Dynamic Re-Configurability Support To Provide Soft QoS Guarantees in Cluster-Based Multi-Tier Data-Centers over InfiniBand

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Master of Science in the

Graduate School of The Ohio State University

By

Savitha Krishnamoorthy, B.E.(Hons)

* * * * *

The Ohio State University

2004

Master's Examination Committee:

Prof. Dhabaleswar K. Panda, Adviser

Prof. P. Sadayappan

Approved by

Adviser

Computer and Information Science Graduate Program

ABSTRACT

Current cluster-based data-centers are configured as multiple tiers, each with specific functionalities and containing multiple nodes. Requests to the data-center have different resource requirements and are satisfied by different tiers of the data-center. Due to the unpredictability of the pattern of such requests, over-provisioning of nodes in these data-center tiers is an accepted approach to provide Quality of Service (QoS) guarantees. However, this approach is not cost-effective since it incurs a potential wastage of resources. 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 thesis we study the current approaches to configure the data-center and capabilities of these approaches to satisfying QoS guarantees. We present a novel architecture to enable dynamic reconfiguration of nodes in a cluster-based multi-tier 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.

In the case of a data-center hosting multiple websites, it is important to provide differentiated service to these websites depending on their individual investments.

In such scenarios, reconfigurability improves the performance and QoS guarantees of higher priority websites with lesser number of nodes. Also when the number of high-priority requests is low, the idle nodes can be re-allocated to serve lower priority requests and hence improve the over-all throughput achieved by the data-center. In a rigidly configured data-center lower priority requests suffer even though the nodes allocated to the high-priority requests may be idle.

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 (i) Single file traces, (ii) Zipf based traces and (iii) Real life world cup trace. Our experimental results show that the dynamic reconfigurability scheme can be used to provide better QoS guarantees (up to 25% better), meet the same guarantees with lesser resources (up to 20% lesser nodes), or even both in some cases.

I dedicate this Appa, Amma, Vidya and Priya

ACKNOWLEDGMENTS

I would like to express my heartfelt gratitude to my adviser Dr. Dhabaleswar K. Panda for his invaluable guidance and motivation. His constant support has gone a long way in the completion of this work.

I am grateful to Dr. P Sadayappan for consenting to serve on my Master's examination committee.

My thanks are due to Pavan Balaji for all his help and constant motivation that was imperative for the successful completion of this project.

I would like to thank Neil, Karthik, Naga and Reshma for their immense help and support.

I am extremely thankful to Bala and Sayantan for their valuable assistance and encouragement.

For generously sharing their thoughts, ideas and comments, I would like to thank Jiesheng, Sundeep, Jin and also my lab mates Adam, Vishnu, Mamidala and Gopal

Special thanks to Tom, Ewana, Elizabeth, Tamera, Catrena and Marty of the CIS department for making my stay at OSU a pleasant one.

I feel greatly indebted to my family for their support, love and affection.

VITA

March 14, 1981	Born - Vellore, India
1998 - 2002	Bachelor of Engineering (Hons) Computer Science
Jan 2002 - June 2002	Intern, DSP Design Texas Instruments (I) Ltd., Bangalore
Sep 2002 - Sep 2003	Distinguished University Fellow The Ohio State University
Sep 2003 - Present	Graduate Research Associate, The Ohio State University.

PUBLICATIONS

S. Krishnamoorthy, P. Balaji, K. Vaidyanathan, H. -W. Jin, D. K. Panda "Dynamic Reconfigurability Support for providing Soft QoS Guarantees in Multi-Tier Data-Centers over InfiniBand". *Super Computing 2004* , April 2004.

P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J.Wu and D. K. Panda "Sockets Direct Protocol over InfiniBand in Clusters: Is it Beneficial?" *IEEE International Symposium on Performance Analysis of Systems and Software* , March 2004.

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" *IEEE International Workshop on System Area Networks* , March 2004.

Savitha Krishnamoorthy, Karthikeyan Vaidyanathan and Mario Lauria "Exploring Advantages of a Simple, but Flexible P2P Lookup Protocol" *International Workshop on Agents and Peer-to-Peer Computing* , July 2004.

Sudha Srinivasan, Savitha Krishnamoorthy and P.Sadayappan "A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs" *IEEE International Conference on Cluster Computing* , December 2003.

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in High Performance Computing: Prof. D.K. Panda

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The Internet technology has grown at a very fast pace and gained enormous popularity. Not only does it serve as one of the primary modes of communication, but also is an effective means of file-sharing and resource-sharing. Today, the Internet has become a primary tool in small and large business transactions. These businesses need to reach out and attract customers through the electronic medium. This has led to the evolution of data-centers or server farms that host web services for these businesses.

The explosive growth of the E-commerce sector has placed higher demands on the performance of data-centers. High availability, scalability and capability to provide QoS guarantees are essential requirements imposed on next generation data-centers. There is a need to decouple resource requirements of requests in the data-center leading to the deployment of multi-tier data-centers.

Online businesses today offer personalized services, in order to increase probability of a financial transaction. Differentiation of service is also a critical component of online business. For instance, some businesses like Amazon and Hallmark, give higher priority to frequent customers. A data-center hosting such services needs to cater to

this requirement of providing differentiated service that may be based on the request, the client making the request, or the business that needs to service the request.

On the other hand, the high-performance community has seen a rapid growth in commodity clusters which is a cost-effective alternative to super-computers. The growth of network-based computing has been made possible by the evolution of high-performance networks, which offer low latency and high bandwidth. Data-centers are now taking advantage of this approach and leaning toward such high-performance clusters for their deployment. These networks rely on user-level networking and one-sided communications for their performance. In our design, we exploit these advanced features in cluster-based data-centers to provide better QoS guarantees and differentiated service.

In this chapter we provide a brief overview of the networking interconnects, protocols and messaging middleware. We also introduce Data-Centers over InfiniBand and present an overview of Multi-Tier Data-Center Architecture.

1.1 Commodity Clusters and Networks

In the past few decades, the high performance computing community has seen a shift from mainframe and host-centric computing to a parallel and network-centric computing approach. Network interconnects that offer very low latency and high bandwidth have continuously emerged to complement the increasing computational power of commodity PCs. This trend has enabled the deployment of high-end production scientific computing environments, built using clusters of commodity PCs and

high speed interconnects. The high performance-to-price ratio of these commodity-off-the-shelf (COTS) clusters has been the most important factor that has accelerated this trend.

1.1.1 Interconnect Technologies And Protocols

Some of the leading products in the network interconnects market include Myrinet [13], Ethernet [20, 1, 23, 19], Quadrics [38, 37, 39, 4, 34] and InfiniBand [6, 2].

Some of these interconnects provide very low latencies (even less than 10 μs) and very high bandwidth (of the order of 10 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 provide hardware-based support for Quality of Service (QoS) and for multicast operations. Layers above the hardware can then take advantage of these advanced features provided at the hardware level to get improved performance and use novel approaches to solving existing problems. 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]. InfiniBand Architecture (IBA) [6, 2] has been recently standardized by the industry to design next generation high-end clusters.

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. The advent of high-performance interconnects further propelled the efforts to develop user-level protocols to take advantage of some of the advanced features offered by them. User-level protocols such as U-Net [46], BIP [35, 36], FM [33], GM [18], EMP [44, 45] Virtual Interface Architecture (VIA) [15, 3, 11, 17, 10] were some of the first to be proposed and deployed.

In the following chapters we show how high-performance interconnects like IBA together with a native implementation of user-level protocols can be used effectively in the design and implementation of next generation data-centers.

1.1.2 IBA: Overview

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. This mainly reduces the system processing overhead by decreasing the number of copies associated with a message transfer, thereby removing the kernel from the critical message passing path. This is achieved by providing the consumer applications direct and protected access to the HCA.

One-sided communication allows the network interface to transfer data between local and remote memory buffers directly and safely without any host 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 [29, 47].

1.2 Multi-Tier Data-Center Architecture

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 [7, 30].

Figure 1.1 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 [42], 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.

1.2.1 QoS Guarantees in Data-Centers

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

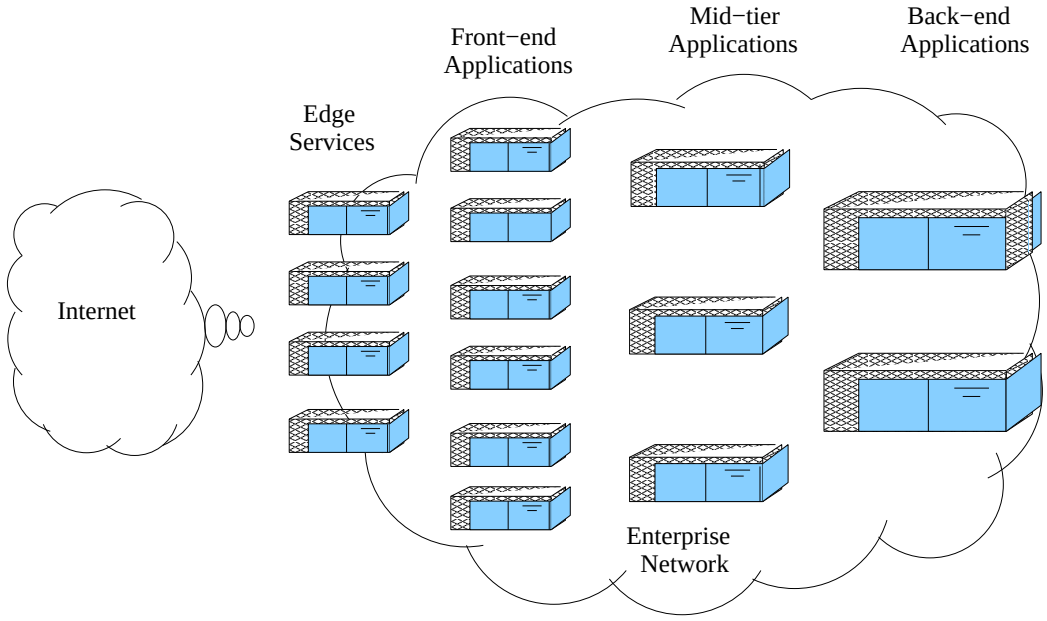

Figure 1.1: A Typical Multi-Tier Data-Center (Courtesy CSP Architecture design [42])

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 better performance to all financial transactions such as web requests involving purchasing goods 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.

1.3 Problem Statement

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, interesting documents such as a recent flash news or dynamic web-pages become available and unavailable. This results in bursty traffic for certain kinds of documents at some times and for some other kinds of documents at a different time. Such scenarios make it difficult to determine and predict request patterns. In this scenario, a burst of one kind of data requests would result in one tier in the data-center environment being over-loaded leaving the nodes in the other tiers idle. Such dynamism in the incoming requests is especially of concern in data-centers which host multiple web-sites or services.

Over-provisioning of nodes in the data-center tiers is an accepted and popular approach to provide Quality of Service (QoS) guarantees due to the unpredictability of resource requirements for an incoming requests. In this approach, nodes are allocated to the multiple tiers in the data-center environment based on the worst case requirements to satisfy 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.

1.4 Our Approach

In this thesis 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.

1.4.1 Introduction to Dynamic Reconfigurability

As mentioned in section 1.3 over-provisioning is a popular solution but suffers from the disadvantages of under-utilization and high costs. On the other hand, dynamic reconfigurability utilizes the existing resources to satisfy guaranteed resources by reconfiguring the data-center to adapt to the current request requirements. Currently, without reconfigurability we can only make QoS guarantees based on the maximum achievable throughput of a single tier considering the worst case when all requests to

the data-center requires to be served by that tier. When the requests are directed to different tiers, keeping all nodes in the data-center busy, we can achieve much higher throughput than that determined by the worst-case guarantee. Hence, the basic idea is to improve utilization of resources in the data-center even during bursts of requests with similar resource requirements. In this scenario a reconfigurable data-center re-allocates some of its resources into the over-loaded tiers, thus improving the overall performance and achievable throughput in the data-center.

This design reduces the cost of a data-center setup by doing away with over-provisioning and allows for improved QoS guarantees provided by the data-center by sustaining performance through all patterns of incoming requests.

1.4.2 Dynamic Reconfigurability for Multiple Websites

A common application of data-centers is to host multiple websites. Depending on the investment made by a client, the data-center is required to provide differentiated service to it. In order to keep the QoS guarantees made to the high priority clients, the rigid data-center may allocate a large fraction of its resources to them. However, this results in under-utilization where in the lower priority clients may suffer even though resources allocated to the high-priority clients may be idle.

We apply reconfigurability to this scenario and show that it benefits low-priority clients while keeping QoS guarantees to the high-priority clients with relatively lesser number of nodes as compared to the rigid case.

The rest of this thesis is organized as follows. Chapter 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. Chapter 2 also describes previous work related to this thesis. In Chapter 3, we describe the design and implementation details of the dynamic reconfigurability approach. In Chapter 4, we discuss more details into the issues related in providing Soft QoS guarantees to websites and the existing approaches used in this domain. We describe our reconfigurability scheme and present the basic performance comparison with respect to the rigid data-center and over-provisioning in Chapter 5. In Chapter 6 we present the performance of the scheme extended to multiple websites. Chapter 7 presents our concluding remarks and possible extensions to this work.

CHAPTER 2

BACKGROUND AND MOTIVATION

In this chapter, 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. In section 2.3, we provide a brief overview of previous research in this direction and the motivation behind this work.

2.1 Data-Center Tiered Architecture

Figure 2.1 shows the architecture of a typical multi-tier data-center. The clients interact with the data-center from the Wide Area Network or the Internet. The data-center itself is composed of tiers which logically partition its resources based on their functionality.

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

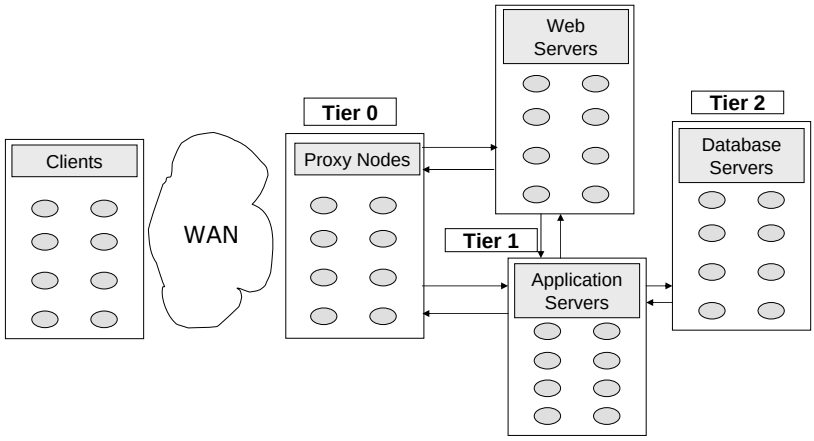

Figure 2.1: Data-Center Tiered Architecture

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 [42] specifies an additional edge service tier which handles security, caching, SAN enclosure of packets for TCP termination and several others. In this thesis, we only deal with the traditional 3-tier data-center architecture.

Since the kind of requests coming into 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 (HCA), which may simply be referred to as Channel Adapters (CA). 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 2.2 illustrates the InfiniBand Architecture model.

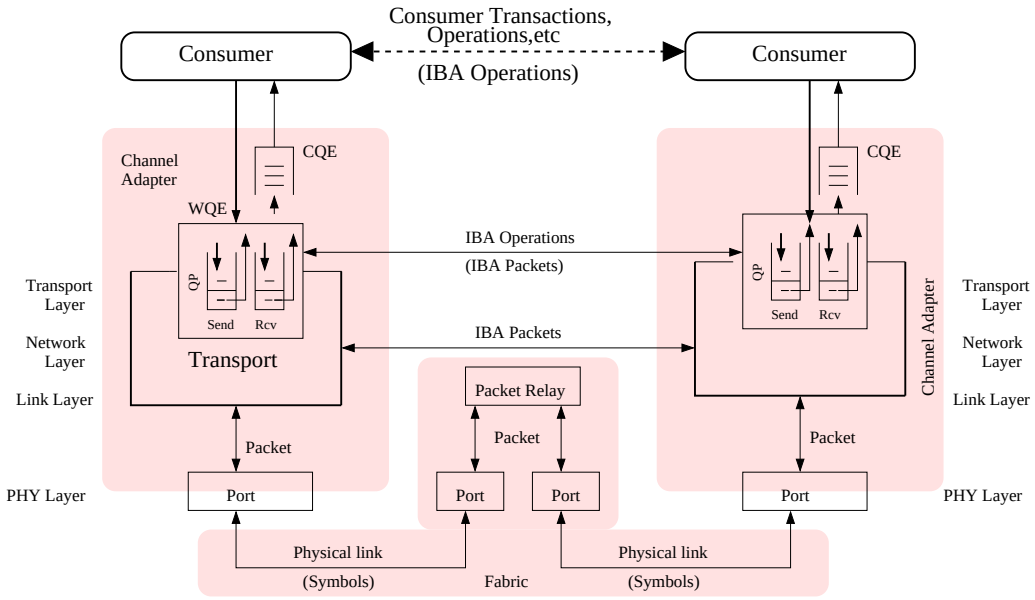

Figure 2.2: InfiniBand Architecture (Courtesy InfiniBand Specifications)

2.2.1 IBA Communication

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, for every connection. Communication requests are initiated by posting Work Queue Requests (WQRs) to these queues. Each WQR is associated with one or more pre-registered buffers which may be used either as the source buffer from which data is transferred out (send WQR) or as the destination buffer into which incoming data is 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 Un-signaled request, no kind of completion event is returned to the user. However, depending on the implementation, the driver cleans up the the Work Queue Request from the appropriate Queue Pair on completion.

IBA supports two types of communication semantics: Channel Semantics (Send-Receive communication model) and memory semantics (RDMA communication model).

2.2.2 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.

RDMA operations have two notable advantages for us to design and implement strong cache coherence. First, it is one-sided communication, that is completely transparent to the peer side. Therefore, the initiator can initiate RDMA operations at its own will. Eliminating involvement of the peer side can overcome the communication performance degradation due to CPU workload of the peer side. This also avoids any interrupt of the peer side processing. Second, RDMA operations provide a "shared-memory illusion". This eases status sharing in caching. In the following chapter, we describe the design details.

2.2.3 Atomic Operations Over IBA

In addition to RDMA, the reliable communication classes also optionally provide support for 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 used to compare two 64-bit values and swap the remote value with the data provided if the comparison succeeds.

Atomics are effectively a variation of 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.

2.3 Motivation

There has been some previous research which focus on dynamism in the data-center environment by HP labs and IBM Research [27, 31]. 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, 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.

Our study of a rigidly configured data-center clearly shows wastage of resources when load is not equally distributed within the tiers. This unequal distribution load is one of the main causes of missing guarantees QoS and is overcome by over-provisioning the tiers with respect to the worst case pattern of requests. As seen in Chapter 1, this is not a cost-effective solution. In this thesis, we address the issue of providing soft QoS guarantees using distributed intelligence within the data-center that can automatically adapt the multi-tier configuration to pattern of requests. We try to satisfy QoS guarantees given by the data-center by making maximum utilization of the resources available within the data-center.

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 [41, 40, 21]. Bhatti et. al., have worked on architectures for supporting end-to-end response time guarantees [12]. Most of the research focus has been on improving QoS guarantees using novel methods of caching and load-balancing [28, 14, 32]. The focus has also been on providing guarantees for real time applications [16]. We believe that these approaches can be used in a complementary manner with our reconfigurability technique to provide better QoS guarantees.

Distributed intelligence for automated re-allocation of resources in the nodes of a cluster-based data-center can be accomplished by using inter-cluster communication to emulate a shared state with load information. Since we focus on data-centers over InfiniBand, we can take advantage of the low latency of the underlying interconnect and use either user-level sockets or the native Verbs interface for communication.

Shah, Kim, Balaji, et. al., have done significant research in User Level High Performance Sockets implementations [43, 24, 25, 8, 9, 7]. In one of our previous works [7], we had evaluated the capabilities of such a pseudo-sockets layer over InfiniBand in the data-center environment.

However, in our work in [30], we designed and architecture for providing coherency for dynamic data in a multi-tier data-center, and evaluated its performance both over sockets and over the native Verbs API provided by InfiniBand. It was apparent that the two-sided nature of sockets API makes the overhead of design-specific helper modules quite huge due to the high load conditions common in data-center environments. The helper modules basically were used to improve performance without changes to the existing application and require inter-tier communication to collect various state information.

Due to this, we focused on the one-sided nature of InfiniBand for this design. 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.

CHAPTER 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.

3.2 Design Challenges

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.

3.2.1 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.

3.2.2 System Wide Shared State

As discussed in Section 3.2.1, 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 3.1) 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.

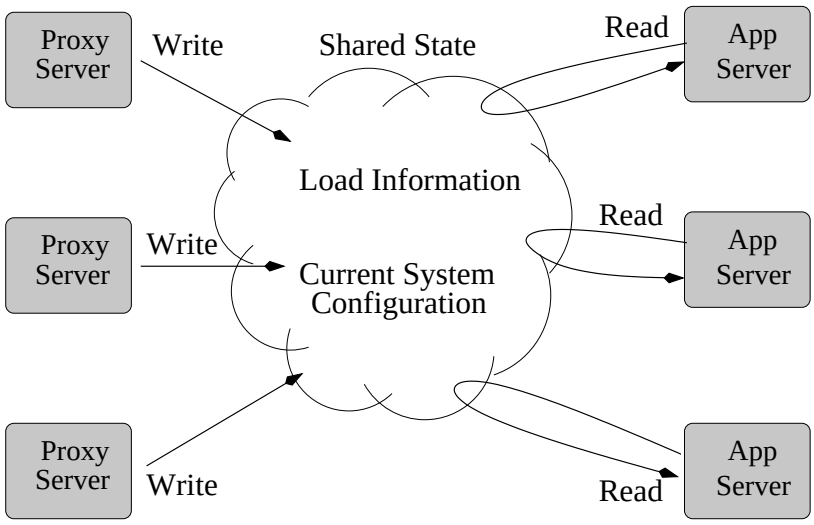

Figure 3.1: Shared State Architecture

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) information 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 [30], 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.

3.2.3 Shared State with Concurrency Control

The logical shared state presented in Section 3.2.2 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 architecture described in Section 3.2.2 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.

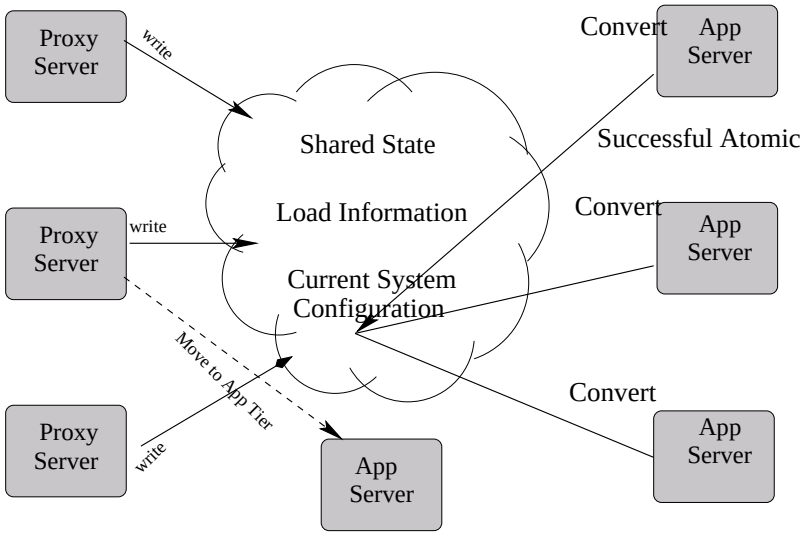

Figure 3.2: Illustration of Concurrency Control Requirement

As shown in figure 3.2 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 we make use of the network based remote atomic operations provided by InfiniBand. As illustrated by 3.2 only one node successfully converts the state of the front-end node, that gets added to the application tier.

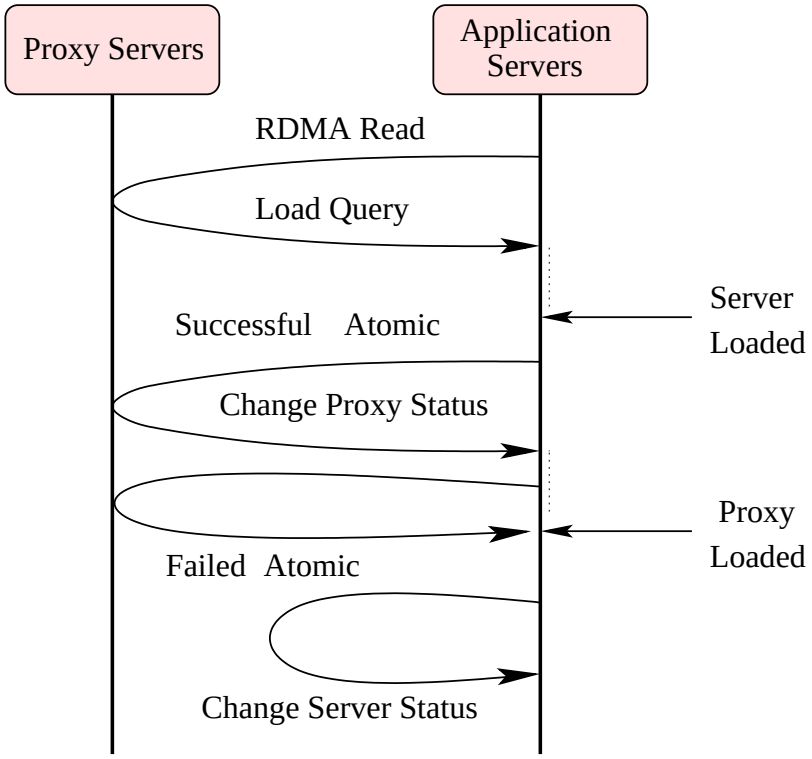

Figure 3.3: Concurrency Control

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 3.3 illustrates the approach used to achieve concurrency control in this architecture.

3.2.4 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 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.

3.2.5 History Aware Reconfiguration

Due to the irregular nature of incoming requests, a small burst of similar requests might potentially trigger a re-configuration 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 responsive 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.

3.2.6 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 responsive

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 responsive 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 Sections 5 and 6.

CHAPTER 4

SOFT QUALITY OF SERVICE GUARANTEES

In this section, we discuss the current trend in the data-center environments to provide some kind of differentiated services 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. This thesis focuses 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 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.

[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.

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.

CHAPTER 5

PERFORMANCE EVALUATION: MICRO-BENCHMARKS AND RECONFIGURABILITY FOR SINGLE WEBSITE

In this chapter, we present the ideal case performance achievable by the native Verbs API (VAPI) over InfiniBand in the form of micro-benchmark results. In Section 5.3, we present the improvement in the basic performance achievable through dynamic reconfigurability for a single website. We also demonstrate the enhanced ability to provide soft Quality of Service guarantees using dynamic reconfiguration in Section 5.4.

In our technical report [26], we presented experimental results with clusters over Myrinet, which used Cluster 0 and Cluster 1 descibed below. Myrinet was used as interconnect and is described below as Interconnect 0:

Cluster 0: 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.

Interconnect 0: 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 experiments described in this thesis, we used 2 clusters whose descriptions are as follows:

Cluster 1: A cluster system consisting of 8 nodes built around SuperMicro SUPER X5DL8-GG motherboards and GC 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 1 GB 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 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.

All nodes in Cluster1 and Cluster2 are interconnected with InfiniBand:

Interconnect 1: 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. We used the IPoIB version 2.0.5_10 for communication over InfiniBand.

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 was 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 Evaluation Methodology

We evaluate our design using some synthetic and real traces. The client program used, emulates Web-Bench. Each client makes a request and waits for the response before making the next request. Throughput (TPS) is measured as the total number of requests serviced for a particular client per second, in the presence of requests from other clients in the data-center. Apache-2.0.48 was configured as proxy for the front-end tier and configured as web server for the back-end.

In our synthetic traces, we define a burst as a burst of requests for the same document or transaction to the proxy or web server respectively. Hence a burst size of 64K (65536) is referred to as a high burst size and corresponds to a trace with alternating bursts of 64K static followed by 64K dynamic requests. Similarly a burst size of 512 is referred to as low burst size and corresponds to a trace with alternating burst of 512 static followed by 512 dynamic requests.

In practice such bursts are seen in traces like the world cup trace [5] which we include in our evaluation. The world cup trace consists of 70% dynamic requests which is bursty since static requests appear scattered in the trace.

5.2 InfiniBand: Performance at Verbs Level

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 5.1a shows the one-way latency achieved by the 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

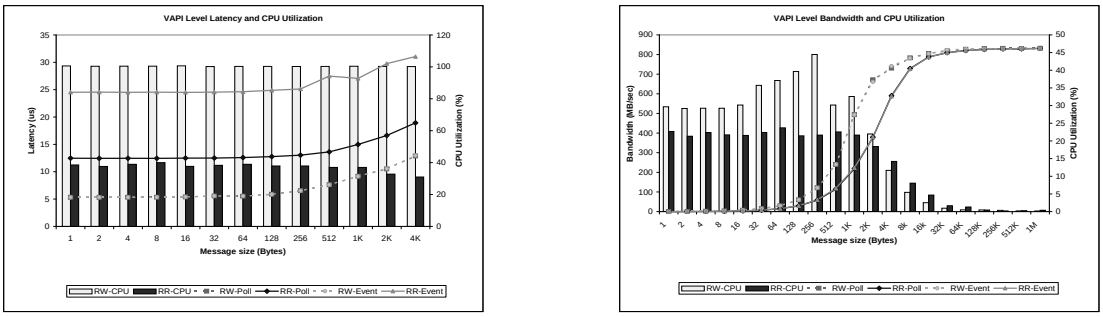

Figure 5.1: Micro-Benchmarks for RDMA write and read: (a) Latency (b) Bandwidth

read approach goes to 30μs. This indifference in the latency for the RDMA write communication model towards 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 5.1b 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 5.2a 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 upto 7.9μs, the Send-Receive scheme achieves upto 6.7μs for 4 byte messages. However, RDMA with immediate data shows a

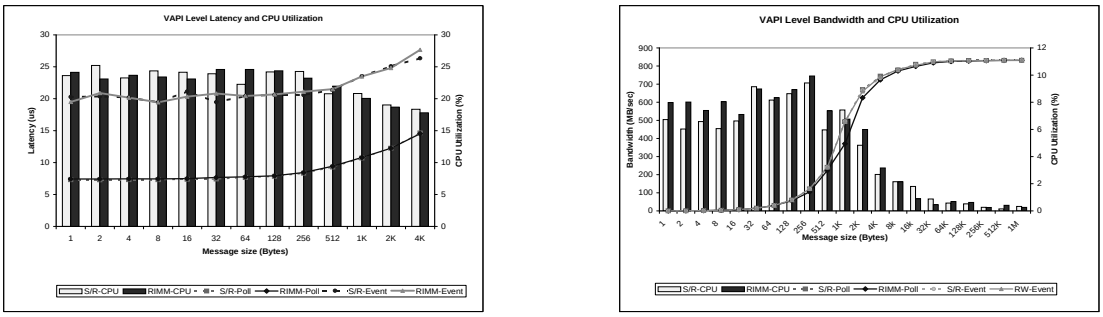

Figure 5.2: Micro-Benchmarks for Send/Recv and RDMA with Immediate: (a) Latency (b) Bandwidth

maximum CPU utilization of 13% and Send-Receive shows a maximum of 30% CPU utilization.

Figure 5.2b 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.

5.3 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.

5.3.1 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.

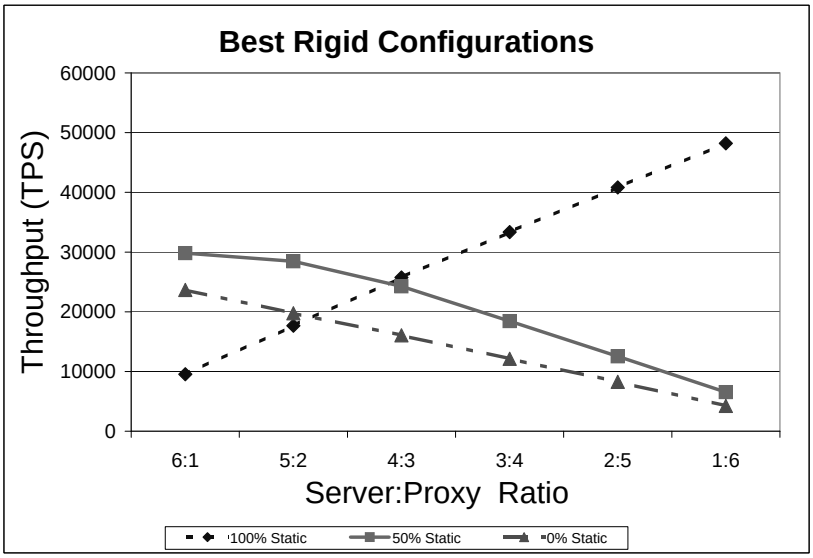

Figure 5.3: Best Rigid Configurations: 7-Node Data-Center

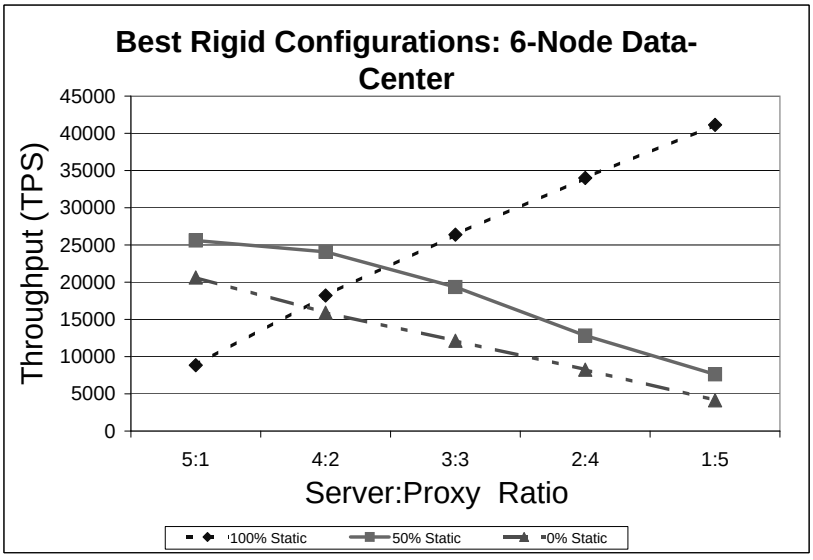

Figure 5.4: Best Rigid Configurations: 6-Node Data-Center

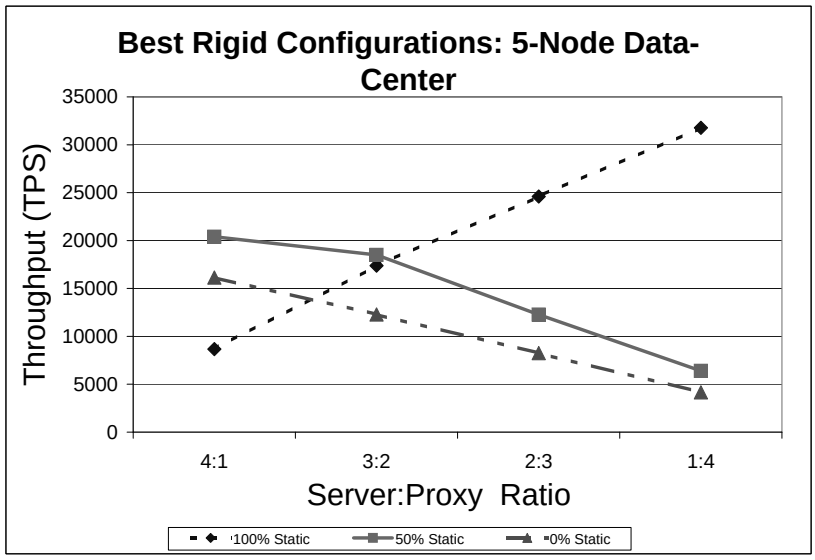

Figure 5.5: Best Rigid Configurations: 5-Node Data-Center

Figure 5.3 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 5.3 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.

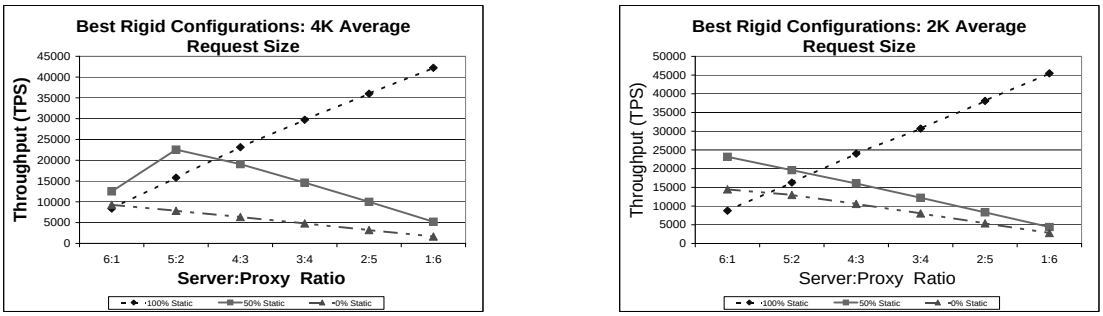

Figure 5.6: Best Rigid Configurations for Different Average Request Sizes: (a) 4KB (b) 2KB

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. Figures 5.4 and 5.5 show the best configuration for various patterns of requests for data-centers consisting of 6 Nodes and 5 Nodes respectively, for various mix of static and dynamic requests. Figures 5.6a and 5.6b show the best configuration for various percentages of static content when the average request size in the data-center is 4KB and 2KB.

5.3.2 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.

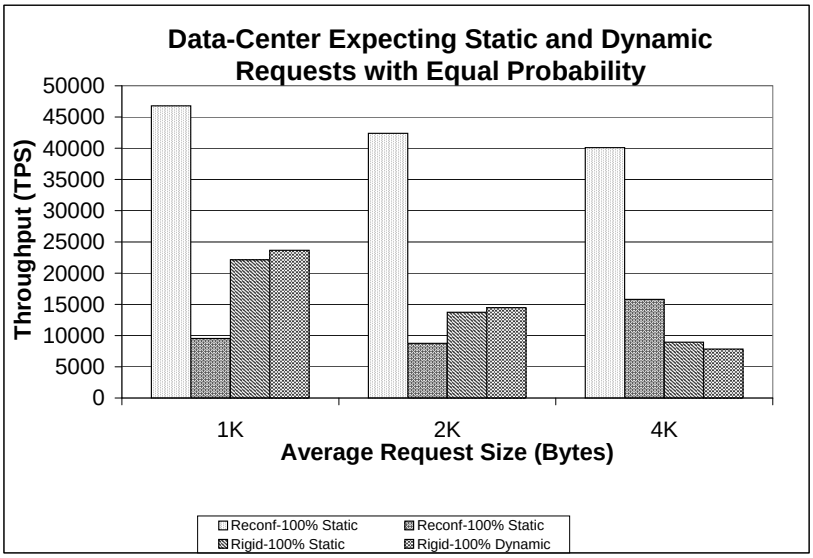

Figure 5.7: Performance For A Snapshot of Requests

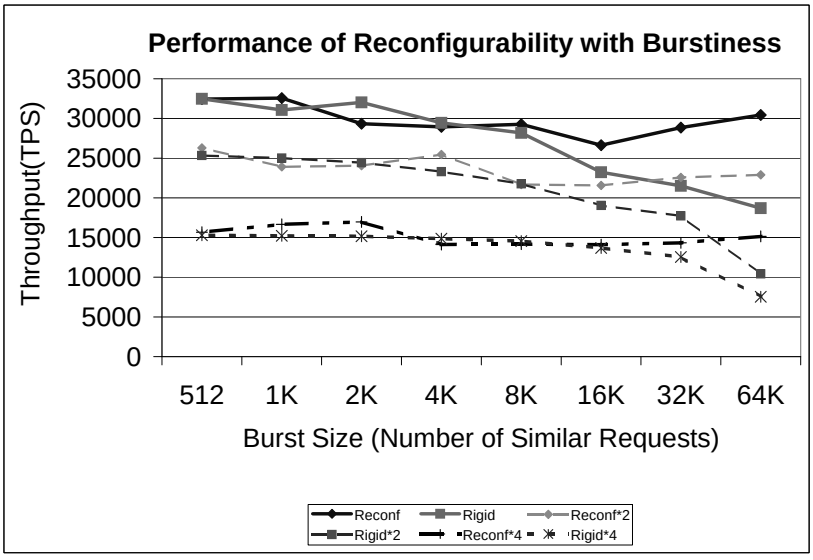

Figure 5.8: Performance For Bursty Traffic

Figure 5.7 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 5.3.1, we chose the best configuration for the rigid scheme for this workload with 6 nodes in the application tier and 1 node in the proxy tier. In this scenario, when there's a burst of dynamic requests, all 6 of the nodes are fully utilized (in the application tier) keeping 1 node in the proxy tier relatively idle. This is coincidentially the best configuration for this request pattern. On the other hand, when there's a burst of static requests, only 1 of the nodes is fully utilized (in the proxy tier) while the 6 nodes in the application tier are idle. However, 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 5.8 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 reconfigurability 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

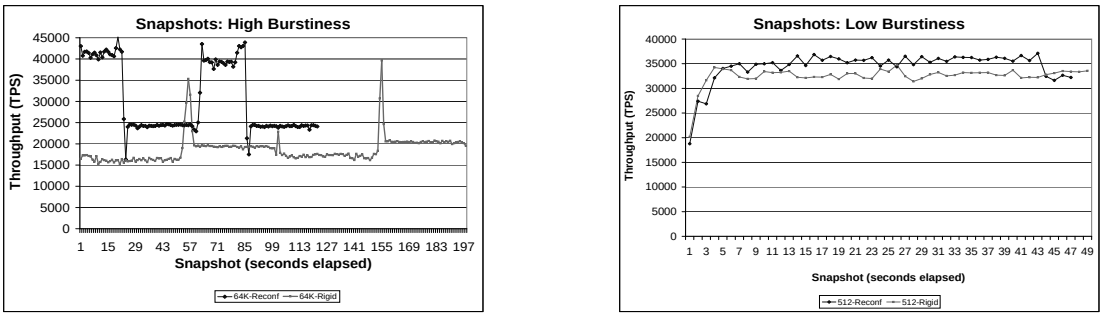

Figure 5.9: Throughput at Various Snapshots for Different Burst Sizes: (a) 65536 (b) 512

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. We can see from this result that as the compute requirement increases an adaptable data-center can outperform a rigid one for bursts of similar requests.

Figures 5.9a and 5.9b show the throughput given by the system at different snapshots 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.

Performance of a Zipf trace: Figure 5.10 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.

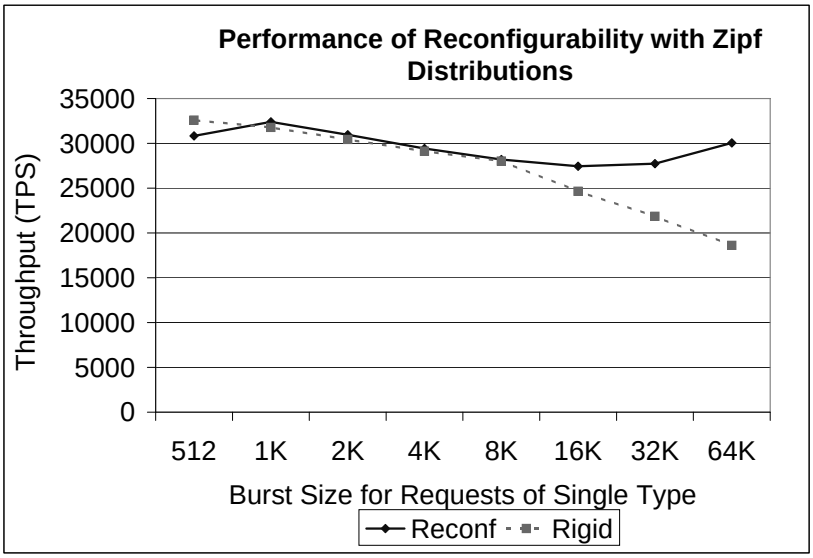

Figure 5.10: Performance of a Zipf Trace

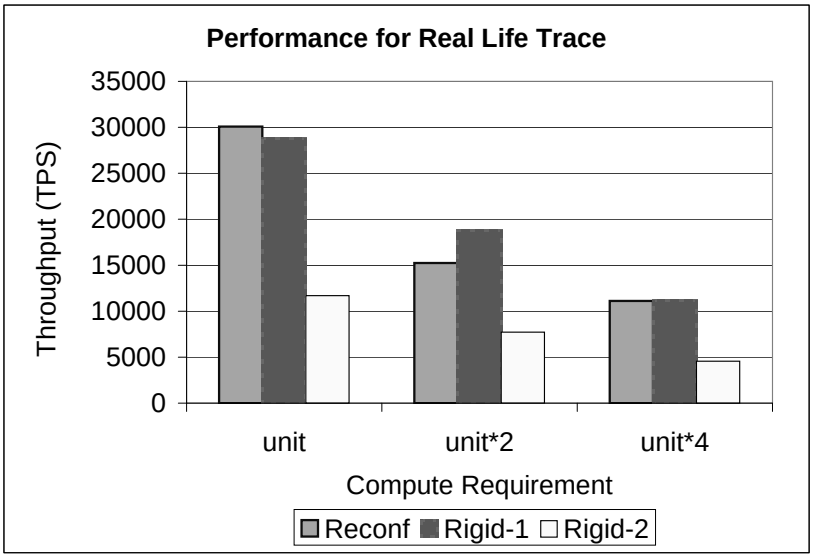

Figure 5.11: Performance of a Real Trace

Performance with real life trace: Figure 5.11 shows the performance of reconfigurability with a real life world cup trace, consisting two-thirds of dynamic requests in bursts, with well spread out static file requests. This performance was evaluated for different assumed requirements of computation for these dynamic requests. When the rigid data-center was configured to accept an equal mix of both type of requests(rigid-1), it coincides with the best configuration for such a trace. Hence the rigid data-center was able to perform as well as a reconfigurable data-center.

However in the case of rigid-2, when the data-center was rigidly configured randomly giving one-third of the servers to the application server tier and two-thirds to the proxy tier, we are able to see upto three times improvement with an adaptable data-center. Such a configuration expecting more static requests is quite practical when documents in demand become available at a website for a brief period of time, like sample question papers before a test. We clearly see that in scenarios like world 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.

Impact of sensitivity on the reconfigurability module: Figure 5.12 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.

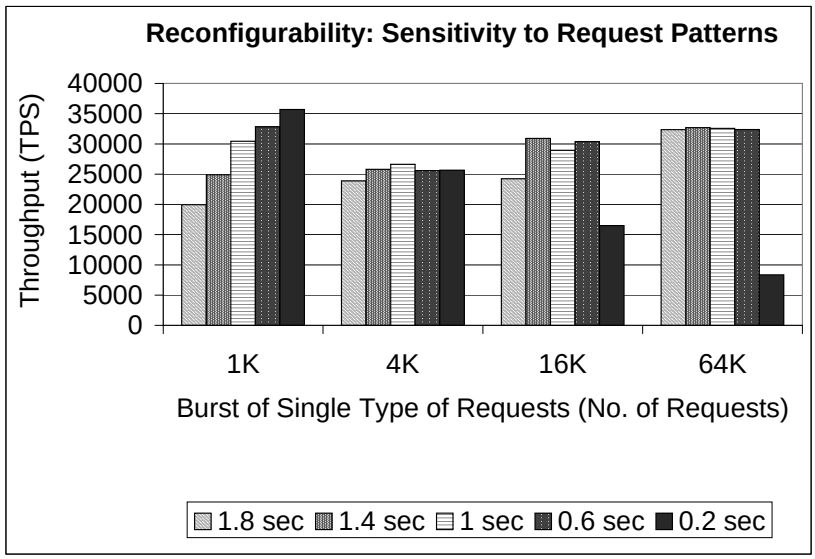

Figure 5.12: Impact of Sensitivity on the reconfigurability module

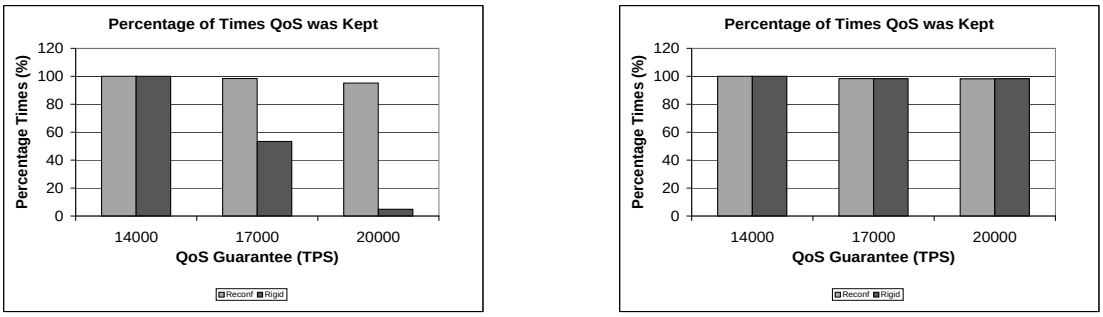

Figure 5.13: QoS guarantee keeping capability: (a) High Burstiness, (b) Low Burstiness

5.4 Soft Quality of Service Support With Reconfigurability

As mentioned in chapter 4, reconfigurability can improve the QoS guarantees given by the data-center. It is also a cost-effective method of guaranteeing Soft QoS than over-provisioning.

Figures 5.13a and 5.13b 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.

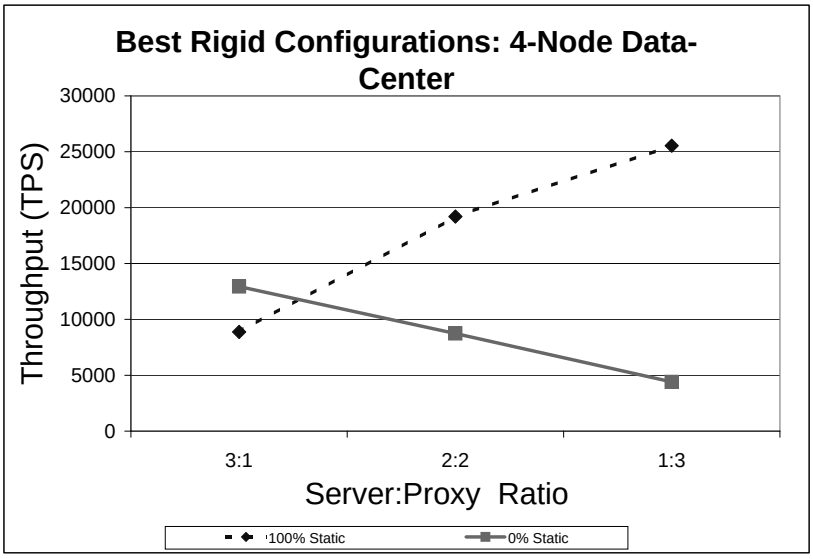

Figure 5.14: Best Rigid Configuration for 4 nodes

5.4.1 Over-Provisioning To Meet QoS Guarantees

Figure 5.14 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 11,000 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 5.15a shows the performance of the reconfigurability scheme compared to the over-provisioning based scheme. We see that when the burst length is low,

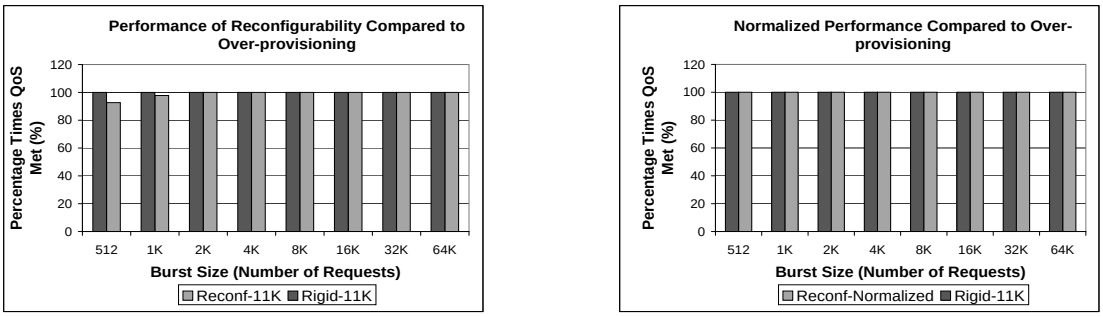

Figure 5.15: QoS guarantee keeping capability: (a) Against Over-Provisioning, (b) Normalized value with number of nodes used

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 5.15b 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.

CHAPTER 6

MULTI-WEBSITE HOSTING SERVERS

In this chapter, 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

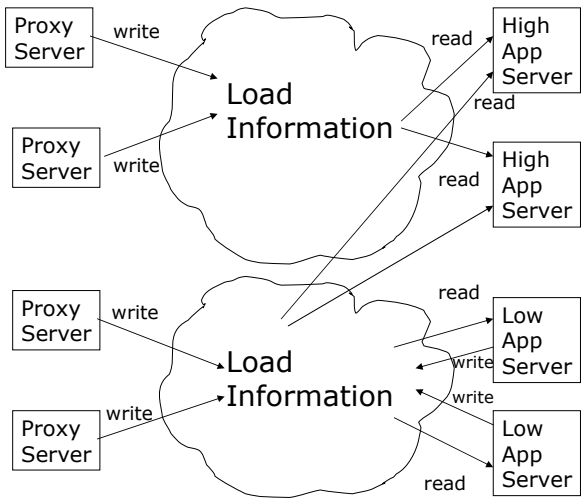

Figure 6.1: Architecture for Multiple Websites

Figure 6.1 shows the architecture for a data-center hosting multiple websites. Each website is allocated a cluster of nodes. Within each cluster, the issues of concurrency control remain. In addition, the higher priority website also keeps the state information of the lower priority website. Resources flow into and out of the lower priority cluster depending on the high priority request load in the data-center.

For a data-center hosting more number of websites at different priority levels, this design can be extended so that the nodes at priority $i+1$ can pull nodes from priority i. This introduces a multi-stage reconfigurability where the shared state only includes nodes from two consecutive priority levels.

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 5.4 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. Due to this, the reconfigurable data-center can achieve equivalent performance with a lesser total number of nodes than the rigid data-center.

Figure 6.2a shows the QoS meeting capabilities for the rigid scheme (with over-provisioning) as well as the reconfigurability scheme. We also show a comparison of this rigid scheme with a reconfigurable data-center of 14% lesser nodes (Reconf-Reduced). On guaranteeing the same QoS as the single website case (8,000 TPS)

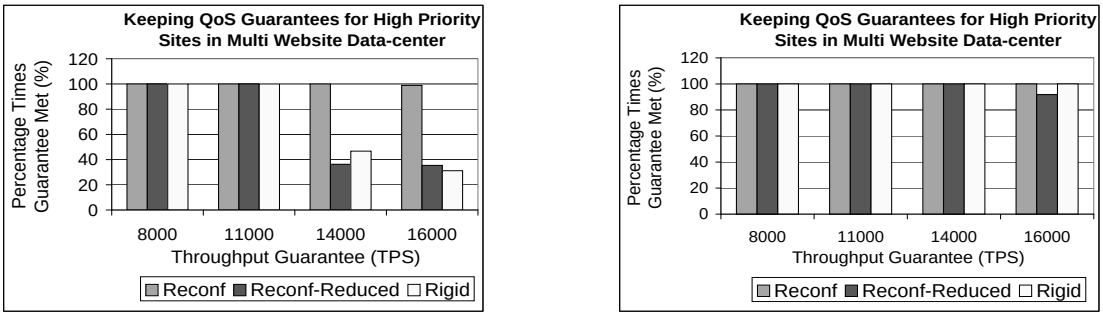

Figure 6.2: QoS guarantees with multiple sites hosted in the data-center: (a) High burstiness, (b) Low burstiness

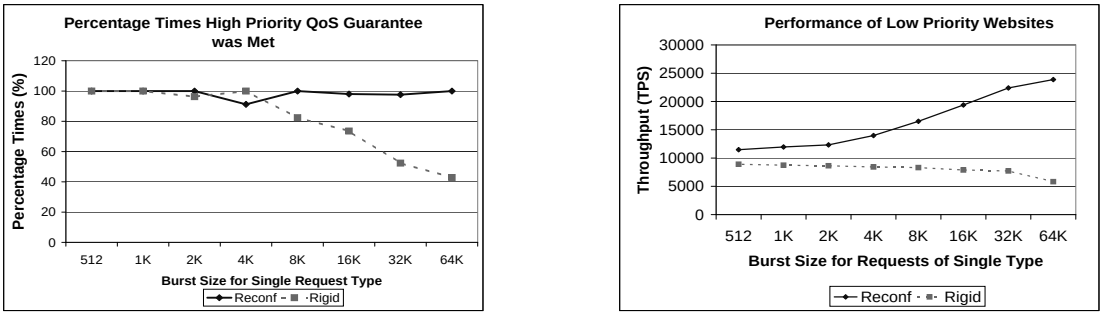

Figure 6.3: Performance with multiple sites hosted in the data-center: (a) QoS meeting capability for high priority requests, (b) Performance for low priority requests

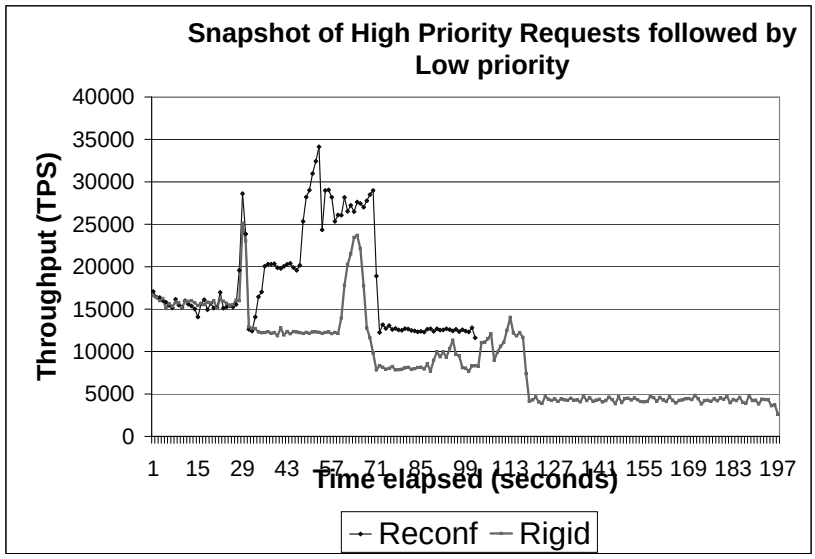

Figure 6.4: Behavior of Reconfigurability with Requests for Multiple sites

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 11,000 TPS, for low burst lengths (512), the rigid scheme is still able to meet the QoS requirements 100% of the times. This is the best guarantee that can be kept with 5 nodes in the data-center. We can, however satisfy the same guarantee with 14% less nodes.

If the QoS guarantee is increased to 14000, the best guarantee given by a datacenter expecting request patterns with alternating resource requirements, on an increased burst length of 65536, the rigid scheme is only able to meet the QoS guarantee around 46% of the times. The reconfigurability scheme with reduced resources performs equally as the rigid scheme in this scenario. The reconfigurability based scheme, with the same resources, meets the QoS guarantee 100% of the times in all cases. This is because as seen in Figure 6.2b, with the existing resources the data-center is capable of guaranteeing upto 14000 but is unable to do so when its resources are not properly utilized and hence is forced to guarantee much less throughput since such bursty patterns with single resource requirement is very practical in present day Internet services.

When the QoS guarantee was further increased to 16000, the rigid model was only able to keep it around 35% of the times whereas the reconfigurable model could keep it close to 100% of the times. This shows that with reconfigurability, we can guarantee upto 40% more QoS than the best guarantee given by the rigid data-center.

Figure 6.2b shows the QoS meeting capabilities of all three schemes for less bursty request patterns. We see that the data-center with reconfigurability with 14% lesser

nodes is able to satisfy QoS upto 90% of the times. The rigid data-center is now able to perform equally with the reconfigurable data-center and thus clearly bringing out the actual potential of the available resources. However, all these resources are not always exploited as request patterns are not always as favourable. For example when a news flash is posted, there may be millions of requests for it in a short period of time and only the proxy tiers catering to these requests. This scenario pertains to the burst size of 65536 when the rigid data-center is unable to satisfy such high guarantees due to resource crunch in the front-end tiers.

Figure 6.3a shows the QoS meeting capability of the over-provisioning scheme and the reconfigurability scheme for the high priority requests with varying burst lengths.

Figure 6.3b 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 6.4 shows the snapshot performance for the data-center with a trace having bursts of high priority and low priority requests alternatively. For this experiment, we used a trace of one million requests with alternating bursts of high and low priority

requests. For low burst sizes, as we have seen earlier, the rigid scheme performs equally as the reconfigurable scheme due to improved utilization. However, for this case we use high burst sizes. We can see that the reconfigurability scheme performs better than the rigid scheme in nearly every case. The reconfigurable scheme completely services one million requests in about 100 seconds whereas the rigid scheme takes twice as much time with very low performance to the low priority bursts in the later parts of the trace.

CHAPTER 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.

This thesis 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 25% better), meet the same guarantees with lesser resources (up to 20% lesser nodes), or even both in some cases.

This work can be extended to further support 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.

The idea of reconfigurability can also be applied in other applications like load-balancing. The proxy tier can use the one-sided communication features to forward requests to the least loaded nodes. Instead of QoS, load-balancing can be the criteria for reconfiguration. It can also be applied in the area of fault tolerance where nodes in the data-center can reconfigure on system failures.

BIBLIOGRAPHY

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

[14] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)*, pages 126–134, 1999.

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

[16] 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.

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

[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. White Paper: 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 2001.

[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] S. Krishnamoorthy, P. Balaji, K. Vaidyanathan, H. W. Jin, and D. K. Panda. Technical Report: Dynamic Reconfigurability Support for providing Soft Quality of Service Guarantees in Multi-Tier Data-Centers over InfiniBand. Technical report, The Ohio State University, Columbus, OH, February 2004.

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

[28] Quanzhong Li and Bongki Moon. Distributed cooperative Apache Web server. In *Proceedings of the 10th International World Wide Web Conference*, Hong Kong, May 2001.

[29] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *the Proceedings of the 17th Annual ACM International Conference on Supercomputing*, June 2003.

[30] 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)*, March 2004.

[31] Daniel OHare, Pankaj Tandon, Hemanth Kalluri, and Phil Mills. White Paper: SNIA SSF Virtualization Demonstration. In *IBM Systems Group - TotalStorage Software*, October 2003.

[32] 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 1998.

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

[34] 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.

[35] 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.

[36] 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.

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

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

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

[40] 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 2003.

[41] 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 2002.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *the Proceedings of the International Conference on Parallel Processing*, October 2003.