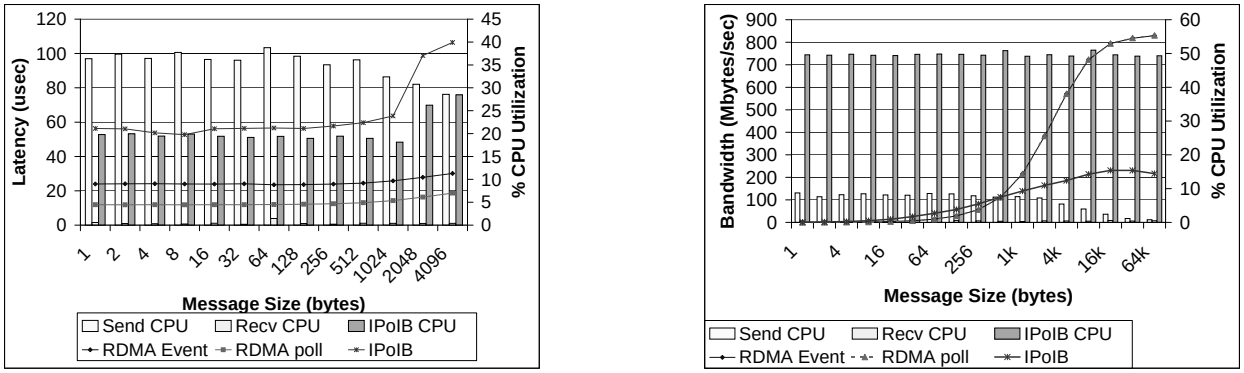


Figure 5. Micro-Benchmarks for RDMA Read and IPoIB: (a) Latency and (b) Bandwidth

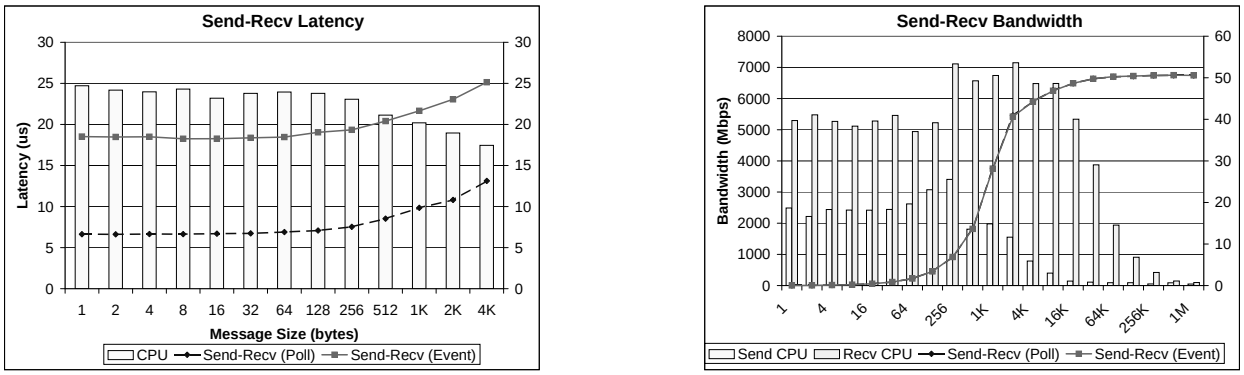


Figure 6. Micro-Benchmarks for the Send-Recv communication model: (a) Latency and (b) Bandwidth

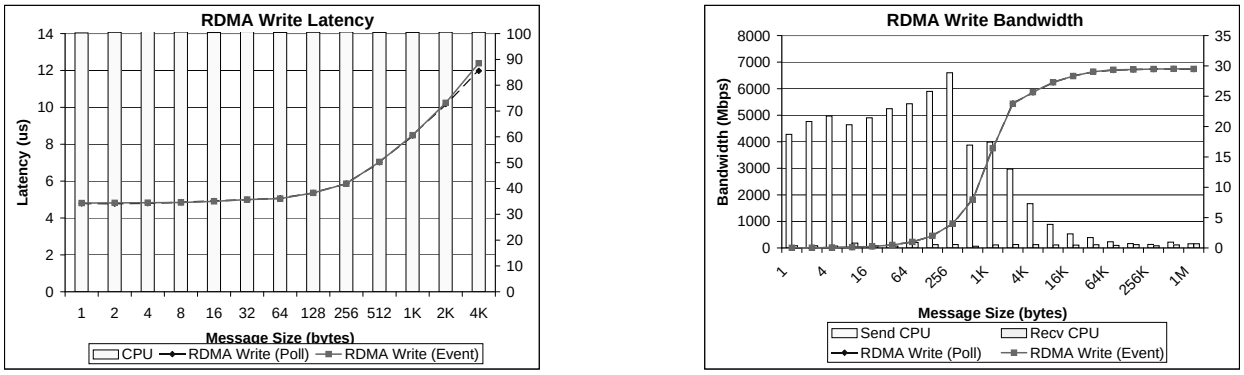


Figure 7. Micro-Benchmarks for the RDMA Write communication model: (a) Latency and (b) Bandwidth

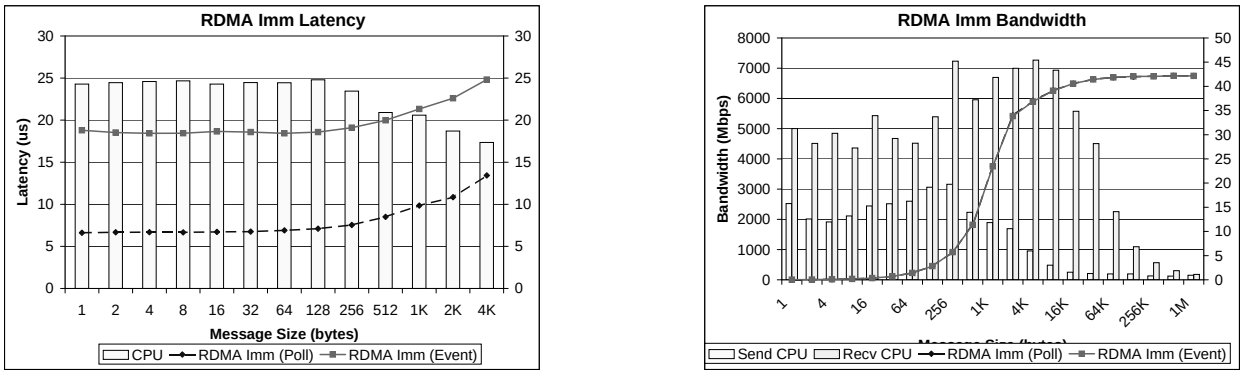


Figure 8. Micro-Benchmarks for the RDMA Write with Immediate Data communication model: (a) Latency and (b) Bandwidth

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.

4.3 Performance of Reconfigurability

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

4.3.1 Performance with Burst Length

In this section, we present the performance of the dynamic reconfigurability scheme as compared to the rigid configuration and the over-provisioning schemes in two scenarios with (i) the data-center hosting three web-sites (nodes alloted in the ratio 3:3:2) and (ii) the data-center hosting four web-sites (nodes alloted in the ratio 2:2:2:2). For the workload, we have used a single file trace with a file size of 1 KB.

Figure 10a shows the performance of the dynamic reconfigurability scheme as compared to the rigid configuration and over-provisioning schemes for varying burst length in the traffic for the first scenario (hosting three web-sites).

In a rigid configuration if there is a burst of traffic for the first website only three nodes are used and the remaining 5 nodes are relatively idle. The rigid configuration achieves an aggregate throughput of about 26,000 Transactions Per Second (TPS) in this scenario. In the over-provisioning scheme, a maximum of 6 nodes are assigned to the web-site being loaded. The maximum throughput achievable in this best configuration is around 51,000 TPS. However, in the reconfiguration scheme we see that the performance depends mainly on the burstiness of the traffic. If the burst length is too short, reconfiguration seems to perform comparably with the rigid scheme but for huge bursts reconfiguration achieves performance close to that of the over-provisioning scheme. The performance of reconfiguration for low burst lengths is comparable with the rigid scheme mainly due to the switching time overhead, i.e., the time required for the nodes to be reconfigured to the optimal configuration. This switching time, however, can be tuned by varying the sensitivity value addressed in Section 3.1.4. For high bursts, the reconfiguration switching time is negligible and gets amortized.

Figure 10b shows a similar trend for the second scenario (hosting four web-sites); reconfigurability scheme achieves up to a factor of 2.5 improvement in the performance while using the same resources as the rigid scheme. Also, for high burstiness, it achieves a comparable performance with that of the over-provisioning scheme while utilizing only half the nodes.

Results for traces with a single file of size 4 KB and 16 KB are shown in Figures 11 and 12. We also present the results for a ZipF [23] based trace in Figure 13 and for a real World-Cup trace [1] in Figure 14. As we can see, the trend is similar to the single file traces.

4.3.2 Node Utilization

In case of shared data-centers, the logically partitioned sets of nodes serve the individual web-sites. Due to this partitioning and possible unbalanced request load, the nodes serving a particular web-site might be over-loaded even when other nodes in the system are not being utilized. These un-utilized servers could typically be used to share the load on the loaded web-site to yield better overall data-center performance.

In this section we measure the effective node utilization of our appraoch and compare it with the rigid and the over-provisioned cases. The effective node utilization is measured as the total number of nodes being fully used by the

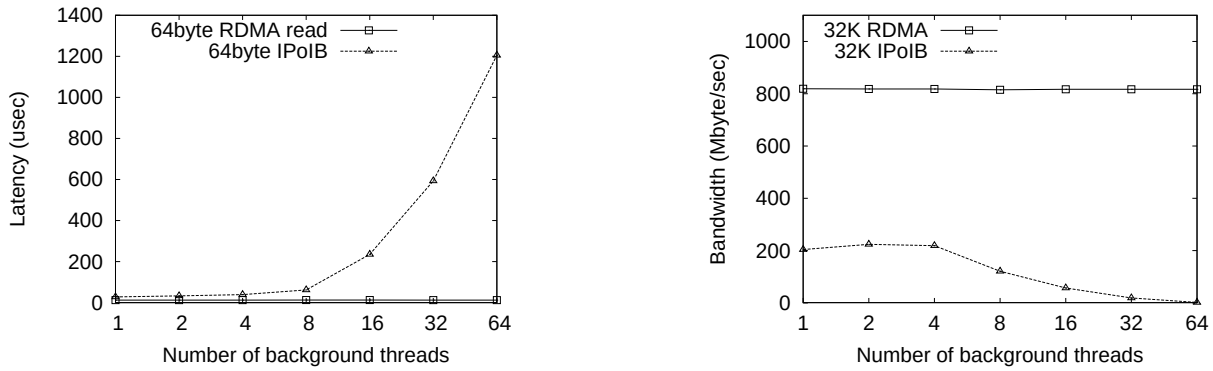


Figure 9. Performance of IPoIB and RDMA Read with background threads: (a) Latency and (b) Bandwidth

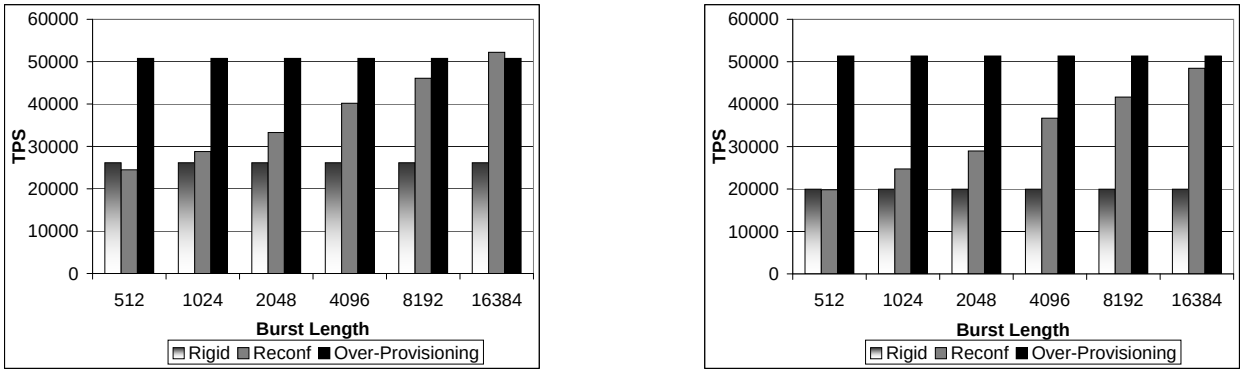


Figure 10. Impact of Burst Length (a) Number of co-hosted web-sites = 3 (b) Number of co-hosted web-sites = 4

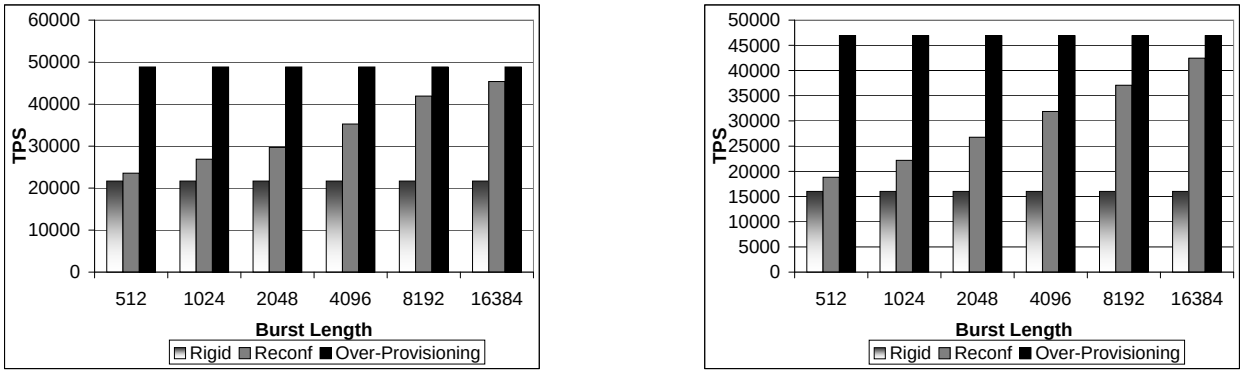
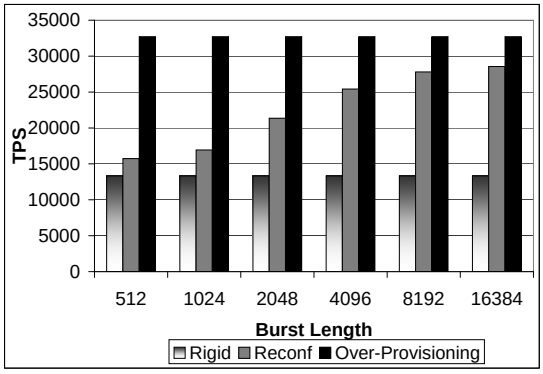


Figure 11. Impact of Burst Length with a request file size 4k(a) Number of co-hosted web-sites = 3 (b) Number of co-hosted web-sites = 4



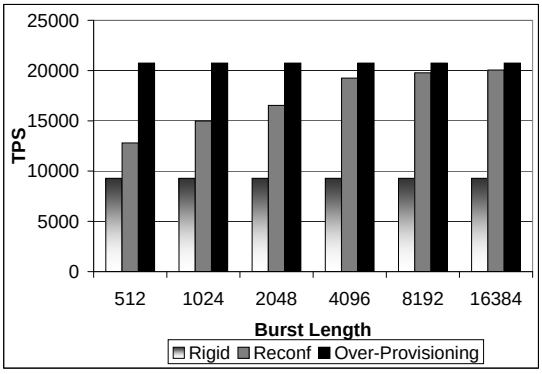
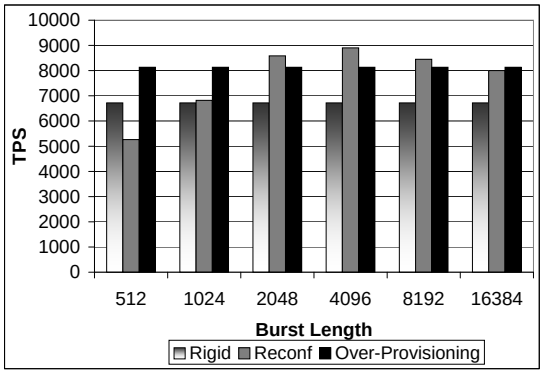


Figure 12. Impact of Burst Length with a request file size 16k (a) Number of co-hosted web-sites = 3 (b) Number of co-hosted web-sites = 4



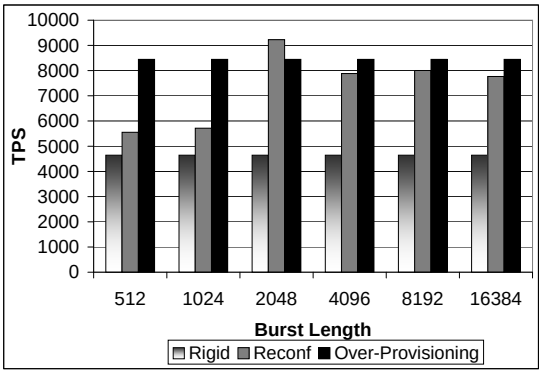
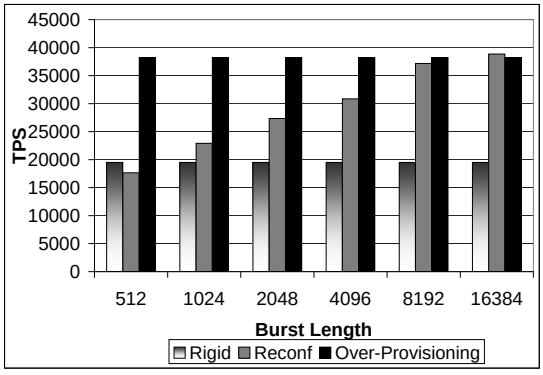


Figure 13. Impact of Burst Length with a Zipf trace (a) Number of co-hosted web-sites = 3 (b) Number of co-hosted web-sites = 4



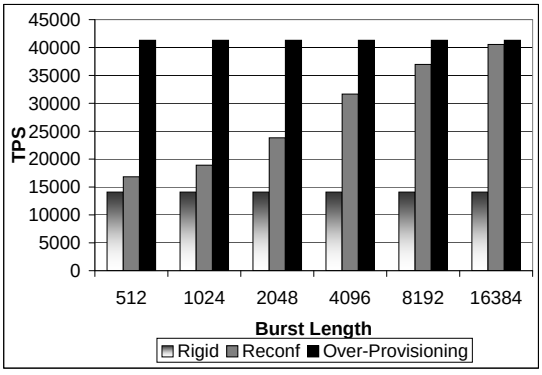


Figure 14. Impact of Burst Length with a worldcup trace (a) Number of co-hosted web-sites = 3 (b) Number of co-hosted web-sites = 4

data-center to serve a particular web-site.

Figure 15 shows the node utilization for a shared data-center having three co-hosted web-sites (single file trace with a file size of 1 KB). The rigid case has a constant node utilization of three nodes since it is statically configured. The over-provisioned case can use a maximum of 6 nodes (leaving 1 node for each of the other web-sites).

In figure 15a, we can see that for non-bursty traffic (burst length = 512), the reconfiguration scheme is not able to completely utilize the maximum available nodes because the switching time between configurations is comparable to the time required to serve the burst length of requests. It is to be noted that it performs comparably with the rigid scheme.

Further, nodes switching to one of the web-sites incurs a penalty for the requests coming to the other web-sites. For example, a burst of requests for web-site 'A' might cause all the nodes to shift accordingly. So, at the end of this burst web-site 'A' would have six nodes while the other 2 web-sites have one node each. At this time, a burst of requests to any of the other web-sites would result in a node utilization of one. This causes a drop in the number of nodes used for reconfigurability as shown in the figure.

Figure 15b shows the node utilization with a burst length of 8096. We can see that for large burst lengths the switching time for our reconfigurability scheme is negligible. And for large periods of time, the maximum number of nodes are fully utilized.

Figure 16 shows similar trends for the scenario with a shared data-center hosting four web-sites. It is to be noted that in this scenario the rigid scheme has only two nodes for each web-site while the over-provisioned scheme has 5 nodes. The reconfigurability scheme starts with the rigid scheme's configuration and reconfigures nodes as required.

Similar results for single file traces with 4 KB and 16 KB file sizes are presented in Figures 17 through 20. We also present results for a ZipF based trace in Figures 21 and 22 and a real world-cup trace in Figures 23 and 24. We can see that all these traces show similar trends as the one described above.

5 Related Work

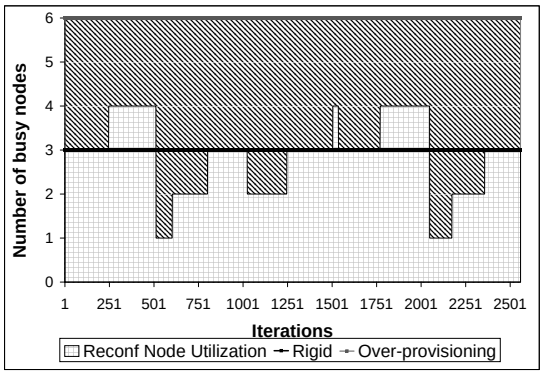
Several researchers have focussed on the design of adaptive systems that can react to changing workloads in the context of web servers [15, 9, 19, 6, 18]. There has been some previous research which focus on dynamism in the data-center environment by HP labs and IBM Research [13, 17]. These are notable in the sense that they were the first to show the capabilities of a dynamic allocation of system resources in the data-center environment. However, some of the solutions focus on lower level architectural requirements mainly for storage related issues, rely on specific hardware and are hard to look at as commodity component based solutions; some others require kernel-level modifications. On the other hand, in our approach, we try to propose a solution that is not geared toward any specific hardware and try to give a generic solution at the application level without requiring any changes to existing applications. Further, some of these approaches rely on the servers to intelligently reconfigure the other nodes. While these approaches are quite intuitive, in a real data-center scenario, the high server loads can make them inefficient and potentially unusable. Our approach of placing the onus of 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.

Shah, Kim, Balaji, et. al., have done significant research in User Level High Performance Sockets implementations [21, 10, 11, 5, 4, 3]. In one of our previous works [3], we had evaluated the capabilities of such a pseudo-sockets layer over InfiniBand in the data-center environment. However, as we had observed in [16], 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. 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.

6 Concluding Remarks and Future Work

In a typical cluster-based data-center, various nodes are logically partitioned to provide various related services including web and messaging services, transaction processing, business logic, databases, etc. In the past few years several researchers have proposed and configured data-centers providing multiple independent services, known as shared data-centers. The increase in such services results in a growing fragmentation of the resources available and ultimately in the degradation of the performance provided by the data-center.

In this paper, 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. 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 our approach, we use the advanced features of InfiniBand such



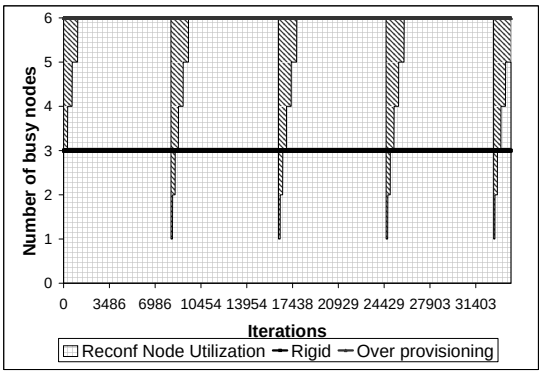
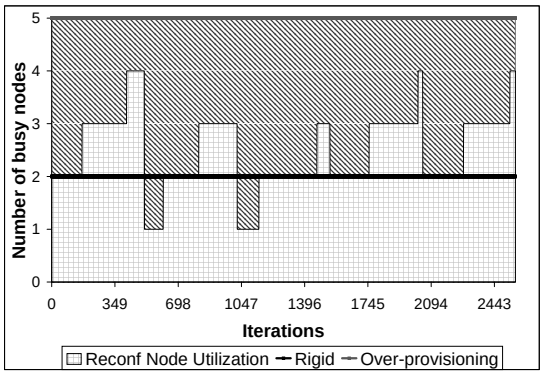


Figure 15. Node Utilization in a data-center requesting a file size of 1k hosting 3 web-sites with burst length (a) 512 (b) 8k



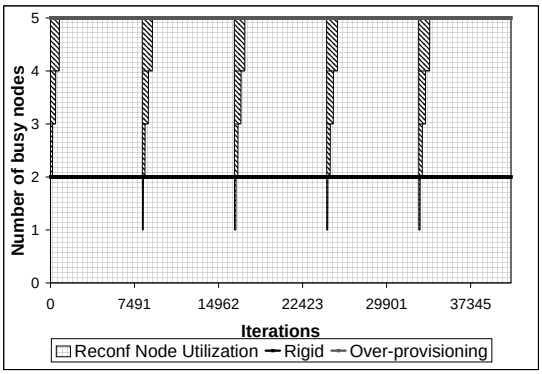
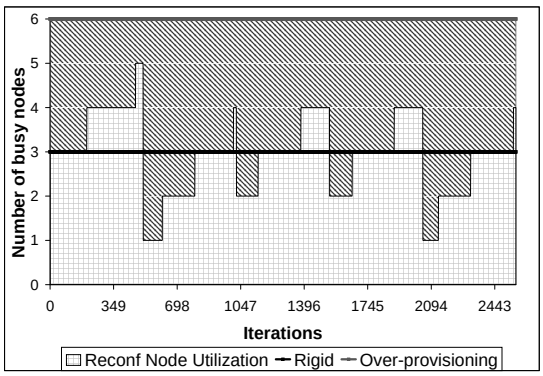


Figure 16. Node Utilization in a data-center requesting a file size of 1k hosting 4 web-sites with burst length (a) 512 (b) 8k



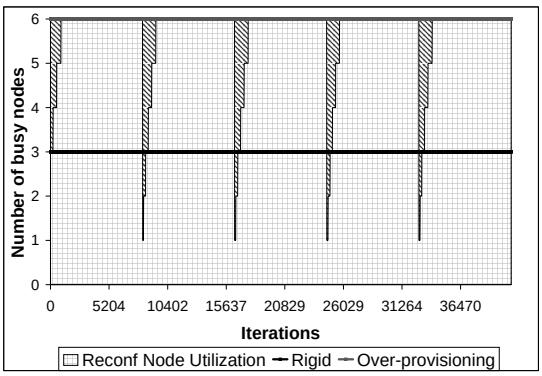
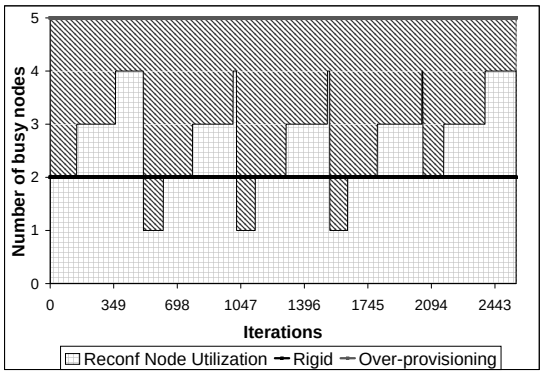


Figure 17. Node Utilization in a data-center requesting a file size of 4k hosting 3 web-sites with burst length (a) 512 (b) 8k



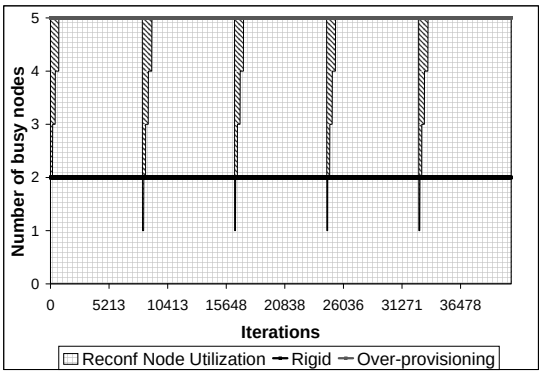
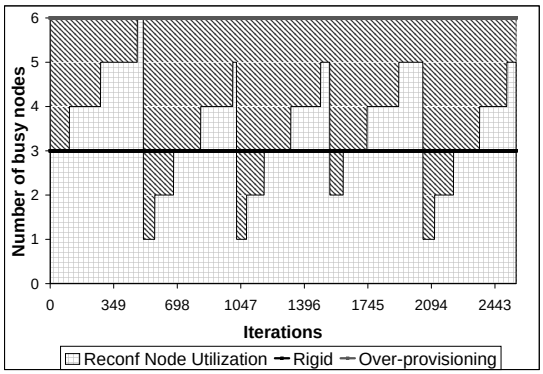


Figure 18. Node Utilization in a data-center requesting a file size of 4k hosting 4 web-sites with burst length (a) 512 (b) 8k



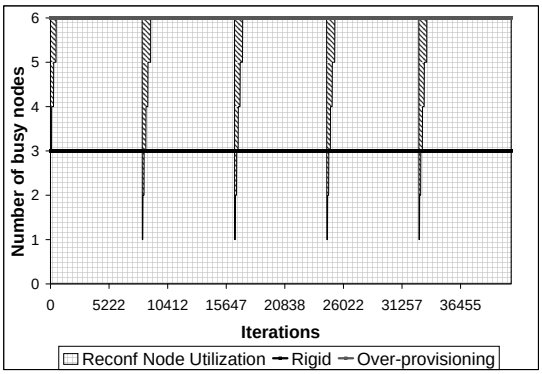
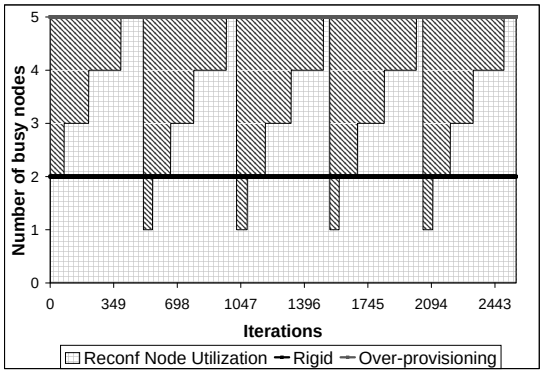


Figure 19. Node Utilization in a data-center requesting a file size of 16k hosting 3 web-sites with burst length (a) 512 (b) 8k



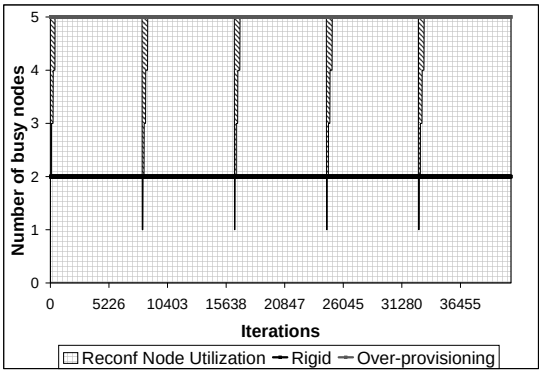
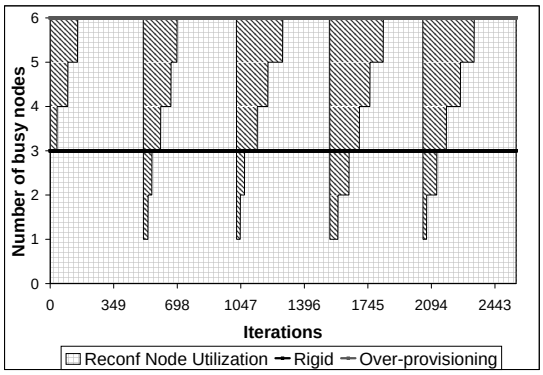


Figure 20. Node Utilization in a data-center requesting a file size of 16k hosting 4 web-sites with burst length (a) 512 (b) 8k



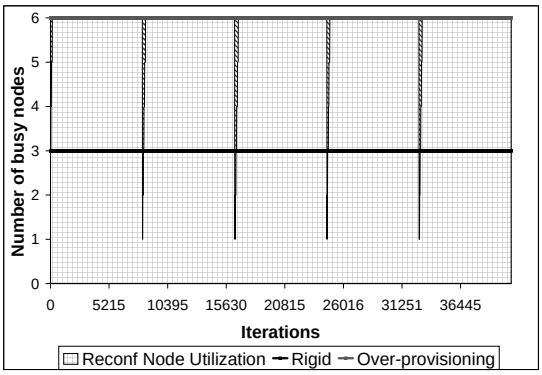
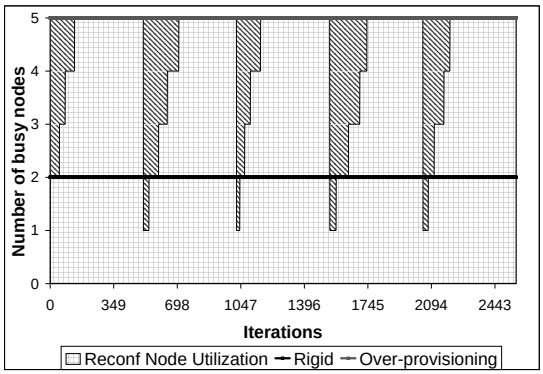


Figure 21. Node Utilization in a data-center for a Zipf trace hosting 3 web-sites with burst length (a) 512 (b) 8k



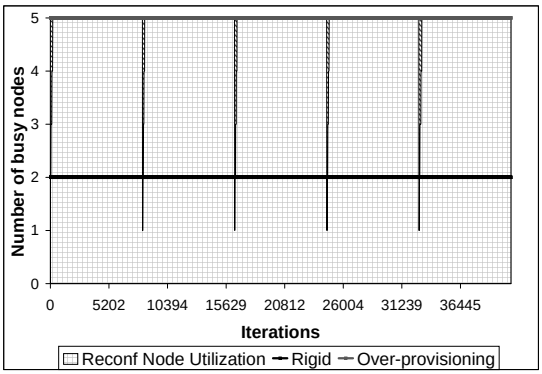
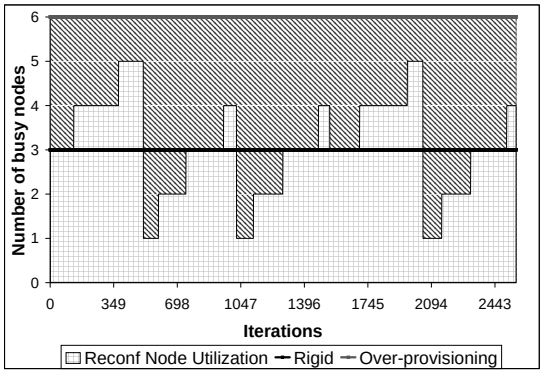


Figure 22. Node Utilization in a data-center for a Zipf trace hosting 4 web-sites with burst length (a) 512 (b) 8k



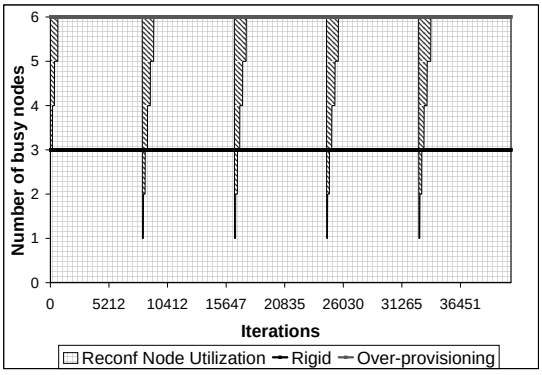


Figure 23. Node Utilization in a data-center for a worldcup hosting 3 web-sites with burst length (a) 512 (b) 8k

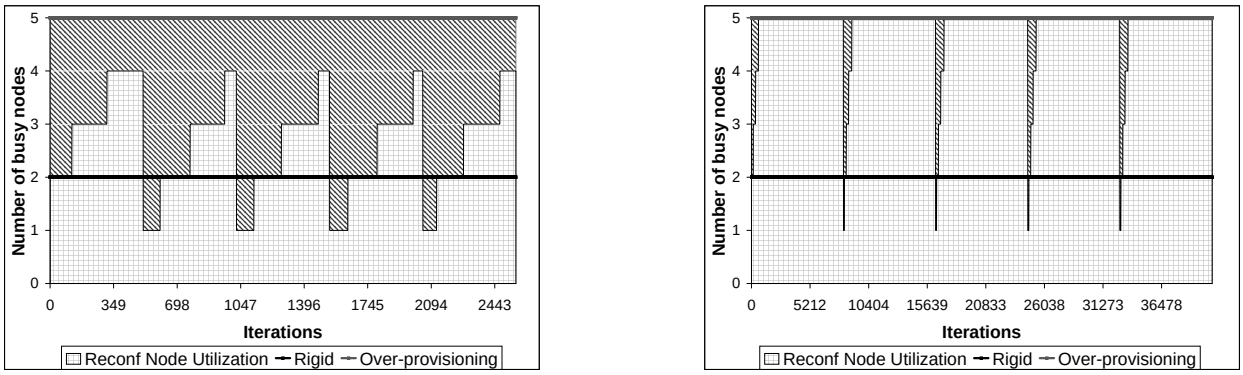


Figure 24. Node Utilization in a data-center for a worldcup trace hosting 4 web-sites with burst length (a) 512 (b) 8k

as Remote Direct Memory Access (RDMA) operations and network based atomic operations to tackle these challenges in an efficient manner without requiring any modifications to the existing data-center applications. Our experimental results show that the reconfigurability scheme provides a significantly higher performance (up to a factor of 2.5 improvement in the throughput) with the same resources or provides a similar performance while using fewer resources (up to half the nodes) as compared to a rigidly configured data-center. More importantly, our scheme takes advantage of the one-sided communication primitives offered by InfiniBand making it resilient and well-conditioned to the load on the servers as compared to two-sided communication protocols such as TCP/IP (sockets).

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. We are also looking at utilizing the reconfigurability scheme for more specific services provided by the data-center such as Quality of Service guarantees for each web-site hosted, etc.

7 Acknowledgments

We would like to thank Jiesheng Wu and other members of the Network Based Computing laboratory at the Ohio State University for all the comments and suggestions they provided during the course of the project.

References

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

[2] Infiniband Trade Association. http://www.infinibandta.org.

[3] 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 *the Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, Texas, March 10-12 2004.

[4] P. Balaji, J. Wu, T. Kurc, U. Catalyurek, D. K. Panda, and J. Saltz. Impact of High Performance Sockets on Data Intensive Applications. In *the Proceedings of the IEEE International Conference on High Performance Distributed Computing (HPDC 2003)*, June 2003.

[5] 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.

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

[7] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. In *Proceedings of ACM Sigmetrics 2003, San Diego, CA*, June 2003.

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

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

[10] 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.

[11] 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.

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

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

[14] 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.

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

[16] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting Strong Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. In *SAN*, 2004.

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

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

[19] 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.

[20] 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*, 2001.

[21] 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.

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

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