DESIGNING HIGH PERFORMANCE AND SCALABLE DISTRIBUTED DATACENTER SERVICES OVER MODERN INTERCONNECTS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Sundeep Narravula, B.Tech, MS

* * * * *

The Ohio State University

2008

Dissertation Committee:

Prof. D. K. Panda, Adviser

Prof. P. Sadayappan

Prof. F. Qin

Approved by

Adviser
Graduate Program in
Computer Science and
Engineering

ABSTRACT

Modern interconnects like InfiniBand and 10 Gigabit Ethernet have introduced a range of novel features while delivering excellent performance. Due to their high performance to cost ratios, increasing number of datacenters are being deployed in clusters and cluster-of-cluster scenarios connected with these modern interconnects. However, the extent to which the current deployments manage to benefit from these interconnects is often far below the achievable levels.

In order to extract the possible benefits that the capabilities of modern interconnects can deliver, selective redesigning of performance critical components needs to be done. Such redesigning needs to take the characteristics of datacenter applications into account. Further, performance critical operations common to multiple datacenter applications like caching, resource management, etc. need to be identified and redesigned utilizing the features of modern interconnects. These operations can be designed and implemented as system services such that they can be provided in a consolidated platform for all other datacenter applications and services to utilize.

In this thesis we explore various techniques to leverage the advanced features of modern interconnects to design these distributed system services. We identify a set of services with high performance requirements that have wide utility in datacenters scenarios. In particular, we identify distributed lock management, global memory aggregation and large scale data-transfers as performance critical operations that need to be carefully redesigned in order to maximize the benefits using of modern interconnects. We further explore the use of these primitive operations and RDMA capabilities of modern interconnects to design highly efficient caching schemes which are critical to most datacenters. We present these components in a layered framework such that applications can leverage the benefits of these services through pre-defined interfaces. We present detailed experimental results demonstrating the benefits of our approaches.

Dedicated to all my family and friends

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. D. K. Panda for guiding me throughout the duration of my study. I would like to thank him for his friendly advice during the past years.

I would like to thank my committee members Prof. P. Sadayappan and Prof. F. Qin for their valuable guidance and suggestions. I am grateful to Prof. S. Parthasarathy and Dr. Jin for their support and guidance at various stages of my study.

I would like to thank all my nowlab colleagues, past and present, Karthik, Jin, Savitha, Ping, Hari, Greg, Amith, Gopal, Abhinav, Wei, Lei, Sayantan, Ranjit, Wei-hang, Qi, Matt, Adam, Prachi, Sita, Jaidev, Tejus, Rahul, Ouyang, Jonathan, Jiux-ing, Jiesheng, Weikuan, Sushmitha and Bala.

I am lucky to have collaborated with Karthik, Savitha and Jin in our initial datacenter projects and I am grateful for the countless hours of work and fun we have had together. I'm especially thankful to Amith, Gopal, Vishnu and Jin and I'm very glad to have collaborated closely with them on various projects.

I would also like to thank my roommates at OSU Amith, Aravind, Aromal, Devi, Jeeth, Naveen and Srinivas for their friendship and support.

Finally, I would like to thank my family, Snehalatha (my mom), Nagarjuna (my dad) and my favorite cousins. I would not have had made it this far without their love and support.

VITA

June 30th, 1981	Born - Hyderabad, India.
July 1998 - June 2002	B. Tech, Indian Institute of Technology - Madras, Chennai, India.
May 2001 - July 2001	Summer Intern, Microsoft IDC, Hyderabad, India.
August 2002 - August 2008	Graduate Teaching/Research Associate, The Ohio State University.

PUBLICATIONS

S. Narravula, H. Subramoni, P. Lai, R. Noronha and D. K. Panda, "Performance of HPC Middleware over InfiniBand WAN", Int'l Conference on Parallel Processing (ICPP '08), September 2008.

K. Vaidyanathan, P. Lai, S. Narravula and D. K. Panda, "Optimized Distributed Data Sharing Substrate for Multi-Core Commodity Clusters: A Comprehensive Study with Applications", IEEE International Symposium on Cluster Computing and the Grid, May 2008.

P. Lai, S. Narravula, K. Vaidyanathan and D. K. Panda, "Advanced RDMA-based Admission Control for Modern Data-Centers", IEEE International Symposium on Cluster Computing and the Grid, May 2008.

S. Narravula, A. R. Mamidala, A. Vishnu, G. Santhanaraman, and D. K. Panda, "High Performance MPI over iWARP: Early Experiences", Int'l Conference on Parallel Processing (ICPP '07), September 2007.

G. Santhanaraman, S. Narravula, and D. K. Panda, "Designing Passive Synchronization for MPI-2 One-Sided Communication to Maximize Overlap", IEEE International Parallel and Distributed Processing Symposium (IPDPS '08), April, 2008.

A. Vishnu, M. Koop, A. Moody, A. Mamidala, S. Narravula, and D. K. Panda, "Topology Agnostic Network Hot-Spot Avoidance with InfiniBand", Journal of Concurrency and Computation: Practice and Experience, 2007.

G. Santhanaraman, S. Narravula, A. R. Mamidala, and D. K. Panda, "MPI-2 One Sided Usage and Implementation for Read Modify Write operations: A case study with HPCC", EUROPVM/MPI '07.

S. Narravula, A. Mamidala, A. Vishnu, K. Vaidyanathan and D. K. Panda, "High Performance Distributed Lock Management Services using Network-based Remote Atomic Operations", IEEE International Symposium on Cluster Computing and the Grid (CCGrid), May 2007.

A. Vishnu, M. Koop, A. Moody, A. Mamidala, S. Narravula, and D. K. Panda, "Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective", IEEE International Symposium on Cluster Computing and the Grid (CCGrid), May 2007.

A. Vishnu, A. Mamidala, S. Narravula, and D. K. Panda, "Automatic Path Migration over InfiniBand: Early Experiences", Third International Workshop on System Management Techniques, Processes, and Services (SMTPS '07), held in conjunction with IPDPS '07, March 2007.

K. Vaidyanathan, S. Narravula, P. Balaji and D. K. Panda, "Designing Efficient Systems Services and Primitives for Next-Generation Data-Centers", Workshop on NSF Next Generation Software (NGS '07) Program; held in conjuction with IPDPS, Greece, March 2007, Long Beach, CA.

A. Mamidala, S. Narravula, A. Vishnu, G. Santhanaraman, and D. K. Panda "Using Connection-Oriented vs. Connection-Less Transport for Performance and Scalability of Collective and One-sided operations: Trade-offs and Impact", ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP '07), March 2007.

K. Vaidyanathan, S. Narravula and D. K. Panda, "DDSS: A Low-Overhead Distributed Data Sharing Substrate for Cluster-Based Data-Centers over Modern Interconnects", IEEE International Symposium on High Performance Computing, December 2006.

H. -W. Jin, S. Narravula, K. Vaidyanathan and D. K. Panda, “NemC: A Network Emulator for Cluster-of-Clusters”, IEEE International Conference on Communication and Networks, October 2006.

S. Narravula, H. -W. Jin, K. Vaidyanathan and D. K. Panda, “Designing Efficient Cooperative Caching Schemes for Multi-Tier Data-Centers over RDMA-enabled Networks”, IEEE International Symposium on Cluster Computing and the Grid, May 2006.

P. Balaji, K. Vaidyanathan, S. Narravula, H. -W. Jin, and D. K. Panda, “Designing Next-Generation Data-Centers with Advanced Communication Protocols and Systems Services”, Workshop on NSF Next Generation Software (NGS ’06) Program; held in conjuction with IPDPS, Greece, April 2006.

H. -W. Jin, S. Narravula, G. Brown, K. Vaidyanathan, P. Balaji and D. K. Panda, “Performance Evaluation of RDMA over IP: A Case Study with the Ammasso Gigabit Ethernet NIC”, Workshop on High Performance Interconnects for Distributed Computing, July 2005.

S. Narravula, P. Balaji, K. Vaidyanathan, H. -W. Jin and D. K. Panda, “Architecture for Caching Responses with Multiple Dynamic Dependencies in Multi-Tier Data-Centers over InfiniBand”, IEEE International Symposium on Cluster Computing and the Grid, May 2005.

P. Balaji, S. Narravula, K. Vaidyanathan, H. -W. Jin and D. K. Panda, “On the Provision of Prioritization and Soft QoS in Dynamically Reconfigurable Shared Data-Centers over InfiniBand”, IEEE International Symposium on Performance Analysis of Systems and Software, March 2005.

P. Balaji, K. Vaidyanathan, S. Narravula, K. Savitha, H. -W. Jin and D. K. Panda, “Exploiting Remote Memory Operations to Design Efficient Reconfiguration for Shared Data-Centers over InfiniBand”, Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations and Technologies, September 2004.

P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu and D. K. Panda, “Sockets Direct Procotol 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", Workshop on System Area Networks, February 2004.

FIELDS OF STUDY

Major Field: Computer Science and Engineering

Studies in:

Computer Architecture	Prof. D. K. Panda
Software Systems	Prof. S. Parthasarathy
Computer Networking	Prof. P. Sinha

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Internet has been growing at a tremendous pace and web applications have now become ubiquitous. The nature of this growth is multi-fold. Unprecedented increases in number of users, amount of data and the complexity of web-applications have created a critical need for efficient server architectures.

Cluster based datacenters have become the defacto standard method for deploying these web-serving applications. The basic technique currently employed to handle the raising complexity of web applications is to logically partition the datacenter into multiple tiers. Each tier handles a portion of the request processing. In this architecture, the number of resources available at each tier are increased as needed in order to handle the increasing volume of web requests. While this method can alleviate the problems to certain extent, the overall effectiveness is often limited. Additional issues like communication/cooperation among the increasing number of servers often imposes a serious limitation in this context. In particular, operations that span multiple nodes tend to limit the scalability and performance of a datacenter. These problems are further magnified for the emerging WAN-based distributed datacenter and cluster-of-cluster scenarios.

On the other hand, datacenter applications usually involve a small subset of basic operations that often involve cooperation among multiple nodes. For example, file-systems, proxy servers and data-base servers often perform data-caching or file-caching and resource managers, file-systems and data-base servers routinely perform locking of files and resources, etc. These operations like caching and distributed locking apply to almost all datacenter deployment scenarios and have been researched on significantly. Providing such operations as system services or primitives is highly desirable and can be extremely beneficial to the designing of datacenter applications. Further, the performance and scalability of such services are critical for building next generation datacenters catering to the enormous application requirements.

In this context, introduction of modern high performance interconnects like InfiniBand and 10 Gigabit Ethernet (iWARP) provide several advanced features like Remote Direct Memory Access (RDMA) and Remote Memory Atomic Operations that can be leveraged to improve the datacenter performance significantly. The datacenter services can be optimized significantly and scaled using the features provided by these modern interconnects. The growing popularity of clusters with modern network interconnects coupled with the ever increasing demands for higher performance and scalability of datacenters leads to the the critical challenge of effectively harnessing the capabilities of the modern interconnects in order to improve the performance of current and next generation datacenters.

In this thesis, we identify distributed lock management, global memory aggregation, large scale data transfers, dynamic and cooperative caching as basic primitives that are widely used performance critical operation in modern datacenters. We present efficient designs for datacenter services and primitives that leverage the capabilities of emerging modern interconnects in order to provide the performance and scalability required for handling the needs of the next generation Web applications. Broadly, we summarize our domain into the following:

1. *Datacenter Service Primitives:* How well can basic primitives such as distributed lock management, large scale data transfers and global memory aggregation be designed to boost the performance of higher level datacenter functions?

2. *Web Caching Services:* Can we design effective caching schemes utilizing RDMA and other interconnect features available to provide dynamic content caching and cooperative caching to effectively scale the overall datacenter performance?

3. *Protocols over WAN:* Can these high performance designs be adapted to cluster-of-cluster scenarios by leveraging WAN capabilities of IB and iWARP enabled Ethernet networks? What are the key issues involved? And what kind of protocols are better suited to WAN interconnections?

The domain objectives described above all involve multiple challenges. These need to be achieved in the light of the various constraints imposed by traditional datacenter applications, the networking mechanisms and the performance requirements. We intend to study and investigate these challenges to design efficient and scalable datacenter services.

To demonstrate the effectiveness of our various designs, we utilize InfiniBand and 10 Gigabit Ethernet (iWARP) and incorporate our designs to work with widely popular open source datacenter applications like Apache, PHP and MySQL.

CHAPTER 2

BACKGROUND AND MOTIVATION

In this section, we present a brief background on modern RDMA enabled interconnects, cluster-based multi-tier data-centers and web-caching.

2.1 RDMA Enabled Interconnects

Many of the modern interconnects such as InfiniBand, 10 Gigabit Ethernet/iWARP, etc. provide a wide range of enhanced features. In these networks, an abstraction interface for adapter is specified in the form of RDMA Verbs. InfiniBand and iWARP support both channel and memory semantics. Several features like Remote Memory Direct Access (RDMA), Remote Atomic Operations, WAN capability, QoS support, NIC level virtualization, etc. are supported by modern adapters.

2.1.1 RDMA Communication Model

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

In channel semantics, every send request has a corresponding receive request at the remote end. Thus there is one-to-one correspondence between every send and receive operation. Failure to post a receive descriptor on the remote node results in the message being dropped and if the connection is reliable, it might even result in the breaking of the connection.

In memory semantics, Remote Direct Memory Access (RDMA) operations are used. These operations are transparent at the remote end since they do not require a receive descriptor to be posted. In this semantics, the send request itself contains both the virtual address for the local transmit buffer as well as that for the receive buffer on the remote end.

Most entries in the WQR are common for both the Send-Receive model as well as the RDMA model, except an additional remote buffer virtual address which has to be specified for RDMA operations.

There are two kinds of RDMA operations: RDMA Write and RDMA Read. In an RDMA write operation, the initiator directly writes data into the remote node's user buffer. Similarly, in an RDMA Read operation, the initiator reads data from the remote node's user buffer.

Apart from improved performance, RDMA operations have two notable advantages that enable highly efficient designs. 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 further enables the design of novel distributed algorithms.

2.1.2 InfiniBand Architecture

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

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 specifications for Verbs includes a queue-based interface, known as a Queue Pair (QP), to issue requests to the HCA. Figure 2.1 illustrates the InfiniBand Architecture model.

Each Queue Pair is a communication endpoint. A Queue Pair (QP) consists of the send queue and the receive queue. Two QPs on different nodes can be connected to each other to form a logical bi-directional communication channel. An application can have multiple QPs. Communication requests are initiated by posting Work Queue Requests (WQRs) to these queues. Each WQR is associated with one or more pre-registered buffers from which data is either transfered (for a send WQR) or received (receive WQR). The application can either choose the request to be a Signaled (SG) request or an Un-Signaled request (USG). When the HCA completes the processing of a signaled request, it places an entry called 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

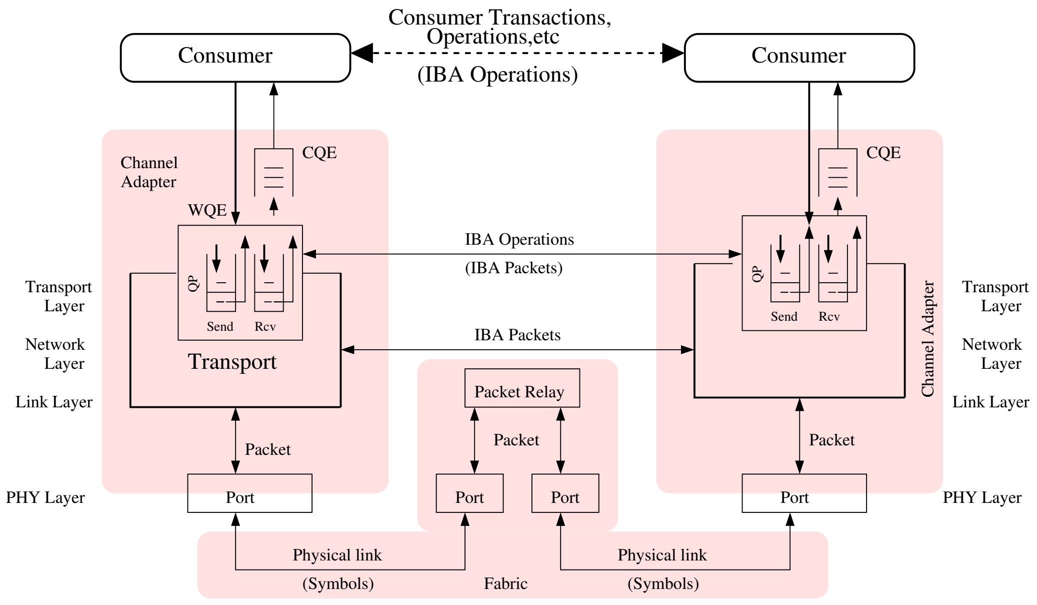


Figure 2.1: InfiniBand Architecture (Courtesy InfiniBand Specifications)

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.

Remote Atomic Operations

InfiniBand provides remote atomic operations, such as *Compare_and_Swap* and *Fetch_and_Add*, that can be used for providing efficient synchronization among the nodes. The advantage of such operations is that the atomicity of these is handled by the NIC. i.e. the synchronization operation does not need a host level agent to perform the actual work. InfiniBand standard currently supports these operations on 64 bit fields. Newer InfiniBand NICs plan to extend these operations to larger data-types [2].

2.1.3 Internet Wide Area RDMA Protocol

The iWARP protocol defines RDMA operations over Ethernet networks [32]. Figure 2.2 shows the basic protocol layering for iWARP detailed in its specification [32]. For iWARP, the basic message transport is undertaken by the TCP layer. Since TCP itself is a stream protocol and does not respect message boundaries, an additional MPA layer is introduced to enforce this. SCTP [78] is also proposed to be used instead of TCP+MPA. The actual zero-copy capability is enabled by the Direct Data Placement (DDP) Layer. The RDMA features provided by DDP are exported to the upper level protocol by the RDMAP layer. It is to be noted that the ordering of data within a single data-transfer is not guaranteed by the specifications of these layers. However, adapters often do guarantee this as an option.

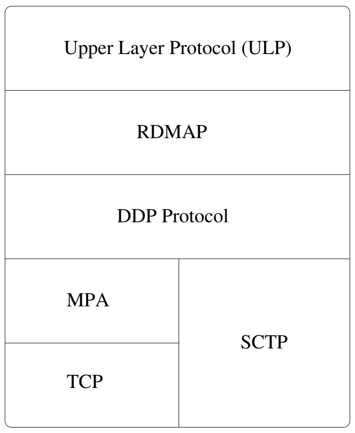


Figure 2.2: RDMAP Layering (Courtesy: RDMA Protocol Specification Draft [71])

The iWARP protocol comprising of RDMAP, DDP, MPA and TCP layers as shown in Figure 2.2 are intended to be implemented in hardware for RNICs resulting in significant performance improvements over the traditional TCP/IP stacks.

The iWARP supports two types of communication semantics: Channel Semantics (Send-Receive communication model) and Memory Semantics (RDMA communication model). Remote Direct Memory Access (RDMA) [45] operations allow processes to access the memory of a remote node process without the remote node CPU intervention. These operations are transparent at the remote end since they do not require the remote end to be involved in the communication. Therefore, an RDMA operation has to specify both the memory address for the local buffer as well as that for the remote buffer. There are two kinds of RDMA operations: RDMA Write and RDMA Read. In an RDMA write operation, the initiator directly writes data into the remote node's memory. Similarly, in an RDMA Read operation, the initiator reads data from the remote node's memory.

The basic communication is achieved over connected end points known as the Queue Pairs (QPs). Each QP consists of a send queue and a receive queue. To enable data transfers, each QP needs to be setup and its state needs to be transitioned into the *connected* state. The communicating process initiates a data transfer by posting a descriptor. The descriptor typically holds all the required information for the data transfer like source/destination buffer pointers, type of transfer, etc. In case of RDMA operations, the descriptor also contains the remote buffer information.

2.1.4 WAN Interconnects

With the advent of InfiniBand WAN, both IB and 10GE/iWARP are now capable of communication over WAN distances. While 10GE/iWARP NICs are naturally capable of WAN communication due to the usage of TCP/IP in the iWARP protocol stack, IB communication needs additional infrastructure for such capabilities.

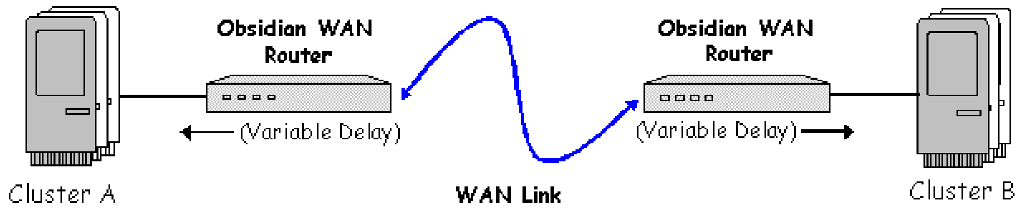


Figure 2.3: Cluster-of-Clusters Connected with Obsidian Longbow XRs

InfiniBand Range Extension: Multiple vendors including Obsidian and Net provide IB WAN capability. Obsidian Longbows [33] primarily provide range extension for InfiniBand fabrics over modern 10 Gigabits/s Wide Area Networks (WAN). The Obsidian Longbows work in pairs establishing point-to-point links between clusters with one Longbow at each end of the link. Figure 2.3 shows a typical deployment of the IB WAN routers. The Longbows communicate using IPv6 Packets over SONET, ATM, 10 Gigabit Ethernet and dark fiber applications. The Longbows can essentially support IB traffic at SDR rates (8 Gbps).

The Obsidian Longbow XR routers also provide a highly useful feature of adding delay to packets transmitted over the WAN link. Each of the Longbows provide a Web interface to specify delay. The packets are then delayed for the specified time before and after traversing over the WAN link. This added delay can indirectly be used as a measure of emulated distance. i.e. this essentially corresponds to the wire delay of about 5 *us* for each *km* of wire length. We leverage this feature to emulate cluster-of-clusters with varying degrees of separation in the following experiments.

2.2 Overview of Web Datacenters

Internet based clients typically connect to web applications over WAN. Due to the cost benefits offered by modern cluster-based architectures, they have become the defacto standard for hosting these applications. These applications present an incredibly diverse set of characteristics and requirements. They can vary from serving of simple image to performing a complex operation involving multiple dynamic backend objects. Catering to this diversity and the need for high performance, web servers have evolved into multi-tier architectures with the capability of providing flexibility, performance, availability and scalability. In this section, we describe the main characteristics of these web datacenters.

2.2.1 Multi-Tiered Architectures

A typical data-center architecture consists of multiple tightly interacting layers known as tiers. Each tier can contain multiple physical nodes. Figure 2.4 shows a typical Multi-Tier Data-Center. Requests from clients are load-balanced by the edge services tier 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 can include data-center security and balancing the request load sent to the back-end based on certain pre-defined algorithms.

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,

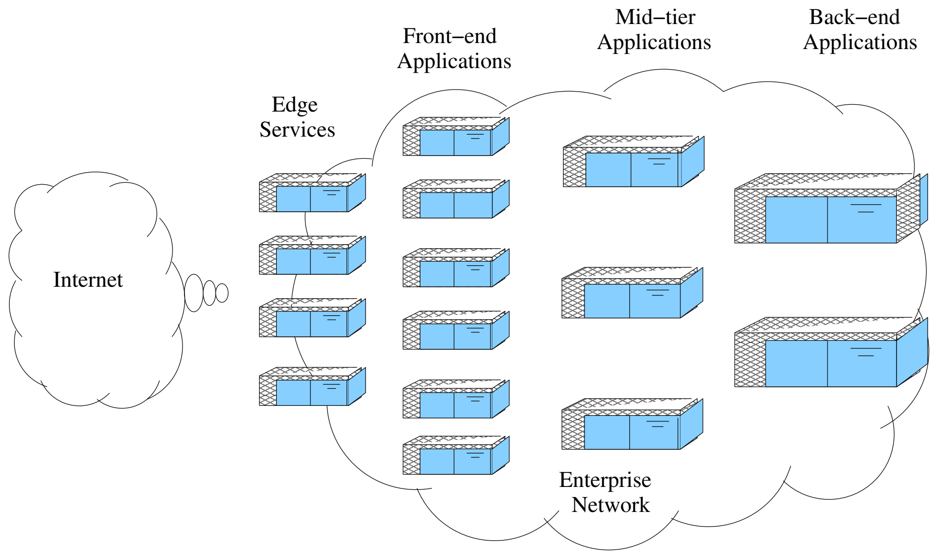


Figure 2.4: A Typical Multi-Tier Data-Center (Courtesy CSP Architecture [75])

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 data repositories. These servers could either be compute intensive or I/O intensive based on the query format. Queries involving non key searches can be more I/O intensive requiring large data fetches into memory. For more complex queries, such as those which involve joins or sorting of tables, these servers can be more compute intensive.

2.2.2 Web Applications

Web applications primarily communicate with the clients over WAN using TCP/IP transport protocol. As mentioned earlier, the characteristics of these applications vary significantly from each other. Large file accesses can be IO bound whereas encryption requirements can lead to high CPU usage. Further, these are designed to handle large scale parallelism. Many applications such as the popular Apache [42] and MySQL [5], are designed as multi-threaded or multi-process servers to handle the expected parallelism.

2.3 Web Caching

In this section we describe a brief outline of web-caching and the basic coherency and consistency issues that it presents.

It has been well acknowledged in the research community that in order to provide or design a data-center environment which is efficient and offers high performance, one of the critical issues that needs to be addressed is the effective reuse of cache content stored away from the origin server. This has been strongly backed up by researchers who have come up with several approaches to cache more and more data at the various tiers of a multi-tier data-center. Traditionally, frequently accessed static content was cached at the front tiers to allow users a quicker access to these documents. In the past few years, researchers have come up with approaches of caching certain dynamic content at the front tiers as well [26].

In the current generation web content, many cache eviction events and uncachable resources are driven by two server application goals: First, providing clients with a *recent* or *coherent* view of the state of the application (i.e., information that is not too old); Secondly, providing clients with a *self-consistent* view of the application's state as it changes (i.e., once the client has been told that something has happened, that client should never be told anything to the contrary).

The web does not behave like a distributed file system (DFS) or distributed shared memory (DSM) system; among the dissimilarities are: (1) the lack of a *write* semantic in common use - while the HTTP protocol does include a *PUT* event which is in some ways comparable to a write, it is rarely used. The most common write-like operation is *POST* which can have completely arbitrary semantics and scope. This generality implies, in the general case, an inability to *batch* user induced updates. (2) The complexity of addressing particular content - URLs or *web addresses* do not in fact address units of contents *per se*, but rather address generic objects (*resources*) which produce content using completely opaque processes. (3) The absence of any protocol-layer persistent state or notion of *transactions* to identify related, batched or macro-operations. These issues are further illuminated by Mogul in [60].

In a DSM or DFS world, the mapping from write events to eventual changes in the *canonical* system state is clearly defined. In the web, non-safe requests from users can have arbitrary application-defined semantics with arbitrary scopes of effect completely unknowable from the parameters of a request, or even from the properties of a response. For this reason, the definitions of consistency and coherence used in the DFS/DSM literature do not fit the needs of systems like the data-center; instead, we use definitions more akin to those in the distributed database literature.

Depending on the type of data being considered, it is necessary to provide certain guarantees with respect to the view of the data that each node in the data-center and the users get. These constraints on the view of data vary depending on the application requiring the data.

Consistency: Cache consistency refers to a property of the responses produced by a single logical cache, such that no response served from the cache will reflect older state of the server than that reflected by previously served responses, i.e., a consistent cache provides its clients with non-decreasing views of the server's state.

Coherence: Cache coherence refers to the average *staleness* of the documents present in the cache, i.e., the time elapsed between the current time and the time of the last update of the document in the back-end. A cache is said to be strong coherent if its average *staleness* is *zero*, i.e., a client would get the same response whether a request is answered from cache or from the back-end.

2.3.1 Web Cache Consistency

In a multi-tier data-center environment many nodes can access data at the same time (*concurrency*). Data consistency provides each user with a consistent view of the data, including all visible (committed) changes made by the user's own updates and the updates of other users. That is, either all the nodes see a completed update or no node sees an update. Hence, for strong consistency, stale view of data is permissible, but partially updated view is not.

Several different levels of consistency are used based on the nature of data being used and its consistency requirements. For example, for a web site that reports football scores, it may be acceptable for one user to see a score, different from the scores as seen by some other users, within some frame of time. There are a number of methods to implement this kind of weak or lazy consistency models.

The *Time-to-Live (TTL)* approach, also known as the Δ-*consistency* approach, proposed with the HTTP/1.1 specification, is a popular weak consistency (and weak coherence) model currently being used. This approach associates a *TTL* period with each cached document. On a request for this document from the client, the front-end node is allowed to reply back from their cache as long as they are within this *TTL* period, i.e., before the *TTL* period expires. This guarantees that document cannot be more *stale* than that specified by the *TTL* period, i.e., this approach guarantees that staleness of the documents is bounded by the *TTL* value specified.

Researchers have proposed several variations of the *TTL* approach including *Adaptive TTL* [30] and *MONARCH* [58] to allow either dynamically varying *TTL* values (as in *Adaptive TTL*) or document category based *TTL* classification (as in *MONARCH*). There has also been considerable amount of work on *Strong Consistency* algorithms [19, 18].

2.3.2 Web Cache Coherence

Typically, when a request arrives at the proxy node, the cache is first checked to determine whether the file was previously requested and cached. If it is, it is considered a cache hit and the user is served with the cached file. Otherwise the request is forwarded to its corresponding server in the back-end of the data-center.

The maximal hit ratio in proxy caches is about 50% [76]. Majority of the cache misses are primarily due to the dynamic nature of web requests. Caching dynamic content pages is much more challenging than static content because the cached object is related to data at the back-end tiers. This data may be updated, thus invalidating the cached object and resulting in a cache miss. The problem of how to provide consistent caching for dynamic content has been well studied and researchers have proposed several weak as well as strong cache consistency algorithms [19, 18, 84]. However, the problem of maintaining cache coherence has not been studied as much.

There are two popularly used coherency models in the current web: *immediate or strong coherence* and *bounded staleness.*

The *bounded staleness* approach is similar to the previously discussed *TTL* based approach. Though this approach is efficient with respect to the number of cache hits, etc., it only provides a weak cache coherence model. On the other hand, *immediate coherence* provides a strong cache coherence.

With *immediate coherence*, caches are forbidden from returning a response other than that which would be returned were the origin server contacted. This guarantees semantic transparency, provides *Strong Cache Coherence*, and as a side-effect also guarantees *Strong Cache Consistency.* There are two widely used approaches to support *immediate coherence.* The first approach is pre-expiring all entities (forcing all caches to re-validate with the origin server on every request). This scheme is similar to a no-cache scheme. The second approach, known as *client-polling*, requires the front-end nodes to inquire from the back-end server if its cache is valid on every cache hit. This cuts down on the cost of transferring the file to the front end on every request even in cases when it had not been updated.

The no-caching approach to maintain *immediate coherence* has several disadvantages:

- Each request has to be processed at the home node tier, ruling out any caching at the other tiers
- The propagation of these requests to the back-end home node over traditional protocols can be very expensive

- For data which does not change frequently, the amount of computation and communication overhead incurred to maintain strong coherence could be very high, requiring more resources

These disadvantages are overcome to some extent by the *client-polling* mechanism. In this approach, the proxy server, on getting a request, checks its local cache for the availability of the required document. If it is not found, the request is forwarded to the appropriate application server in the inner tier and there is no cache coherence issue involved at this tier. If the data is found in the cache, the proxy server checks the *coherence status* of the cached object by contacting the back-end server(s). If there were updates made to the dependent data, the cached document is discarded and the request is forwarded to the application server tier for processing. The updated object is now cached for future use. Even though this method involves contacting the back-end for every request, it benefits from the fact that the actual data processing and data transfer is only required when the data is updated at the back-end. This scheme can potentially have significant benefits when the back-end data is not updated very frequently. However, this scheme has its own set of disadvantages, mainly based on the traditional networking protocols:

- Every data document is typically associated with a home-node in the data-center back-end. Frequent accesses to a document can result in all the front-end nodes sending in *coherence status* requests to the same nodes potentially forming a *hot-spot* at this node.

- Traditional protocols require the back-end nodes to be interrupted for every cache validation event generated by the front-end.

CHAPTER 3

PROBLEM STATEMENT

Cluster-based architectures have been envisioned as one of the main platforms for handling large scale web applications and their massive requirements. With significant increase in volume of data, number of users and application complexity, the performance and scalability of such systems has become critical.

Modern interconnects like InfiniBand and 10 Gigabit Ethernet have introduced a range of novel features while delivering excellent performance. Increasing number of cluster-based datacenters are being deployed with these modern interconnects due to their high performance-to-cost ratios. Further, due to the added WAN capabilities of these high performance networks, datacenter deployments in cluster-of-cluster scenarios is becoming a promising option. However, the extent to which the current deployments manage to benefit from these interconnects is often far below the achievable levels.

In order to maximize the potential benefits that the advanced capabilities of these modern interconnects can deliver, selective redesigning of performance critical datacenter components is needed. Any such redesigning will need to take the typical characteristics of datacenter applications into account. Performance critical operations common to multiple datacenter applications like caching, resource management, etc. need to be identified and redesigned utilizing the features of modern interconnects. These operations can be designed and implemented as system services in a consolidated framework such that all other datacenter applications and services can utilize them effectively. So the main challenge that we address in this thesis is:

> *How can we effectively utilize the novel capabilities of modern interconnects to design efficient and scalable services for cluster-based datacenters and demonstrate the potential performance benefits thereof?*

3.1 Open Challenges and Issues

Traditionally datacenters include applications that are very diverse in nature. Not only it is common that each datacenter typically has multiple applications running concurrently to provide the required services, but also the kinds of services that these applications provide vary a lot. Several aspects ranging from the type of workloads, the amount of data handled, the level of security required, interaction with legacy codes and systems, concurrency of operations, QoS requirements, etc., all significantly vary from deployment to deployment. The datacenter workloads themselves can vary in terms of complexity of requests, rate at which requests arrive, size of responses, etc. making the datacenters a very diverse environment. Further, many of the legacy applications have been developed over time and due to the sheer volume of such legacy application code, it is not practical to undertake any large scale redesigning.

While many of the characteristics of datacenters differ among the different deployments, there are two main similarities as well: (i) the applications are typically based on TCP/IP for all communications and (ii) the applications are largely multi-threaded and usually have a significant number of threads running. These aspects have had a tremendous influence on traditional datacenter application designs and based on the current trends they are likely to do so in future.

While modern interconnects provide a range of novel features with high performance, the above mentioned requirements of traditional datacenters present a very difficult set of constraints that need to be addressed in order to efficiently utilize the capabilities of high performance interconnects. In this context, we plan to design a set of common datacenter services using a multi-layered approach providing a common framework for leveraging the benefits of modern networks to deliver higher datacenter performances.

To present our case, we identify dynamic and cooperative caching, large scale data transfers, distributed lock management and global memory aggregation as a set of distributed datacenter services with high performance requirements and have wide utility in datacenters scenarios. These services as described below have high performance requirements and have wide applicability in modern datacenters. Common applications like web-servers, data-base servers, file-systems, resource managers, etc. can benefit significantly from improved performance of these ubiquitous datacenter services.

- Caching has become an essential component of current datacenters. Currently, web and proxy servers, file-systems, data-bases, etc. all routinely perform caching where possible, implementing their own specific solutions for this purpose. Due to the performance costs associated with traditional design approaches dynamically changing data is usually either not cached or cached to a very limited extent (such as in the case of data-bases). Further, such caching is often done independently on each node leading to a large scale fragmentation of caching resources. Addressing these caching requirements is a considerable challenge.

- Distributed lock management forms an important component with very high performance requirements in many applications due to the fact that locking operations are often in the critical path of application logic. i.e. locks are usually in place blocking and unblocking the progress of the application. Applications sharing data or resources in a distributed scenario need the services of a distributed lock manager. Examples of such scenarios include, cooperative caching, resource management, distributed data-bases, etc. Since these operations are most often used in critical execution paths high performance is of vital importance.

- Due to the ubiquitous nature of large scale data transfers in datacenters, they remain critical to the overall performance of these systems. Operations such as data staging, data backup, replication, etc. all require efficient mechanisms for transferring large amounts of data, often across WAN links. Further, it is highly desirable to achieve good performance for such data transfers with minimal overhead on the end nodes.

- Memory requirements of various nodes in datacenters can vary significantly based on the application hosted on the node and over time. Effective pooling of such memory resources can benefit applications with large memory requirements. In particular, components like remote memory swap devices and cooperative caching can utilize such memory to boost the performances of higher level applications. Pooling of available memory at the lowest level by means of a global memory aggregating service can lead to the most effective consolidation strategies across a datacenter.

While several different designs for these services have been proposed in the past, redesigning these efficiently in light of the capabilities of modern interconnects and technologies is a challenging task and providing them as a part of system-level services is very valuable to higher level applications. Further, exploring the design space and understanding the trade-offs involved in this context is extremely important for the design of the next generation Web applications and datacenter architectures.

3.2 Objectives

In order to address the issues listed above and to demonstrate the benefits of the novel capabilities of modern interconnects in the context of current and next generation datacenters, we propose a multi-layered set of datacenter services utilizing the features of modern interconnects. We divide the main objectives involved into the following:

- Can we design efficient distributed locking services leveraging the RDMA and remote atomic operations provided by InfiniBand? A lot of datacenter applications utilize both shared and exclusive locking modes very frequently. Designing an efficient system service with fair load distribution, better load resilience and minimal costs in the critical path is extremely desired.

  In addition, can we design an efficient all purpose global memory aggregator? Also, how efficiently can we pool memory resources across the datacenter such that free resources can be most effectively consolidated? What kind of allocation and placement policies would such an aggregation of memory resources require for maximum utility of such a memory pool and can the overheads of such an aggregation be minimized on other applications? Finally, can we provide the proposed services in a scalable manner?

- Can we design an efficient coherency protocol for caches that can support strong cache coherency while delivering a high performance and high load resiliency? How effectively can this protocol be designed in the context of modern interconnects and how significant would the quantitative benefits of such a design be?

  Further, the basic dynamic content caching protocol designs need to be extended to scenarios with complex data dependencies. For example, modern dynamic web-pages are usually made up of multiple smaller dynamic components. Can we leverage the higher performance and advanced features that modern interconnects provide to redesign protocols that can handle the increased complexity of modern cache entities that include multiple dynamic sub-objects?

  In addition, can the basic caching and cooperative caching protocols be extended to RDMA based modern interconnects such that the effective total memory usage be improved significantly? Based on the trade-offs involved, what kind of policies and schemes be supported such that performance benefits are maximized?

  Further, what are the scalability implication of such designs? How effectively can these be deployed in large scale datacenter scenarios?

- Can we deign an advanced data transfer service that achieves efficient large scale data transfers across LAN and WAN links? What are the fundamental issues involved in optimizing such data transfers? Can we design such high performance capabilities while minimizing the overheads on the host systems? What are the fundamental trade-offs in this context?

  With the increasing number of cluster-of-cluster scenarios, can the above designs be extended to WAN based protocols? What are the trade-offs involved? What kind of new flow control and communication protocols are beneficial for managing efficient performance on these WAN links? What are the fundamental limitations?

We intend to address the above challenges and design these components in a layered architecture such that applications can leverage the benefits of these services through pre-defined interfaces. Further, the services themselves would be able to benefit from other services provided.

3.3 Proposed Framework

Figure 3.1 provides an overview of all the above mentioned components. In the figure, the white-shaded components form existing datacenters. The components we proposed in this thesis are dark shaded. We divide the services into higher-layer services including (i) Dynamic Content Caching, (ii) Multi-Dependency Dynamic Content Caching and (iii) Cooperative Caching and lower-layer services including (i) Distributed Lock Manager, (ii) Global Memory Aggregator and (iii) Advanced Data Transfer Service. These services are designed in the context of modern interconnects.

The WAN framework extends the InfiniBand and 10GE/iWARP capabilities (and services designed using them) across longer WAN links.

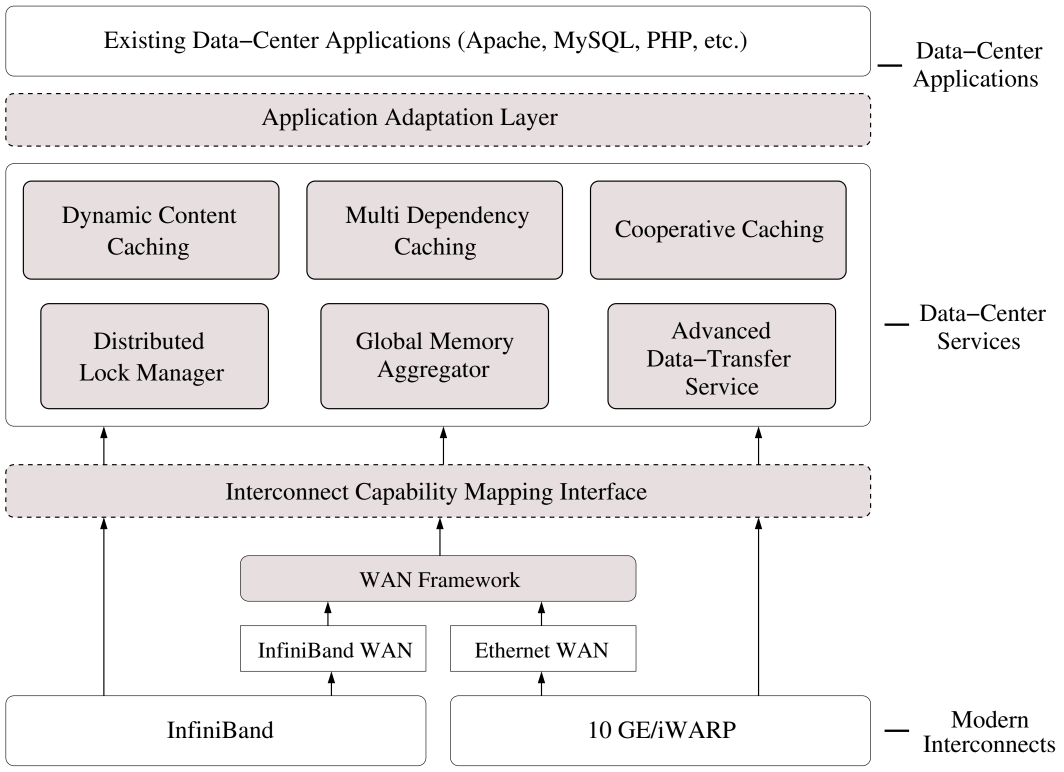


Figure 3.1: Proposed Framework for Datacenter Services over Modern Interconnects

Applications invoke these services through an application adaptation layer that traps service requests into the service layers and returns responses to the applications. This applications adaptation layer is intended to provide access to the services while keeping the application changes minimal. The Interconnect Capability Mapping Interface provides a uniform mechanism of access to network capabilities for the proposed services.

3.4 Dissertation Overview

In this section, we present the overview of the dissertation. Detailed designs and evaluations are presented in the following chapters.

In Chapter 4, we present a novel approach to efficient distributed lock management. We utilize the remote atomic operations provided by InfiniBand to provide one-sided mechanisms to enable both shared mode locking and exclusive mode locking. We discuss limitations of existing approaches and the benefits of our designs. By distributing the lock management related workload appropriately, we achieve fair load balancing among participating nodes. Further, we demonstrate the advantages of our design with detailed evaluation.

In Chapter 5, we present a highly efficient mechanism for dynamic content caching. We extensively utilize RDMA Read operations to obtain cache validity information in real-time to enable strong cache coherence required for several modern applications. Our results demonstrate an order of magnitude improvement of performance over existing approaches. Further, we also demonstrate excellent scalability of our approach under loaded conditions that datacenters often encounter.

In Chapter 6, we present an extended approach for dynamic content caching in order to handle complex web-objects. Modern applications often generate complex objects with multiple dynamic components. We discuss the issues involved in providing strong cache coherency for such complex objects. We present an extended protocol that enables such capability for current and next-generation datacenter applications.

In Chapter 7, we study the various alternatives for enabling cooperative caching of web objects. In particular, we discuss several optimizations to maximize system resource usage and overall performance. We evaluate the effect of selective automated replication of cached data to achieve maximum performance benefits. Further, we explore the possibility of utilizing system-wide free resources for caching in terms of possible benefits and involved overheads.

In Chapter 8, we present an evaluation framework of high performance communications over WAN. We study the various mechanisms for emulating WAN characteristics for performing accurate higher level evaluations. We study the performance characteristics of both IB WAN and iWARP using a multitude of hardware and software environments. Based on our analysis of such environments, we isolate the benefits and overheads of high performance protocols when used over WAN.

In Chapter 9, we present the design of high performance data transfer capabilities across IB WAN. We explore the possibility of using zero-copy communications, including RDMA, for WAN data transfers. We study the tradeoffs involved in such communications.

In Chapter 10, we present our conclusions and describe topics for future research.

CHAPTER 4

NETWORK-BASED DISTRIBUTED LOCK MANAGER

Effective cooperation among the multiple processes distributed across the nodes is needed in a typical data-center environment where common pools of data and resources like files, memory, CPU, etc. are shared across multiple processes. This requirement is even more pronounced for clusters spanning several thousands of nodes. Highly efficient distributed locking services are imperative for such clustered environments. In this chapter, we present our novel network based distributed locking protocol.

4.1 Background and Related Work

While traditional locking approaches provide basic mechanisms for this cooperation, high performance, load resiliency and good distribution of lock management workload are key issues that need immediate addressing. Existing approaches [35, 10, 34, 49, 57] handle these requirements either by distributing the per-lock workload (i.e. one server manages all operations for a predefined set of locks) and/or by distributing each individual lock's workload (i.e. a group of servers share the workload by distributing the queue management for the locks). While the former is popularly used to distribute load, it is limited to a high granularity of workload distribution.

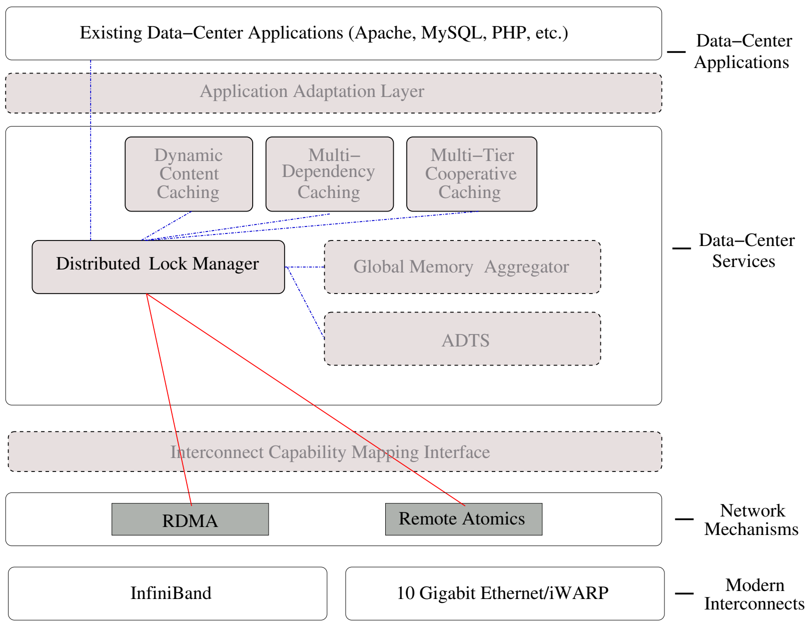


Figure 4.1: Proposed Distributed Lock Manager

Further, some locks can have significantly higher workload as compared to others and thereby possibly causing an unbalanced overall load.

The second approach of distributed queue management has been proposed by researchers for load-sharing fairness and better distribution of workload. In such approaches, employing two-sided communication protocols in data-center environments is inefficient as shown by our earlier studies [62]. Devulapalli. et. al. [35], have proposed a distributed queue based locking protocol which avoids two-sided communication operations in the locking critical path. This approach only supports locking of resources in exclusive access mode. However, supporting all popular resource access patterns needs two modes of locking: (i) Exclusive mode locking and (ii) Shared mode locking. Lack of efficient support for shared mode locking precludes the use of these locking services in common high performance data-center scenarios like multiple concurrent readers for a file (in file system caching), or multiple concurrent readers for a data-base table, etc. Hence the distributed lock management needs to be designed taking into account all of these issues.

In the rest of the chapter, we describe the various design aspects of our RMA based complete DLM locking services. Section 4.1.1 describes the common implementation framework for our system. Section 4.2 describes the design details of our locking designs.

4.1.1 External Module-based Design

The DLM works in a client-server model to provide locking services. In our design we have the DLM server daemons running on all the nodes in the cluster. These daemons coordinate over InfiniBand using OpenFabrics Gen2 interface [43] to provide the required functionality. Figure 4.2 shows the basic setup on each node. The applications (i.e. clients) contact their local daemons using IPC message queues to make lock/unlock requests. These requests are processed by the local daemons and the response is sent back to the application appropriately. Since typical data-center applications have multiple (often transient) threads and processes running on each node, this approach of having one DLM server daemon on each node provides optimal sharing of DLM resources while providing good performance. These DLM processes are assigned rank ids (starting from one) based on their order of joining the DLM group.

The DLM maintains the information on each lock with an associated key. These keys and related lock information is partitioned among the participating nodes; i.e. each key has a *homenode* that represents the default location of the locking state

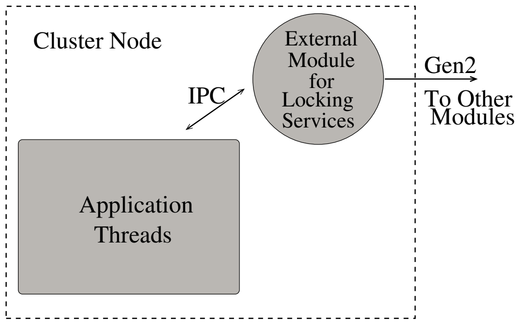


Figure 4.2: External Module-based Services

information for that lock (and the keys themselves are randomly distributed among all the nodes).

In order to support these operations, we have three threads in each of our design: (i) Inter-node communication thread, (ii) IPC thread and (ii) Heartbeat thread. The inter-node communication thread blocks on gen2-level receive calls. The IPC thread performs the majority of the work. It receives IPC messages from application processes (lock/unlock requests) and it also receives messages from the other threads as needed. The heartbeat thread is responsible for maintaining the work queues on each node. This thread can also be extended to facilitate deadlock detection and recovery. This issue is orthogonal to our current scope and is not dealt in the current research.

In our design we use one-sided RDMA atomics in the critical locking path. Further, we distribute the locking workload among the nodes involved in the locking operations. Hence our design maintains basic fairness among the cluster nodes.

4.2 Design of the Proposed Network-based Distributed Lock Manager

In this section, we describe the various aspects of our high performance design for providing shared and exclusive locking using network based atomic operations. In particular, we provide the details of the various protocols and data-structures we use in order to accomplish this. This section is organized as the following. First, we explain the organization of the data-structures used in protocols. We then explain the Network-based Combined Shared/Exclusive Distributed Lock Design (N-CoShED) protocol proposed in this chapter.

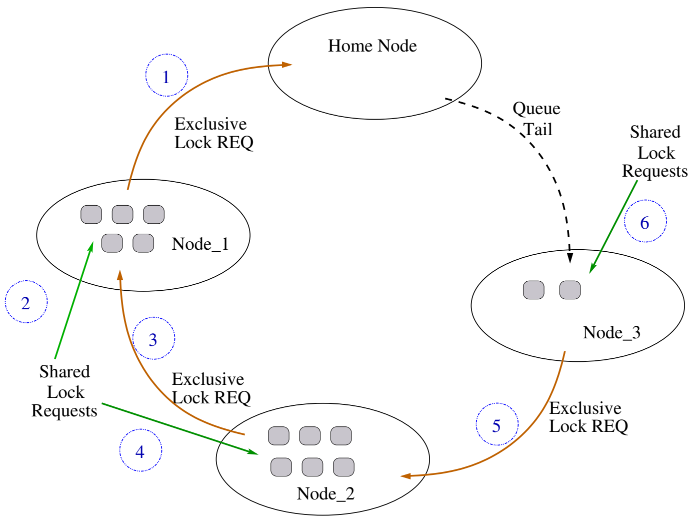


Figure 4.3: An Example Scenario of the DLM Protocol - N-CoSED

Global Shared Data-Structures: The primary data element used in our proposed DLM design is a 64-bit value. The required attributes of this value is that it should be globally visible and accessible (i.e. RDMA Atomics are enabled on this memory field) by all the participating processes. Each 64-bit value used for lock management is divided equally into two regions: Exclusive region and Shared region, each making up 32-bits. These fields are initialized to zero at the start and the details of the the usage are described in the following subsections.

We now explain the combined distributed locking protocol for shared and exclusive locks. To simplify understanding, we break this protocol into four broad cases:(i) Only Exclusive locks are issued, (ii) Only Shared locks are issued, (iii) Exclusive locks are issued following Shared locks and (iv) Shared locks are issued following Exclusive locks.

Figure 4.3 shows a sample snapshot of the state of the distributed queue for locks in our design. The circled numbers label the lock request arrows to show the order in which the queue locks are granted. The three nodes shown have exclusive lock requests and each of them have a few shared lock requests queued that will be granted after they are done with the exclusive lock.

4.2.1 Exclusive Locking Protocol

In this section we outline the locking and unlocking procedures when only exclusive locks are issued. As explained above a 64-bit value (on the home node) is used for each lock in the protocol. For exclusive locking, only the first 32 bits of the 64-bit value are used. The following steps detail the exclusive lock/unlock operation. Figure 4.4(a) shows an example of this case.

- **Locking Protocol:**

  Step 1. To acquire the lock the requesting client process issues an atomic compare-and-swap operation to the home node. In this operation, two values

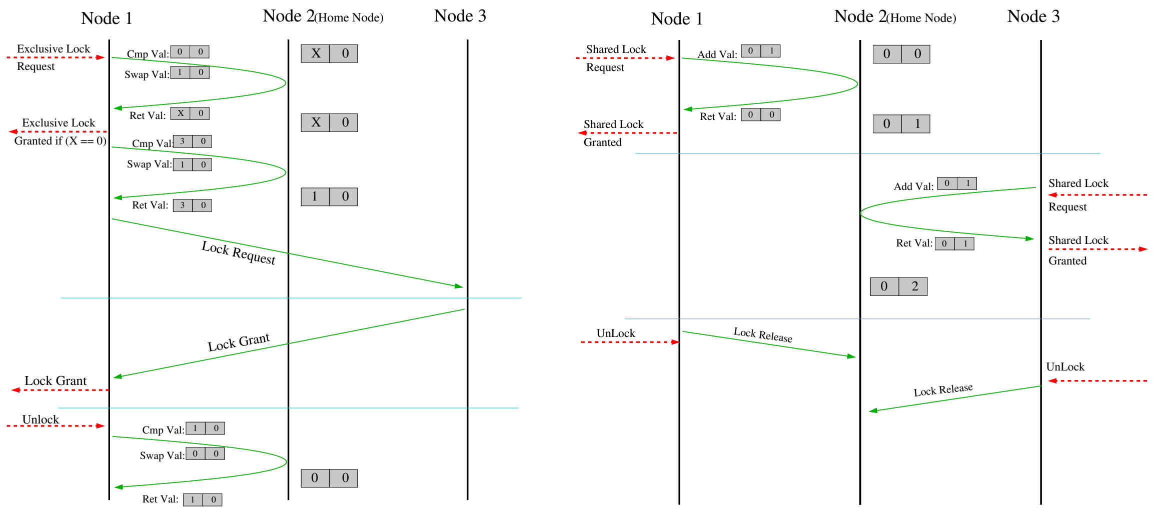


Figure 4.4: Locking protocols: (a) Exclusive only (b) Shared Only

are provided by this process, the swap value and the compare value. The swap value is a 64-bit value whose first 32 bits correspond to the rank of the issuing process and the next 32 bits are zeros $[rank : 0]$. The compare value $[0 : 0]$ is passed for comparison with the value at the home node. If this value equals the value at the home node, the compare operation succeeds and the value at the home node is swapped with the supplied swap value. If the comparison fails then the swapping does not take place. The issuing process is returned with the original 64-bit value of the home node after the atomic operation completes.

Step 2. If the exclusive region of the returned value corresponds to zero, it indicates that no process is currently owning the lock. The process can safely acquire the lock in this circumstance.

Step 3. If the value is not zero, then the exclusive region of the returned value corresponds to the rank of the process at the end of the distributed queue waiting for the lock. In this case, the issued atomic comparison would have failed and the entire atomic operation has to retried. However, this time the exclusive region of the compare value [*current_tail* : 0] is set to the rank of the last process waiting in the queue. Once the atomic operation succeeds, the local DLM process sends a separate lock request message (using Send/Recv) to the last process waiting for the lock. The rank of this process can be extracted from the 64-bit returned value of the atomic operation. This approach is largely adequate for performance reasons since this operation is not in critical path.

- **Unlocking Protocol:**

  Step 1. After the process finishes up with the lock, it checks whether it has any pending requests received from other processes. If there is a pending lock request, it sends a message to this process indicating that it can go ahead and acquire the lock. This process is the next in the distributed queue waiting for the lock.

  Step 2. If there are no pending lock requests, the given process is the last in the queue and it resets the 64-bit value at the home-node to zero for both the exclusive and shared regions.

4.2.2 Shared Locking Protocol

In this section we explain the protocol steps when only requests for the shared lock are issued. In this part of the protocol, the shared region portion of the 64-bit value is employed which makes up the last 32 bits. The basic principle employed is that the shared region is atomically incremented using Fetch-and-Add operation every time a shared lock request arrives at the home node. Thus, at any given time the count in the shared region represents the number of shared lock requests arrived at the home node. The following are the detailed steps involved.

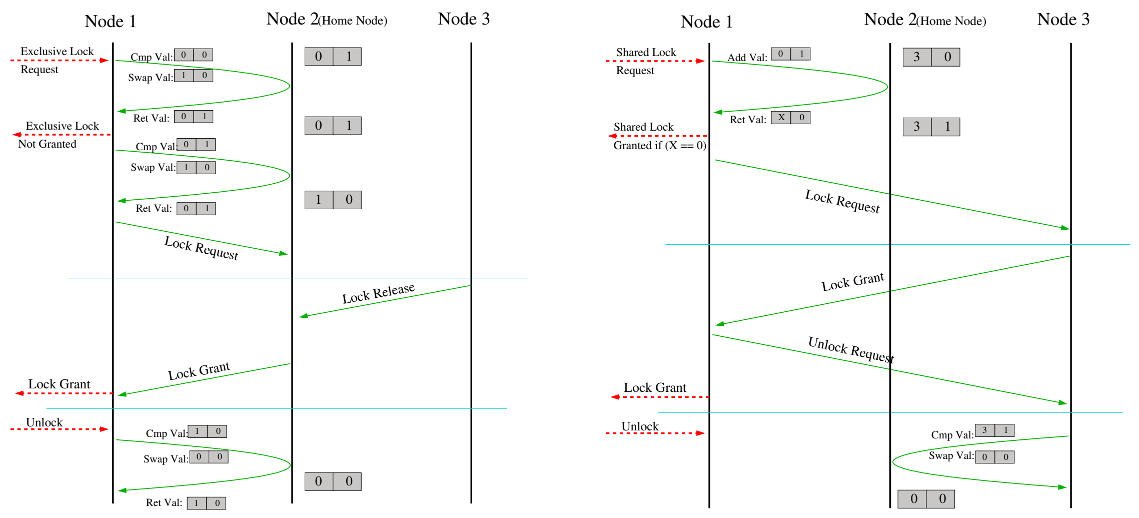


Figure 4.5: Locking protocols: (a) Shared followed by Exclusive (b) Exclusive followed by Shared

- **Locking Protocol:**

  Step 1. The process acquiring the shared lock initiates an atomic fetch-and-add increment operation on the 64-bit value at the home node. Please note that in effect, the operation is performed on the shared region of the value. The first 32 bits are not modified.

  Step 2. If the exclusive portion of the returned value corresponds to zero then the shared lock can be safely acquired.

  Step 3. If the exclusive portion of the returned value contains a non-zero value, it implies that some other process has issued an exclusive lock request prior to the shared lock request on the lines of the exclusive locking protocol described earlier. We explain this scenario in detail in the following sections.

- **Unlocking Protocol:**

  Step 1. The process after acquiring the shared lock issues a lock release message to the home node.

  Step 2. Once all the lock release messages from the shared lock owners have arrived, the shared region is re-set to zero atomically by the home node.

4.2.3 Shared Locking followed by Exclusive locking

We now outline the steps when an exclusive lock request arrives after the shared locks have been issued. In this case, the value at the home node reads the following. The first 32 bits corresponding to the exclusive portion would be zero followed by the next 32 bits which contain the count of the shared locks issued so far. The process acquiring the exclusive lock issues an atomic compare-and-swap operation on the 64-bit value at the home node as described in the above exclusive protocol section. The following steps occur during the operation. Figure 4.5(a) shows the basic steps.

- **Locking Protocol:**

  Step 1. Similar to the exclusive locking protocol, the issuing client process initiates an atomic compare-and-swap operation with the home node. Since shared locks have been issued, the atomic operation fails for this request. This is because the value in the home node does not match with the compare value supplied which is equal to zero. The atomic operation is retried with the new compare value set to the returned value of the previous operation.

Step 2. Once the retried atomic operation succeeds, the 64-bit value at the home node is swapped with a new value where the shared region is re-set to zero and the exclusive region contains rank of the current issuing process.

Step 3. The issuing process then gets the number of shared locks issued so far from the last 32 bits of the returned value. It also obtains the value of the first 32 bits which is the exclusive region. In our case, since we are assuming that only shared locks have been issued so far this value is zero. It then sends an exclusive lock acquire message to the home node. It also sends the count of the number of shared locks to this process. This count helps the home node keep track of the shared locks issued so far and hence needs to wait for all these unlock messages before forwarding the lock to the node requesting the exclusive lock.

Step 4. The exclusive lock is acquired only when the home node process receives the shared lock release messages from all the outstanding shared lock holders in which case it grants the exclusive lock request.

The case of subsequent exclusive lock requests is the same as described in the exclusive locking protocol section outlined above. Unlock procedures are similar to the earlier cases.

4.2.4 Exclusive Locking followed by Shared Locking

The following are the sequence of operations when shared locks are issued after exclusive locks. Figure 4.5(b) shows an example scenario.

- **Locking Protocol:**

Step 1. The issuing client process initiates a fetch-and-add atomic operation in the same fashion described in the locking protocol for shared locks. However, the value of exclusive region in the returned value may not match with the rank of the home process. This is because the exclusive region contains the rank of the last process waiting for exclusive lock in the queue.

Step 2. The shared lock requests are sent to the last process waiting for the exclusive lock. This is obtained from the exclusive portion of the returned value of Fetch-and-Add operation.

Step 3. The shared lock is granted only after the last process waiting for the exclusive lock is finished with the lock.

The same procedure is followed for any shared lock issued after the exclusive locks.

4.3 Experimental Evaluation

In this section, we present an in-depth experimental evaluation of our Network-based Combined Shared/Exclusive Distributed Lock Management (N-CoSED). We compare our results with existing algorithms (i) Send/Receive-based Centralized Server Locking (SRSL) and (ii) Distributed Queue-based Non-Shared Locking (DQNL) [35].

All these designs are implemented over InfiniBand's OpenFabrics-Gen2 interface [43]. Message exchange was implemented over IBA's Send/Receive primitives. The one-sided RDMA operations were used (*compare-and-swap* and *fetch-and-add*) for all the one-sided operations for DQNL and N-CoSED.

For our experiments we used the a 32-node Intel Xeon cluster. Each node of our testbed has two 3.6 GHz Intel processor and 2 GB main memory. The CPUs

Primitive	Polling (us)	Notification (us)
Send/Recv	4.07	11.18
RDMA CS	5.78	12.97
RDMA FA	5.77	12.96

Table 4.1: Communication Primitives: Latency

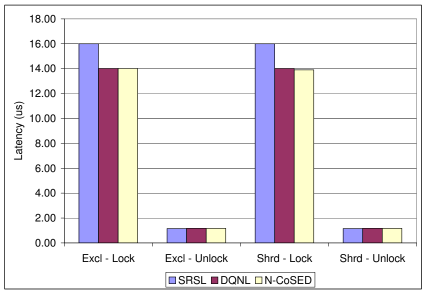
support the EM64T technology and run in 64 bit mode. The nodes are equipped with MT25208 HCAs with PCI Express interfaces. A Flextronics 144-port DDR switch is used to connect all the nodes. OFED 1.1.1 software distribution was used.

4.3.1 Microbenchmarks

The basic latencies observed for each of the InfiniBand's primitives used in our experiments are shown in Table 4.1 . The latencies of each of these is measured in polling and notification mode. The three primitives shown are *send/recv*, RDMA *compare-and-swap* (RDMA CS) and RDMA *fetch-and-add* (RDMA FA). For the *send/recv* operation we have used a message size of 128 bytes.

As clearly seen from the numbers, the polling approach leads to significantly better latencies. However, the polling-based techniques consume many CPU cycles and hence are not suitable in typical clustered data-center scenarios.

In addition to network based primitives, a DLM needs an intra-node messaging primitive as explained in Section 4.1.1. In our experiments we use System V IPC message queues. The choice is orthogonal to our current scope of research. We observe a latency of 2.9 microseconds for communicating with IPC message queues for a 64 byte message. The cost for initiating such a request is observed to be 1.1 microseconds. It is to be noted that while the network primitives operate in both



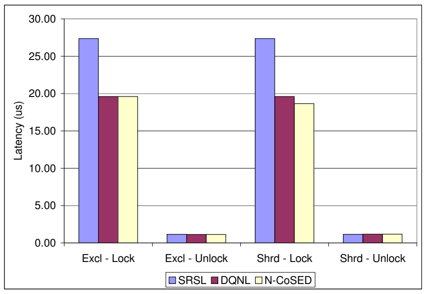


Figure 4.6: Basic Locking Operations' Latency: (a) Polling (b) Notification

polling and notification mode, the intra-node messaging is used only in notification mode. This is because multiple processes that require locking services usually exist on a single node and the situation of having all of these processes polling practically block the node from doing any useful computation and needs to be avoided.

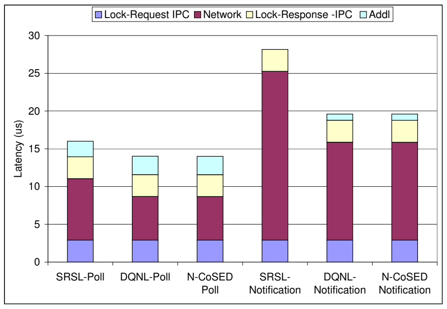


Figure 4.7: Timing breakup of lock operations

All the locking mechanisms dealing with the distributed locking service daemon, the total lock/unlock latency is divided into two parts: (i) the intra-node messaging latency and (ii) the lock wait + network messaging. While various distributed locking schemes differ significantly in the second component, the first component is usually

Scheme	Lock	Unlock
SRSL	$2 * T_{Send} + 2 * T_{IPC}$	$T_{IPC-Initiate}$
DQNL	$T_{RDMAAtomic} + 2 * T_{IPC}$	$T_{IPC-Initiate}$
N-CoSED	$T_{RDMAAtomic} + 2 * T_{IPC}$	$T_{IPC-Initiate}$

Table 4.2: Cost Models

common to all the different designs and hence can be eliminated for the sake of comparing the performance across different designs.

4.3.2 Detailed Performance Evaluation

In this section, we compare the locking performance of the various lock/unlock operations involved across the three schemes: SRSL, DQNL and N-CoSED.

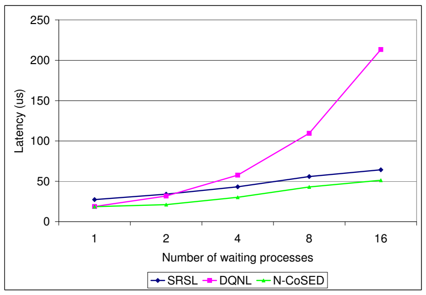
Figure 4.6 shows the average latency of the basic locking operations as seen by the applications. In this experiment, we have only one outstanding lock/unlock request (serial locking) for a given resource at any given point of time. Both polling and notification modes are shown.

As seen in Figure 4.6(a) for polling based latencies, basic locking latency for SRSL is 16.0 microseconds and is identical for shared and exclusive mode locking. This basically includes work related to two Send/Recv messages of IBA, two IPC messages plus book keeping. The DQNL and N-CoSED schemes perform identically for serial locking and show a lower latency of 14.02 microseconds. As shown in Figure 4.7 the main benefit here is from the fact that two network *send/recv* operations are replaced by one RDMA atomic operation.

Figure 4.6(b) shows the same latencies in notification mode. The lock latencies for DQNL and N-CoSED show an average latency of 19.6 microseconds whereas SRSL shows a latency of 27.37 microseconds.

The more interesting aspect to note is that in case of polling based approach the SRSL lock latency is 14% more than the RDMA based DQNL and N-CoSED, while in the notification case the SRSL latency is 39.6% higher than the the RDMA based designs. As shown in Figure 4.7 this higher increase of latency for SRSL is in the network communication part which is due to the fact that it requires notifications for each of the two *send/recv* messages needed for it. On the other hand the RDMA based schemes incur only one notification. Hence the RDMA based schemes offer better basic latencies for locking over two sided schemes.

The basic unlocking latency seen by any process is just about 1.1 microseconds. This is because for unlock operations the applications just initiate the unlock operation by sending the command over messages queues. This actual unlock operation latency is hidden from the process issuing the unlock operation. Table 4.2 shows the cost models for each of the operations.



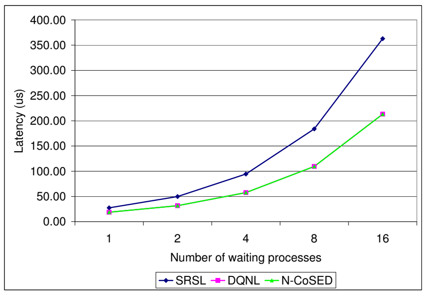


Figure 4.8: Lock Cascading Effect: (a) Shared Lock Cascade (b) Exclusive Lock Cascade

4.3.3 Cascading Unlock/Lock delay

While the basic unlock latency seen by any unlocking process is minimal, the actual cost of this operation is seen by processes next in line waiting for the lock to be issued. This aspect is equally important since this directly impacts the delay seen by all processes waiting for locks. The two extreme cases of locking scenarios are considered: (i) All processes waiting for exclusive locks and (ii) all waiting processes are waiting for shared locks. In this experiment, a number of processes wait in the queue for a lock currently held in exclusive mode, once the lock is released it is propagated and each of the processes in the queue unlock the resource as soon as they are granted the lock. This test intends to measure the propagation delay of these locks.

In Figure 4.8(a), the basic latency seen by DQNL increases at a significantly higher rate as compared to SRSL and N-CoSED. This is due to the fact that DQNL is as such incapable of supporting shared locks and grants these in a serialized manner only, whereas the other two schemes release all the shared locks in one go. It is to be noted that all the shared lock holders are immediately releasing the locks in this test. This effect is heightened when the lock holding time of each of the shared lock holders increases. As compared to N-CoSED, DQNL and SRSL incur 317% and 25% higher latencies, respectively. The difference in SRSL and N-CoSED is the extra message SRSL required from the last lock holder to the home-node server before the release can be made.

The increase in the latency for the N-CoSED scheme for longer wait queues is due to the contention at the local NIC for sending out multiple lock release messages using Send/Recv messages. This problem can be addressed by the use of multicast or other optimized collective communication operations [54].

Figure 4.8(b) captures this lock cascading effect by measuring the net exclusive lock latency seen by a set of processes waiting in a queue. The latency for propagation of exclusive locks is similar for both DQNL and N-CoSED, each of which incurs the cost of one IB Send/Recv message per process in the queue. On the other hand, the SRSL scheme incurs two Send/Recv messages per unlock/lock operation since all the operations have to go the server before they can be forwarded. In all these cases, N-CoSED performs the best.

4.3.4 Benchmark with Shared Locking Trace

In this section we detail our experiment in which we evaluate our shared locking protocol with the world cup caching trace. In this benchmark several participating nodes perform shared locking operations according to the caching trace. We measure the rate of locking with increasing number of participating nodes. Figure 4.9 shows that our mechanism for shared locking can sustain heavy loads simulated with the world cup trace quite easily. The performance scales linearly with number of participating nodes.

4.4 Summary

The massive increase in cluster based computing requirements has necessitated the used of highly efficient DLMs. In this chapter, we have presented a novel distributed locking protocol utilizing the advanced network level one-sided operations provided by InfiniBand. Our approach arguments the existing approaches by eliminating the need for two sided communication protocols in the critical path for locking operations.

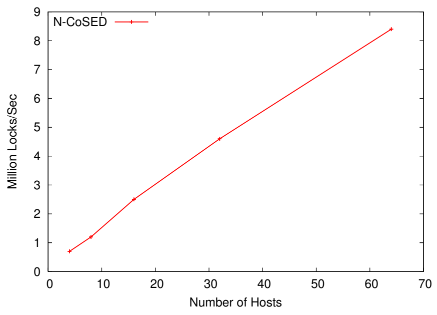


Figure 4.9: Benchmark with Shared Locking Trace

Further, we have also demonstrated that our approach provides significantly higher performance in scenarios needing both shared and exclusive mode access to resources. Since our design distributes the lock management load largely on some of the nodes using the lock, basic fairness is maintained.

Our experimental results have shown that we can achieve 39% better locking latency as compared to basic *send/recv* based locking schemes. In addition, we have also demonstrated that our design provides excellent shared locking support using RDMA FA. This support is not provided by existing RDMA based approaches. In this regard we have demonstrated the performance of our design which can perform an order of magnitude better than the basic RDMA CS based locking proposed is prior approaches [35].

CHAPTER 5

DYNAMIC CONTENT CACHES WITH STRONG CACHE COHERENCY

In this chapter, we describe the architecture we use to support strong cache coherence. We first provide the basic design of the architecture for any generic protocol. Next, we point out several optimizations possible in the design using the various features provided by InfiniBand. Figure 5.1 shows the overview of our approach.

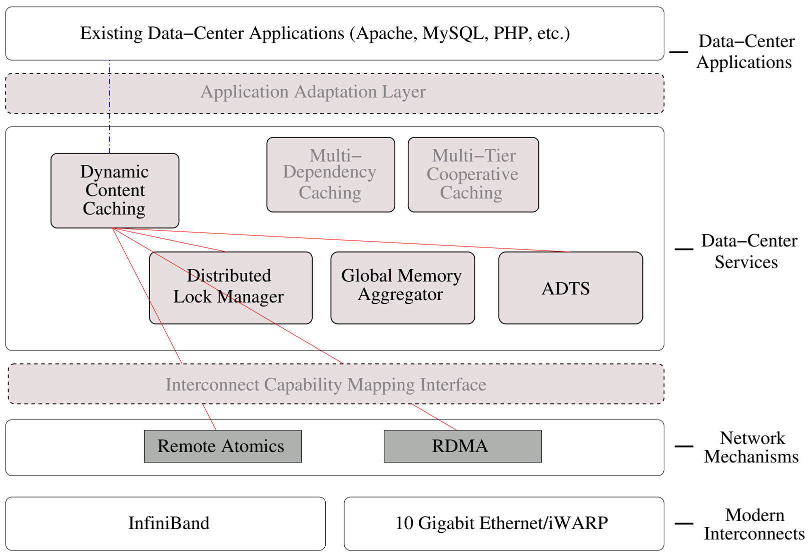


Figure 5.1: Proposed Dynamic Content Caching with Strong Coherency

5.1 Background and Related Work

With ever increasing on-line businesses and services and the growing popularity of personalized Internet services, dynamic content is becoming increasingly common [26, 84, 76]. This includes documents that change upon every access, documents that are query results, documents that embody client-specific information, etc. Large-scale dynamic workloads pose interesting challenges in building the next-generation data-centers [84, 75, 39, 79]. Significant computation and communication may be required to generate and deliver dynamic content. Performance and scalability issues need to be addressed for such workloads.

Reducing computation and communication overhead is crucial to improving the performance and scalability of data-centers. Caching dynamic content, typically known as *Active Caching* [26] at various tiers of a multi-tier data-center is a well known method to reduce the computation and communication overheads. However, it has its own challenges: issues such as cache consistency and cache coherence become more prominent. In the state-of-art data-center environment, these issues are handled based on the type of data being cached. For dynamic data, for which relaxed consistency or coherency is permissible, several methods like TTL [41], Adaptive TTL [30], and Invalidation [51] have been proposed. However, for data like stock quotes or airline reservation, where old quotes or old airline availability values are not acceptable, strong consistency and coherency is essential.

Providing strong consistency and coherency is a necessity for *Active Caching* in many web applications, such as on-line banking and transaction processing. In the current data-center environment, two popular approaches are used. The first approach is pre-expiring all entities (forcing data to be re-fetched from the origin server on every request). This scheme is similar to a no-cache scheme. The second approach, known as *Client-Polling*, requires the front-end nodes to inquire from the back-end server if its cache entry is valid on every cache hit. Both approaches are very costly, increasing the client response time and the processing overhead at the back-end servers. The costs are mainly associated with the high CPU overhead in the traditional network protocols due to memory copy, context switches, and interrupts [75, 39, 15]. Further, the involvement of both sides for communication (two-sided communication) results in performance of these approaches heavily relying on the CPU load on both communication sides. For example, a busy back-end server can slow down the communication required to maintain strong cache coherence significantly.

In this chapter, we focus on leveraging the advanced features provided by Modern Interconnects to support strong coherency for caching dynamic content in a multi-tier data-center environment. In particular, we study mechanisms to take advantage of InfiniBand's features to provide strong cache consistency and coherency with low overhead and to provide scalable dynamic content caching. We detail our designs and demonstrate the obtained benefits in the following sections.

5.2 Design of Dynamic Content Caching

As mentioned earlier, there are two popular approaches to ensure cache coherence: *Client-Polling* and *No-Caching*. In this chapter, we focus on the *Client-Polling* approach to demonstrate the potential benefits of InfiniBand in supporting strong cache coherence.

While the HTTP specification allows a cache-coherent client-polling architecture (by specifying a TTL value of NULL and using the `‘‘get-if-modified-since’’`

HTTP request to perform the polling operation), it has several issues: (1) This scheme is specific to sockets and cannot be used with other programming interfaces such as InfiniBand's native Verbs layers (e.g.: VAPI), (2) In cases where persistent connections are not possible (HTTP/1.0 based requests, secure transactions, etc), connection setup time between the nodes in the data-center environment tends to take up a significant portion of the client response time, especially for small documents.

In the light of these issues, we present an alternative architecture to perform *Client-Polling.* Figure 5.2 demonstrates the basic coherency architecture used in this chapter. The main idea of this architecture is to introduce *external helper modules* that work along with the various servers in the data-center environment to ensure cache coherence. The external module used is described in Section 4.1.1. All issues related to cache coherence are handled by these modules and are obscured from the data-center servers. It is to be noted that the data-center servers require very minimal changes to be compatible with these modules.

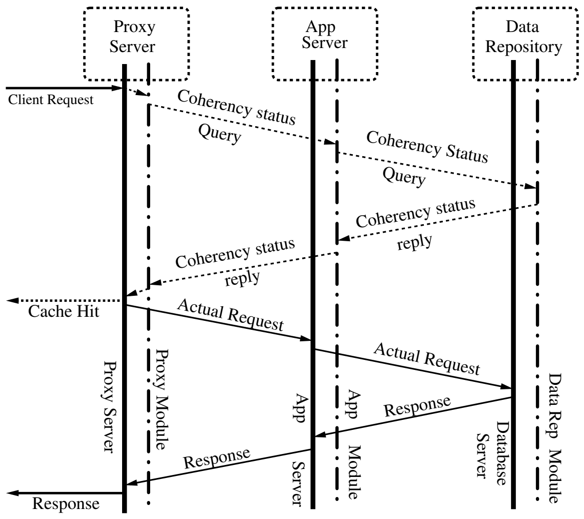


Figure 5.2: Strong Cache Coherence Protocol for Dynamic Content Caches

The design consists of a module on each physical node in the data-center environment associated with the server running on the node, i.e., each proxy node has a proxy module, each application server node has an associated application module, etc. The proxy module assists the proxy server with validation of the cache on every request. The application module, on the other hand, deals with a number of things including: (a) Keeping track of all updates on the documents it owns, (b) Locking appropriate files to allow a multiple-reader-single-writer based access priority to files, (c) Updating the appropriate documents during update requests, (d) Providing the proxy module with the appropriate version number of the requested file, etc. Figure 5.3 demonstrates the functionality of the different modules and their interactions.

Proxy Module

On every request, the proxy server contacts the proxy module through IPC to validate the cached object(s) associated with the request. The proxy module does the actual verification of the document with the application module on the appropriate application server. If the cached value is valid, the proxy server is allowed to proceed by replying to the client's request from cache. If the cache is invalid, the proxy module simply deletes the corresponding cache entry and allows the proxy server to proceed. Since the document is now not in cache, the proxy server contacts the appropriate application server for the document. This ensures that the cache remains coherent.

Application Module

The application module is slightly more complicated than the proxy module. It uses multiple threads to allow both updates and read accesses on the documents in a multiple-reader-single-writer based access pattern. This is handled by having a separate thread for handling updates (referred to as the *update thread* here on). The main thread blocks for IPC requests from both the application server and the update thread. The application server requests to read a file while an update thread requests to update a file. The main thread of the application module, maintains two queues to ensure that the file is not accessed by a writer (update thread) while the application server is reading it (to transmit it to the proxy server) and vice-versa.

On receiving a request from the proxy, the application server contacts the application module through an IPC call requesting for access to the required document (IPC_READ_REQUEST). If there are no ongoing updates to the document, the application module sends back an IPC message giving it access to the document (IPC_READ_PROCEED), and queues the request ID in its *Read Queue (RQ)* . Once the application server is done with reading the document, it sends the application module another IPC message informing it about the end of the access to the document (IPC_READ_DONE). The application module, then deletes the corresponding entry from its *Read Queue*.

When a document is to be updated (either due to an update server interaction or an update query from the user), the update request is handled by the *update thread*. On getting an update request, the update thread initiates an IPC message to the application module (IPC_UPDATE_REQUEST). The application module on seeing this, checks its *Read Queue*. If the *Read Queue* is empty, it immediately sends an IPC message (IPC_UPDATE_PROCEED) to the update thread and queues the request ID in its *Update Queue (UQ)*. On the other hand, if the *Read Queue* is not empty, the update request is still queued in the *Update Queue*, but the IPC_UPDATE_PROCEED message is not sent back to the update thread (forcing it to hold the update), until

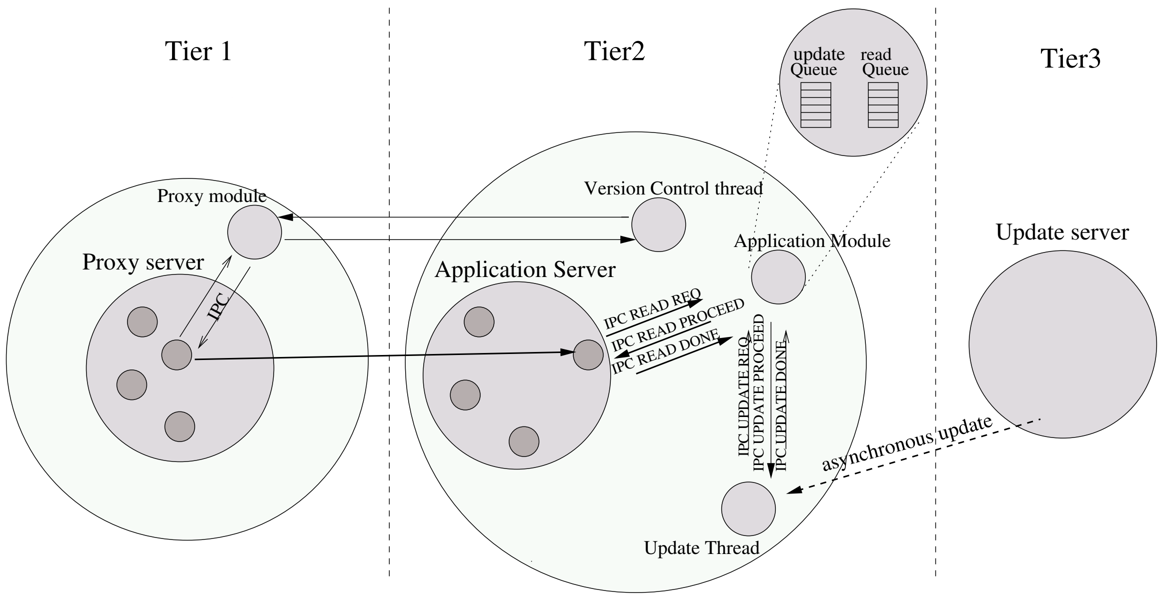


Figure 5.3: Interaction between Data-Center Servers and Caching Coherency Modules

the *Read Queue* becomes empty. In either case, no further read requests from the application server are allowed to proceed; instead the application module queues them in its *Update Queue*, after the update request. Once the update thread has completed the update, it sends an IPC_UPDATE_DONE message to the update module. At this time, the application module deletes the update request entry from its *Update Queue*, sends IPC_READ_PROCEED messages for every read request queued in the *Update Queue* and queues these read requests in the *Read Queue*, to indicate that these are the current readers of the document.

It is to be noted that if the *Update Queue* is not empty, the first request queued will be an update request and all other requests in the queue will be read requests. Further, if the *Read Queue* is empty, the update is currently in progress. Table 5.1 tries to summarize this information. The key for the table is as follows: IRR –

TYPE	RQ State	UQ State	Rule
IRR	E	E	1. Send IPC_READ_PROCEED to proxy 2. Enqueue Read Request in RQ
IRR	NE	E	1. Send IPC_READ_PROCEED to proxy 2. Enqueue Read Request in RQ
IRR	E	NE	1. Enqueue Read Request in UQ
IRR	NE	NE	Enqueue the Read Request in the UQ
IRD	E	NE	Erroneous State. Not Possible.
IRD	NE	E	1. Dequeue one entry from RQ
IRD	NE	NE	1. Dequeue one entry from RQ 2. If RQ is *now* empty, Send IPC_UPDATE_PROCEED to head of UQ
IUR	E	E	1. Enqueue Update Request in UQ 2. Send IPC_UPDATE_PROCEED
IUR	E	NE	Erroneous state. Not Possible
IUR	NE	E	1. Enqueue Update Request in UQ
IUR	NE	NE	Erroneous State. Not possible
IUD	E	E	Erroneous State. Not possible
IUD	E	NE	1. Dequeue Update Request from UQ 2. For all Read Requests in UQ: - Dequeue Read Requests from UQ - Send IPC_READ_PROCEED - Enqueue in RQ
IUD	NE	NE	Erroneous State. Not Possible.

Table 5.1: IPC message rules for Dynamic Content Caching

IPC_READ_REQUEST, IRD – IPC_READ_DONE, IUR – IPC_UPDATE_DONE, IUD – IPC_UPDATE_DONE, E – Empty and NE – Not Empty.

5.2.1 Strong Coherency Model over RDMA-enabled Interconnects

In this section, we point out several optimizations possible in the design described, using the advanced features provided by InfiniBand. In Section 4.3 we provide the performance achieved by the InfiniBand-optimized architecture.

As described earlier, on every request the proxy module needs to validate the cache corresponding to the document requested. In traditional protocols such as TCP/IP, this requires the proxy module to send a version request message to the *version thread*[1], followed by the *version thread* explicitly sending the version number back to the proxy module. This involves the overhead of the TCP/IP protocol stack for the communication in both directions. Several researchers have provided solutions such as SDP to get rid of the overhead associated with the TCP/IP protocol stack while maintaining the sockets API. However, the more important concern in this case is the processing required at the version thread (e.g. searching for the index of the requested file and returning the current version number).

Application servers typically tend to perform several computation intensive tasks including executing CGI scripts, Java applets, etc. This results in a tremendously high CPU requirement for the main application server itself. Allowing an additional version thread to satisfy version requests from the proxy modules results in a high CPU usage for the module itself. Additionally, the large amount of computation

[1]Version Thread is a separate thread spawned by the application module to handle version requests from the proxy module

carried out on the node by the application server results in significant degradation in performance for the version thread and other application modules running on the node. This results in a delay in the version verification leading to an overall degradation of the system performance.

In this scenario, it would be of great benefit to have a one-sided communication operation where the proxy module can directly check the current version number without interrupting the version thread. InfiniBand provides the RDMA read operation which allows the initiator node to directly read data from the remote node's memory. This feature of InfiniBand makes it an ideal choice for this scenario. In our implementation, we rely on the RDMA read operation for the proxy module to get information about the current version number of the required file. Figure 5.4 demonstrates the InfiniBand-Optimized coherency architecture.

5.2.2 Potential Benefits of RDMA-based Design

Using RDMA operations to design and implement client polling scheme in data-center servers over InfiniBand has several potential benefits.

Improving response latency: RDMA operations over InfiniBand provide very low latency of about 5.5μs and a high bandwidth up to 840Mbytes per second. Protocol communication overhead to provide strong coherence is minimal. This can improve response latency.

Increasing system throughput: RDMA operations have very low CPU overhead in both sides. This leaves more CPU free for the data center nodes to perform other processing, particularly on the back-end servers. This benefit becomes more attractive when a large amount of dynamic content is generated and significant computation is needed in the data-center nodes. Therefore, clients can benefit from active caching with strong coherence guarantee at little cost. The system throughput can be improved significantly in many cases.

Enhanced robustness to load: The load of data center servers with support of dynamic web services is very bursty and unpredictable [76, 82]. Performance of protocols to maintain strong cache coherency over traditional network protocols can be degraded significantly when the server load is high. This is because both sides should get involved in communication and afford considerable CPU to perform communication operations. However, for protocols based on RDMA operations, the peer side is transparent to and nearly out of the communication procedure. Little overhead is paid on the peer server side. Thus, the performance of dynamic content caching with strong coherence based on RDMA operations is mostly resilient and well-conditioned to load.

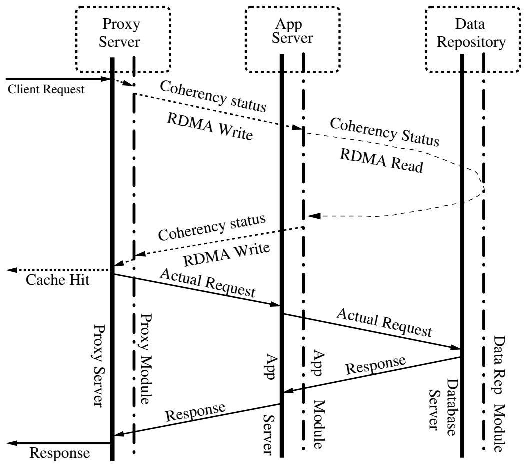


Figure 5.4: Strong Cache Coherency Protocol: InfiniBand based Optimizations

5.3 Experimental Evaluation

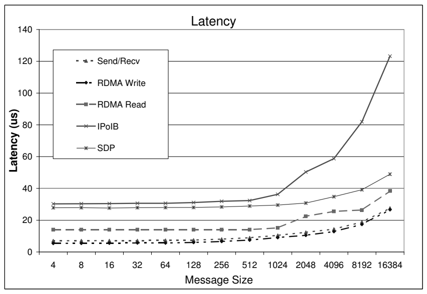
In this section, we first show the micro-benchmark level performance given by VAPI, SDP and IPoIB. Next, we analyze the performance of a cache-coherent 2-tier data-center environment. Cache coherence is achieved using the *Client-Polling* based approach in the architecture described in Section 2.3.

All our experiments used a cluster system consisting of 8 nodes built around Super-Micro 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. The machines are connected with Mellanox InfiniHost MT23108 DualPort 4x HCA adapter through an InfiniScale MT43132 Eight 4x Port InfiniBand Switch. The Mellanox InfiniHost HCA SDK version is thca-x86-0.2.0-build-001. The adapter firmware version is fw-23108-rel-1_18_0000. We used the Linux 2.4.7-10 kernel.

5.3.1 Microbenchmarks

In this section, we compare the ideal case performance achievable by IPoIB and InfiniBand VAPI using a number of micro-benchmark tests.

Figure 5.5 a shows the one-way latency achieved by IPoIB, VAPI Send-Receive, RDMA Write, RDMA Read and SDP for various message sizes. Send-Receive achieves a latency of around 7.5μs for 4 byte messages compared to a 30μs achieved by IPoIB, 27μs achieved by SDP and 5.5μs and 10.5μs achieved by RDMA Write and RDMA Read, respectively. Further, with increasing message sizes, the difference between the latency achieved by native VAPI, SDP and IPoIB tends to increase.



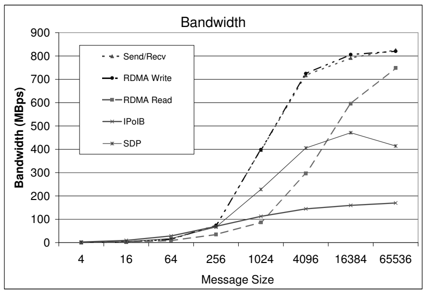
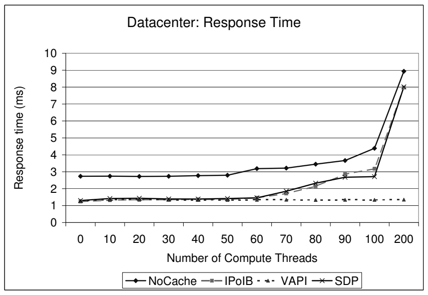


Figure 5.5: Micro-Benchmarks: (a) Latency, (b) Bandwidth

Figure 5.5b shows the uni-directional bandwidth achieved by IPoIB, VAPI Send-Receive and RDMA communication models and SDP. VAPI Send-Receive and both RDMA models perform comparably with a peak throughput of up to 840Mbytes/s compared to the 169Mbytes/s achieved by IPoIB and 500Mbytes/s achieved by SDP. We see that VAPI is able to transfer data at a much higher rate as compared to IPoIB and SDP. This improvement in both the latency and the bandwidth for VAPI compared to the other protocols is mainly attributed to the zero-copy communication in all VAPI communication models.



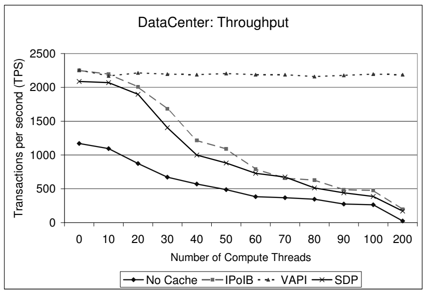


Figure 5.6: Data-Centers Performance Analysis for Dynamic Content Caching

5.3.2 Strong Cache Coherence

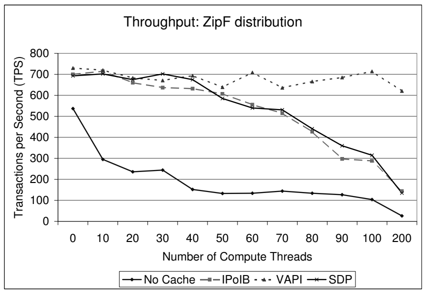
In this section, we analyze the performance of a cache-coherent 2-tier data-center environment consisting of three proxy nodes and one application server running Apache-1.3.12. Cache coherency was achieved using the *Client-Polling* based approach described in Section 2.3. We used three client nodes, each running three threads, to fire requests to the proxy servers.

Three kinds of traces were used for the results. The first trace consists of a single 8Kbyte file. This trace shows the ideal case performance achievable with the highest possibility of cache hits, except when the document is updated at the back-end. The second trace consists of 20 files of sizes varying from 200bytes to 1Mbytes. The access frequencies for these files follow a Zipf distribution [86]. The third trace is a 20,000 request subset of the WorldCup trace [13]. For all experiments, accessed documents were randomly updated by a separate update server with a delay of one second between the updates.

The HTTP client was implemented as a multi-threaded parallel application with each thread independently firing requests at the proxy servers. Each thread could either be executed on the same physical node or on a different physical nodes. The architecture and execution model is similar to the WebStone workload generator [59].

As mentioned earlier, application servers are typically compute intensive mainly due to their support to several compute intensive applications such as CGI script execution, Java applets, etc. This typically spawns several compute threads on the application server node using up the CPU resources. To emulate this kind of behavior, we run a number of compute threads on the application server in our experiments.

Figure 5.6a shows the client response time for the first trace (consisting of a single 8Kbyte file). The x-axis shows the number of compute threads running on the application server node. The figure shows an evaluation of the proposed architecture implemented using IPoIB, SDP and VAPI and compares it with the response time obtained in the absence of a caching mechanism. We can see that the proposed architecture performs equally well for all three (IPoIB, SDP and VAPI) for a low number of compute threads; All three achieve an improvement of a factor of 1.5 over the no-cache case. This shows that two-sided communication is not a huge bottleneck in the module as such when the application server is not heavily loaded.



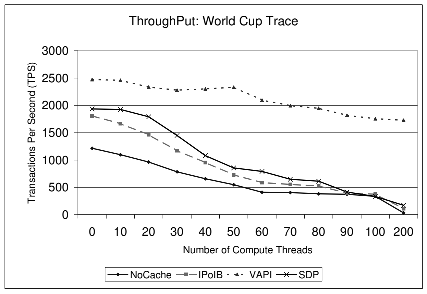


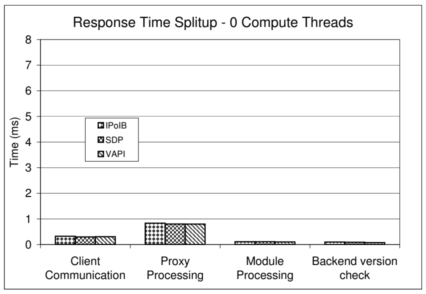
Figure 5.7: Data-Center Throughput: (a) Zipf Distribution, (b) WorldCup Trace

As the number of compute threads increases, we see a considerable degradation in the performance in the no-cache case as well as the Socket-based implementations using IPoIB and SDP. The degradation in the no-cache case is quite expected, since all the requests for documents are forwarded to the back-end. Having a high compute load on the back-end would slow down the application server's replies to the proxy requests.

The degradation in the performance for the Client-Polling architecture with IPoIB and SDP is attributed to the two sided communication of these protocols and the context switches taking place due to the large number of threads. This results in a significant amount of time being spent by the application modules just to get access to the system CPU. It is to be noted that the version thread needs to get access to the system CPU on every request in order to reply back to the proxy module's version number requests.

On the other hand, the Client-Polling architecture with VAPI does not show any significant drop in performance. This is attributed to the one-sided RDMA operations supported by InfiniBand. For example, the version number retrieval from the version thread is done by the proxy module using a RDMA Read. That is, the version thread does not have to get access to the system CPU; the proxy thread can retrieve the version number information for the requested document without any involvement of the version thread. These observations are re-verified by the response time breakup provided in Figure 5.8.

Figure 5.6b shows the throughput achieved by the data-center for the proposed architecture with IPoIB, SDP, VAPI and the no-cache cases. Again, we observe that the architecture performs equally well for both Socket based implementations (IPoIB and SDP) as well as VAPI for a low number of compute threads with an improvement of a factor of 1.67 compared to the no-cache case. As the number of threads increases, we see a significant drop in the performance for both IPoIB and SDP based client-polling implementations as well as the no-cache case, unlike the VAPI-based client-polling model, which remains almost unchanged. This is attributed to the same reason as that in the response time test, i.e., no-cache and Socket based client-polling mechanisms (IPoIB and SDP) rely on a remote process to assist them. The throughput achieved by the WorldCup trace (Figure 5.7b) and the trace with Zipf distribution (Figure 5.7a) also follow the same pattern as above. With a large number of compute threads already competing for the CPU, the wait time for this remote process to acquire the CPU can be quite high, resulting in this degradation of performance. To demonstrate this, we look at the component wise break-up of the response time.



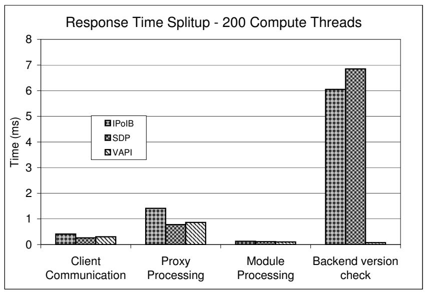


Figure 5.8: Data-Center Response Time Breakup: (a) 0 Compute Threads, (b) 200 Compute Threads

Figure 5.8a shows the component wise break-up of the response time observed by the client for each stage in the request and the response paths, using our proposed architecture on IPoIB, SDP and VAPI, when the backend has no compute threads and is thus not loaded. In the response time breakup, the legends *Module Processing*, and *Backend Version Check* are specific to our architecture. We can see that these components together add up to less than 10% of the total time. This shows that the computation and communication costs of the module as such do not add too much overhead on the client's response time.

Figure 5.8b on the other hand, shows the component wise break-up of the response time with a heavily loaded backend server (with 200 compute threads). In this case, the module overhead increases significantly for IPoIB and SDP, comprising almost 70% of the response time seen by the client, while the VAPI module overhead remains unchanged by the increase in load. This indifference is attributed to the one-sided communication used by VAPI (RDMA Read) to perform a version check at the backend. This shows that for two-sided protocols such as IPoIB and SDP, the main overhead is the context switch time associated with the multiple applications running on the application server which skews this time (by adding significant wait times to the modules for acquiring the CPU).

5.4 Summary

Caching content at various tiers of a multi-tier data-center is a well known method to reduce the computation and communication overhead. In the current web, many cache policies and uncachable resources are driven by two server application goals: Cache Coherence and Cache Consistency. The problem of how to provide consistent caching for dynamic content has been well studied and researchers have proposed several weak as well as strong consistency algorithms. However, the problem of maintaining cache coherence has not been studied as much. Further, providing such capabilities in a standardized manner needs consideration.

In this chapter, we proposed an architecture for achieving strong cache coherence based on the previously proposed client-polling mechanism for multi-tier data-centers. The architecture as such could be used with any protocol layer; we also proposed optimizations to better implement it over InfiniBand by taking advantage of one sided operations such as RDMA. We evaluated this architecture using three protocol platforms: (i) TCP/IP over InfiniBand (IPoIB), (ii) Sockets Direct Protocol over InfiniBand (SDP) and (iii) the native InfiniBand Verbs layer (VAPI) and compared it with the performance of the no-caching based coherence mechanism. Our experimental results show that the InfiniBand-optimized architecture can achieve an improvement of upto a factor of two for the response time and nearly an order of magnitude for the throughput achieved by the TCP/IP based architecture, the SDP based architecture and the no-cache based coherence scheme. The results also demonstrate that the implementation based on RDMA communication mechanism can offer better performance robustness to the load of the data-center servers.

CHAPTER 6

DYNAMIC CONTENT CACHING FOR COMPLEX OBJECTS

In this chapter, we describe all aspects of our design to handle complex dynamic objects. We detail each of the requirements along with the corresponding design solution. We broadly divide this section into three parts: (i) Section 6.2: The basic protocol for the cache coherency, (ii) Section 6.2.2: Application server interaction and (iii) Section 6.2.3: The study of the effect of multiple dependencies on cached documents.

6.1 Background and Related Work

Online services such as personalized services, e-commerce based services, etc. have recently increased several folds in volume. Scenarios like online banking, auctions, etc. are constantly adding to the complexity of content being served on the Internet. The responses generated for these can change depending on the request and are typically known as dynamic or active content. Multi-tier data-centers process these complex requests by breaking-up the request processing into several stages with each data-center tier handling a different stage of request processing. With the current

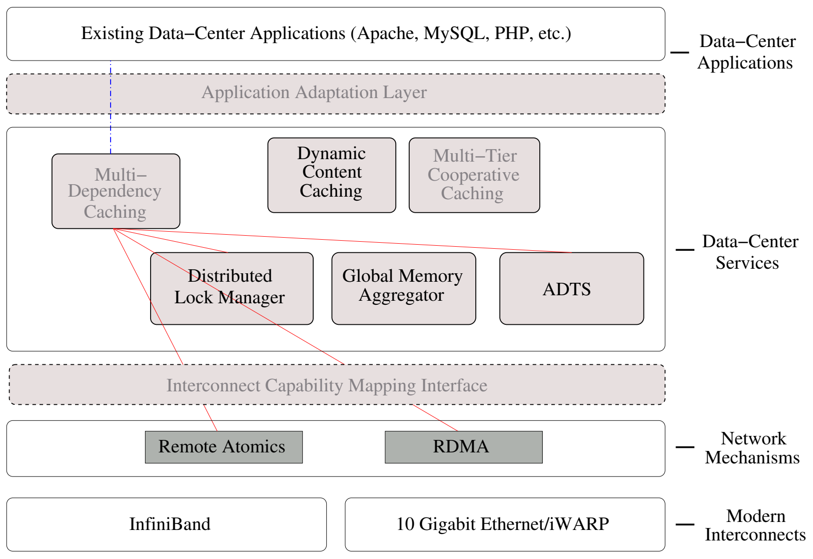

Existing Data–Center Applications (Apache, MySQL, PHP, etc.)
Data–Center Applications
Application Adaptation Layer
Multi– Dependency Caching
Dynamic Content Caching
Multi–Tier Cooperative Caching
Data–Center Services
Distributed Lock Manager
Global Memory Aggregator
ADTS
Interconnect Capability Mapping Interface
Remote Atomics
RDMA
Network Mechanisms
InfiniBand
10 Gigabit Ethernet/iWARP
Modern Interconnects

Figure 6.1: Proposed Dynamic Content Caching with Complex Objects

processing needs and growth trends in mind, the scalability of data-centers has become an important issue.

Traditionally, caching has been an important technique to improve scalability and performance of data-centers. However, simple caching methods are clearly not applicable for dynamic content caching. Documents of dynamic nature are typically generated by processing one or more data objects stored in the back-end database, i.e., these documents are dependent on several persistent data objects. These persistent data objects can also be a part of multiple dynamic documents. So in effect these documents and data objects have several many to many mappings between them. Thus, any change to one individual object can potentially affect the validity of multiple cached requests.

In the previous chapter, we have presented a simple architecture that supports strong cache coherency for proxy caches. However, the previous scenario is optimized for file level granularity for coherency, i.e., each update affects a single object which corresponding to each cached request. However, most data-centers allow and support more complex web documents comprising of multiple dynamic objects. These additional issues necessitate more intricate protocols to enable dynamic content caching and make the design of strongly coherent caches extremely challenging. Further, since an updated object can potentially be a part of multiple documents across several servers, superior server coordination protocols take a central role in these designs.

Several researchers have focused on the aspect of dynamic content caching. Popular approaches like TTL [41], Adaptive TTL [30], MONARCH [58], etc. deal with lazy consistency and coherency caches. These cannot handle strong cache coherency. Approaches like [18, 19] deal with strong cache consistency. Other approaches like *get-if-modified-since* specified in [41] can handle coherent changes to the original source file, but are not designed to handle highly dynamic data. These essentially will cause the file to be re-created for each request negating the benefits of caching. Solutions proposed in [24] handle the strong cache coherency but use two-sided communication and are less resilient to load.

In this chapter, we present an extended architecture to support strong cache coherency service for dynamic content caches. Our architecture is designed to handle caching of responses composed of multiple dynamic dependencies. We propose a complete architecture to handle two issues: (i) caching documents with multiple dependencies and (ii) being resilient to load on servers. We also study the effect of varying dependencies on these cached responses. The following sections detail our proposed designs.

Cachable Requests: Data-center serving dynamic data, usually have HTTP requests that may be reads (select based queries to the database) or writes (update or insert based queries to the database). While read based queries are cachable, writes cannot be cached at the proxies. Since, caching the popular documents gives the maximum benefits, its a popular practice to cache these. Most simple caching schemes work on this principle. Similarly, in our design, a certain number of top most frequent requests are marked down for caching. Naturally, caching more requests leads to better performance but requires higher system resources. The actual number of requests that are cached are chosen based on the availability of resources. Based on these constraints, for each request the proxy server decides if the request is cachable. And if it is cachable, the proxy decides if caching that particular request is beneficial enough. Significant amount of research has been done on cache replacement policies [36, 48, 73]. Our work is complimentary to these and can leverage those benefits easily.

6.2 Caching Documents with Multiple Dependencies

In our approach we divide the entire operation into two parts based on the tier functionalities. Proxy servers that maintain the cache need to validate the cache entity for each request. The application servers need to maintain the current version of the cached entity for the proxies to perform validations. We use the external module-based design described in Section 4.1.1 for providing these services.

RDMA based Strong Cache Coherence:

Caches in our design are located on the proxy server nodes. On each request, the primary issues for a proxy are as follows: (i) Is this request cachable? (ii) Is the response currently cached? and (iii) Is this cached response valid?

These services are provided to the proxy server by our module running on the proxy node. The apache proxy server is installed with a small handler that contacts the local module with an *IPC-Verify* message and waits on the IPC queue for a response. The module responds with a *use cache* or *do not use cache* depending on the choices. If the request is not cachable or if the cache is not present or invalid, the module responds with *do not use cache.* And if the request is cachable, cache is present and valid, the module responds with *use cache.*

The module verifies the validity of the cached entry by contacting the home node application server module which keeps track of the current version for this particular cache file. InfiniBand's one sided operation (RDMA Read) is used to obtain the current version from the shared version table on the home node application server thereby avoiding interrupts at that application server. Figure 6.2 shows the basic protocol. The information on cachability and presence of cache are available locally on each proxy.

Each proxy maintains a version number for each of the cached entries. The same cache files also have a current version number maintained on the home node application server modules. When necessary, the application server modules increment their cache version numbers. For each proxy verify message, if the application server cache file version number and the proxy's local cache file version number match, then it implies that the cache file is current and can be used to serve the request. This basic protocol was proposed in our previous work [62].

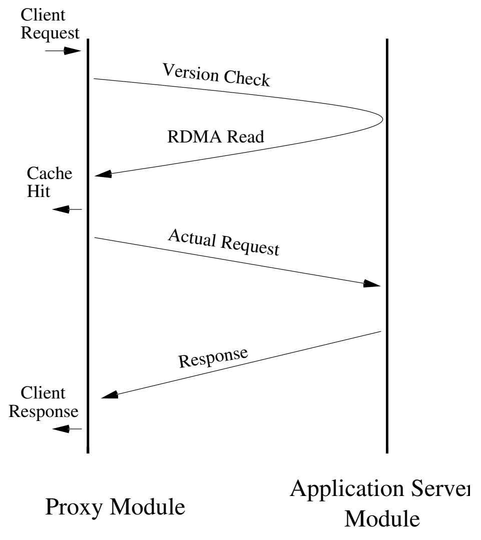


Figure 6.2: RDMA based Strong Cache Coherence

6.2.1 Multi-Dependency Maintenance

All cache misses from the proxies are serviced by application servers. Since all accesses to the database need to be routed through an application server and since an application server (unlike a proxy) has the capability to analyze and process database level queries, we handle all coherency issues at this tier.

An application server module needs to cater to two cases: (i) version reads from the proxy server and (ii) version updates from local and other application servers. The main work of the application server module lies in updating the shared version table readable by the proxy server modules based on the updates that occur to the data in the system.

As mentioned, a single cached request contains multiple dynamic objects which can get updated. For any version updates to take place, it is necessary to know the following: (i) Which updates affect which dynamic objects? and (ii) Which dynamic objects affect which cache files? Since, typically dynamic objects are generated as results of queries to a database, knowledge of the database records that a query depends on is sufficient to answer the above.

There are three cases that arise in this context: (i) The application server understands the database schema, constraints, each query and its response thereby knowing all the dependencies of a given request, (ii) Each query response contains enough information (e.g. list of all database keys) to find out the dependencies or (iii) The application server is incapable of gauging any dependencies (possibly for cases with very complex database constraints). The first two cases can be handled in the same manner by the application server module since the dependencies for all requests can be obtained. The third case needs a different method. We present the following two sample schemes to handle these cases. It is to be noted that these schemes are merely simple schemes to show proof of concept. These can be further optimized or be replaced by complex schemes to handle these cases.

Scheme - Invalidate All: For cases where the application servers are incapable of getting any dependency information, the application servers modules can invalidate the entire cache for any update to the system. This makes sure that no update is hidden from the clients. But this also leads to a significant number of false invalidations. However, the worst performance by this scheme is lower bounded by the base case with no caching.

Scheme - Dependency List: In cases where all the dependencies of the required queries are known, the application server module maintains a list of dependencies for each cached request (for which it is a home node) along with the version table. In case the application server module is notified of any update to the system, it checks these lists for any dependencies matching the update. All cache files that have at least one updated dependency are then invalidated by incrementing the version number on the shared version table. This scheme is very efficient in terms of the number of false invalidations but involves slightly higher overhead as compared to the *Invalidate All* scheme.

6.2.2 Protocol for Coherent Invalidations

In addition to the issues seen above, requests comprising multiple dynamic objects in them involve additional issues. For example, two different cache files with different home nodes might have a common dependency. So, any update to this dependency needs to be sent to both these home nodes. Similarly, the application server modules need to communicate all updates with all other application server modules. And the update can be forwarded to the database for execution only after all the application server modules invalidate all the dependent cache files.

Figure 6.3 shows the interaction between the application servers and the database for each update. As shown, the application server on getting an update, broadcasts the same to all the other application server modules. These modules then perform their local invalidations depending on the scheme chosen (*Invalidate All* or *Dependency List* search). After the invalidations, the modules send an acknowledgment to the original server, which forwards the request to the database and continues with the rest as normal.

In our design, we use *VAPI-SEND/VAPI-RECEIVE* for the initial broadcasts. The acknowledgments are accumulated by the original process using *VAPI-ATOMIC-FETCH-AND-ADD* : poll : yield cycle. For each application server module, an

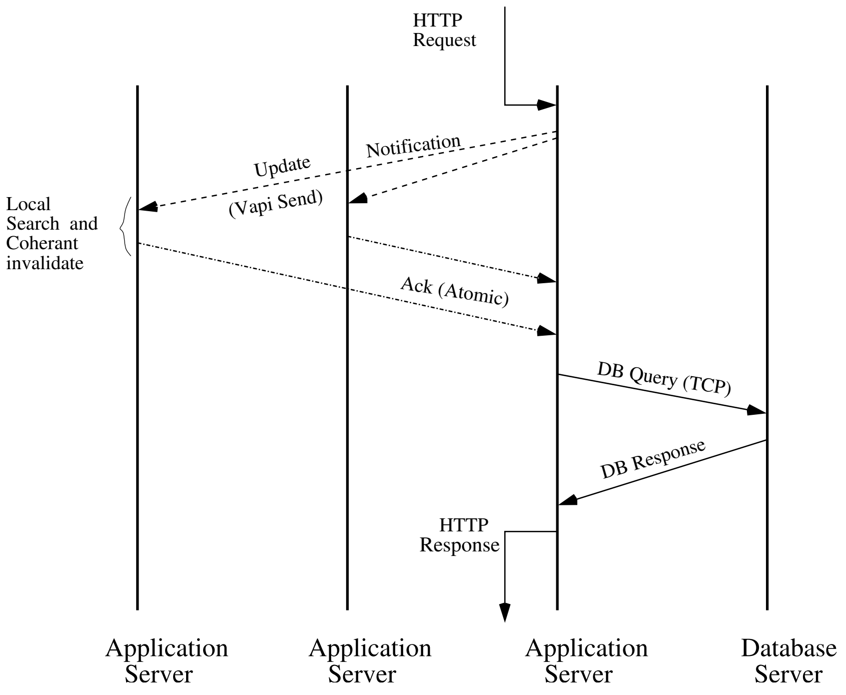
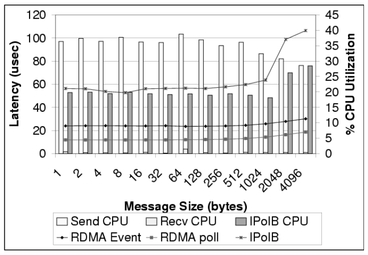


Figure 6.3: Protocol for Coherent Invalidations

acknowledgment collection variable is defined and set to zero for each update. All the other application server modules perform a *VAPI-ATOMIC-FETCH-AND-ADD* incrementing the value of this variable by one. The original server module checks this ack collection variable to see if all the remaining modules have performed this operation. If the value of the ack collection variable is less than the number of other servers, then the original application server module process yields the CPU to other processes in the system using the system call *sched_yield()*. This kind of polling cycle makes sure that the module process does not waste any CPU resources that other processes on that node could have used.

6.2.3 Impact of Multiple Dependencies on Caching

We have seen that there is significant complexity in managing multiple dependencies per cache file on strong coherency caching. In addition, having multiple dependencies also affect the overall cache. Typically, caching is expected to yield maximum benefits when the number of updates is low and the benefits of caching are linked with the number of updates in all calculations.



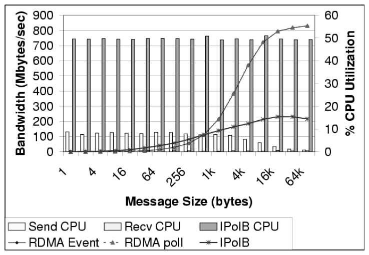


Figure 6.4: Micro-Benchmarks for RDMA Read and IPoIB: (a) Latency and (b) Bandwidth

However, the actual value that affects caching is the number of invalidations that occur to the cached entries. The main difference between the number of invalidations and the number of updates to an object is the magnification factor for updates. This magnification factor represents the average number of dependencies per cached entry. Hence, the cache effectiveness is dependent on the product of *system update rate* and *average dependency magnification factor*.

In our design each application server module maintains its own set of cache file versions and the corresponding dependency lists. So, for each update, the number of messages between the application servers is not affected by this magnification factor. Each application server module is just notified once for each update, and all the invalidations on that node are taken care of locally by the corresponding module. However, the overall cache hit ratio remains directly affected by this factor.

6.3 Experimental Evaluation

In this section, we describe our experimental testbed and a set of relevant micro-benchmark results followed by overall data-center results.

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

Cluster 1: 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.

Cluster 2: A cluster system consisting of 8 nodes built around SuperMicro SUPER X5DL8-GG motherboards with ServerWorks GC LE chipsets which include 64-bit 133 MHz PCI-X interfaces. Each node has two Intel Xeon 3.0 GHz processors with a 512 kB L2 cache and a 533 MHz front side bus and 512 MB of main memory. We used the RedHat 9.0 Linux distribution.

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

Interconnect: InfiniBand network with Mellanox InfiniHost MT23108 DualPort 4x HCA adapter through an InfiniScale MT43132 twenty-four 4x Port completely non-blocking InfiniBand Switch. The Mellanox InfiniHost HCA SDK version is thca-x86-3.2-rc17. The adapter firmware version is fw-23108-rel-3_00_0002. The IPoIB driver for the InfiniBand adapters was provided by Mellanox Incorporation as a part of the Golden CD release 0.5.0.

Cluster 1 was used for all the client programs and Cluster 2 was used for the data-center servers. In our experiments, we used apache servers 2.0.48 as proxy servers, apache 2.0.48 with PHP 4.3.7 as application servers and MySQL 4.1 [5] as the database. Our system was configured with *five* proxy servers, *two* application servers and *one* database server.

6.3.1 Microbenchmarks

We show the basic micro-benchmarks that characterize our experimental testbed. We present the latency, bandwidth and CPU utilizations for the communication primitives used in our design. Figure 6.4 shows the performance achieved by VAPI RDMA read and TCP/IP over InfiniBand (IPoIB).

The latency achieved by the VAPI RDMA Read communication model and IPoIB (round-trip latency) for various message sizes is shown in Figure 6.4a. RDMA Read, using the polling based approach, achieves a latency of 11.89μs for 1 byte messages compared to the 53.8μs achieved by IPoIB. The event based approach, however, achieves a latency of 23.97μs. Further, with increasing message sizes, the difference between the latency achieved by VAPI and IPoIB tends to increase significantly. The figure also shows the CPU utilized by RDMA Read (notification based) and IPoIB. The receiver side CPU utilization for RDMA as observed is negligible and close to zero, i.e., with RDMA, the initiator can read or write data from the remote node without requiring any interaction with the remote host CPU. In our experiments, we benefit more from the one sided nature of RDMA and not just due to the raw performance improvement of RDMA over IPoIB.

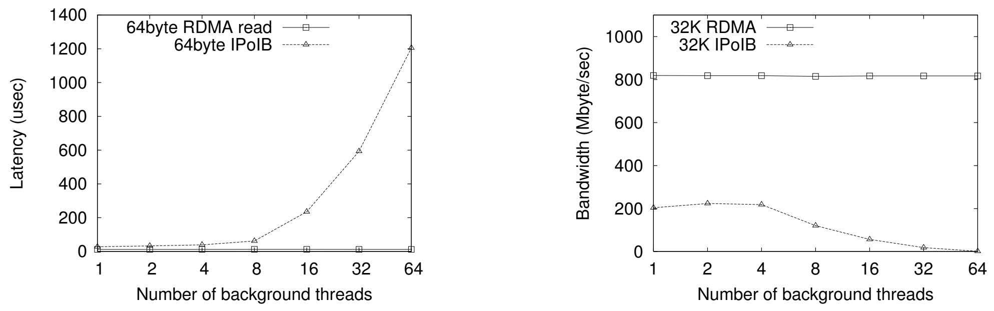
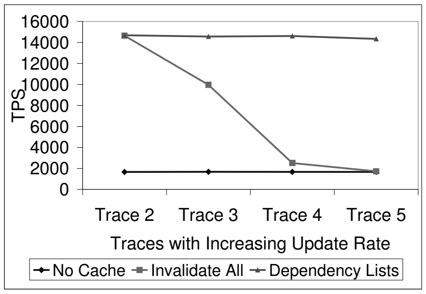


Figure 6.5: Performance of IPoIB and RDMA Read with background threads: (a) Latency and (b) Bandwidth

Figure 6.4b shows the uni-directional bandwidth achieved by RDMA Read and IPoIB. RDMA Read is able to achieve a peak bandwidth of 839.1 MBps as compared to a 231 MBps achieved by IPoIB. Again, the CPU utilization for RDMA is negligible on the receiver side.

In Figure 6.5, we present performance results showing the impact of the loaded conditions in the data-center environment on the performance of RDMA Read and IPoIB on Cluster 2. We emulate the loaded conditions in the data-center environment by performing background computation and communication operations on the server while the read/write test is performed by the proxy server to the loaded server. This environment emulates a typical cluster-based multiple data-center environment where multiple server nodes communicate periodically and exchange messages, while the proxy, which is not as heavily loaded, attempts to get the version information from the heavily loaded machines. Figure 6.5 shows that the performance of IPoIB degrades significantly with the increase in the background load. On the other hand, one-sided communication operations such as RDMA show absolutely no degradation in the performance. These results show the capability of one-sided communication primitives in the data-center environment.



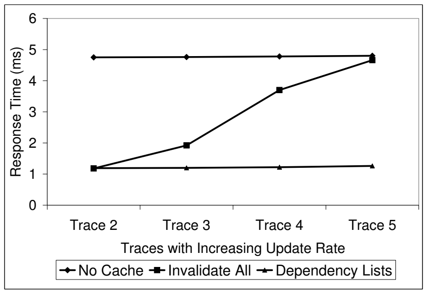


Figure 6.6: Performance of Data-Center with Increasing Update Rate: (a) Throughput and (b) Response Time

6.3.2 Coherent Active Caching

In this section, we present the basic performance benefits achieved by the strong coherency dynamic content caching as compared to a traditional data-center which does not have any such support. To measure and make these comparisons, we use the following traces: (i) *Trace 1:* Trace with 100% reads, (ii) *Trace 2 - Trace 5:* Traces with update rates increasing in the order of milliseconds to seconds. (iii) *Trace 6:* Zipf like trace [20, 86] with medium update rate.

6.3.3 Overall Performance and Analysis

In this section, we present results for overall data-center performance with dynamic content caching. We also analyze these results based on the actual number of

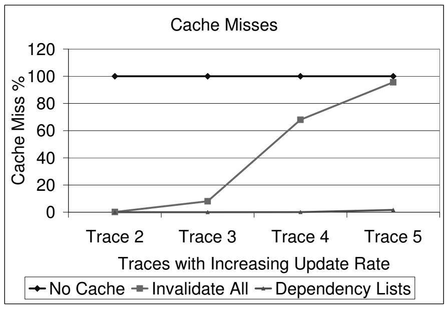


Figure 6.7: Cache Misses With Increasing Update Rate

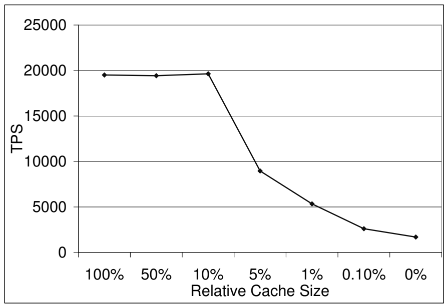


Figure 6.8: Effect of Cache Size for Trace 6

cache misses and cache hits. All results presented in this section are taken in steady state (i.e. eliminating the effects of cold cache misses, if any). We non-cached data-center throughput of about 1,700 TPS for *Trace 1* and 15,000 TPS for a fully cached data-center. These values roughly represent the minimum and maximum throughput achievable for our setup.

Figure 6.6a compares the throughput achieved by dynamic content caching schemes and the base case with no caching. We observe that the caching schemes always perform better than the *no cache* schemes. The best case observed is about 8.8 times better than the *no cache* case.

Figure 6.6a also shows the two schemes (*Invalidate All* and *Dependency Lists*) for the traces 2 - 5. We observe that *Invalidate All* scheme drops in performance as the update rate increases. This is due to the false invalidations that occur in the *Invalidate All* scheme. On the other hand, we observe that *Dependency Lists* scheme is capable of sustaining performance even for higher update rates. The latter sustains a performance of about 14,000 TPS for our setup. Figure 6.6b shows the response time results for the above three cases. We observe similar trends in these results as above. *No Cache* case has a response time of about 4 milliseconds where as the best response time for dynamic content caching schemes is about 1.2 milliseconds.

Figure 6.7 shows the cache misses that occur in each of the runs in the throughput test. The *No Cache* scheme obviously has 100% cache misses and represents the worst case scenario. We clearly observe that the cache misses for *Invalidate All* scheme increase drastically with increasing update rate, leading to the drop in performance. Data-center scenarios in which application servers cannot extract dependencies from the requests can take advantage of our dynamic content caching architecture for lower update rates. For higher update rates, *Invalidate All* performs slightly better than or almost equal to the performance of *No Cache* case. The difference in the number of cache misses between *Invalidate All* and *Dependency Lists* is the number of false invalidations occurring in the system for the *Invalidate All* scheme.

Selective Caching

As mentioned earlier, in real scenarios only a few popular files are cached. In Figure 6.8, we present the results of an experiment showing the overall data-center performance for varying cache sizes. We used *Trace 6* for this experiment. We observe that even for very small cache sizes the performance is significantly higher than the

No Cache case. The throughput achieved by caching 10% of the files is close to the maximum achievable. Hence, data-centers with any amount of resources can benefit from our schemes.

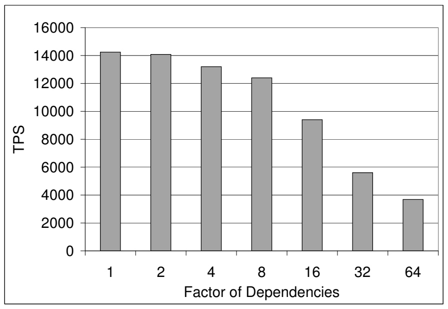


Figure 6.9: Effect of Varying Dependencies on Overall Performance

Effect of Varying Dependencies on Overall Performance

Figure 6.9 shows the effect of increasing the number of dependencies on the overall performance. The throughput drops significantly with the increase in the average number of dependencies per cache file. This is because the number of coherent cache invalidations per update request increase with the average number of dependencies tending toward *Invalidate All* in the worst case. We see that as the ratio of object updates to file invalidations representing the dependency factor increases to 64 in Figure 6.9 the throughput achieved drops by about a factor of 3.5.

6.3.4 Effect on Load

In this section, we study the effect of increased back-end server load on the data-center aggregate performance. In this experiment, we emulate artificial load as described in Section 6.3.1. We use *Trace 5* (trace with higher update rate) to show the results for the *Dependency Lists* scheme. Figure 6.10 shows the results.

We observe that our design can sustain high performance even under heavy back-end load. Further, the factor of benefit for the *Dependency Lists* scheme to the *No Cache* scheme increases from about 8.5 times to 21.9 times with load. This clearly shows that our approach is much more resilient to back-end load than the *No Cache* scheme. In addition, since loaded back-end servers can support the proxy caching with negligible overhead, our approach can scale to bigger data-centers with significantly higher number of caching servers. The results in [62] show that these benefits are largely due to the one-sided communication in the basic client polling protocols.

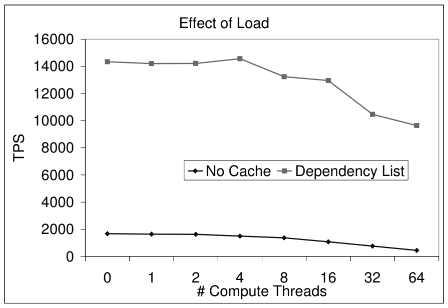


Figure 6.10: Effect of Load

6.4 Summary

In this chapter, we have presented an extended load resilient architecture that supports caching of dynamic requests with multiple dynamic dependencies in multi-tier data-centers. Our architecture is designed to support existing data-center applications with minimal modifications. We have used one sided operations like RDMA and Remote Atomics in our design to enable load resilient caching. We have performed our experiments using native InfiniBand Verbs Layer (VAPI) for all protocol communications. Further, we have presented two schemes *Invalidate All* and *Dependency Lists* to suite the needs and capabilities of different data-centers. Our experimental results show that in all cases the usage of our schemes yield better performance as compared to *No Cache* case. Under loaded conditions, our architecture can sustain high performance better than the *No Cache* case, and in some cases being more than an order of magnitude better. The results also demonstrate that our design can scale well with increasing number of nodes and increasing system load.

CHAPTER 7

EFFICIENT MULTI-TIER COOPERATIVE CACHING

In this work, we present four schemes for efficient cooperative caching to optimize the utilization of the system resources. At each stage we also justify our design choices. This section is broadly categorized into four main parts: (i) RDMA based design and implementation of basic cooperative caching (Section 7.2.1) , (ii) Design of a non-redundancy scheme (Section 7.2.2), (iii) Multi-tier extensions for cooperative caches (Section 7.2.3) and (iv) A combined hybrid approach for cooperative caches (Section 7.2.4).

7.1 Background and Related Work

As mentioned earlier, caching of processed content in a data-center has been a long standing technique to help web systems to scale and deliver high performance. Researchers [9] have looked at various aspects of basic web caching. It has been well acknowledged in the research community that single larger cache performs significantly better than multiple smaller caches. Even though the total cache size remains the same in both cases, having multiple independent caches leads to very high localization of caching decisions and individual servers do not take advantage of the neighboring caches. Due to this disadvantage and the fact that current clusters support efficient data transfers, [70, 74, 83, 65], for example, have proposed cooperation among caches within the data-center. Several nodes in a data-center participate in this cooperative caching process and try to present a logical illusion of having a single cache. While such cooperative caching schemes show better performance than the basic case of nodes caching independently, several trade-offs exist and a through study of needs to be done and caching schemes need to be modified appropriately to reflect these.

Currently many approaches do not cautiously eliminate the possibility of duplicating cached entries redundantly on more than a single node. Some approaches can handle this in a limited manner with no direct control over the cache content [67, 27]. In particular, such approaches [16] are explored in the context of storage systems. While this leads to better performance for the duplicated cache content, it is often at the cost of not caching other content that could have occupied this cache space. On the other hand, lack of redundant duplicates in the system necessitates the transfer of the cached entries from remote nodes on each request. In addition, cache replacement for each local cache is also considerably complicated due to the additional checks, logic and data transfers needed. A good balance between these trade offs is critical to achieve good performance. Consequently, designing these more complex protocols to deliver high performance in the light of these constraints, is a central challenge. Researchers [29] have evaluated and confirmed these needs empirically. In this chapter, we present schemes to eliminate these redundant copies to achieve high performance through effective cache to cache cooperations.

7.2 Efficient Cooperative Caching

In this section we describe the various design details of our cooperative caching schemes.

The traditional data-center applications service requests in two ways: (i) by using different server threads for different concurrent requests or (ii) by using single asynchronous server to process to service requests. Catering to both these approaches used by applications, our design uses the asynchronous external helper module described in Section 4.1.1 to provide cooperative caching support.

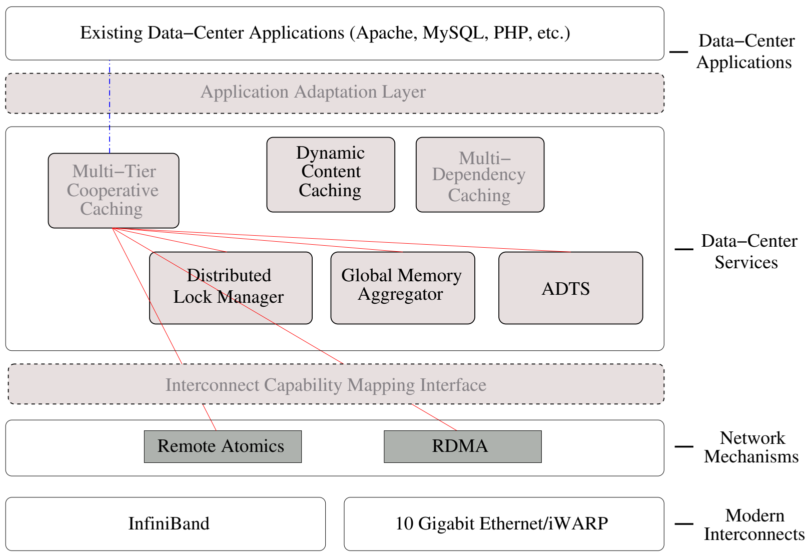


Figure 7.1: Proposed Multi-Tier Cooperative Caching

The cache's meta-data information is maintained consistently across all the servers by using a home node based approach. The cache entry key space (called *key-table*) is partitioned and distributed equally among the participating nodes and hence all the nodes handle a similar amount of meta-data key entries. This approach is popularly known as the home node based approach. It is to be noted that in our approach we handle only the meta-data on the home node and since the actual data itself can reside on any node, our approach is much more scalable than the traditional home node based approaches where the data and the meta-data reside on the home node.

All modifications to the file, such as invalidations, location transfers, etc., are performed on the home node for the respective file. This cache meta-data information is periodically broadcasted to other interested nodes. Additionally, this information can also be requested by other interested nodes on demand. The information exchange uses RDMA Read operations for gathering information and send-receive operations for broadcasting information. This is done to avoid complex locking procedures in the system. Please note that the actual remote data placement and remote data transfer is handled separately from the control protocol and the meta-data management and we intend to allow usage of these Global Memory Aggregation services directly to other applications in future.

Basic Caching Primitives: The basic caching operations can be performed using a small set of primitives. The internal working caching primitives needs to be designed efficiently for scalability and high performance. Our various schemes implement these primitives in different ways and are detailed in the following subsections. The basic caching primitives needed are:

- *Cache Fetch*: To fetch an entity already present in cache
- *Cache Store*: To store an entity in cache
- *Cache Validate*: To verify the validity of a cached entity
- *Cache Invalidate*: To invalidate a cached entity

Buffer Management: The cooperative cache module running on each node reserves a chunk of memory. This memory is then allocated to the cache entities as needed. Since this memory needs to be pooled into the global cooperative cache space, this memory is registered (i.e. locked in physical memory) with the InfiniBand HCA to enable efficient memory transfers by RDMA. Several researchers [25, 23, 85] have looked at the different aspects of optimizing this limited buffer usage and have suggested different cache replacement algorithms for web caches. Our methods are orthogonal to these issues and can easily leverage the benefits of the proposed algorithms.

In the following subsections, we describe the details of our following schemes: (i) Basic RDMA-based Cooperative Caching (BCC), (ii) Cooperative Cache Without Redundancy (CCWR), (iii) Multi-Tier Aggregate Cooperative Cache (MTACC) and (iv) Hybrid Cooperative Cache (HYBCC).

7.2.1 Basic RDMA based Cooperative Cache

In our design, the basic caching services are provided by a set of cooperating modules residing on all the participating server nodes. Each cooperating module keeps track of the local cache state as a set of local page-tables and places this information in the soft shared state for global access.

The Basic RDMA based Cooperative Caching (BCC) is achieved by designing the cache primitives using RDMA operations. The communication messages between the modules are divided into two main components: (i) control messages and (ii) data messages. The control messages are further classified into (i) meta-data read messages and (ii) meta-data update messages. Since data messages form the bulk volume of the total communication load we use one-sided RDMA operations for these. In addition, the meta-data read messages use the RDMA Read capabilities. Meta-data update messages are exchanged using send-receive operations to avoid concurrency control related issues.

The basic cache primitives are handled by BCC in the following manner:

Cache Fetch involves three simple steps: (i) finding the cache entry, (ii) finding a corresponding amount of local free space and (iii) fetching the data using RDMA Read operation.

Cache Store involves the following steps: in case the local node has enough free space the entity is cached and *key-table* holding the meta-data information is updated. In cases where the local node has no free memory, the entity is stored into a temporary buffer and the local copies of all page tables are searched for a suitable candidate remote node for a possible free space. A control message is sent to that node which then performs an RDMA Read operation of this data and notifies the original node of the transfer. Once a control message is sent with a store request to a remote node, then the current entity is considered to be a responsibility of the remote node. For both these primitives, in cases where free space is not available system-wide, a suitable replacement is chosen and data is stored in place of the replacement.

Cache Validate and *Cache Invalidate* involve a meta-data read or a meta-data update to the home node respectively. As mentioned earlier, RDMA Read is used for the read operation.

Although this scheme provides a way to share cache across the proxy nodes, there may be redundancy in the cache entities across the system.

7.2.2 Cooperative Cache Without Redundancy

In this scheme, the main emphasis is on the redundant duplicates in the system. At each step of request processing, the modules systematically search the system for possible duplicate copies of cache entities and these are chosen for replacement. In aggregate, the cache replacement decisions are taken in the following priority: (i) Local free space, (ii) Remote node free space, (iii) Local redundant copies of entries cached elsewhere in the system, (iv) remote redundant copies having duplicates in the system and (v) replacement of suitable entity by removing an existing entry to make space for the new entry. We again describe the details of designs of the cache primitives.

The case of *Cache Fetch* presents interesting design options. The data from remote node is fetched into local free space or in place of local redundant copy in the priority described above. However, in case there are no free buffer spaces or local duplicates available for getting the data, remote cache entity is swapped with some local cached entity. In our design, we select a suitable local replacement, send a store message to the remote cache for this local replacement and followed by a RDMA Read of the required remote cache entity. The remote node follows a similar mechanism to decide on storage and sends back an acknowledgment. Figure 7.2 shows the swap case of this scheme. The dotted lines shown in the figure are control messages.

Cache Store design in this case is similar to the previous approach, the main difference being the priority order described above. The memory space for storing new cache entries is searched in the order of free space, redundant copies and permanent replacements.

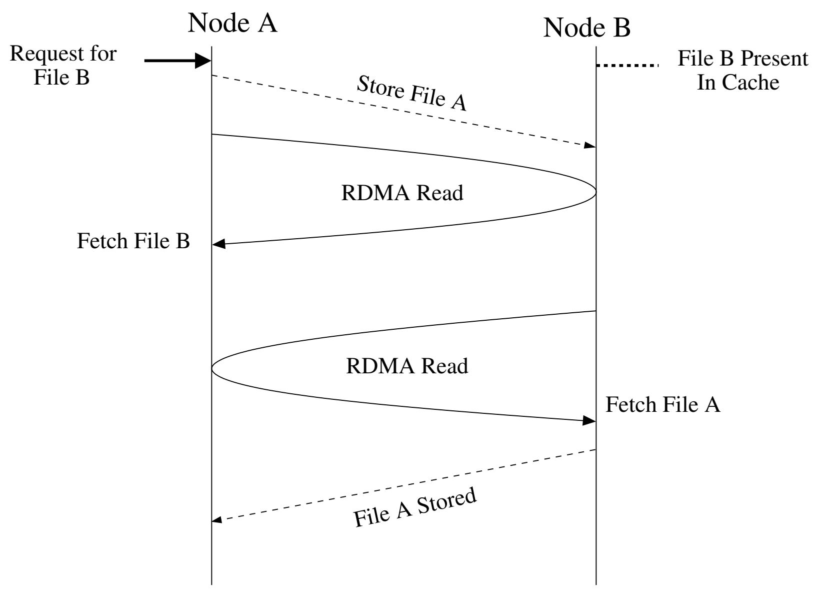


Figure 7.2: Cooperative Caching Without Redundancy

The CCWR scheme benefits significantly by increasing the total amount of memory available for cooperative caching by removing redundant cache entities. For large working sets this yields higher overall performance.

7.2.3 Multi-Tier Aggregate Cooperative Cache

In typical multi-tier data-centers proxy servers perform all caching operations. However, the system can benefit significantly by having access to additional memory resources. There are several back-end nodes in the data-center that might not be using their memory resources to the maximum extent. In MTACC, we utilize this additional free memory on servers from other tiers of the multi-tier data-center. This provides us with more aggregate system memory across the multiple tiers for cooperative caching. Further, the involvement back-end modules in caching can be possibly extended to the caching support for dynamically changing data [62].

The MTACC scheme is designed with passive cooperative caching modules running on the back-end servers. These passive modules do not generate cache store or retrieve requests themselves, but help the other modules to utilize their pooled memory. In addition, these passive modules do not act as home nodes for meta-data storage, minimizing the necessity for cache request processing overheads on these back-end servers.

In addition, in certain scenarios such as cache invalidates and updates, the back-end servers need to initiate these operations [62]. Utilizing the modules existing on the back-end nodes, the back-end nodes can perform operations like invalidations, etc. efficiently with the help of the closer and direct access to cache to achieve significant performance benefits. Figure 7.3 shows a typical setup for MTACC.

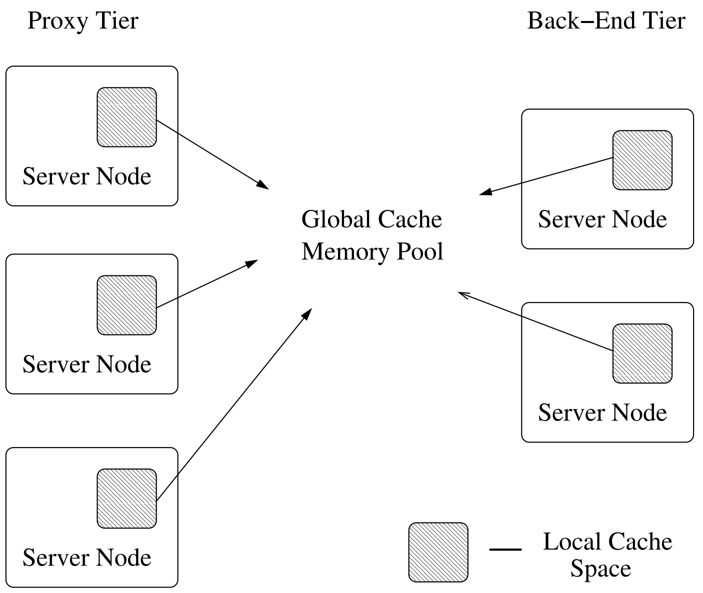


Figure 7.3: Multi-Tier Aggregate Cooperative Caching

7.2.4 Hybrid Cooperative Cache

Though the schemes CCWR and MTACC can achieve good performance by catering to larger working sets, they have certain additional working overhead to remove redundant cache entities. While this overhead does not impact the performance in cases when the working set is large or when the requested file is large, it does impact the performance of the smaller cache entities or smaller working set files to a certain extent.

CCWR adds certain overhead to the basic cache processing. The added lookups for duplicates and the higher cost of swapping make up these overheads. MTACC also adds similar overheads. This larger aggregated cache system size can cause higher overheads for request processing.

To address these issues, we propose the Hybrid Cooperative Caching Scheme. In this scheme, we employ different techniques for different file sizes. To extent possible, smaller cache entities are not checked for duplications. Further, the smaller cache entities are stored and their lookups are performed on only the proxy servers without using the back-end servers. Hence, smaller cache entities are not stored on the passive nodes and are duplicated to the extent possible reducing the effect of the associated overheads.

7.3 Performance Evaluation

In this section, we present a detailed experimental evaluation of our designs. Here, we compare the following levels of caching schemes: (i) Apache default caches (AC) (ii) BCC, (iii) CCWR, (iv) MTACC and (v) HYBCC.

Trace	2 nodes	4 nodes	8 nodes	10 nodes
8k-trace	80M/128M	80M/256M	80M/512M	80M/640M
16k-trace	160M/128M	160M/256M	160M/512M	160M/640M
32k-trace	320M/128M	320M/256M	320M/512M	320M/640M
64k-trace	640M/128M	640M/256M	640M/512M	640M/640M

Table 7.1: Working Set and Cache Sizes for Various Configurations

For our experiments we used 20 nodes with dual Intel Xeon 2.66 GHz processors. InfiniBand network connected with Mellanox InfiniHost MT23108 Host Channel Adapters (HCAs). The clusters are connected using a Mellanox MTS 14400 144 port switch. The Linux kernel version used was 2.4.20-8smp. Mellanox IBGD 1.6.1 with SDK version 3.2 and the HCA firmware version 3.3 was used.

These nodes were setup with two web-servers and with the number of proxy servers varying from two to eight. The client requests were generated from multiple threads on 10 nodes. The web-servers and application servers used in the reference implementation are Apache 2.0.52. All proxy nodes we configured for caching of data. Web server nodes were also used for caching for the schemes MTACC and HYBCC as needed. Each node was allowed to cache 64 MBytes of data for any of the experiments.

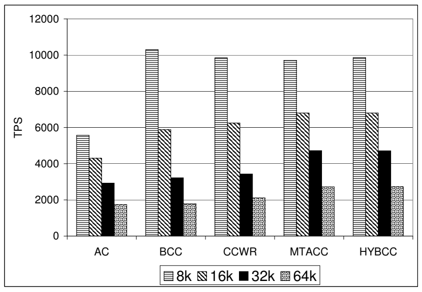
Traces Used: Four synthetic traces representing the working sets in Zipf [86] traces were used. The files sizes in the traces were varied from 8KBytes to 64KBytes. Since the working sets of Zipf traces all have similar request probabilities, a trace comprising of just the working set is seemingly random. The working set sizes for these traces are shown in the Table 7.1. These present us with a number of cases in which the working sets are larger than, equal to or smaller than the total cache space available to the caching system.

7.3.1 Basic Performance Analysis

As an indication of the potential of various caching schemes, we measure the overall data-center throughput. Figures 7.4(a) and 7.4(b) show the throughput measured for the four traces. We see that the basic throughput for all the cooperative caching schemes are significantly higher than the base case of basic Apache caching (AC) - the default single node caching provided by apache.

Impact of Working Set Size: We notice that the performance improvements from the AC scheme to the other schemes show steep improvements when the cooperative caching schemes can hold the entire working set of that trace. For example, the throughput for the cooperative caching schemes for the 8k-trace for two nodes in Figure 7.4(a) are about 10,000 TPS, where as the performance for AC is just above 5000 TPS. This shows a performance improvement of about a factor of two. This is because the AC scheme cannot hold the working set of the 8k-trace which is about 80 MBytes. Since each node can hold 64 MBytes, AC incurs cache misses and two node cooperative caching shows good performance. We see similar performance jumps for all cases where the working set fits in cache. Figure 7.5(a) clearly shows a marked improvement for larger traces (32k-trace and 64k-trace) for MTACC and HYBCC. This benefit comes from the fact that MTACC and HYBCC can accommodate more of the working set by aggregating cache from nodes across several tiers.

Impact of Total Cache Size: The total cache size of the system for each case is as shown in Table 7.1. For each configuration, as expected, we notice that the overall system performance improves for the cases where the working-set sizes are larger then the total system cache size. In particular, the performance of the 64k-trace for the 8 node case achieves a throughput of about 9,500 TPS while using the memory



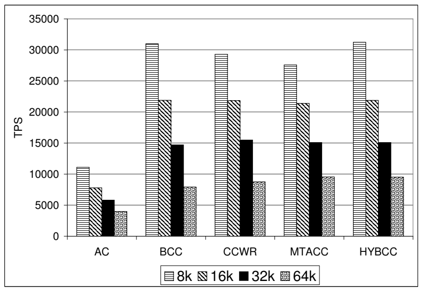
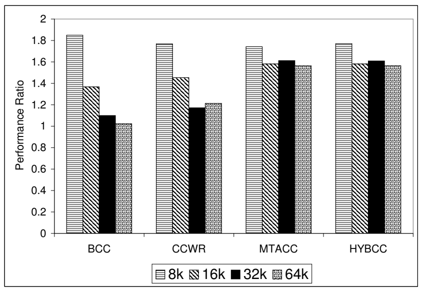


Figure 7.4: Data-Center Throughput: (a) Two Proxy Nodes (b) Eight Proxy Nodes



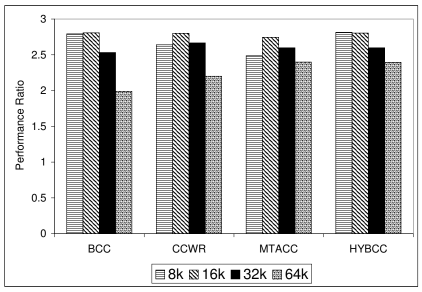


Figure 7.5: Performance Improvement: (a) Two Proxy Nodes (b) Eight Proxy Nodes

aggregated from the web server for caching. This clearly shows an improvement of close to 20.5% improvement over basic caching scheme BCC.

Impact of System Size: The performance of the 8k-trace in Figure 7.5(b) shows a drop in performance for the CCWR and the MTACC cases. This is because as a result of aggregated cache across tiers for MTACC its total system size increases, hence the total overheads for each lookup also increases as compared to CCWR. On the other hand, since HYBCC uses CCWR for small cache entities and MTACC for large cache entities, its improvement ratios of HYBCC in Figure 7.5(b) clearly show

Scheme	Fetch	Store	Validate	Invalidate
BCC	2,1	2,1	1,0	1,0
CCWR	2,2	2,1	1,0	1,0
MTACC	2,2	2,1	1,0	1,0
HYBCC	2,2	2,1	1,0	1,0

Table 7.2: Maximum number of messages required (control-messages/data-messages)

that the HYBCC scheme does well in all cases. It is to be noted that the benefit of HYBCC will increase as the system size increases.

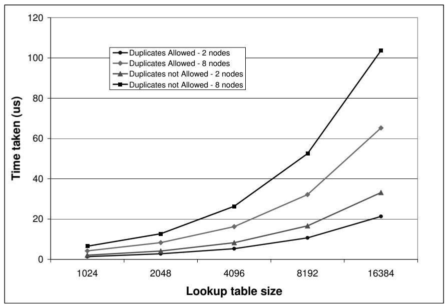


Figure 7.6: Bookkeeping and Lookup delay

We now discuss the performance benefits seen for each of the schemes and analyze the same.

7.3.2 Additional Overheads for Cooperative Caching

Our approaches incur different costs for lookup for different schemes. The primary difference is in the lookup times of schemes with redundancy allowed and schemes without redundancy. Figure 7.6 shows the worst case lookup latency for each request in steady state. We have seen that as the total size of the cache increases with number of nodes the lookup times also increase correspondingly. In addition, searching for redundant copies also incurs additional cost.

The number of network messages required for cache operations is shown in the Table 7.2. We see that the expected worst case number of control and data messages remain the same for all mechanisms with lower redundancy.

7.3.3 Detailed Data-Center Throughput Analysis

In the following sections, detailed analysis is presented for the each scheme to evaluate their effectiveness.

- **AC:** These numbers show the system throughput achievable by using the currently available and widely used simple single node caching. Since all the nodes here take local decisions the performance is limited by the amount of cache available on individual nodes.

- **BCC**: As shown by researchers earlier, the performance of the BCC scheme marks significant performance improvement over the AC scheme. These performance numbers hence represent throughput achievable by basic cooperative caching schemes. In addition, the trends for the BCC performance also show the effect of working-set size as mentioned earlier. We see that as we increase the number of proxy servers, the performance benefit seen by the BCC scheme with respect to AC increases. The performance benefit ratio as shown in Figures 7.5(a) and 7.5(b) clearly shows this marked improvement.

- **CCWR**: From the Figures 7.4(a) and 7.4(b), we observe that the performance for the CCWR method shows two interesting trends: (i) the performance for the traces 16k-trace, 32k-trace and 64k-trace show improvement of up to 32% as compared to the BCC scheme with the improvement growing with higher size traces and (ii) the performance of the 8k-trace shows a drop of about 5% as compared to the BCC scheme. The primary reason for this performance drop is the cost of additional book-keeping required for eliminating copies (as shown in Figure 7.6). We measured this lookup cost for this scheme to be about 5-10% of the total request processing time for a file of 8 Kbytes size. Since this cost does not grow with file size, its effect on larger file sizes is negligible.

- **MTACC**: The main difference between the CCWR scheme and the MTACC scheme is the increase in the total system cache size and the total system meta-data information size. The additional system size improves performance by accommodating more entities in cache. On the other hand, larger lookup table size incurs higher lookup and synchronization costs. These reasons both show effect on the overall performance of the data-center. The 8 node case in Figure 7.4(b) shows that the performance of 8k-trace decreases with MTACC as compared to BCC and CCWR and the performance improves for 16k-trace, 32k-trace and 64k-trace. We observe similar trends for the 2 node case in Figure 7.4(a).

- **HYBCC**: HYBCC overcomes the problems of lower performance for smaller files as seen above by using a hybrid scheme described in Section 7.2.4. In this case, we observe in Figures 7.4(a) and 7.4(b) that the HYBCC scheme matches the best possible performance. Also, we notice that the improvement of the HYBCC scheme over the BCC scheme is up to 35%.

7.4 Summary

The importance of caching as an instrument for improving the performance and scalability of web-serving data-centers is immense. Existing cooperative cache designs often partially duplicate cached data redundantly on multiple servers for higher performance while optimizing the data-fetch costs for multiple similar requests. With the advent of RDMA enabled interconnects these cost estimates have changed the basic factors involved. Further, the utilization of the large scale of resources available across the tiers in today's multi-tier data-centers is of obvious importance.

In this chapter, we have presented cooperative cache schemes that have been designed to benefit in the light of the above mentioned trends. In particular, we have designed schemes that take advantage of RDMA capabilities of networks and the resources spread across the multiple tiers of modern multi-tier data-centers. Our designs have been implemented on InfiniBand based clusters to work in conjunction with Apache based servers. We have evaluated these with appropriate request traces. Our experimental results have shown that our schemes perform up to 35% better than the basic cooperative caching schemes for certain cases and 180% better than the simple single node caching schemes.

We further analyze the performance of each of our schemes and propose a hybrid caching scheme that shows high performance in all our cases. We have observed that simple caching schemes are better suited for cache entities of small sizes and advanced schemes are better suited for the larger cache entities.

CHAPTER 8

WAN FRAMEWORK

In this section we present the details on our WAN communication framework. We briefly describe the possible alternatives RDMA communication over WAN. We further study the tradeoffs of RDMA communication involved in the various WAN scenarios.

8.1 Background and Motivation

Compute clusters have become increasingly popular due to the high performance to cost ratios they offer. Rapid technology advances at largely affordable costs have led to the wide spread deployment of these clusters, with several organizations having multiple cluster deployments. TCP/IP has been the most popular protocol for all inter-node communication requirement. While TCP/IP based communication has its advantages in being the most popular protocol and supporting both across LAN and WAN communication, CPU and memory related costs for driving traditional TCP/IP stacks often impact communication performance and hence limit the scalability and efficiency of the clusters.

To improve communication performance within clusters, modern interconnects like 10 Gigabit Ethernet, Quadrics [68], Myrinet [61] and InfiniBand [4] offer higher network bandwidths and lower communication latencies. In addition, interconnects like InfiniBand include features such as Remote Direct Memory Access (RDMA) that have enabled communication mechanisms providing a significantly higher performance. To extend the benefits of RDMA to the traditional Ethernet-based networks, a new Internet Wide Area RDMA Protocol (iWARP) standard has been recently introduced [32]. The iWARP standard basically allows for zero-copy transfer of data over the legacy TCP/IP communication stacks. Hence iWARP provides for a significantly higher communication performance. Applications need to leverage this available communication performance into better overall application performance. Additionally, the iWARP protocol being based on TCP/IP also allows high performance data transfers across WANs enabling users to run high performance applications in cluster-of-clusters scenarios. Figure 8.1 shows a typical scenario with cluster-of-clusters. Currently several vendors including Chelsio [31], NetEffect [64] and Ammasso provide iWARP capable RNICs.

Further, InfiniBand has also extended its capability to WANs. Obsidian Research Corporation [33] has recently announced products capable of performing InfiniBand communications over WAN. While these modern interconnects have now enabled the use of advanced features like RDMA across WANs, it is important to understand the benefits and limitations that these protocols see in such scenarios.

While these Obsidian IB WAN routers provide limited capability of emulating long distance WAN links, networks without such capabilities cannot be evaluated.

Hence, appropriate evaluation methodologies also need to be established for these emerging cluster-of-cluster scenarios.

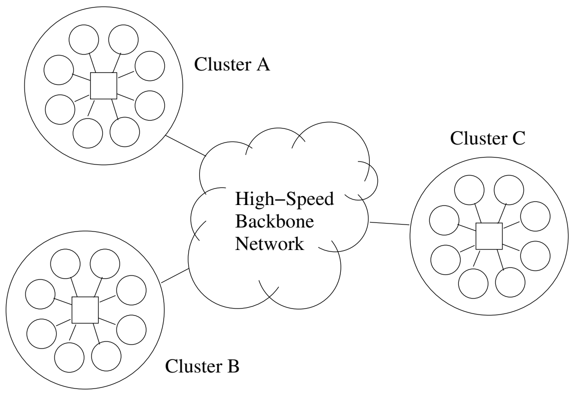


Figure 8.1: Typical Cluster-of-Clusters Environment

In this chapter, we detail our work on evaluation methodology of WAN capable adapters. We first describe our design of NemC - a Network Emulator for Cluster-of-Clusters to handle the emulation of specific characteristics of these emerging cluster-of-cluster scenarios.

8.2 Network Emulator for Cluster-of-Cluster Scenarios

In this section we describe our fine-grain network emulator for cluster-of-cluster scenarios.

8.2.1 Background and Related Work

The cluster-of-clusters environment poses several research challenges including performance, compatibility, security, authentication, etc. However, before addressing such research challenges, one of the foremost critical issues is how to construct the experimental environment of cluster-of-clusters. Since research groups usually do not have the actual backbone networks for cluster-of-clusters, which can be reconfigured with respect to delay, packet loss, etc. as needed, it is hard to carry out practical research over realistic environments. Accordingly, the demand for an efficient way to emulate the backbone networks for cluster-of-clusters is overreaching. Approaches involving simulations and modeling are widely accepted [40, 17, 22]; however, these approaches have the limitations that they cannot run actual software (i.e., applications, middleware, and system software). On the other hand, if we can emulate only the backbone networks running actual clusters, it will provide very close environments to the real-world systems but also give flexibility to change the system parameters, such as network delay, packet loss, etc. For the emulation, a workstation can be configured as a router with multiple Network Interface Cards (NICs), of which each is connected to a cluster. By running a network emulation software that generates artificial network delay, packet loss, etc. on the workstation-based router we can emulate the backbone networks for cluster-of-clusters while running actual software over the clusters in a transparent manner.

Though there are several existing network emulators [28, 44, 72, 80], they are focusing on large scale Wide Area Networks (WANs) such as Internet. However, there are many prominently different characteristics between such WANs and the backbone networks for cluster-of-clusters. For example, the backbone networks usually have a much lower delay than typical WAN environments though the backbone networks have a higher delay than the intra-cluster LAN environments. The emulators that can emulate a millisecond network delay resolution may not be enough to emulate the high-speed backbone networks. In addition, the bandwidth provided by the backbone networks for cluster-of-clusters is higher than the WAN case. Hence the emulator should be able to emulate higher bandwidth networks.

In this context, we present a novel design for emulating the backbone networks of cluster-of-clusters. The emulator named *NemC* (Network Emulator for Cluster-of-Clusters) can support the fine-grained network delay resolution minimizing the additional overheads. We design a new packet scheduling mechanism that performs on-demand scheduling, which is independent on any system timers. Also we minimize the additional overhead by designing it at the kernel-level to emulate high bandwidth networks. In addition to the network delay emulation, current implementation of NemC can emulate packet losses and out-of-order packets. To the best of our knowledge, no research has focused on the network emulation for cluster-of-cluster environments and NemC is the first emulator to address this.

8.2.2 Design and Implementation of NemC

In this section, we detail the design and implementation of our network emulator for cluster-of-clusters named NemC. NemC is implemented using the `netfilter` hooks provided by Linux, which can be dynamically inserted to the kernel's chain of packet processing. A run-time loadable kernel module which runs on Linux-based routers is used to perform all operations. Its design does not require any kernel modifications. The current implementation can insert network delay with fine-grained resolution, packet drops, and out-of-order packets.

Figure 8.2 shows the overall design of NemC. As shown in the figure, NemC consists of four components: (i) NemC netfilter, (ii) NemC scheduling demon, (iii) NemC kernel module and (iv) NemC user applications. The NemC netfilter intercepts the packets arrived at the router node after the IP routing decision. Based on the parameters set by the user applications, the NemC netfilter can drop packets, generate out-of-order packets, or introduce network delays. These parameters can be controlled at run-time by using the NemC user applications. The NemC scheduling daemon is a user-level process, which requests the netfilter to search the packets that has been sufficiently delayed and reinject them into the network. The kernel module takes care of insertion of the netfilter in the initialization phase but also provides access to the internal data structures and parameters of the NemC netfilter to the scheduling daemon and the user applications.

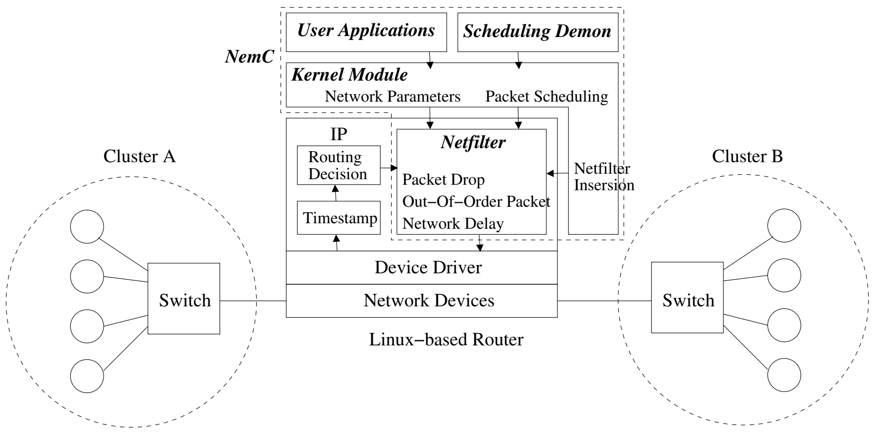


Figure 8.2: Overall Design of NemC

Packet Scheduling for Fine-Grained Delay Resolution:

The backbone networks for cluster-of-clusters have low network delay compared to general WANs such as Internet. To emulate such networks, the emulator is required to support fine-grained delay resolution. The delay resolution of a network emulator is mainly decided by the triggering mechanism of packet scheduling. The packets delayed more than the given time, *net_delay*, at the router node are reinjected into the network by the packet scheduling routine. The most widely used mechanism to trigger the packet scheduling is to invoke the scheduling routine for every timer interrupt. This mechanism is simple to design and implement; however, since it depends on the system timer resolution, it may not be able to support fine-grained delay resolution. For example, if the network emulator uses Linux timer then it can support only 10ms (with kernel version 2.4) or 1ms (with kernel version 2.6) delay resolution, which is too coarse-grained to emulate the backbone networks for cluster-of-clusters. On the other hand, if the network emulator directly uses a hardware timer in the system, the interrupt can be generated very high frequency and can delay the actual packet processing.

To overcome these limitations of the timer based mechanism, we suggest the on-demand packet scheduling mechanism. In this mechanism, the packet scheduling routine is triggered by either incoming packet or scheduling daemon. That is, whenever there is a new packet arrived at the router node, it triggers the packet scheduling routine, while the user-level scheduling demon continually tries to invoke the packet scheduling routine if there are no packets waiting to be processed in the protocol stacks and the system is idle. It is to be noted that the user-level scheduling daemon has lower priority than the kernel-level packet processing context. Thus, if packets arrive at the router node in a bursty manner the scheduling routine will be invoked very frequently by those packets. On the other hand, if packets arrive intermittently then the user-level daemon will continuously trigger the packet scheduling. In this manner, we can trigger the scheduling routine as much as possible (i.e., in a fine-grained mode) without any effect on the actual packet processing of the protocol stacks. In this mechanism, since both newly arrived packets and the user-level daemon invoke the scheduling routine, which accesses the same data structures in the NemC netfilter, we guarantee that only one can access the data structures at a time by locking. We use the time stamp in the `sk_buff` data structure of the Linux kernel to calculate the total time duration spent by the packet in the router node.

Low Overhead Emulation for High Bandwidth Support:

Another important characteristic of the backbone networks for cluster-of-clusters is high bandwidth. To emulate the high bandwidth networks, we need to address two critical issues: i) delay cascading and ii) emulation overhead.

If an emulator holds a packet for a given time to add a delay without yielding the CPU resource, this delay will be cascaded to the next packets that have been already arrived at the router node. For example, if an emulator is implemented as a high priority kernel-level process and polls the timer occupying the CPU resource, the delay can be cascaded on subsequent packets. To avoid this delay cascading problem, we place the packets that need to be delayed into a queue and immediately return the context to the original routine. The packets queued are re-injected by the packet scheduling mechanism described earlier.

On the other hand, higher emulation overheads can reduce the effective bandwidth between the clusters in the experimental systems, which is a undesired side effect. Broadly, the emulator can be implemented at the user-level or the kernel-level. The user-level emulation requires two data copies between user and kernel buffers for each packet. This copy operation is a well-known bottleneck of packet processing. Hence, our network emulator is designed at the kernel-level to prevent any additional data copy.

Packet Drop and Out-of-Order Packet Generation:

Since the backbone networks for cluster-of-clusters can use store-and-forward networks there can be packet drops because of network congestion. To emulate such case, we generate packet drops based on the packet drop rate value, *drop_rate*, given by a NemC user application. NemC chooses a packet randomly for every *drop_rate* packets and simply drops this packet freeing all the resources occupied by this packet.

Out-of-order packets can occur in cluster-of-clusters due to multi-path and adaptive routing. To emulate such case, we generate out-of-order packets using a given out-of-order packet generation rate, *ooo_rate*, and a delay for out-of-order packets, *ooo_delay*. These values are set by a NemC user application. It is guaranteed that the value of *ooo_delay* is always larger than that of *net_delay*. NemC chooses a packet randomly for every *ooo_rate* packets and delays this packet as much as *ooo_delay*. Since this packet has been delayed more than other packets it becomes an out-of-order packet if the packet interval between this packet and the next is smaller than *ooo_delay*.

Experimental WAN Setup

Our performance evaluations have been performed on the experimental system shown in Figure 8.3, where we have two different IP networks and they are connected through a workstation-based router emulating WAN. The end nodes are SuperMicro SUPER P4DL6 nodes - each has dual Intel Xeon 2.4GHz processors with a 512KB L2 cache and an Ammasso 1100 Gigabit Ethernet NIC. The router node is a SuperMicro

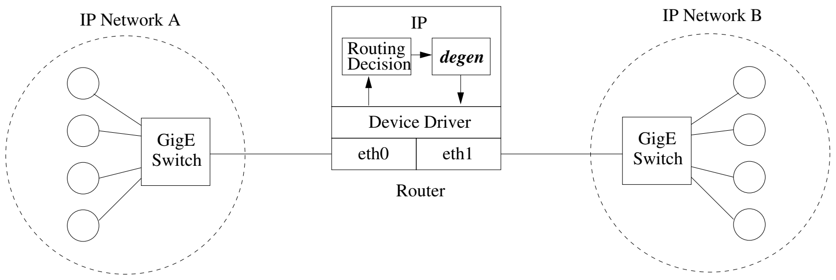


Figure 8.3: Experimental WAN Setup: A Simplistic View

SUPER X5DL8-GG workstation with dual Intel Xeon 3.0GHz processors, 512KB L2 cache, and 2GB of main memory. The router node is connected to IP networks A and B with Broadcom BCM5703 and Intel PRO/1000 Gigabit Ethernet NICs, respectively. All nodes use Linux kernel version 2.4.20. The switches used for each IP network are Foundry FastIron Edge X448 Switch and Netgear GS524T Gigabit Switch, respectively.

To reflect the characteristics of high latency in the WAN environment, we utilize a simplistic version of our network emulator NemC (named *degen*) to introduce delays. It delays the forwarding of each packet on the router by a given value after the corresponding routing decision has taken place as shown in Figure 8.3.

8.3 Evaluation of iWARP protocols

In this section, we compare RDMA with traditional TCP/IP sockets on WAN environments with respect to (i) latency, (ii) computation and communication overlap, (iii) communication progress, (iv) CPU resource requirements, and (v) unification of communication interface.

We perform our experiments over the Ammasso Gigabit Ethernet [1] adapter as described in the following subsection.

8.3.1 Overview of Ammasso Gigabit Ethernet NIC

The iWARP capable Ammasso Gigabit Ethernet NIC [1] provides an implementation of the RDMA over TCP/IP enabled NIC. Based on the RDMA Protocol Verbs (RDMAVS 1.0) [45] specified by the RDMA consortium, the RDMA interface of the Ammasso Gigabit Ethernet NIC provides low latency and high bandwidth on Gigabit Ethernet network. As shown in Figure 8.4, Ammasso Gigabit Ethernet NIC supports the legacy sockets interface and the Cluster Core Interface Language (CCIL) interface.

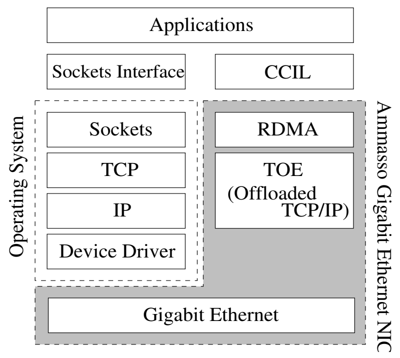


Figure 8.4: Protocol Stacks on Ammasso Gigabit Ethernet NIC

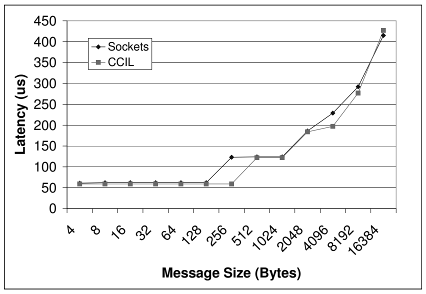
The CCIL interface is an implementation of the Verbs layer to utilize RDMA over IP. The CCIL interface uses the RDMA layer and offloaded TCP/IP on the NIC to transmit the data. On the other hand, the sockets interface still sends and receives the data through the traditional TCP/IP implemented in the operating system kernel. The CCIL interface enables zero-copy and kernel-bypass data transmission.

8.3.2 Communication Latency

The basic communication latency is one of the most important performance metrics. In this section, we carry out the latency test in a standard ping-pong fashion to report one-way latency. The client sends a message and waits for a reply message of the same size from the server. The time for this is recorded by the client and it is divided by two to find out one-way latency. In the case of RDMA, we use RDMA write to transmit the data.

Figure 8.5(a) shows the results of latency without the delay by *degen*. We can see that the latencies of RDMA and sockets are almost the same regardless of the message size even without the delay. It is because the latency added by the default experimental setup described in Section 8.2.2 is relatively large compared to the overhead on the end nodes. Although RDMA achieves a zero-copy data transmission, since the MTU size is only 1500 Bytes, we cannot expect a large benefit with respect to the latency. In the case of messages larger than the MTU size, TCP constructs several segments so that each segment can fit into the MTU sized IP fragment. Hence the transmission of the segments are pipelined and we can obtain the benefit of the zero-copy only for the first segment in a high delay environment. However, if the network delay is smaller than the processing overhead of the end node, the zero-copy transmission helps to deliver low latency.

Figure 8.5(b) shows the latency varying the network delay with 1KB message. As we can see, RDMA and sockets report almost the same latency. This reveals that



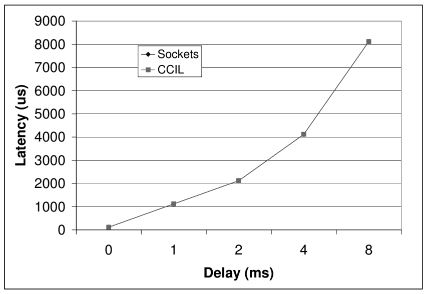


Figure 8.5: Communication Latency: (a) Varying Message Size and (b) Varying Network Delay

the basic communication latency is not an important metric to distinguish RDMA from the sockets in a high delay WAN environment because the overheads on the end nodes is too small compared with the network delay in the order of milliseconds.

8.3.3 Computation and Communication Overlap

In this section, we evaluate how well the process is able to overlap computation with communication. In our test shown in Figure 8.6(a), the client performs a computation loop that does a dummy work for a given time in between ping (i.e., sending) and pong (i.e., receiving) operations of the latency test described in Section 8.3.2. We evaluate the computation and communication overlap ratio with $(Computation\_Time)/(Total\_Time)$ on the client side. Thus a value closer to 1 represents a better computation and communication overlap. It is to be noted that our sockets latency test is using non-blocking sockets to maximize the computation and communication overlapping.

Figure 8.7(a) shows the overlap ratios with varying computation time and without network delay. The message size used is 1KB. As we can see, RDMA can achieve better overlap even with smaller computation time compared to the sockets. This is because RDMA provides asynchronous communication interface. In addition, we do not need to utilize CPU resources to get the data from remote node because the remote node uses RDMA write. On the other hand, in the case of the sockets, some of the receiving functions of the sockets (e. g., kernel buffer to user buffer copies) are performed in the context of the application process. As a result, these receiving functions cannot be overlapped with actual application processing while some other functions (e. g., interrupt handling and bottom half) can run in parallel with the application processing by another CPU in a SMP system. Further, the offloaded TCP/IP of the RDMA case increases the chance of overlapping between packet processing overhead and computation overhead.

Figure 8.7(b) shows the overlap ratio values of RDMA and sockets for varying network delay, where we have fixed the computation time to 242ms and message size to 1KB. It can be observed that the difference between RDMA and sockets reduces with large network delays. It is mainly because the network delay is the dominant overhead and it can be overlapped with computation time regardless of RDMA or sockets. Since the packet processing overhead of end nodes is not a critical overhead anymore on a high delay WAN, its overlapping with other overheads does not affect much to the overlap ratio. However, still we can see that RDMA can provide better overlap than sockets for delays in order of a few milliseconds.

8.3.4 Communication Progress

In many distributed systems, we often observe the communication pattern that a node requests some data to a remote node and it returns the data. This operation

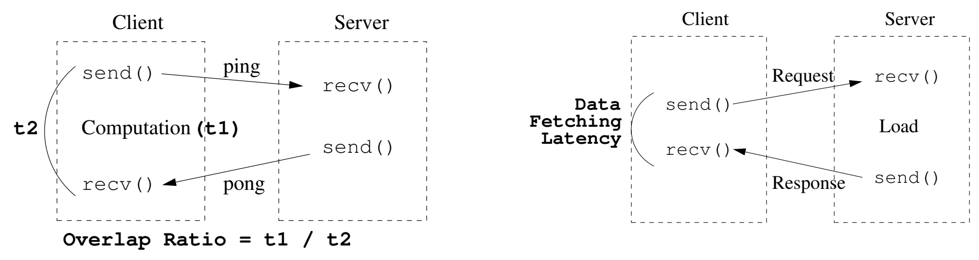


Figure 8.6: Pseudo Code for Benchmarks: (a) Overlap Test and (b) Communication Progress Test

can be implemented with either by using a pair of send and receive calls or by using RDMA read. Moreover, the remote node can be heavily loaded because of burst requests on the data or CPU intensive computations. To compare the performance of RDMA read with the traditional sockets in this scenario, we simulate the load on the remote node by adding a dummy loop running for a given time. We measure the latency to get 1KB of data from remote node as shown in Figure 8.6(b).

Figure 8.8(a) shows the data fetching latency varying the load on the remote node, where the load is represented as the response delay. Since RDMA read does not require any involvement of remote process for data transmission, it can read data from remote memory without any impact from the load on the target. It is to be noted that the sockets interface is not able to deliver good performance as the load increases. It is because the overall communication progress of the sockets highly depends on that of both sides (sender and receiver).

Figure 8.8(b) shows the data fetching latency varying the network delay for a fixed response load of 16ms. With increase in the network delay, both of the interfaces (sockets and RDMA) perform similarly because the network delay tends to dominate the performance costs more than the load on the remote node. However, it can still be seen that RDMA is more tolerant of the network delay achieving better communication progress.

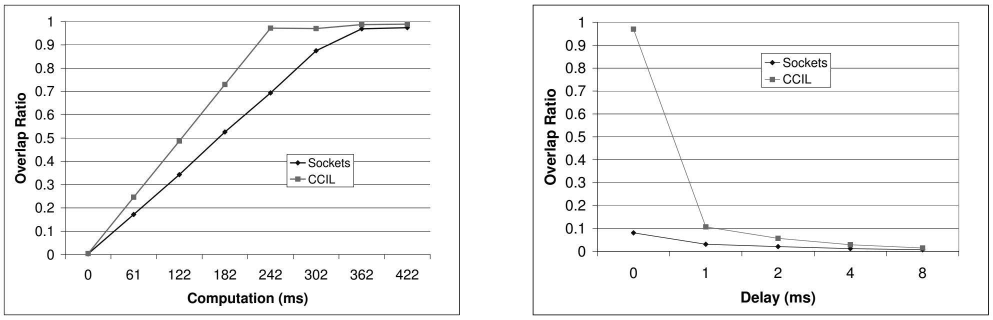


Figure 8.7: Overlap Ratio: (a) Varying Computation Time and (b) Varying Network Delay

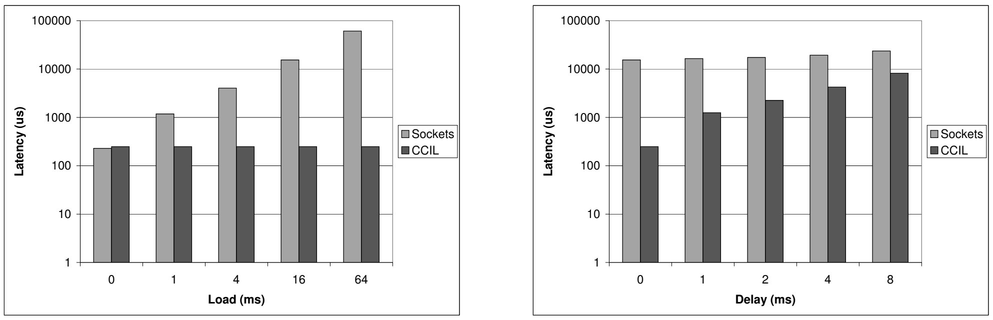
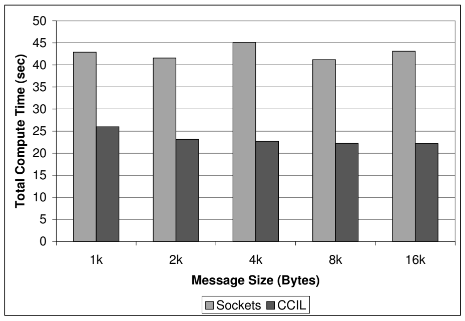


Figure 8.8: Communication Progress: Data Fetching Latency (a) Varying Load and (b) Varying Network Delay



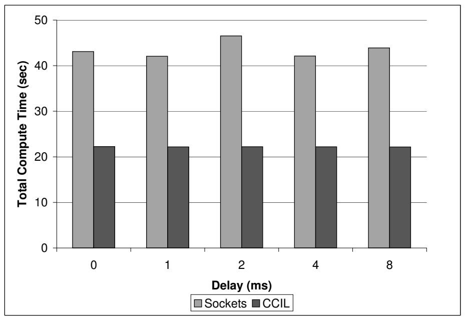


Figure 8.9: Impact on Application Execution Time: (a) Varying Message Size and (b) Varying Network Delay

8.3.5 CPU Resource Requirements

To measure the effect of CPU resource requirements for communication on the application performance, in this experiment, we run an application on a server node that performs basic mathematical computations while 40 clients continuously send data to this server. We report the total execution time of the application under this scenario.

Figure 8.9(a) shows the application execution time varying the message size with no added network delay. As we can see, the execution time with background sockets communications is very high for all message sizes while it does not have any significant performance impact with background RDMA communication.

Figure 8.9(b) shows the execution time of the application varying the network delay with 16KB message size. We can observe that the background sockets communication significantly degrades the application performance even on high delay WANs. The reason is that, in the case of sockets, the remote CPU is involved in the communication effort, with packet processing and interrupt handling at the kernel level and receive request posting at the application level. This results in stealing of the CPU resource from the computing application. However, RDMA can place the data to the remote memory without any CPU requirement on the remote node. Hence we can see in the figure that the application execution time is not affected by RDMA and constant for all network delays. This reveals that RDMA has a strong potential of saving the CPU resource on the server side even on a high delay WAN environment.

8.3.6 Unification of Communication Interface

In addition to direct performance metrics detailed in the previous sections, WAN and LAN interoperability is a very important feature of RDMA over IP and IB-WAN. Scenarios in which several inter-cluster and intra-cluster nodes communicate with each other need common communication interfaces for the job. Traditionally, the sockets over TCP/IP has been the main interface with this feature. However, with RDMA over IP (or IB-WAN), this interoperability can be achieved and it can be achieved with all the benefits described in the previous sections. Further, RDMA over IP performs significantly better then sockets for within LAN communications.

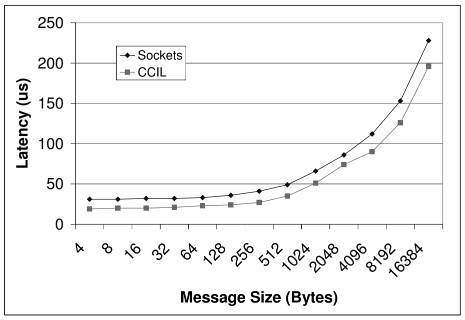


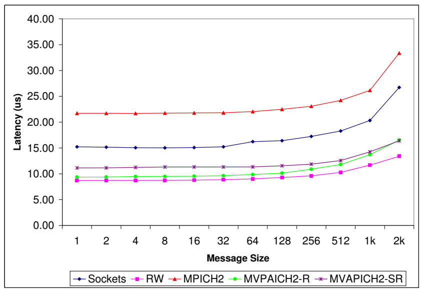
Figure 8.10: Communication Latency within a LAN with varying Message Sizes

Figure 8.10 shows the latency of CCIL and Sockets communications within a LAN. We see that the small message latency differs by almost 50% with RDMA being better. Hence, RDMA over IP benefits these multi-cluster applications with better communication both over the WAN as well as in the LAN. It is to be noted that the benefit of zero-copy with RDMA is not significant even in the LAN as we have discussed in Section 8.3.2. However, 10 Gigabit Ethernet [46] is expected to provide very low propagation delay within LAN and show the benefit of zero-copy on the communication latency with large messages.

8.3.7 Evaluation of the Chelsio 10 Gigabit Ethernet Adapter

In this section, we show the performance potential of modern iWARP capable 10 Gigabit Ethernet interconnects. We have designed MPI-iWARP to study the issues in designing iWARP based communication middleware. We further use our MPI-iWARP implementation to evaluate the Chelsio [8] iWARP capable 10 Gigabit Ethernet adapter. The details of our MPI-iWARP design are available in [63]. The main issues in the design of MPI-iWARP include RDMA-CM based connection management and iWARP protocol's requirement for the first message to be initiated from client to server. The implementation is released as part of MVAPICH2 [52].

Experimental Testbed: For all our experiments we have used a two node cluster equipped with nodes having two quad core Intel Xeon 2.33GHz Nodes and a memory of 4GB each. These systems are equipped with a Chelsio T3B 10 GigE PCI-Express adapters (Firmware Version 4.2) plugged into an x8 PCI-Express slot. The Chelsio adapters are connected through a 24 port Fulcrum 10 GigE evaluation switch [3]. The



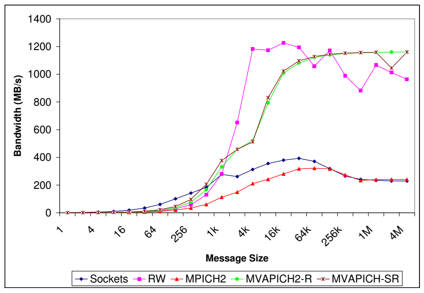


Figure 8.11: Verbs level Performance of 10 GE/iWARP: (a) Latency and (b) Bandwidth

MTU used for the path is 9000 bytes. The software stack we have used is OFED 1.2 rc4 and the operating systems is RH4 U4. MPICH2 1.0.5p3 is used for comparisons.

We present the performance numbers for the following: (i) MPICH2: the basic MPI-2 implementation over TCP/IP [47], (ii) MVAPICH2-R: the MPI-iWARP implementation using MVAPICH2's RDMA fast path, (iii) MVAPICH2-SR: the MPI-iWARP implementation without RDMA fast path and (iv) MVAPICH2-1SC: the MPI-iWARP implementation with RDMA based direct one-sided operations enabled. Complete set of MPI-level results showing the performance of MPI-iWARP are available in our work [63].

Figure 8.11(a) shows the basic latencies that we observe. The latency for the verbs level RDMA write operation over the T3 adapter is about 6.49 microseconds which is quite lower than the basic sockets over TCP/IP number which is about 22.3 microseconds. In thic context we also show the numbers for a popular RDMA capable MPI library MVAPICH2 and compare it with sockets based implementation of MPICH2. The corresponding latency for MPICH2 is about 27.84 microseconds and the latencies for MVAPICH2-R and MVAPICH2-SR are 6.89 us and 8.43 us, respectively. As we clearly observe, MVAPICH2-R adds a minimal overhead to the basic RDMA write latency. The difference in the performance of MVAPICH2-R and MVAPICH2-SR is the absence of RDMA fast path in the latter. Further we also note that the latency observed by the MVAPICH2-R is about 75% better than the latency observed by MPICH2. It is to be noted that large messages are bandwidth bound and hence for clarity of presentation we show the latencies of only the small messages.

The peak bandwidth that we observe for our test bed is about 1287 Million Bytes per second (MB/s) using the verbs level RDMA write operations. MPICH2 shows a peak bandwidth of about 895 MB/s out of a maximum bandwidth of 1034 MB/s that the sockets interface offers. The MVAPICH2-R and MVAPICH2-SR implementations both offer a peak bandwidth of about 1231 MB/s. The performance gain that we observe for MPI-iWARP variations over MPICH2 is about 37%. Figure 8.11(b) shows the basic bandwidth numbers.

8.4 Evaluation of InfiniBand WAN

In this section we present a brief evaluation of various protocols over IB WAN. Figure 2.3 shows our experimental setup for IB WAN.

8.4.1 Evaluation Methodology

In order to study and analyze the performance of IB communication and IO middleware, we first perform a basic low-level evaluation of IB protocols. These results provide a base line for understanding the results for higher level protocols. We perform all the tests with varying WAN delays. We then evaluate and examine the performances of IPoIB (with both RC and UD transports). For all these scenarios, we perform basic tests followed by optimized tests such as parallel stream tests.

Experimental Testbed: In our experiments we use the following two clusters connected by a pair of Obsidian Longbow XRs: (i) *Cluster A* consists of 32 Intel Xeon dual 3.6 Ghz processor nodes with 2GB of RAM and (ii) *Cluster B* consists of 64 Intel Xeon Quad dual-core processor nodes with 6GB RAM. Both the clusters are equipped with IB DDR memfree MT25208 HCAs and OFED 1.2 [37] drivers were used. The OS used was RHEL4U4. The WAN experiments are executed using nodes from each of the clusters as shown in Figure 2.3.

8.4.2 Verbs-level Performance

In this section, we use the IB verbs-level tests (*perftests*) provided with the OFED software stack to evaluate the performance of the basic IB protocols in cluster-of-clusters scenarios. The experiments evaluate the latency, bandwidth and bidirectional bandwidth between the nodes of the two clusters shown in Figure 2.3.

The Obsidian Longbows are capable of providing full bandwidth at SDR rates. We measure the bandwidth performance across our clusters (with increasing network delays) using RC and UD transports, respectively.

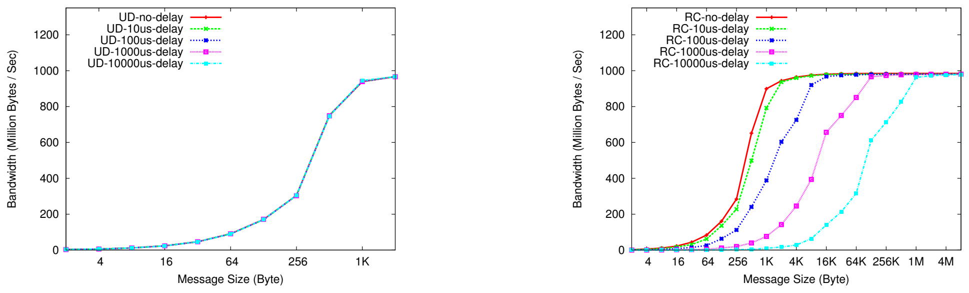


Figure 8.12: Verbs level Throughput over IB WAN using (a) UD (b) RC

Verbs-level UD Bandwidth

In this experiment, we utilize *perftests* to measure the Send/Recv UD bandwidth with varying network delays. We observe that the bandwidth seen in this context is independent of the network delay. We achieve a peak bandwidth of about 967 MillionBytes/sec for a message size of 2k in all cases. This is primarily due to the fact that UD bandwidth tests do not involve any acknowledgements from the remote side and the data can be pushed at the full rate possible. Figure 8.12(a) which shows the UD bandwidth performance, indicates that UD is scalable with higher delays. It is to be noted that higher level protocols using UD transport will need to include their own reliability/flow control mechanisms (such as message acks, etc.) which can impact the performance.

Verbs-level RC Bandwidth

Figure 8.12(b) shows the bandwidth using RC transport mode, with varying delay between the clusters. We observe a peak bandwidth of about 984 MillionBytes/sec in all cases. However, the bandwidth observed for small and medium messages is progressively worse with increasing network delays. i.e. in order to leverage the high bandwidth capability of the IB WAN connectivity under higher network delays, larger messages need to be used. This is due to the fact that RC guarantees reliable and in-order delivery by ACKs and NACKs. This limits the number of messages that can be in flight to a maximum supported window size. While using larger messages, the pipeline can be filled with fewer messages, so it is seen that larger messages do quite well with larger delays. Higher level applications can fill the message transmission pipelines well in several different ways including message coalescing, overlapping multiple streams, etc. We observe similar trends with bidirectional bandwidth.

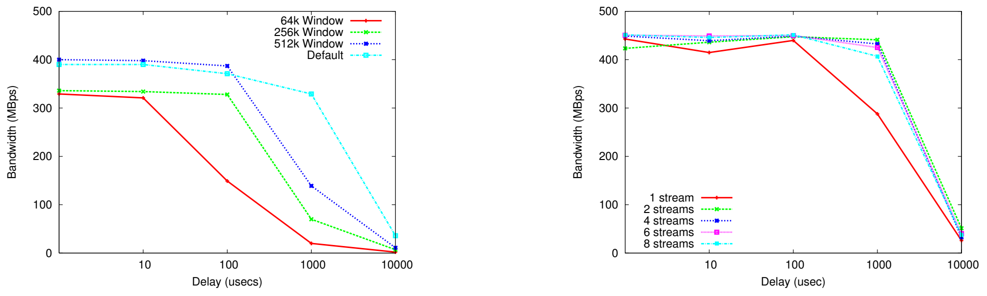


Figure 8.13: IPoIB-UD throughput: (a) single stream (b) parallel streams

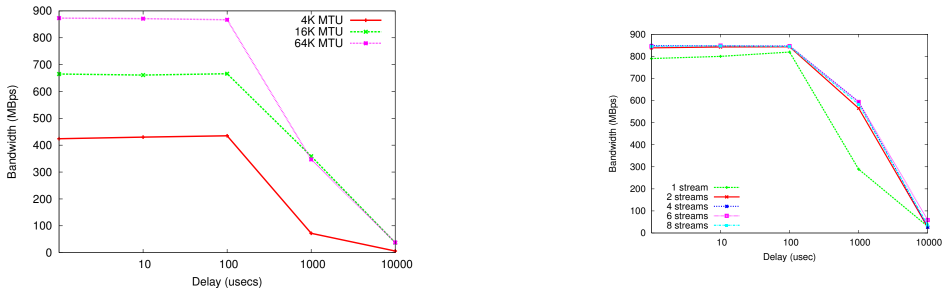


Figure 8.14: IPoIB-RC throughput over WAN: (a) Single Stream (b) Parallel Streams

8.4.3 Performance of TCP/IPoIB over WAN

In this section, we aim to characterize the IPoIB throughput and provide insights to the middleware and application design in the cluster-of-clusters scenarios. Four main factors affect the bandwidth performance, i.e., MTU size, the TCP buffer size, the number of parallel streams and the WAN delays. Therefore, we vary these parameters in the following experiments. Messages with size 2M are used in all the experiments.

IPoIB UD Bandwidth

We evaluate the IPoIB bandwidth using the UD transport with varying WAN delays in both the single-stream and the parallel streams tests. Also, we vary the protocol window sizes in the single-stream experiment and the number of connections in the parallel stream experiment. The results are shown in Figures 8.13 (a) and (b), respectively. The MTU size used for IPoIB UD is 2KB.

From Figure 8.13(a), we see that larger bandwidth is achieved with larger window sizes. It is well known that TCP needs larger window sizes in order to achieve good bandwidth over large bandwidth networks. However, when the WAN delay increases, we observe that the performance of all the cases degrades. It is to be noted that the peak bandwidth that IPoIB UD achieves is significantly lower than the peak verbs-level UD bandwidth due to the TCP stack processing overhead. Overall, the default window size (>1M) in Figure 8.13(a) shows good performance in most cases. Thus, we use this default window size in all of the following experiments.

In order to improve the overall bandwidth performance, we measure the parallel stream bandwidth with various WAN delays as shown in Figure 8.13(b). We see that by using more streams, significant improvements (up to 50%) are achieved in the higher delay scenarios. We observe that the peak IPoIB-UD bandwidth can be sustained even with the delay of 1ms using multiple streams. This is because of the fact that higher number of TCP streams lead to more UD packets with independent flow control (at TCP level), allowing for better utilization of the IB WAN long haul pipe, i.e. there are more outstanding packets that can be pushed out from the source at any given time frame.

IPoIB RC Bandwidth

For the IPoIB using RC transport mode, we also evaluate the single-stream and the parallel stream bandwidth with various WAN delays. One significant advantage of using RC transport mode for IPoIB is the that RC can handle larger packet sizes. This has the following advantages: (i) larger packets can achieve better bandwidth and (ii) per byte TCP stack processing decreases.

As expected in Figure 8.14(a), we see that the best bandwidth of 890 Million-Bytes/sec is achieved with largest MTU size of 64KB (the maximum allowed for an IP packet). This is significantly higher than the bandwidth achieved for IPoIB-UD. That is because the IPoIB-UD test has an MTU size of just 2KB, which means that more packets need to be transferred for the same amount of data and correspondingly more overhead is introduced. In addition, the number of packets required to utilize the WAN link bandwidth fully is significantly higher. On the other hand, we also observe that the bandwidth drops sharply with the longer WAN delay (i.e., larger than 100 us) in this case. This drop corresponds to the drop of verbs level bandwidth for 64K message sizes (at 1000us delay) as seen in Figure 8.12(b) as well.

As in the earlier section, we measure the parallel stream bandwidth of IPoIB-RC. The results are shown in Figure 8.14 (b). We observe the similar trend that with two or more connections, the bandwidth performance can be better sustained across a wider range of cluster separations. Hence, applications with parallel TCP streams have high potential to maximize the utility of the WAN links.

8.5 Summary

In this chapter, we have presented our network emulator for cluster-of-clusters. Further, we have evaluated several WAN scenarios including IB WAN and iWARP with Ammasso and Chelsio NICs. Our results have shown that applications usually absorb smaller network delays fairly well. However, many protocols get severely impacted in high delay scenarios. Further, we have shown that communication protocols can be optimized for high delay scenarios to improve the performance. In particular, the benefits of RDMA in terms of communication latency and communication progress are significantly diminished on long delay links, while CPU utilization remains an important benefit. Hence, our experimental results show that optimizing communication protocols (i.e. WAN-aware protocols), transferring data using large messages and using parallel data streams (upto 50% improvement for high delay networks) is necessary for the cluster-of-cluster scenarios.

CHAPTER 9

ADVANCED DATA TRANSFER SERVICE

Ever increasing needs in High End Computing (HEC) and cost effectiveness of high performance commodity systems have led to massive deployments of highly capable systems on a global scale. This tremendous increase in compute and storage capabilities has necessitated bulk data transfers among various storage systems and/or compute systems that are often physically separated. Typical uses for such large scale data transfers include distribution of scientific data-sets, content replication, remote site backup, etc. Traditionally, File Transfer Protocol (FTP) [69] has been used for handling such bulk data transfers. In this chapter, we present our advanced data transfer service that utilizes zero-copy transfers to achieve high performance.

9.1 Background and Related Work

Large data transfers have traditionally been explored in the context of FTP. While FTP provides the basic data transfer functionality, it also suffers from several critical performance overheads. In particular, since FTP is based on TCP/IP, it inherits the fundamental limitations imposed by TCP/IP itself, such as high CPU utilization, multiple memory copies and limited peak bandwidths achieved over high delay links. Researchers in [11, 12] have looked at several improvements to FTP for optimizing performance. Techniques such as TCP tuning, striped access, persistent data connections, etc. have been proposed for providing improved performance. However, these approaches still bank on TCP/IP (along with its limitations) for data-transfers and fail to achieve good performance for high-delay, high-bandwidth links.

In order to drive higher bandwidth performance, approaches utilizing *"TCP unfriendly"* mechanisms such as multi-streaming, FTP over UDP or SCTP, etc. [21, 77, 50, 11] have recently gained prominence. Even though these approaches achieve better bandwidth as compared to TCP based approaches, they still suffer from multiple memory copies and high CPU overheads.

On the other hand, a rapid growth of modern high performance networking interconnects, such as InfiniBand (IB) and 10 Gigabit Ethernet/iWARP, have revolutionized the communication capabilities of modern systems. As described earlier, with the advent of InfiniBand WAN and iWARP, communication libraries are now capable of zero-copy communication over WAN. This presents the scope for designing high performance communication protocols in wide area networks as well. However, in order to maximize the possible benefits of the IB WAN capabilities, the data transfer mechanisms required for FTP need to be designed carefully, which presents a considerable challenge.

In this chapter, we present a novel zero-copy approach for bulk data transfers across IB WAN. In our approach, we design a high performance Advanced Data Transfer Service (ADTS) that utilizes zero-copy capabilities of modern networks to achieve scalable and efficient data transfers. We leverage the performance of the ADTS layer to design a high performance FTP client/server library. We further investigate the issues involved and study the benefits of our approach in various IB WAN scenarios.

9.1.1 File Transfer Protocol

In this section we provide a brief background on the File Transfer Protocol (FTP).

FTP [69] was initially proposed to promote file sharing and the implicit use of remote computers. It is an application level protocol used to copy files between the local machine (*user*) and the remote machine (*server*) over the network, and fundamentally relies on the TCP/IP protocol. The typical FTP model, as shown in Figure 9.1, includes control connection and data connection between the user and the server FTP processes for communicating the control commands and the data (file) respectively. The FTP server plays a passive role of listening on the specific port when started, and the user initiates a connection which may involve negotiation of authentication and certification with the server. After that, control commands can be transferred over the connection. The data connections are established as needed for the data transfers.

Through the years since the release of the protocol, FTP has seen a variety of extensions and improvements [6, 56, 55, 11, 77, 21, 50], largely due to the explosive development of Internet and data-intensive applications. This trend is expected to grow with the availability of newer networking technologies and other related approaches.

Performance Limitations of Existing FTP Mechanisms

In this section we briefly discuss the main limitations[2] in the communication performance of popular FTP designs: (i) the standard FTP over TCP/IP, (ii) FTP over UDP and (iii) FTP over SCTP.

FTP over TCP/IP: TCP/IP based data transport imposes certain well known limitations on raw communication performance. The fundamental limitations in the current context are as follows: (i) TCP's inability to achieve good bandwidth for high-delay, high-bandwidth links, (ii) high CPU utilization required for TCP processing and (iii) multiple memory copies causing high memory bandwidth utilization and increased end to end latency. These limitations are inherited by all FTP transfers based on TCP as well.

In order to address the first limitation, researchers have proposed a multi-stream approach. Such an approach requires multiple TCP connections, requiring additional resources at the server, thereby limiting the scalability of the server. Further, multi-stream approaches (along with UDP based data transfers) are considered *"TCP Unfriendly"* [50]. The use of TCP Offload Engines (TOE) have been proposed to alleviate the second limitation. However, this approach still incurs overheads including multiple memory copies and hence higher end-to-end latencies.

FTP over UDP/IP: UDP based data transfers suffer from both (i) high CPU utilization and (ii) high memory bandwidth utilization due to multiple memory copies.

[2]It is to be noted that researchers have identified certain limitations such as having a separate control and data connection, sequential nature of command exchanges, etc. in the FTP standard. These issues are orthogonal to our approach and will not be addressed in this chapter. The main focus of our approach is to alleviate communication performance overheads.

However, UDP based transfers are capable of achieving higher bandwidths as compared to TCP due to lack of flow control and congestion control for UDP transfers. FTP mechanisms based on UDP inherit all its limitations.

FTP over SCTP/IP: Data transfers using SCTP (Stream Control Transmission Protocol)[7] incur limitations similar to the those mentioned above for TCP/IP. The primary benefit of utilizing SCTP is the capability of managing multiple streams without much of the overhead as seen for TCP. In particular, high bandwidth through multiple streams can be achieved with fewer connections. However, FTP based on SCTP also suffers from the above mentioned CPU utilization overhead and memory copy overheads.

In this chapter, we address these limitations by leveraging the zero-copy capabilities of modern interconnects. Our design alternatives and optimizations are presented in the following sections.

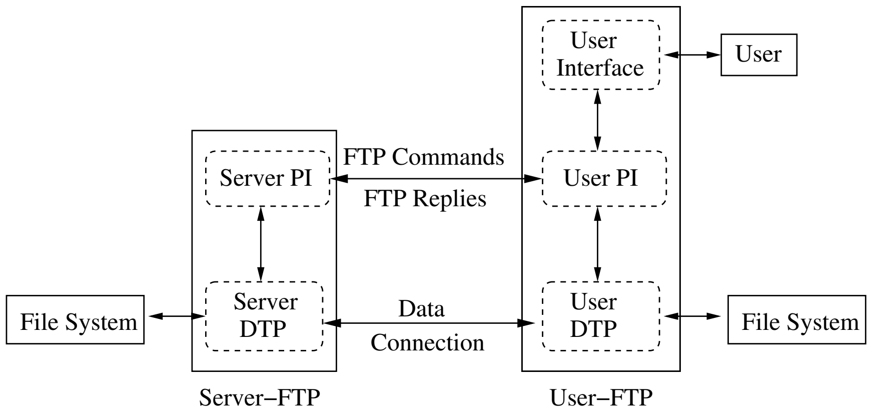


Figure 9.1: Basic FTP Model (Courtesy: RFC 959 [69])

9.2 Proposed Advanced Data Transfer Service

In this section we describe the details of our zero-copy design. In particular, we present our high performance data transfer service layer that is used to provide FTP services to the applications. In the following subsections we present the main design alternatives followed by the design details of the overall FTP library.

9.2.1 Design Alternatives

In order to enable zero-copy data-transfers, we consider the following two alternatives: (i) Memory semantics using RDMA and (ii) Channel semantics using Send and Receive.

RDMA based data-transfer requires allocating and registering buffers on both the source and the destination nodes. The data-transfer is initiated from the sender side by specifying the precise destination target buffer address. It is known that RDMA based approaches [53] achieve better latencies. However, RDMA based approaches have two significant drawbacks. Firstly, the target RDMA buffers need to pre-allocated, registered and the information (including addresses and memory registration keys) need to be communicated to the source process before each buffer can be used. Further, the flow control for the data-transfer using RDMA is very explicit. i.e. the sender can initiate data-transfers only after explicit notification of buffer availability on the remote target. Secondly, since RDMA does not involve remote node CPU in data-transfer, notifying the completion of each data-transfer requires additional mechanisms. In particular, additional messages based on send-recv (or RDMA-write with immediate) are needed for handling the remote notification, which adds a further overhead. And finally, in the case of WAN links, the latency benefits of RDMA seen for small messages are dominated by the actual network delay. Hence RDMA does not see any specific benefits over send-recv in WAN scenarios. These issues present critical constraints for data-transfers over WAN links.

On the other hand, send-recv based mechanisms show good benefits. Firstly, zero copy benefits of send-recv mechanism are identical to those seen with RDMA. i.e. the remote data can be directly received in the FTP buffers, which can then be written to the destination file. Secondly, send-recv mechanisms present opportunities for enabling flow control mechanisms. For example, with the use of SRQ [14], the receiver can post buffers when needed automatically. Such a capability eliminates the need for strict (or explicit) flow control for the data-transfers. This benefit can be quite significant on WAN links because the sender is not throttled due to lack of buffers on the remote node. In addition, as mentioned in Section 2.1.2 the InfiniBand's send/recv communications can be used over both the RC and UD transports. Due to these benefits, we utilize channel semantics (send/recv) for all WAN communication protocols[3].

9.2.2 Overview of the Proposed ADTS Architecture

In order to provide robust and efficient data transfers over modern WAN interconnects, in this chapter we propose the Advanced Data Transfer Service (ADTS). Figure 9.2 shows the main components of the ADTS layer.

As mentioned earlier, several high performance clients can utilize zero-copy mechanisms for data transfers. Once the type of channel to be used is negotiated by the

[3]It is to be noted that RDMA operations provide one-sided data-transfer capabilities which can be highly beneficial for certain kinds of applications. As mentioned in the previous chapter, in this context zero-copy send/recv based mechanisms are more beneficial

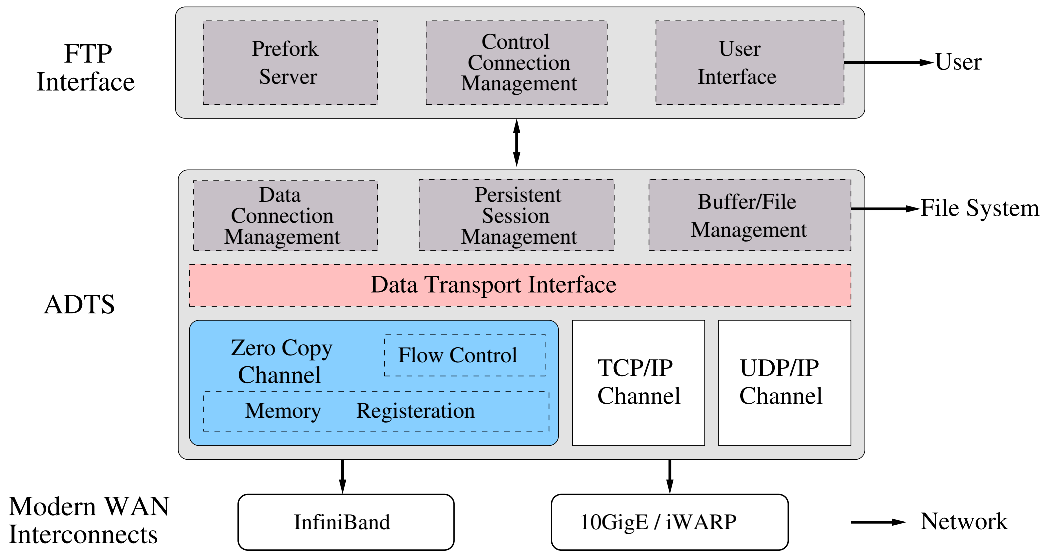


Figure 9.2: Overview of the Proposed ADTS Architecture

client and the server, the *Data Connection Management* component shown in Figure 9.2, initiates a connection to the remote peer (based on the negotiated PORT, PASV and/or TPRT commands) on either the zero-copy channel, the TCP/IP channel or the UDP/IP. Transport channels can dynamically be selected by the ftp server processes on a per connection basis to handle different kinds of clients. Thus improving robustness and interoperability. In addition to zero-copy, TCP/IP and UDP/IP channels, the Advanced Transport Interface provides scope for enabling support for other emerging transport protocols for next generation architectures.

We present our zero-copy channel design and the possible optimizations in the following sections.

Zero-Copy Channel

Once the client and server negotiate the use of zero-copy (detailed in Section 9.3) and the appropriate connections are setup by the *Data Connection Management* component, the channel is marked for zero-copy data transfers. However, in order to utilize zero-copy operations (send/recv), the client and the server need to handle flow control and buffer management.

Each buffer that the underlying layer (IB or iWARP) accesses needs to be registered and pinned in memory. This requirement adds a significant overhead to the actual data transfer. In order to alleviate this problem, the *Buffer and File Management* component keeps a small set of pre-allocated buffers. The data that needs to be transmitted is first read into these buffers while additional buffers are being allocated and registered as needed. Once the data is ready and the buffers are registered, the data transfers are initiated from these buffers. The buffers that are allocated on-demand are unregistered and released on completion of the complete data transfer.

Unlike the sockets interface where the underlying kernel TCP/IP stack performs the flow control, the ADTS needs to perform flow control for the buffers being sent out. i.e. data cannot be sent out unless the sender is assured of buffer availability on the receiver. In our design, we perform our flow control on the receiver side (using SRQ) as mentioned earlier in Sections 2.1.2 and 9.2.1. This enables the ADTS to push the data out of the source node at a high rate.

Performance Optimizations

In order to optimize the performance of data transfers over ADTS, we present the following optimizations: Persistent Sessions and Memory Registration Cache, Pipelined Data Transfers.

Persistent Sessions and Memory Registration Cache: While we utilize a set of pre-allocated buffers to speed up processing, these buffers need to be registered for each use. This in turn impacts the performance. Existing RDMA based libraries such as MVAPICH [66], amortize the registration cost by avoiding multiple registration/deregistration calls for multiple transfers by using the same buffers. This technique of maintaining registered buffers is popularly known as registration cache.

In typical FTP data transfers, each transferred file is transmitted on a different data-connection. Due to this, memory registration caching would not help significantly in our case. Further, such an approach would also incur multiple data connection setup costs as well.

In order to alleviate these multiple data-connection costs, in our design we enable persistent data-sessions that keep data connection and the related buffer associations alive during the transfer of multiple files. The maximum number of files to be transmitted on a given connection is negotiated in advance and the connection is not closed until the specified number of files are transferred or the connection becomes idle. This approach also allows for an efficient use of buffers and memory registrations which boosts the performance significantly.

Pipelined Data Transfers: In order to maximize the utility of both the network bandwidth and the local disk bandwidth (or the bandwidth of the local file system being used), the ADTS layer is designed with two threads. The network thread deals with processing of network related work queues, completion notifications, flow control and memory registrations while the disk thread handles the reads and writes from the disk. With this multi-threaded capability, all data transfers are packetized and pipelined and hence increased performance is obtained.

9.3 Design of FTP over ADTS

Figure 9.2 shows the basic architecture of our FTP client/server library (*FTP-ADTS*). In our approach we utilize the low-overhead zero-copy ADTS layer to provide high performance FTP transfers. The FTP Interface deals with the rest of the features needed for the FTP library. This interface provides a basic client user interface to enable all client interactions. The other main components are described in the following sections.

FTP Control Connection Management

Based on the user information, the client FTP engine initiates a sockets based control connection to the remote FTP server. This connection is used to relay all control information such as FTP commands and error messages. In addition, it is also used to negotiate the transport protocols and modes to be used for the data-transfers. In particular, the client negotiates with the server on Active/Passive (PORT/PASV commands in the FTP specifications [69]) mode connections.

Further, in our FTP client/server library we negotiate the use of zero-copy channel (or TCP/UDP channels) as well. Hence, clients that are capable of zero-copy transfers with the servers can benefit from higher performance. To enable this negotiation, we require the zero-copy enabled client to send an additional command TPRT (Transport

PRoTocol) advertising its transport preference. Once this negotiation is complete, the ATDS layer initiates the appropriate data connection.

Parallel Prefork Server

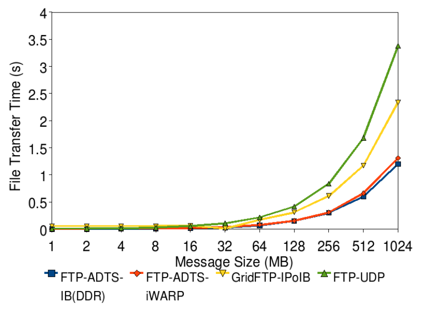
Multiple parallel accesses to the FTP server is a common scenario in most large data-centers. In order to efficiently support such parallelism, we design our FTP server as a multi-process server. The main FTP server daemon forks multiple processes that handle the client requests. In order to handle bursts of requests, our server maintains a small pool of pre-forked processes that handle burst of requests efficiently.

9.4 Performance Evaluation

In this section, we present the experimental results demonstrating the capabilities of our design. We evaluate the performance of our FTP designs in both LAN and WAN scenarios. We further measure the overheads of our approach and compare them with existing popular approaches to analyze the performance and scalability aspects of our design.

In a typical scenario, the primary bottleneck for large file transfer is disk I/O. In order to demonstrate the performance benefits of our design, we use RAM disks for all data storage purposes. Please note that modern HEC systems usually employ the use of high performance parallel or distributed file systems and advanced data storage technologies such as RAID, to obtain improved I/O performance.

Experimental Setup: In our experiments we use a cluster consisting of 64 Intel Xeon Quad dual-core processor nodes with 6GB RAM. The nodes are equipped with IB DDR ConnectX HCAs with OFED 1.3 [37] drivers. The OS used was



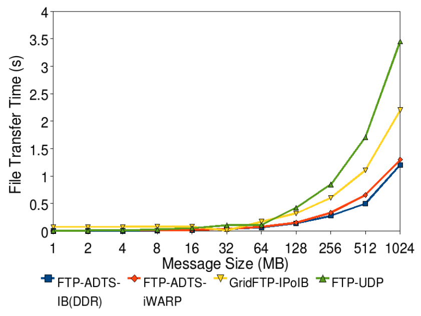


Figure 9.3: FTP File Transfer Time in LAN: (a) FTP (get) and (b) FTP (put)

RHEL4U4. These nodes are also equipped with Chelsio T3b 10 Gigabit Ethernet/iWARP adapters. We use the GridFTP and FTP-UDP (FTP using the UDP/IP channel in ADTS) as the base case for our performance comparisons. The connectivity for TCP and UDP are provided by IPoIB-RC and IPoIB-UD, respectively. The nodes in the cluster are divided into *Cluster A* and *Cluster B* and are connected with Obsidian InfiniBand WAN routers as shown in Figure 2.3.

9.4.1 Performance of FTP in LAN Scenarios

This experiment shows the basic performance improvement achieved by our FTP-ADTS as compared to FTP-UDP and GridFTP in low-delay, high-bandwidth scenarios. We evaluate the file transfer time of both file *put* and file *get* operations.

GridFTP and FTP-UDP are used for the TCP/IP and UDP/IP cases, respectively, and FTP-ADTS is used for the zero-copy cases (IB-DDR[4] and iWARP).

[4]It should be noted that currently the Obsidian Longbows can only support the packets at SDR rate, However, in LAN scenarios our cluster is capable of DDR speeds.

Figure 9.3(a) compares the client-perceived file transfer time of *get* operation with varying file sizes. We can see that our design (FTP-ADTS) achieves significantly better performance for larger file sizes. The FTP-ADTS over IB presents an improvement of by up to 95% and 181% as compared to GridFTP and FTP-UDP, respectively. Similarly FTP-ADTS over iWARP also outperforms these two cases by huge margins. This is expected since the zero-copy operations that FTP-ADTS employs has much lower latency than the IPoIB operations. We also observe that GridFTP does not perform well for small file sizes, but does better as the file sizes increase. This confirms what has been indicated about GridFTP in [81]. We can observe the similar trends for *put* operations as shown in Figure 9.3(b). Also, note that the performance of IPoIB itself limits the performance of the FTP operations using it. GridFTP and FTP-UDP are capable of saturating the available bandwidths in LAN scenarios, however, at the cost of high CPU utilization and memory bandwidth usage. We study this aspect further in the following sections.

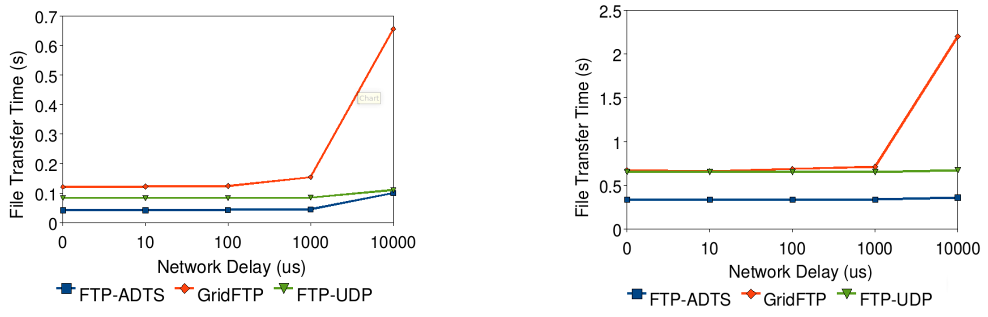


Figure 9.4: FTP File Transfer Time in WAN (get): (a) 32 MBytes and (b) 256 MBytes

9.4.2 Performance of FTP in WAN Scenarios

In order to study the impact of our design in WAN scenarios, we evaluate the performance of basic FTP operations with server and client running on a pair of nodes connected by a pair of Obsidian routers. The distance between them is emulated by varying the WAN delays ($5us$ corresponds to one km of wire length).

Basic Performance

We compare the performance of our design, GridFTP and FTP-UDP with varying WAN delays of 0 us (no delay), 10 us, 100 us, 1000 us and 10000us (corresponding to the distance of 0 km, 2 km, 20 km, 200 km and 2000 km). Figure 9.4 presents the time for transferring a 32 MByte file and a 256 MByte file. It is obvious that our FTP outperforms the GridFTP and FTP-UDP especially for data transfers over high delay networks. We observe that the *FTP-ADTS* sustains performance for larger WAN delays quite well, while the GridFTP shows a steep latency increase when the WAN delay is 10000 us. In the high delay scenario, our FTP delivers six times better performance, which shows significant promise for our FTP-ADTS design. The improvement is not only due to the underlying zero-copy operations being faster than the TCP/UDP, but also because the network throughput is the bottleneck for IPoIB over WAN, where issues such as RTT time, MTU size and buffer size can severely impact the performance. Another interesting observation is that here the FTP-UDP performs better than GridFTP, which is contrary to the results in the LAN scenario as shown earlier. This is due to the well-known trait that UDP can achieve good performance on high-bandwidth, high-latency networks in which TCP has fundamental limitations [38].

Detailed Analysis

In order to demonstrate the fundamental reasons that explain the above observations, we further carried out the following two experiments.

The first experiment is to measure the peak bandwidth of the WAN link with different transmission protocols. Figure 9.5 shows the results of IPoIB (including TCP/IP and UDP/IP) and verbs (including verbs-RC and verbs-UD) with increasing network delays. We can see that the IB verbs achieves the highest bandwidth and it sustains very well through the whole range of delays (WAN distance), while the TCP/IP bandwidth drops fast with the increasing delay, which in turn jeopardizes the GridFTP performance. On the other hand, although the UDP/IP bandwidth with smaller delays is lower than the TCP/IP bandwidth, it shows no significant degradation as the delay increases, which even makes it better than TCP/IP when the delay is high (> 10000 us). This is because that UDP can swamp the network by firing many packets, but TCP is limited by the congestion control and flow control that constrains it from using the available bandwidth in high bandwidth, high latency scenarios. (Please note that researchers have shown that TCP bandwidth over longer pipes can be improved by using techniques such as multiple parallel streams. While this improves the bandwidth performance of the application, this also needs additional resources at the end nodes. We intend to study the impact of parallel zero-copy protocol and parallel TCP/IP streams in future.)

The second experiment is to characterize the impact of transmitted packet size on bandwidth performance. Usually, larger packet sizes are preferred as they can make more use of the available link bandwidth. We claim that one of the reasons for the benefits of our design is that very large packet size (i.e. 1 MByte) can be used in the zero-copy send-recv (or RDMA) based approaches. Comparatively, the largest packet size that IP can use is 64 KByte (IPv4). In this experiment, we vary the packet size of IB RC Verbs bandwidth test (which is used in our design) with different WAN delays. From the results shown in Figure 9.6, we observe that the bandwidth for small and medium messages is progressively worse with increasing network delays. i.e. in order to leverage the high bandwidth capability of the IB WAN connectivity under higher network delays, larger messages need to be used. This demonstrates that some of the benefits of our design can be attributed to the use of large packet sizes. Further, we also show that IB WAN is currently not ideally suited for IP traffic (IPoIB), especially over high delay links.

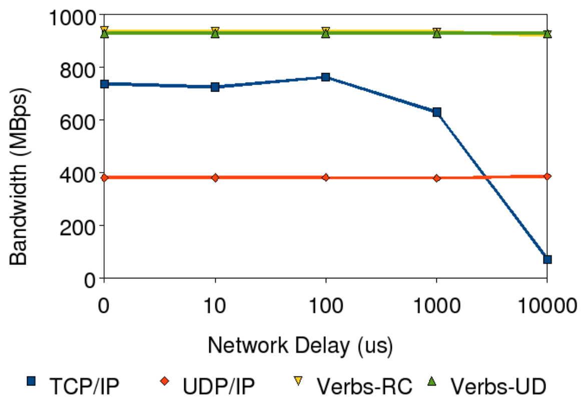


Figure 9.5: Peak Network Bandwidth with Increasing WAN Delay

9.4.3 CPU Utilization

TCP and UDP based communication libraries often suffer from the added CPU utilization for TCP(UDP)/IP stack processing. In basic implementations the FTP sender reads the data into its buffers and then sends this using the sockets interface.

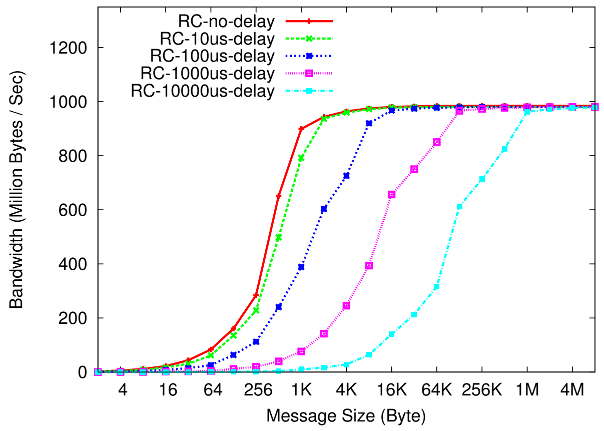


Figure 9.6: InfiniBand RC Bandwidth with Increasing WAN Delay

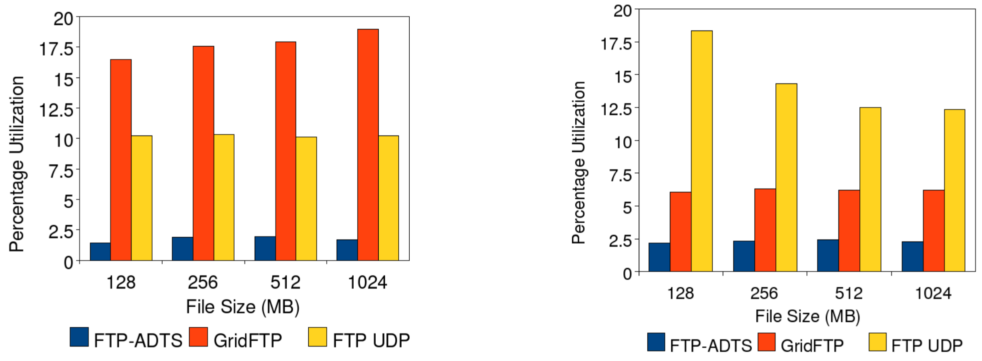


Figure 9.7: End Node CPU Utilization (a) Server; (b) Client

This would then be copied into the kernel's socket buffer before being sent out to the network. Similarly, the receiver would have to get this data into its kernel's internal socket buffers, then into its own buffers before it can be written to the destination disk. While TCP-based FTP implementations optimize the sender side overhead by utilizing the *sendfile* system call, the receiver side overhead is usually unavoidable.

Figures 9.7 (a) and (b) show the server and client CPU utilization, respectively, of the FTP-ADTS, GridFTP and FTP-UDP while performing multiple back-to-back large file *put* operations. The y-axis shows a normalized CPU utilization metric that represents the total percentage of CPU time being used on our 8-core system.

As expected, the GridFTP and FTP-UDP utilizes a significant amount of CPU on both the server and client for performing the two copies. On the other hand, our ADTS based approach has much lower CPU utilization, due to the use of zero-copy protocol which eliminates the need for additional copies on both the sender and the receiver. Further, we observe that the CPU utilization of GridFTP client is significantly lower. This demonstrates the benefits of using *sendfile* to reduce one memory copy on the client side. FTP-UDP does not show such trends as it can not use this optimization in UDP. Overall, our approach requires fairly smaller amounts of CPU time for all file sizes on both the server and client. This shows that our zero-copy ADTS design requires less CPU per request even at very high data rates and hence is more scalable than the corresponding IPoIB based designs.

9.4.4 Performance of Multiple File Transfers

In several scenarios such as site replication, mirroring, etc. FTP is used to transfer multiple individual files. Such scenarios present opportunities for several performance optimizations (including ones presented in Section 9.2.2). In this section, we present the following two experiments: (i) FTP performance for content replication and (ii) Analysis of performance benefits due to ADTS optimizations.

FTP Performance for Content Replication

In order to demonstrate the benefits of our design, we measure the performance of FTP-ADTS and FTP-UDP using a zipf [86] trace. This trace has a high α value with an average file size of about 66 MB. The average amount of time taken to replicate these traces over WAN is shown in Figure 9.8. We can see that the FTP-ADTS speeds up the data transfer by up to 65%. This demonstrates that the FTP-ADTS is a promising candidate for FTP design for the zero-copy-enabled networks especially in the long-distance WAN scenarios. These benefits are in addition to the CPU benefits mentioned earlier. We observe that the total transfer time in both cases increases for very large network delays. This is due to the fact that the zipf trace used consists of a large number of requests for smaller sized files.

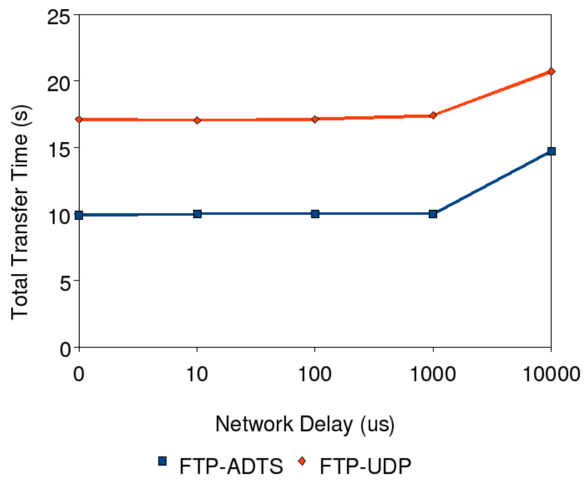


Figure 9.8: Site Content Replication using FTP

Benefits of the Proposed Optimizations

In this experiment we break up the performance of the FTP-ADTS while transferring a set of small files into Connection time (*Conn*) and Data Transfer time (*Data*). Figure 9.9 shows this breakup of the per transfer performance for FTP-ADTS with the following two cases: (i) *Basic*: with all optimizations disabled and (ii) *Opt*: with all optimizations enabled.

We clearly observe the following two trends: (i) pipelined data transfers, buffer reuse and memory registration caches improve the performance significantly (upto 55% improvement for the transfer of 16 files of size 1MB) and (ii) the use of persistent sessions improves the connection setup time considerably. i.e. the cost of initiating the connections is incurred only once instead of incurring on a per transfer basis.

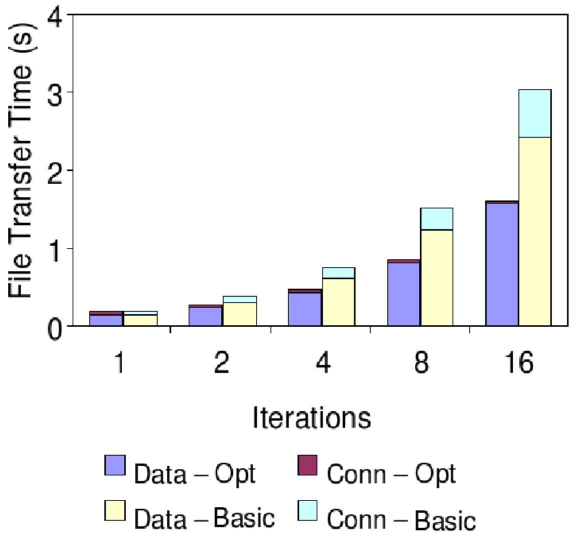


Figure 9.9: Benefits of ADTS Optimizations

9.5 Summary

FTP has been the most popular method to transfer large files for data-staging, replication, and the like. While existing FTP mechanisms have improved gradually with newer networks, they still inherit the fundamental limitations imposed by the underlying networking protocol TCP/IP. This includes limitations on the achievable bandwidth and the amount of CPU utilization that TCP/IP causes on the end-nodes, thereby limiting both the performance and scalability of such systems. On the other hand modern WAN capable interconnects such as InfiniBand WAN and 10 Gigabit Ethernet/iWARP have enabled the use of high performance RDMA capabilities in these scenarios. In this chapter we have presented an efficient zero-copy Advanced Data Transfer Service (ADTS) that has enabled a high performance FTP design (FTP-ADTS) capable of efficiently transferring data across WANs. Further, we reduced the CPU utilization required for the data-transfers and demonstrated significantly higher FTP server scalability.

In our experimental results, we have observed that our *FTP-ADTS* design outperforms existing approaches by more that 80% in transferring large amounts of data. In addition, we have utilized the WAN emulation capabilities of Obsidian InfiniBand WAN routers to study the impact of our designs in a wide range of WAN scenarios. We also observed that our approach achieves peak transfer rates at significantly lower (up to 6 times) CPU utilization.

Our studies have demonstrated that the IB WAN-based solutions are highly beneficial in WAN scenarios as well and can enable designing of next generation high performance parallel and distributed environments in a radically different manner.

CHAPTER 10

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

10.1 Summary of Research Contributions

In this thesis, we have demonstrated various benefits that modern interconnects like InfiniBand and 10 Gigabit Ethernet/iWARP bring to data-centers. The high performance and scalable services proposed are beneficial to a wide gamut of data-center applications and scenarios. Specifically, we have demonstrated how we can use the advanced features of modern interconnects to improve the performance of existing and emerging data-center scenarios. The following are the broad research contributions:

- Our research has demonstrated the feasibility of developing high performance system services based on the advanced capabilities of modern interconnects. The various designs proposed in this thesis are also intended to expose the design space that the advanced features of modern interconnects, such as RDMA and Remote Memory Atomics, present for distributed system services.

- Traditional communication protocols over WAN are limited largely to TCP/IP. Our study has demonstrated the benefits and limitations of WAN-based cluster-of-cluster environments. Our research will have a great impact on such current and next-generation scenarios with modern interconnects. In particular, the proposed designs will significantly impact the design of applications for the rapidly emerging distributed and hierarchical datacenter scenarios.

- Although we have concentrated on web-caching and data-centers, many of our research contributions are directly applicable to other scenarios like file-systems, grid computing, etc. Hence, our work is expected to have impact in those areas as well.

Below, we summarize the specific contributions and results of this dissertation.

10.1.1 Distributed Lock Management using Remote Atomic Operations

In Chapter 4, we presented a novel approach for efficient distributed lock management. We utilized the remote atomic operations provided by InfiniBand to provide one-sided mechanisms to enable both shared mode locking and exclusive mode locking. We discussed limitations of existing approaches and the benefits of our designs. By distributing the lock management related workload appropriately, we achieved fair load balancing among participating nodes. Further, we demonstrated the advantages of our design with detailed evaluation. Our results showed that our approaches consistently outperform existing approaches for distributed locking over InfiniBand.

10.1.2 Dynamic Content Caching with Strong Cache Coherency

In Chapter 5, we presented a highly efficient mechanism for dynamic content caching. We extensively utilized RDMA Read operations to obtain cache validity information in real-time to enable strong cache coherence required for several modern applications. Our results demonstrate a significant performance improvement over existing approaches. Further, we also demonstrate excellent scalability of our approach under loaded conditions that datacenters often encounter. Our results show an improvement of upto 10 times in the overall throughput (and upto 2 times improvement for average response time) achieved under loaded circumstances.

10.1.3 Dynamic Content Caching for Complex Objects

In Chapter 6, we have presented an extended approach for dynamic content caching in order to handle complex web-objects. Modern applications often generate complex objects with multiple dynamic components. We discussed the issues involved in providing strong cache coherency for such complex objects. We have evaluated the various tradeoffs involved in providing such caching capabilities. Our results have demonstrated an improvement of upto 1000% over existing approaches.

10.1.4 Efficient Multi-Tier Cooperative Caching

In Chapter 7, we have studied the various alternatives for enabling cooperative caching of web objects. In particular, we have discussed several optimizations to maximize system resource usage and overall performance. We evaluated the effect of selective automated replication of cached data to achieve maximum performance benefits. Further, we have explored the possibility of utilizing system-wide free resources for caching in terms of possible benefits and involved overheads. Our experimental results showed an improvement of upto 35% over existing cooperative caching approaches.

10.1.5 WAN Framework and High Performance Protocols over WAN

In Chapter 8, we have presented a framework for study of high performance communications over WAN. We have analyzed various mechanisms for emulating WAN characteristics for performing accurate higher level evaluations. We studied the performance characteristics of both IB WAN and iWARP using a multitude of hardware and software environments. Based on our analysis of such environments, we isolated the benefits and overheads of high performance protocols when used over WAN. We observed that the latency oriented benefits of RDMA are largely diminished in the context of WAN communications. However, zero-copy data transfers and asynchronous communication benefits still play an important role in such scenarios.

10.1.6 Efficient Data Transfers over WAN

In Chapter 9, we have presented the design of high performance data transfer capabilities (ADTS) across IB WAN. We have explored the possibility of using zero-copy communications, including RDMA and Send/Recv, for WAN data transfers. We have studied the tradeoffs involved in such communications and have designed an extended FTP client/server architecture that leverages our ADTS service to achieve efficient data transfers over IB WAN. Our experimental results showed significant performance benefits (upto 80% as compared to FTP over TCP/IP) over IB WAN interconnects.

10.2 Future Research Directions

Deployment of cluster based datacenters has long become the main stream approach for several applications. However, with the rapid advancement in interconnect and system technologies, many interesting research directions are open to be pursued. In particular, the recent WAN capability of popular RDMA interconnects is presenting great scope for new research directions. Below we describe some of these areas of future research.

10.2.1 Zero-copy WAN protocols for HTTP

HTTP has been the single most popular data transfer protocol over WAN. Web clients access content by communicating with servers over the HTTP protocol. In order to accommodate for wide portability, HTTP is primarily structured over TCP/IP communications. However, this can lead to known limitations in server scalability and sub-optimal communication performances.

While high performance communication protocols are being widely used in LAN scenarios, recent advances in WAN technologies have enabled the use of these high performance protocols across WAN links. InfiniBand WAN and 10GE/iWARP have enabled the use of RDMA and other zero copy mechanisms over wide area networks. In this context, the use of zero-copy high performance protocols need to be explored for HTTP servers. While such use of these protocols for HTTP can be beneficial, appropriate extensions to the HTTP protocol need to be explored, defined and standardized.

10.2.2 Caching and Content Dissemination with Internet Proxies

Web caching has been an important technique for improving web server performance. Traditionally, Internet proxy servers have been used to provide certain levels caching capabilities for web users. However, the usage of such servers is limited and these servers are often coupled loosely with the backend web-servers. With the WAN capabilities of modern interconnects such as InfiniBand and 10GE/iWARP, high performance protocols can now be used in the context of these Internet proxy servers. Several aspects of caching such as providing coherency and large scale cooperation among Internet proxies, can now be explored and need to be designed in order to handle the requirements of the next generation datacenters and web-applications. Further, with the added capabilities of modern interconnects, aggressive strategies for content dissemination are feasible and need to be studied.

10.2.3 Exploiting Network QoS features to Boost Performance

InfiniBand includes mechanisms to provide network-level QoS. In particular, the traffic in the network can be effectively prioritized using the SL-VL mapping mechanism. While providing user-level QoS is the usual goal, mechanisms like segregating traffic into control and data packets and prioritizing them appropriately at the network level are also possible. Further, modern generation IB NICs provide additional QoS capabilities including best latency guarantees, best bandwidth guarantees, etc. that can be exploited for improving the performance of the data-center services.

For example, the bulk of the data traffic in data-centers uses TCP/IP. During high network load, critical communication messages of the various data-center services (such as a lock request) can be delayed by the bulk data-transfer. Providing appropriate QoS guarantees to such messages might lead to better overall performance. i.e. a lock request message could be guaranteed best possible latency.

Detailed investigation is needed in order to ascertain the feasibility of exploiting InfiniBand's QoS mechanisms in enabling highly efficient communications for the framework providing the data-center services. Further the effect of this traffic segregation and prioritization on overall performance needs to be studied.

BIBLIOGRAPHY

[1] Ammasso, Inc., www.ammasso.com.

[2] ConnectX: 4th Generation Server and Storage Adapter Architecture. http://www.mellanox.com/.

[3] Fulcrum Microsystems. `http://www.fulcrummicro.com/`.

[4] InfiniBand Trade Association. http://www.infinibandta.com.

[5] MySQL - Open Source Database Server. http://www.mysql.com/.

[6] Secure File Transfer Protocol (SFTP). `www.openssh.com`.

[7] Stream Control Transmission Protocol (SCTP). `www.sctp.com`.

[8] Virtual Interface Architecture Specification. http://www.viarch.org.

[9] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: limitations and potentials. In *Proceedings of the 4th International WWW Conference*, Boston, MA, December 1995.

[10] M. Aldred, I. Gertner, and S. McKellar. A distributed lock manager on fault tolerant MPP. *hicss*, 00:134, 1995.

[11] W Allcock. GridFTP: Protocol Extensions to FTP for the Grid. Global Grid ForumGFD-R-P.020,2003.

[12] M. Allman and S. Ostermann. Multiple Data Connection FTP Extensions. Technical report, Ohio University, 1996.

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

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

[15] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Sockets Direct Protocol over InfiniBand in Clusters: Is it Beneficial? In *the Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, Texas, March 10-12 2004.

[16] Alexandros Batsakis and Randal Burns. NFS-CD: Write-Enabled Cooperative Caching in NFS. In *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 2008.

[17] W. Bell, D. Cameron, L. Capozza, A. Millar, K. Stockinger, and F. Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.

[18] Adam D. Bradley and Azer Bestavros. Basis Token Consistency: Extending and Evaluating a Novel Web Consistency Algorithm. In *the Proceedings of Workshop on Caching, Coherence, and Consistency (WC3)*, New York City, 2002.

[19] Adam D. Bradley and Azer Bestavros. Basis token consistency: Supporting strong web cache consistency. In *the Proceedings of the Global Internet Worshop*, Taipei, November 2002.

[20] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the INFOCOM '99 conference*, March 1999.

[21] Dennis Bush. UFTP. http://www.tcnj.edu/ bush/uftp.html.

[22] J. Cao. ARMSim: a Modeling and Simulation Environment for Agent-based Grid Computing. *Simulation*, 80(4), 2004.

[23] P. Cao and S. Irani. Greedydual-size: A cost-aware www proxy caching algorithm, 1997.

[24] P. Cao, J. Zhang, and K. Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, 2002.

[25] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.

[26] Pei Cao, Jin Zhang, and Kevin Beach. Active cache: Caching dynamic contents on the Web. In *Middleware Conference*, 1998.

[27] Enrique V. Carrera, Srinath Rao, Liviu Iftode, and Ricardo Bianchini. User-level communication in cluster-based servers. In *HPCA*, 2002.

[28] M. Carson and D. Santay. NIST Net: A Linux-based Network Emulation Tool. *Computer Communication Review*, 33(3):111–126, June 2003.

[29] Zhifeng Chen, Yan Zhang, Yuanyuan Zhou, Heidi Scott, and Berni Schiefer. Empirical Evaluation of Multilevel Buffer Cache Collaboration for Storage Systems. In *SIGMETRICS*, 2005.

[30] Michele Colajanni and Philip S. Yu. Adaptive TTL schemes for load balancing of distributed Web servers. *SIGMETRICS Perform. Eval. Rev.*, 25(2):36–42, 1997.

[31] Chelsio Communications. http://www.chelsio.com/.

[32] RDMA Consortium. Architectural Specifications for RDMA over TCP/IP. `http://www.rdmaconsortium.org/`.

[33] Obsidian Research Corporation. http://www.obsidianresearch.com.

[34] Nirmit Desai and Frank Mueller. A Log(n) Multi-Mode Locking Protocol for Distributed Systems.

[35] A. Devulapalli and P. Wyckoff. Distributed queue based locking using advanced network features. In *ICPP*, 2005.

[36] John Dilley, Martin Arlitt, and Stephane Perret. Enhancement and validation of the Squid cache replacement policy. In *Proceedings of the 4th International Web Caching Workshop*, April 1999.

[37] Open Fabrics Enterprise Distribution. http://www.openfabrics.org/.

[38] Phil Dykstra. Gigabit Ethernet Jumbo Frames. http://sd.wareonearth.com/ phil/jumbo.html.

[39] E. V. Carrera, S. Rao, L. Iftode, and R. Bianchini. User-Level Communication in Cluster-Based Servers. In *the 8th IEEE International Symposium on High-Performance Computer Architecture (HPCA 8)*, Feb. 2002.

[40] K. Fall and K. Varadhan. The NS Manual (Formerly NS Notes and Documentation), February 2006.

[41] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP 1.1. RFC 2616. June, 1999.

[42] The Apache Software Foundation. URL: http://www.apache.org/.

[43] Open Fabrics Gen2. http://www.openfabrics.org.

[44] S. Hemminger. Network Emulation with NetEm. In *Proceedings of LCA*, April 2005.

[45] J. Hilland, P. Culley, J. Pinkerton, and R. Recio. RDMA Protocol Verbs Specification (Version 1.0). Technical report, RDMA Consortium, April 2003.

[46] IEEE. IEEE Std 802.3ae-2002, Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation, August 2002.

[47] J. Liu and W. Jiang and P. Wyckoff and D. K. Panda and D. Ashton and D. Buntinas and B. Gropp and B. Tooney. High Performance Implementation of MPICH2 over InfiniBand with RDMA Support. In *IPDPS*, 2004.

[48] Terence P. Kelly, Yee Man Chan, Sugih Jamin, and Jeffrey K. MacKie-Mason. Biased replacement policies for Web caches: Differential quality-of-service and aggregate user value. In *Proceedings of the 4th International Web Caching Workshop*, April 1999.

[49] H. Kishida and H. Yamazaki. SSDLM: architecture of a distributed lock manager with high degree of locality for clustered file systems.

[50] Sourabh Ladha and Paul D. Amer. Improving Multiple File Transfers Using SCTP Multistreaming. In *IEEE International on Performance, Computing, and Communications Conference*, April 2004.

[51] D. Li, P. Cao, and M. Dahlin. WCIP: Web Cache Invalidation Protocol. IETF Internet Draft, November 2000.

[52] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and Implementation of MPICH2 over InfiniBand with RDMA Support. In *Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS '04)*, April 2004.

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

[54] Jiuxing Liu, Amith R.Mamidala, and Dhabaleswar K. Panda. Fast and Scalable MPI-Level Broadcast using InfiniBand's Hardware Multicast Support. In *Proceedings of IPDPS*, 2004.

[55] S. Ostermann M. Allman and C. Metz. FTP Extensions for IPv6 and NATs. RFC 2428. Network Working Group, Sep. 1998.

[56] C. Solutions M. Horowitz and S. Lunt. FTP Security Extensions. RFC 228. Network Working Grou, Oct., 1997.

[57] David McWherter, Bianca Schroeder, Anastassia Ailamaki, and Mor Harchol-Balter. Improving preemptive prioritization via statistical characterization of oltp locking. In *21st International Conference on Data Engineering (ICDE 05)*, 2005.

[58] Mikhail Mikhailov and Craig E. Wills. Evaluating a New Approach to Strong Web Cache Consistency with Snapshots of Collected Content. In *WWW2003, ACM*, 2003.

[59] Inc Mindcraft. http://www.mindcraft.com/webstone.

[60] Jeffrey C. Mogul. Clarifying the fundamentals of HTTP. In *the Proceedings of WWW-2002*, Honolulu, HI, May 2002.

[61] Myricom. Myrinet Software and Customer Support. http://www.myri.com/scs/GM/doc/, 2003.

[62] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting strong cache coherency for active caches in multi-tier data-centers over infiniband. In *SAN-3 held in conjunction with HPCA 2004*, 2004.

[63] S. Narravula, A. Mamidala, A. Vishnu, G. Santhanaraman, and D. K. Panda. High Performance MPI over iWARP: Early Experiences. In *Proceedings of International Conference on Parallel Processing*, 2007.

[64] Inc. NetEffect. http://www.neteffect.com/.

[65] V. Olaru and W.F Tichy. Request distribution-aware caching in cluster-based web servers. In *Third IEEE International Symposium on Network Computing and Applications, 2004. (NCA 2004). Proceedings.*, 2005.

[66] MVAPICH2: High Performance MPI over InfiniBand and iWARP. http://mvapich.cse.ohio-state.edu/.

[67] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich M. Nahum. Locality-aware request distribution in cluster-based network servers. In *Architectural Support for Programming Languages and Operating Systems*, pages 205–216, 1998.

[68] F. Petrini, W. C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network (QsNet): High-Performance Clustering Technology. In *Hot Interconnects*, 2001.

[69] J. Postel and J Reynolds. File Transfer Protocol. RFC 959. Internet Engineering Task Force. 1985.

[70] L. Ramaswamy, Ling Liu, and A. Iyengar. Cache clouds: Cooperative caching of dynamic documents in edge networks. In *25th IEEE International Conference on Distributed Computing Systems, 2005. ICDCS 2005. Proceedings.*, 2005.

[71] R. Recio, P. Culley, D. Garcia, and J. Hilland. An RDMA Protocol Specification (Version 1.0). Technical report, RDMA Consortium, Oct 2003.

[72] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *Computer Communication Review*, 27(1):31–41, January 1997.

[73] Luigi Rizzo and Lorenzo Vicisano. Replacement policies for a proxy cache. Technical Report RN/98/13, UCL-CS, 1998.

[74] J.B. Sartor, Subramaniam Venkiteswaran, K.S. McKinley, and Z. Wang. Cooperative caching with keep-me and evict-me. In *9th Annual Workshop on Interaction between Compilers and Computer Architectures, 2005. INTERACT-9.*, 2005.

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

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

[77] Karen R. Sollins. The Trivial File Transfer Protocol – TFTP 2 RFC 1350. July, 1992.

[78] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol(SCTP), RFC 2960, October 2000.

[79] Mellanox Technologies. InfiniBand and TCP in the Data-Center.

[80] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of OSDI*, December 2002.

[81] W. Allock, J. Bresnahan, R. Kettimuthu, and M. Link. The Globus Striped GridFTP Framework and Server. In *Super Computing, ACM Press*, 2005.

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

[83] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna R. Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles*, pages 16–31, 1999.

[84] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering Web Cache Consistency. *ACM Transactions on Internet Technology, 2:3,*, August. 2002.

[85] Q. Zhang, Z. Xiang, W. Zhu, and L. Gao. Cost-based cache replacement and server selection for multimedia proxy across wireless networks.

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