Republic of Iraq
Ministry of Higher Education
and Scientific Research
Al-Nahrain University
College of Science

Cooperative Caching for a Distributed System

A Thesis
Submitted to the College of Science, Al-Nahrain University
In Partial Fulfillment of the Requirements for
The Degree of Master of Science in Computer Science

By

Wurood Saad Ibraheem Al-Obaidi

(B.Sc. 2004)

Supervisors

Dr. Lamia H. Khalid ***Dr. Ban N. Al-Kallak***

December 2007 *Dhulhejja 1428*

بسم الله الرحمن الرحيم

(الحمد لله الذي هدانا لهذا وما كنا

لنهتدي لولا أن هدانا الله)

صدق الله العظيم

الأعراف ٤٣

Supervisor Certification

We certify that this thesis was prepared under our supervision at the Department of Computer Science/College of Science/Al-Nahrain University, by **Wurood Saad Ibraheem Al-Obaidi** as partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Supervisors

Signature:
Name: **Lamia H. Khalid**
Title: **Assist. Prof.**
Date: 16 /12 / **2007**

Signature:
Name: **Ban N. Al-Kallak**
Title: **Lecturer**
Date16 /12 / **2007**

The Head of the Department Certification

In view of the available recommendations, I forward this thesis for debate by the examination committee.

Signature:
Name: **Dr. Taha S. Bashaga**
Title: **Head of the department of Computer Science, Al-Nahrain University.**
Date: 17 /12 / **2007**

Examining Committee Certification

We certify that we have read this thesis and as an examining committee, examined the student in its content and what is related to it, and that in our opinion it meets the standard of a thesis for the degree of Master of Science in Computer Science.

Supervisors Certification

Signature:
Name: **Dr. Lamia H. Khalid**
Title: **Assist. Prof.**
Date:16 /4 / **2008**

Signature:
Name: **Dr. Ban N. Al-Kallak**
Title: **Lecturer**
Date16 /4 / **2008**

Examining Committee Certification

Signature:
Name: **Dr. Loay E. George**
Title: **Assist. Prof. (Chairman)**
Date: 16 /4 / **2008**

Signature:
Name: **Dr. Bara'a A. Attea**
Title: **Assist. Prof. (Member)**
Date: 16 /4 / **2008**

Signature:
Name: **Dr. Sawsan K. Thamer**
Title: **Lecturer (Member)**
Date: 16 /4 / **2008**

The Dean of the College Certification

Approved by the Council of the College of Science

Signature:
Name: **Dr. LAITH ABDUL AZIZ AL - ANI**
Title: The Dean of College of Science**, Al-Nahrain University.**
Date: / / **2008**

Dedication

I would like to dedicate this work to the member of my beloved family and all those who supported me in finishing this thesis.

Specially to my **father**, my **sisters**, my aunt **Dr. Muna Fadhel**, my aunt **Eng. Khanssa abdoul Alrahman**, my friends for continuous support and encouragement.

To my best friend **Eng. Zainab Hamid** and her mother for taken my hand to this college and their support during the period of my studies.

To the spirit of my grandfather **Ibraheem Al-Obaidi** who believed in me, to the spirit of my beloved uncle **Muneer Fadhel** the most person who I really miss, and to the spirit of my beloved **grandmother** who prayed and wished the best for me through out her life.

Finally, to the candle of my life who lights the darkness for me, **my mother**, the most kind and generous person I have ever known.

I hope you are all proud of me

Wurood

Acknowledgment

First, I would like to thank ***God****, for all the blessings that have given us.*

Second, I would like to express my sincere gratitude and appreciation to my supervisors ***Dr. Lamia H. Khalid*** *and* ***Dr. Ban N. Al-Kallak*** *for their valuable guidance, supervision and untiring efforts during the course of this work.*

Grateful thanks for the Head of Department of Computer Science ***Dr. Taha S. Bashaga,*** *staff and employees, especial thanks to* ***Dr. Sawsan Kamal, Shatha Alhassani, Haider Majeed****, and everyone who teached me.*

Finally, my very special thanks to my family especially my ***mother****, my friends M.Sc. classmates especially* ***Dunia Hamid****, for her continuous supports and encouragement during the period of my studies.*

Wurood

Abstract

Caching is the technique of storing the frequently used data in a fast memory, either at a client or at a server, which is connected to clients via a network. *Cooperative Caching* seeks to improve network file system performance by coordinating the contents of client caches and allowing requests not satisfied by a client's local in-memory file cache to be satisfied by the cache of another client.

This thesis aims to built and implement a cooperative caching for a distributed system (CCDS), which manages remote and local caches in a Local Area Network (LAN) working under windows operating system. It is developed using Java programming language. The CCDS consists of three components: manager, client, and server. The manager is the controller of the CCDS, which includes locating the required blocks in the local and global caches and decides from which cache to get the block. The manager controls the whole cooperative caches. Client accesses the blocks stored on the servers. It is the sender of the request to the distributed caches. The client controls local client cache. Server serves the requested clients. It is the receiver of the requests from the distributed caches client. The server controls server cache. Every machine in the LAN contains the CCDS with its three entities**.**

The advantage of CCDS can effectively support the scalability of the cooperative caching system because the communication and the data distribution are based on multicast and unicast routing techniques and support sharing resources of distributed data.

Table of Content

Table of Content

List of Abbreviations

API	Application Program Interface
BGP	Border Gateway Protocol
BSD	Berkeley Software Distribution
CCDS	Cooperative Caching for a Distributed System
DEC	Digital Equipment Corporation
DRAM	Dynamic Random Access Memory
FAT	File Allocation Table
FTP	File Transfer Protocol
HDD	Hard Disk Drive
HTTP	Hyper Text Transfer Protocol
IGMP	Internet Group Multicast Protocol
IP	Internet Protocol
IPC	InterProcess Communication
ISP	Internet Service provider
LAN	Local Area Network
LFU	Least Frequently Used
LRU	Least Recently Used
LRV	Least Relative Value
MAN	Metropolitan Area Network
MANET	Mobile Ad-Hoc Network
NTFS	New Technology File System
OS	Operating System
PGMS	Prefetching and caching in a Globally-managed Memory System
RAM	Random Access Memory
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
URL	Universe/Uniform Resource Locater
WAN	Wide Area Network

Chapter One

General Introduction

Chapter One

General Introduction

1.1 Introduction

This chapter explains distributed system, caching, distributed caching, some related works, aim of thesis, and thesis layout.

A ***distributed system*** is one in which components located at networked computers communicate and coordinate their actions only by passing messages. This definition leads to the following characteristics of distributed systems: concurrency of components, lack of global clock and independent failures of components [Cou01, Gab01]. There are three examples of distributed system [Cou01]:

- *The internet.*
- *The intranet, which is apportion of internet managed by an organization.*
- *Mobile and ubiquitous computing.*

The sharing of resources is the main motivation for constructing distributed system. Resources may be managed by servers and accessed by clients or they may be encapsulated as objects and accessed by other client objects. The web is discussed as an example of resource sharing. The challenges arising from the construction of distributed system are the heterogeneity of its components, openness which allows component to be added or replaced, security, scalability which is the ability to work well when the number of users increases, failure handling, concurrency of components, and transparency. The term resource is rather abstract one, but it best characterizes the range of things that can usefully be shared in a networked computer system. It extends from hardware components such as disk and printers to software-defined entities such as files, databases, and data objects

of all kinds. It includes the stream of video frames that emerges from a digital video camera and the audio connection that a mobile phone call represents [And01, Tan02].

A distributed system consists of four types of components, as depicted in figure (1.1) [Gab01]. These are:

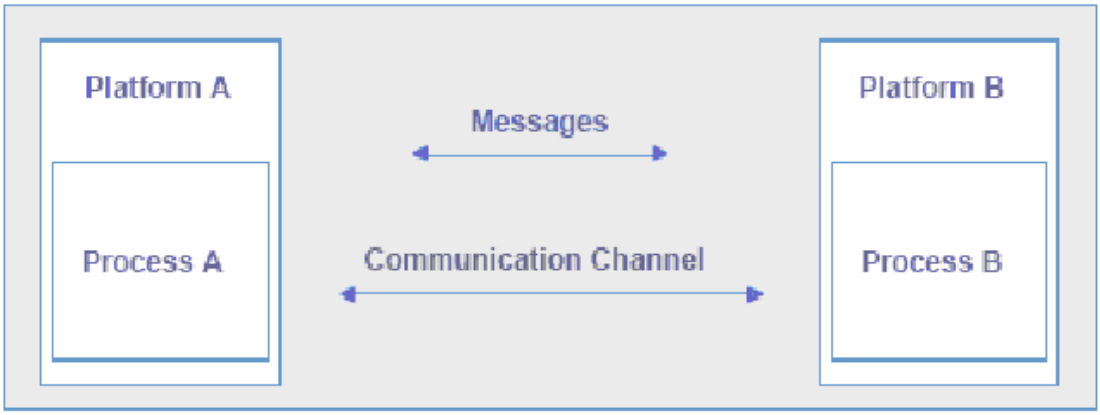

Figure (1.1) Distributed system components [Gab01]

1. *Platforms*—Platforms are the individual computing environments in which programs execute. These can be heterogeneous hardware components, operating systems, and device drivers that system architects and developers must integrate into a seamless system.
2. *Processes*—Processes are independent software components that collaborate with one another over channels. The terms client, server, peer, and service are often substituted for the term process, and each has a more specific meaning. Process can mean different things depending on the granularity with which one uses it. A process can represent an individual software object with a remote interface, a client or server that implements a particular protocol, some proprietary business application, or many other things.
3. *Communication channels*—Communication channels are pipelines between processes that enable them to interact. The term usually refers to the computer network(s) that logically connect processes and physically connect platforms. Communication channels have

both physical and logical aspects that are accounted for in any distributed system design.

4. *Messages*—Messages are the data sent from one process to another over a communication channel. How these data flow between processes in a reliable and secure manner is a question that requires much thought in the analysis and design stages of the development cycle.

The four types of distributed system components identified above are typically arranged in one of three distinct architectures, based on the ways in which individual processes interact with one another. These models are summarized in table (1.1) [Cou01, Gab01].

Table (1.1) Distributed system types [Gab01]

System architecture	**Description**
Client/Server	A distributed interaction model in which processes do things for one another
Peer processing	A distributed interaction model in which processes do things together
Hybrid	A combination of client/server and peer processing models

Distributed systems offer a number of advantages [Cou01, Kos95]:

- Users can be geographically separate: This is important for large corporations, where business decisions must be made by people in different locations, but those decisions must be based on company-wide data.

- Multiple machines can improve performance and scalability: Because a client-server system is distributed over several machines, the user can improve the performance and scalability in several ways. There might be multiple replicas of a server running on separate machines, so each handles only a fraction of the total number of clients. Redundant servers on separate machines can provide fail-over capability, to ensure service in the event of a problem on one machine.

- Heterogeneous systems can use the best tools for each task: Different components of an application can run on hardware that is optimized for a specific task. For example, an application might need to retrieve large amounts of statistical or experimental data from a database, perform complex computations on that data (such as computing a weather model), and display the results of that computation in the form of maps. By running the database, the computational engine, and the graphics rendering engine on hardware that is optimized for each task, performance can improve dramatically.

- Distributed systems can reduce maintenance costs: For example, by upgrading an application image on a single server, it is possible to upgrade thousands of clients.

- Resource sharing ability to use any hardware, software or data anywhere in the system. Resource manager controls access provides naming scheme and controls concurrency, and resource sharing model (e.g. client/server or object-based).

1.2 Caching

Pronounced cache, a special high-speed storage mechanism. It can be either a reserved section of main memory or an independent high-speed storage device. Caching is the technique of storing of frequently used data in a fast memory, either at a client or at a server, which is connected to clients via a network. Huge increases in performance can be obtained by storing frequently used data in a local file of memory data which would normally be accessed over a slow network connection or from some slow file device. Caching is an excellent way of speeding up a system for data which is not subject to much change such as simple web pages which do not contain dynamic data [Boy06].

A buffer is a temporary storage location where a large block of data is assembled or disassembled. This may be necessary for interacting with a storage device that requires large blocks of data, or when data must be delivered in a different order than that in which it is produced. The benefit is present even if the buffered data are written to the buffer once and read from the buffer once. A cache is a kind of buffer. However, it operates on the premise that the same datum will be read from it multiple times, that written data will soon be read, or that there is a good chance of multiple reads or writes to combine to form a single larger block. Its purpose is to reduce accesses to the underlying slower storage. Cache is also usually an abstraction layer that is designed to be invisible [Wik07d].

Caches, which deal with dynamically updated data, are known as write-back caches or write-through caches. For such caches when a transaction updates some stored data which appears in a cache at a client computer the following must occur [Are95, Has04]:

- The data that is stored at the client's cache must be updated to reflect the change.
- The stored data corresponding to the cached data must also be updated at its server.
- All other caches at other clients must be changed to reflect the changed data.

1.3 Distributed Caching

Distributed caching is the technique of keeping frequently accessed information in a location close to the requester. By reducing the amount of traffic on a network, caching provides significant benefits to Internet Service Providers (ISPs), enterprise networks, and end users. There are four key benefits [Wan03, Lan02, Lag02]:

- ***Cost savings due to Wide Area Network (WAN) bandwidth reduction*** ISPs can place cache engines at strategic points on their networks to improve response times and lower the bandwidth demand on their backbones. ISPs can station cache engines at strategic WAN access points to serve requests from a local disk rather than from distant or overrun servers. In enterprise networks, the dramatic reduction in bandwidth usage due to caching allows a lower-bandwidth (lower-cost) WAN link to serve the same user base. Alternatively, the organization can add users or add more services that use the freed bandwidth on the existing WAN link.

- ***Improved productivity for end users*** the response of a local cache is often three times faster than the download time for the same content over the WAN. End users see dramatic improvements in response times, and the implementation is completely transparent to them.

- ***Secure access control and monitoring*** the cache engine provides network administrators with a simple, secure method to enforce a site-wide access policy through Universe Resource Locater (URL) filtering.

- ***Operational logging*** Network administrators can learn which URLs receive hits, how many requests per second the cache is serving, what percentage of URLs are served from the cache, and other related operational statistics.

1.4 Literature survey

Many studies and researches have been introduced in the field of utilizing the network storage, especially the cooperative caching, the following are some of these researches:-

1. *Cooperative Caching: Using Remote Client Memory to Improve File System Performance* [Dah95].

In this research, five cooperative caching algorithms were used as trace-driven simulation. These algorithms are: *Direct Client Cooperation* allows an active client to use an idle client's memory as backing store. *Greedy Forwarding* treats the cache memories of all clients in the system as a global resource that may be accessed to satisfy any client's request. *Centrally Coordinated Caching* adds coordination to the Greedy Forwarding algorithm by statically partitioning each client's cache into a locally managed section, managed greedily by that client, and a globally managed section, coordinated by the server as an extension of its central cache. *N-Chance Forwarding*, dynamically adjusts the fraction of each client's cache managed cooperatively, depending on client activity. *Hash-Distributed Caching* differs from Centrally Coordinated Caching in that Hash-Distributed Caching partitions the centrally managed cache based on block

identifiers, with each client managing one partition of the cache. . These simulations indicate that for the systems studied cooperative caching can halve the number of disk accesses and improve file system read response time. Based on these simulations, the researchers conclude that cooperative caching can significantly improve file system read response time and that relatively simple cooperative caching algorithms are sufficient to realize most of the potential performance gain.

2. ***Implementing Cooperative Prefetching and Caching in a Globally-Managed Memory System*** **[Voe98].**

This research presents cooperative prefetching and caching—the use of network-wide global resources (memories, CPUs, and disks) to support prefetching and caching in the presence of hints of future demands. Cooperative prefetching and caching effectively unites disk-latency reduction techniques from three lines of research: prefetching algorithms, cluster-wide memory management, and parallel I/O. When used together, these techniques greatly increase the power of prefetching relative to a conventional (nonglobal- memory) system. The authors have designed and implemented *prefetching and caching in a globally-managed memory system* (PGMS), a cooperative prefetching and caching system, under the Unix operating system running on a 1.28 Gb/sec Myrinet connected cluster of Digital Equipment Corp.(DEC) Alpha workstations. Their measurements and analysis show that by using available global resources, cooperative prefetching can obtain significant speedups for I/O-bound programs.

3. ***Cooperative caching and prefetching in parallel/distributed file systems [Cor97].***

In this thesis, the author proposed a solution to improve the file system performance by decoupling the performance of the file-system from the performance of the disk. He achieved it by designing a new cooperative cache and some aggressive-prefetching algorithms. Both mechanisms decrease the number of times the file system has to access the slow disk in the critical path of the user request. Furthermore, the resources used in this solution are large memories and high-speed interconnection networks which at a similar pace as the rest of the components in a parallel machine.

4. ***Exploiting Idle Memories in LAN Using Cooperative Caching Technique*** **[Has04].**

In this thesis, the author proposes a distributed system sharing proxy cooperative caching using N-chancing forwarding algorithm but this algorithm is modified by allowing the client to request more than one block at each session. The proposed system is a centralized control using one server with a database contains knowledge about all files, blocks, and clients. The system works in user mode using LAN working under any network operating system, and it implemented using Java programming language. The results showed a high speed up ratio in accessing remote memory and it reached on average 17 times faster than accessing server disk.

5. ***Shark: Scaling File Servers via Cooperative Caching [Ann04]***

The authors present Shark, a novel system that retains the best of both worlds-the scalability of distributed systems with the simplicity of central servers. Shark is a distributed file system designed for large scale, wide-area deployment, while also providing a drop-in replacement for local-area file

systems. Shark introduces a novel cooperative-caching mechanism, in which mutually-distrustful clients can exploit each others' file caches to reduce load on an origin file server. Using a distributed index, Shark clients find nearby copies of data, even when files originate from different servers. Performance results show that Shark can greatly reduce server load and improve client latency for read-heavy workloads both in the wide and local areas, while still remaining competitive for single clients in the local area. Thus, Shark enables modestly -provisioned file servers to scale to hundreds of read-mostly clients while retaining traditional usability, consistency, security, and accountability.

6. *A Stateless Neighbor-Aware Cooperative Caching Protocol for Ad-Hoc Networks [Mir05]*

Replication of data items among different nodes of a Mobile Ad-Hoc NETwork (MANET) is an efficient technique to increase data availability and improve access latency. This work proposes a novel algorithm to distribute cached data items among nodes in a MANET. The algorithm combines a probabilistic approach with latency constraints such as the distance from both the source and the clients of the data item. In most scenarios, the proposed approach allows any node to retrieve a data item from a nearby neighbor (often, just one hop away). The work describes the algorithm and provides its performance evaluation for several different network configurations.

7. *Online Hierarchical Cooperative Caching [Lix06]*

The authors address a hierarchical generalization of the well-known disk-paging problem. In the hierarchical cooperative caching problem, a set of n machines residing in an ultra metric space cooperate with one another to satisfy a sequence of read requests to a collection of read-only files. A seminal result in the area of competitive analysis states that the ”least

recently used" (LRU) paging algorithm is constant-competitive if it is given a constant-factor blowup in capacity over the offline algorithm. So such a constant-competitive deterministic algorithm, with a constant-factor blowup in the machine capacities, exist for the hierarchical cooperative caching problem.

1.5 Aim of Thesis

The aim of this research is to apply distributed cooperative caching using multicast and unicast routing techniques to coordinate the file caches of up to 10 machines distributed on a LAN to form an effective overall file cache. The cooperative caching is a group caching sharing their contents and workloads via network. By using cooperative caching, the implemented system expected to gain an ease of ever-growing bandwidth need and to speed the information delivery. The use of multicast and unicast routing techniques can improve the quality of caching system. In addition, cooperative caching provides resource-sharing ability to use any data anywhere in the system.

1.6 Thesis layout

The thesis layout is as follows:

- **Chapter Two:** explains the theoretical basis of file system, cooperative caching and caching replacement techniques.
- **Chapter Three**: explains the theoretical basis of Interprocess communication and routing techniques.
- **Chapter Four:** explains the design and implementation of the proposed system with the experiments and results.
- **Chapter Five:** concludes this thesis and gives points for future work.

Chapter Two

Cooperative Caching

Chapter Two

Cooperative Caching

2.1 Introduction

In computing, a file system is a method for storing and organizing computer files and the data, they contain to make it easy to find and access them. File systems may use a data storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files, they might provide access to data on a file server by acting as clients for a network protocol, or they may be virtual and exist only as an access method for virtual data. A file system is a set of abstract data types that are implemented for the storage, hierarchical organization, manipulation, access, and retrieval of data. The most familiar file systems make use of an underlying data storage device that offers access to an array of fixed-size blocks, sometimes called sectors, generally 512 bytes each. The file system software is responsible for organizing these sectors into files and directories, and keeping track of which sectors belong to which file and which are not being used. However, file systems need not make use of a storage device at all. A file system can be used to organize and represent access to any data, whether it is stored or dynamically generated (e.g., from a network connection). Traditional file systems offer facilities to create, move and delete both files and directories [Wik07a]. This chapter explains the caching in file system, virtual memory vs. file system cache, cooperative caches, cache coherency, cache consistency and replacement policies.

2.2 Caching in File System

Caching file system data is an important performance optimization that virtually every modern operating system performs. The premise behind caching is that most applications access data that is primarily localized within a few files. Bringing those files into memory and keeping them there for the duration of the application's accesses minimizes the number of disk reads and writes the system must perform. Without caching, applications require relatively expensive disk operations every time they access a file's data [Rus98].

Operating Systems (OSs) uses two types of file-system data caching: ***logical block caching*** and ***virtual block caching***. The two types store data at different levels of abstraction. A logical drive resides on a disk partition that's composed of physical storage units called sectors. When an application accesses data in a particular file, the file system responsible for the drive (e.g., File Allocation Table (FAT), New Technology File System (NTFS)) determines which sectors of the disk hold the data in the file. The file system then issues disk I/O requests to read from or write to those sectors [Rus98, Sun94].

In ***logical block caching***, the OS caches sector data in memory so that the memory associated with the target sectors, rather requires disk operations, can satisfy disk I/O requests. Older variants of the Unix OS, including Berkeley Software Distribution (BSD) 4.3 OS, every Microsoft OS (Windows 98, Win95, Windows 3.x, and DOS) except Windows NT, and Novell NetWare, cache file system data at the logical block level [Rus98, Sun94].

Virtual block caching caches data at the file system level rather than the disk level. When an application accesses data in a file, the file system checks to see whether the data resides in the cache. If the data is in the cache, the file system doesn't need to determine which sectors of the disk store the data and issue disk I/O requests. The file system simply operates on the data in the cache. Windows NT relies on ***virtual block caching***, as do newer versions of Unix OS, including Linux, Solaris, and BSD 4.4 [Rus98, Sun94].

Virtual block caching has a couple of advantages over logical block caching. First, when file data the application is reading is in a ***virtual block cache***, the file system performs no file-to-sector translations. In fact, in some cases, the I/O system can bypass the file system altogether and retrieve requested data directly from the cache. Second, the cache subsystem knows which files and which offsets within the files an application is asking for. The cache subsystem can monitor the access patterns of each file and make intelligent guesses about which data an application is going to ask for next. Using its guesses as guidelines, the cache subsystem reads the data from disk in anticipation of future requests. This process is known as ***read-ahead,*** and when the cache subsystem's predictions are accurate, read-ahead boosts system performance. Although read-ahead is possible with logical block caching, virtual block caching makes read-ahead simple to implement [Rus98, Sun94].

2.3 Virtual memory vs. File system cache

It is a good idea to have big file system caches in order to be very effective. On the other hand, if the cache is too big, the physical memory available for the virtual-memory systems may not be enough. When this happens, the system starts trashing and all the applications slowdown their execution. In order to solve this problem, some operating systems propose a

variable-size cache. This mechanism decides dynamically the portion of physical memory given to the cache and to the virtual-memory system. With this mechanism memory-bound applications are not affected by file-system cache. Furthermore, when memory is available, the I/O bound applications may get the advantage of a big cache [Cor97].

In particular, if there is insufficient memory to run application programs, then the programs may slowdown by factors of 10 to 100 because of excessive paging activity. Thus, if a cache is allowed to become too large, the improvement in file system performance may be more than offset by degradation in virtual memory performance. In order to provide both good file system performance and good virtual memory performance, several operating systems have implemented variable-size cache mechanisms. In these operating systems the portion of memory used for file data and virtual memory varies in response to the file and virtual-memory needs of the application programs being executed. These mechanisms will obviously work well when there is little or no contention for memory between file and virtual-memory pages. The approach that has been commonly used to provide variable-size file data caches is to combine the virtual memory and file systems together, this is generally called the mapped-file approach. To access a file, it is first mapped into a process's virtual address space and then read and written just like virtual memory.

This approach eliminates the file cache entirely; the standard page replacements mechanisms automatically balance physical memory usage between file and program information [Nel90].

The approach to providing variable-size caches is quite different from the mapped file approach. In this approach the file system and virtual memory system are separate. Users invoke system calls such as read and write to access file data. These system calls copy data between the file cache and the virtual address spaces of user processes. Variable-size caches are provided by having the virtual memory system and file system modules negotiate over physical memory usage [Cor97, Nel90].

2.4 Cooperative Caches

Cooperative caching seeks to improve network file system performance by coordinating the contents of client caches and allowing requests not satisfied by a client's local in-memory file cache to be satisfied by the cache of another client. Two technology trends push computer scientist to consider cooperative caching. First, processor performance is increasing much more rapidly than disk performance. This divergence makes it increasingly important to reduce the number of disk accesses by the file system. Second, emerging high-speed low-latency switched networks can supply file system blocks across the network much faster than standard [Dah95].

This coordination allows a request from a given node to be served by the local cache of a different node. Until cooperative come into sight, all client caches were isolated and uncoordinated [Cor97].

The cooperative cache differs from the other levels of the storage hierarchy in that it is distributed across the clients and it therefore shares the same physical memory as the local caches of the clients. A local client cache is controlled by the client, and a server cache is controlled by the server, but it is not clear who should control the cooperative cache. For the cooperative cache to be effective, the clients must somehow coordinate their actions. Although it is enticing to think of cooperative caching as simply another layer in the storage hierarchy, management of the cooperative cache can

potentially involve every machine in the system since the cache is distributed across all the clients [Sar96].

The implementations of cooperative caches are designed with either distributed or centralized control. The cooperative caches with distributed control are designed to avoid bottlenecks. This distributed control is necessary if many nodes take part in the cooperation. On the other hand, it may not be always necessary when a small number of machines are used in the cooperative cache. Scalability is the first problem, which solved in the distributed control. Important objectives of this distributed control consist of proving that avoiding replication to avoid coherence problems (to be discussed later) still works when the control is distributed. The cooperative caches with centralized control are much easier to implement than a distributed one, see figure (2.1). It avoids all the communication and synchronization problems. Moreover, centralized algorithm has better knowledge of the whole system as they keep all the information of the system. The main disadvantage of centralized control resides in its lack of scalability [Cor97, Voe98].

A centralized system becomes a bottleneck and has to be redesigned in a distributed fashion.

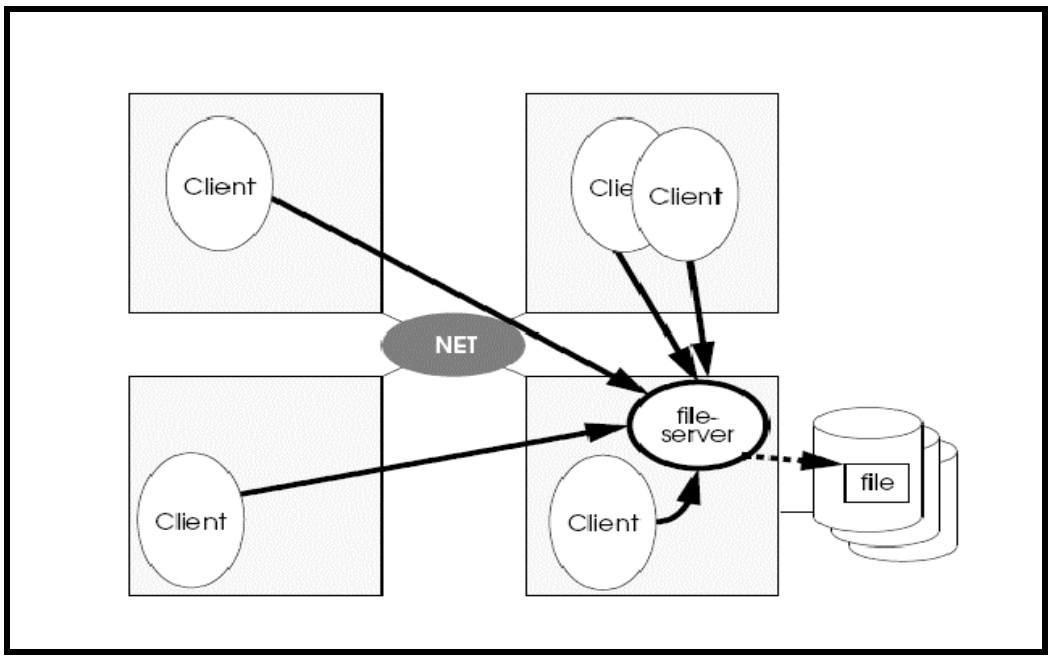

Figure (2.1) File system architecture of the centralized control [Cor97]

2.5 Cache Coherency

A typical shared memory multiprocessor contains multiple levels of caches in the memory hierarchy. Each processor may read data and store it in its cache. This results in copies of the same data being present in different caches at the same time. The problem occurs when a processor performs a write to data. If only the value in the writing processor's cache is modified, no other processor will see the change. If some action is not taken, other processors will read a stale copy of the data. Intuitively, a read by another processor should return the last value written. To avoid the problem of reading stale data, all processors with copies of the data must be notified of the changes. Two properties must be ensured. First, changes to a data location must be made visible to all processors, which is called write propagation. Second, the changes to a location must be made visible in the same order to all processors, which is called write serialization. Using file-system caches in different node may rise coherence problem. Two nodes may be caching the same file block and this one should be kept coherent when one of the nodes modifies it [Cor97, Grb03].

Coherence defines the behavior of reads and writes to the same memory location. 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) [Wik07b, Nar05].

To solve the cache coherence problemThere are three main mechanisms of cache coherence [Wik07b]:

- **Directory-based cache coherence** mechanisms maintain a central directory of cached blocks.
- **Snooping** is the process where the individual caches monitor address lines for accesses to memory locations that they have cached. When a write operation is observed to a location that a cache has a copy of, the cache controller invalidates its own copy of the snooped memory location.
- **Snarfing** is where a cache controller watches both address and data in an attempt to update its own copy of a memory location when a second master modifies a location in main memory.

2.6 Cache Consistency

The 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). So, either every client sees an update or no client sees that particular update. The value of caching is greatly reduced, however, if cached copies are not updated when the original data change. 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. In the current web, 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). 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 based on the application requiring the data [Wik07b, Nar05].

Cache consistency mechanisms ensure that cached copies of data are eventually updated to reflect changes to the original data. There are three main cache consistency mechanisms currently in use on the Internet [Gwe97, Nar05]:

- **Time-To-Live fields (TTL)** are pointing as a prior estimate of an object's lifetime that is used to determine how long cached data remain valid [Gwe97].
- **Client polling** is a technique where clients periodically check back with the server to determine if cached objects are still valid. The specific variant of client polling is based on the assumptions that young files are modified more frequently than old files and that the older a file is the less likely it is to be modified [Gwe97].
- **Invalidation protocols** are required when weak consistency is not sufficient; many distributed file systems rely on invalidation protocols to ensure that cached copies never become stale. [Gwe97].

2.7 Replacement Policies

The role of the cooperative cache replacement policy is to determine the order in which blocks are replaced. The cooperative cache replacement policy is activated when a client decides to replace a block from its local cache. A replacement policy can use two factors in deciding whether or not to forward a block. First, a block is discarded if the replacement algorithm decides that the block is less valuable than any block in the cooperative cache. Otherwise, the block is forwarded to a target client, which then replaces a block in its cache. The second factor in deciding whether or not a block should be forwarded to the cooperative cache is duplicate avoidance. Since the cooperative cache is a resource used by all of the clients, the potential exists for uncoordinated client actions to result in several copies of the same block in the cooperative cache. These duplicate copies pollute the cooperative cache and reduce its hit rate. Thus a block should not be forwarded to the cooperative cache if it is going to become a duplicate copy. In particular, if several clients have a copy of a block in their local caches, only one of the copies should be forwarded to the cooperative cache, and only if the cooperative cache does not already contain a copy.

If the client decides to forward the block, the choice of the target client becomes important in determining the effectiveness of the replacement policy. Thus, the target client should be chosen such that the forwarded block replaces a block less valuable than itself. The replaced block in the target client may be in the client's local cache or in the cooperative cache [Sar96].

The main four replacement algorithms that have obtained the best result are [Cor97, Arl99, Bit02, And03]:

- **Least-Recently Used (LRU):**

  This algorithm always replaces the least recently used block in the cache. This algorithm tries to take advantage of the temporal locality. If a block has been recently used it will probably be used again in a short period of time. On the other hand, if a block has not been used recently, it will probably not be used shortly. This is the replacement algorithm most widely used in commercial file-system caches. Its popularity is due to its high effectiveness and simplicity of its implementation. The policy is based on the same principle as page replacement policies in operating systems. Every cached item is associated with a time stamp that stores the last time the item was accessed, since the data server started execution. The item with the minimum time stamp is replaced when a new item must be stored in a full cache [And03, Cor97].

- **Segmented LRU :**

The idea behind this algorithm is that blocks which have been requested more than once will probably be used again. In order to take advantage of this heuristic, the replacement algorithm will try to avoid replacing blocks that have been accessed more than once [Cor97].

Segmented LRU is a frequency-based variation of LRU designed for fixed-size page caching in file-systems. Observing that objects with two accesses are much more popular than those with only one access, the cache space is partitioned into two LRU segments: probationary segment and protected segment. Objects brought to the cache are initially put in the probationary segment, and will only be moved to the protected segment if they get at least one more access. When an object has to be evicted, it will be taken from the probationary segment first. The protected segment has a fixed size, and when it gets full the objects that don't have space in it will be kept in the probationary segment. Segmented LRU is not suitable for Web caching as it ignores the size of cached objects and assumes fixed-size objects. Furthermore, it has the problem of needing parameterization of the number and sizes of segments [Arl99].

- **Least Frequently Used (LFU):**

This strategy the intermediate aggregate which is accessed least frequently. It is based on the assumption that collaborative environment are likely to request the same or closely related regions of interest, with the same or similar processing requirements. A reference count is associated with each cached blocks. The count is incremented when the block is reused. The objective of this algorithm is to find the most popular blocks and keep them in the cache. In order to do it, the system keeps counter with the number of times each block has been referenced.

Whenever a block has to be discarded, the one with the smaller number of references is replaced with a new block when the cache is full. This policy keeps track of the number of requests that are made for each document in the cache, evicting the document of documents that have been less frequently requested first if space is needed. As it ignores the document sizes, it can lead to an inefficient use of the space on the cache. Where LRU is equivalent to sorting by last access time, LFU is equivalent to sorting by number of accesses [And03, Bit02].

Least Relative Value (LRV):

This policy replaces the intermediate result that has the least value. The value metric can be computed in several different ways. Ideally, it should be a relative measure of how expensive it is to generate a given intermediate result. LRV is based on the relative value (V), a function of the probability that a document is accessed again (Pr). The LRV algorithm simply selects the document with the Lowest Relative Value as the candidate for eviction. As V is proportional to Pr, the issue is to find this probability. The parameters used for computing Pr are the following [Bit02, And03]:

- Time from the last access.
- Number of previous accesses.
- Document size.

Chapter Three

Interprocess Communication and Routing

Chapter Three

Interprocess communication and Routing

3.1 Introduction

The connection between the client and server portions must allow data to flow in both directions. There are number of ways to establish this connection. OSs support several mechanisms of Interprocess communication (IPC). Many of these mechanisms are similar in its function but different in name from one OS to another. Windows OS is taken as an example, since this project is implemented under windows [Cou01, Saf06]. This chapter explains the IPC and routing techniques.

3.2 Interprocess communication (IPC)

Windows provides several different IPC mechanisms. The most common windows IPCs are [Cou01, Saf06 and Dei01]:

1. **Remote Procedure Calls (RPCs):** The RPC was designed as a way to abstract the procedure-call mechanism for use between systems with network connections. In RPC, the client program calls a procedure in another program running in server process. Servers may be clients to other servers to allow chains in RPCs. RPC allows a procedural program to call a function residing on another computer as conveniently as if that function where part of the same program running on the same computer. A disadvantage of RPC is that it supports a limited set of simple data types. [Cou01, Saf06].

2. **Sockets:** The computer can offer different kinds of services, one of these is the opportunity to send and receive data by using Hyper Text Transfer Protocol (HTTP), another to give the exact time and date for the computer. To keep track of different kind of services, a ***port*** number is used which is a connection point on a computer. Each service is assigned a particular port, identified by the whole number such as port number 80 is used for HTTP communication on internet, port number 21 for File Transfer Protocol (FTP) communication, port number 13 gives the exact time and date and port number 25 for Simple Mail Transfer Protocol (SMTP) . All upper-layer applications that use Transmission Control Protocol (TCP) have a ***port*** number those identifies the application. In theory, port numbers can be assigned on individual machines however the administrator desires, but some conventions have been adopted to enable better communications between TCP implementations, which enables the port number to identify the type of service that one TCP system is requesting from another. Port numbers can be changed, although this can cause difficulties. Most systems maintain a file of port numbers and their corresponding service. When the programmer wants to get on the LAN to communicate with other computers through a program, he has to connect a "virtual line" through the port. This "virtual line" is called a ***socket***. Therefore, a socket is a kind of channel for data. Only one socket is usually drown through a particular port but it is possible to have a several sockets running through a port. A program can also make use of several ports and sockets. Figure (3.1) gives dramatic view of ports and sockets [Ska00, Cou01, Saf06].

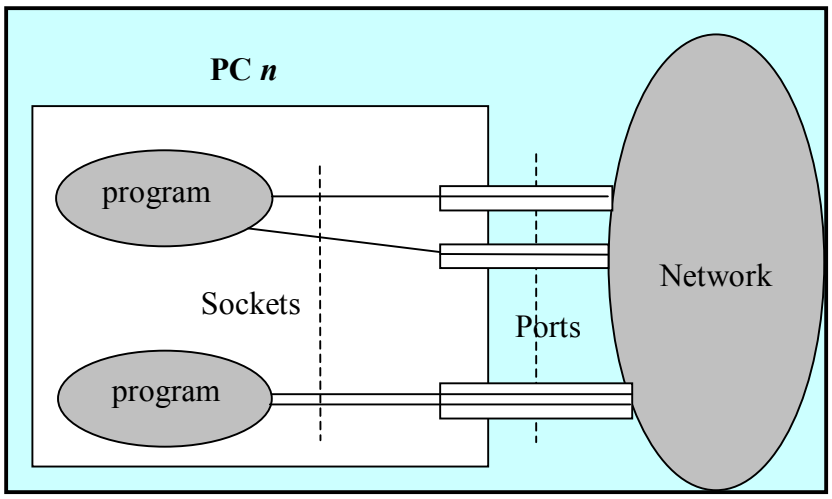

Figure (3.1) ports and sockets [Ska00]

The ***socket*** is a mechanism between network applications running on the same computer, or on different computers connected using a LAN or WAN. It defines a set of standard Application Program Interface (API) that an application uses to communicate with one or more other applications, usually across a network. The socket supports [Cou01, Saf06]:

1. Initiating an outbound connection for a client application.
2. Accepting an inbound connection for server application.
3. Sending and receiving data on a client/server connection.
4. Terminating a client/server connection.

The specification includes a standard set of APIs supported by all Windows-based Transmission Control Protocol/Internet Protocol (TCP/IP) stack, and to be used by network applications. In ***sockets***, application communications channels are represented by data structures called ***sockets***. Two items are used to identify ***a socket*** [Cou01]:

- An Internet Protocol address.
- A port number.

Both the sending and receiving machines have sockets. Because the IP address is unique across the internetwork, and the port numbers are unique to the individual machine, the socket numbers are also unique across the entire internetwork. This enables a process to talk to another process across the network, based entirely on the socket number.Sockets are highly useful in at least three communications contexts [Cou01]:

- Client/Server models.
- Peer-to-Peer scenarios, such as chat applications.
- Making RPC by having the receiving application interpret a message as a function call.

There are two main principles according to which communication can proceed between two computers: one way is by sending ***datagrams***, which are an independence packets of data send over network that contain the address of both the sender (source) and receiver (destination). These addresses are indicated as the computer address and port number. Another way is by setting up a ***connection (socket)*** between two computers and send data through this connection. In the first case, the option of using ***multicast*** is available, which means sending data packets at the same time to several connected members of particular group Communicating by means of datagrams has the advantage of being quit simple, its disadvantage is the lack of security. When sending a datagram, it is not certain that it will really reach the receiver; neither it will be certain that the datagrams will arrive in the same order as they were sent. In some applications, this will be not a problem but in other applications, this will be a problem of losing data. If this is the case, the datagrams is not used. When the ***socket*** (client-server) technique is used, there is a computer, or *server*, which provides a service. Other computers, or *clients*, can be associated with the server to gain access to this service[Ska00].

A server can often handle several clients at the same time It is also important for communication between a server and its clients to be reliable. A client who wants to connect to a server will create a new socket, where the server's IP address and port number are indicated. When this connection is ready, the input and output streams for both directions will be transferred [Ska00].

3. **Remote Method Invocation (RMI):** RMI is a Java's implementation of RPC for java-object-to-java-object distributed communication. Sockets are built around sending bytes while RMI provides a way to call methods on objects on other systems. It operates at a higher level of abstraction than socket-based programming, even though RMI does use sockets under the covers. RMI enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. A Java program can make a call on a remote object once it obtains a reference to the remote object, either by looking up the remote object in the bootstrap naming service provided by RMI or by receiving the reference as an argument or a return value. A client can call a remote object in a server, and that server can also be a client of other remote objects. RMI uses object serialization to marshal and unmarshall parameters and does not truncate types, supporting true object-oriented polymorphism [Saf06].

3.3 Routing

Routing (or routeing) is the process of selecting paths in a network along which to send data or physical traffic. Routing is performed for many kinds of networks, including the telephone network, the Internet, and transport networks. Routing directs forwarding, the passing of logically addressed packets from their source toward their ultimate destination through intermediary nodes; typically, hardware devices called routers, bridges, gateways, firewalls, or switches. Ordinary computers with multiple network cards can also forward packets and perform routing, though with more limited functionality and performance. The routing process usually directs forwarding based on routing tables, which maintain a record of the routes to various network destinations. Thus constructing routing tables, which are held in the router's memory, becomes very important for efficient routing [Wik07c].

Figure (3.2) shows the routing schemes, there are four different schemes, which differ in their delivery semantics [Cou01, Kha00, Gri02, Sno01, Pau02, and Wik07c]:

- ***Unicast*** delivers a message to a single specified node.
- ***Broadcast*** delivers a message to all nodes in the network.
- ***Multicast*** delivers a message to a group of nodes that have expressed interest in receiving the message.
- ***Anycast*** delivers a message to any one out of a group of nodes, typically the one nearest to the source.

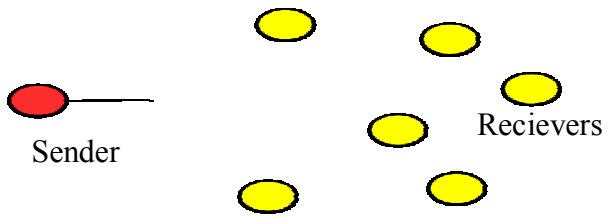

Figure (3.2) Routing Schemes [Wik07c]

3.3.1 Unicast

It is a one-to-one relationship where source will send one signal to one destination as shows in figure (3.3). In situations where many users need the same data, unicast can quickly consume the bandwidth of a network, as many copies of the data be sent through the same routers to reach individual users. The most widely used protocols today are the traditional unicast protocols such as User Datagram Protocol (UDP) and Transmission Control Protocol (TCP), which provide point-to-point delivery. The UDP provides an unreliable datagram service and TCP provides a reliable stream. With a UDP connection, messages are sent with best effort delivery. There is no congestion control in the protocol to allow adaptation to the current network congestion level. A TCP connection provides for reliable ordered delivery of bits from sender to receiver. TCP makes the assumption that the connection is simply a stream and that there are no explicit message boundaries. The semantics of TCP completely rule out prioritizing messages within a stream and delivering messages out of order. TCP and UDP are only intended for use in connecting two processes. These protocols are inefficient when used to send messages to multiple destinations. With TCP and UDP, a message intended for multiple destinations must be copied and a separate copy of the message sent to each individual receiver [Gri02, Sno01].

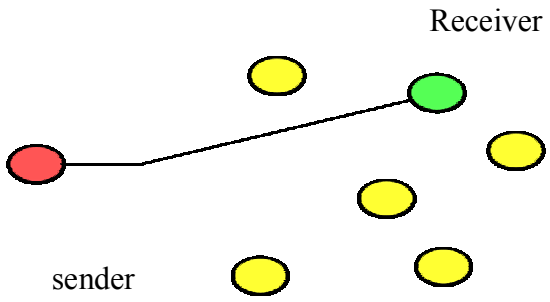

Figure (3.3) Unicast [Wik07c]

3.3.2 Broadcast

A broadcast allows a sender to communicate with every user that can listen to the channel using a single broadcast message. A satellite link, allowing a news source to broadcast to all users in the shadow of the satellite, is one example. Other examples include cable TV, microwave, and the Ethernet. With *n* users, broadcast can be *n* times cheaper than sending *n* separate unicast messages [Sno01].

A broadcast means that the network delivers one copy of a packet to each destination. On bus technologies like Ethernet, broadcast delivery can be accomplished with a single packet transmission. On networks composed of switches with point-to-point connections, software must implement broadcasting by forwarding copies of the packet across individual connections until all switches have received a copy [Kha00, Wik07c]. Figure (3.4) illustrates the broadcast routing.

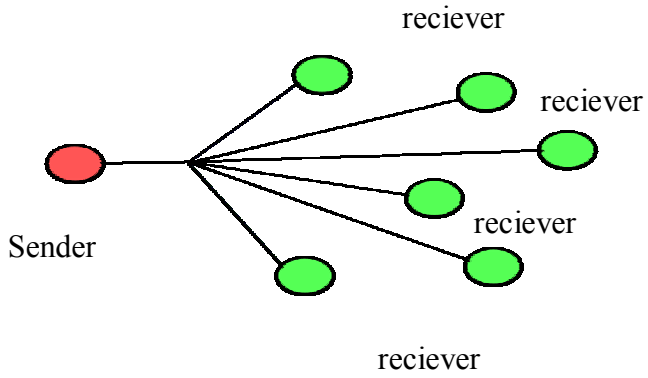

Figure (3.4) Broadcast [Wik07c]

3.3.3 Multicast

It is the idea of sending data from a single source to multiple destinations with a minimum use of network resources as shown in figure (3.5). There are wide varieties of applications that use multicast technology. Video conferencing, Internet stock market tickers, whiteboards, data warehousing, distributing product updates, and company wide updates all find great value in transmitting the same data to groups of users.

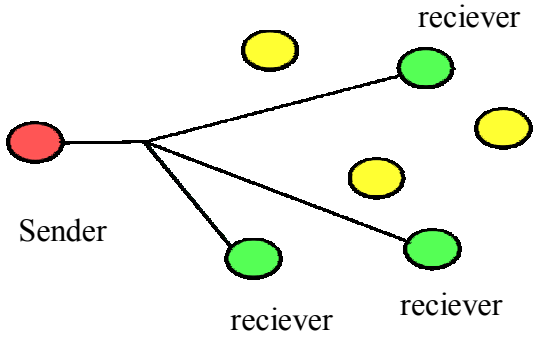

Figure (3.5) Multicast [Wik07c]

When the number of sender and receiver involved in a data communication is one-to-many or many-to-many or many-to-one, multicast is used as the means of data communication. The sender(s) and receivers are assumed to be part of a group. The features of a multicast group are described below [Cou01, Pau02]:

- A host can be a member of any number of multicast groups.
- The membership to a multicast group is dynamic, the sender(s) and receivers can join or leave the group at any time. For scalability, the join and leave operation has to be simple without any side effects.
- To be a sender of a group, it is not necessary that the host is a member of the group.
- Each group is identified by a given IP (from 224.0.0.0 to 239.255.255.255 address in networks.

- Data communication is done using UDP. This is to avoid the overhead of reliability and flow control that is associated with TCP.

Multicast, for the most part, is UDP based. As such, it is a best-effort delivery of services. This is fine for most multimedia applications, since there is no need to retrieve lost frames of live video, etc. Some great research is being done in the field of reliable multicast to make it more reliable as well as practical for more diverse usage [Gri02].

The multicast groups can be classified either as permanent or transient groups. The transient groups remain in existence as long as there are members in the group. However, permanent groups remain in existence even when the number of members in the group is zero. A part from this, the multicast groups can be classified either as dense or sparse groups based on the distribution of the group members in the network. With the advent of multicasting, many applications have emerged that can derive maximum benefit from multicasting of data. The multicast applications can be divided into the following categories [Cou01, Pau02]:

- Single-point to multi-point (e.g., Audio-Video broadcasts, Database updates, Push applications).
- Multi-point to multi-point (e.g., Video-conferencing, Distance Learning, Multiplayer Games).
- Multi-point to single-point (e.g., Resource Discovery, Data Collection, Auctions).

Multicast massages provide a useful infrastructure for constructing distributed systems with the following characteristics [Cou01]:

1. ***Fault tolerance based on replicated services:*** A replicated service consists of a group of servers. Client requests are multicast to all the members of the group, each of which performs an identical operation. Even when some of the members fail, clients can still be served.
2. ***Finding the discovery servers:*** in spontaneous networking: multicast massages can be used by servers and clients to locate available discovery services in order to register their interfaces or to look up the interfaces of other services in the distributed system.
3. ***Better performance through replicated data:*** data are replicated to increase the performance of services in some cases replicas of the data are placed in user's computers. Each time the data change, the new value is multicast to the processes managing the replicas.
4. ***Propagation of event notifications*:** multicast to a group may be used to notify processes when something happens. For example, a news system may notify interested users when a news massage has been posted on a particular newsgroup.

The use of a single multicast operation instead of multiple send operations amounts to much more than a convenience for the programmer. It is enables the implementation to be efficient and allows it to provide stronger delivery guarantees than would otherwise be possible [Cou01].

- ***Efficiency:*** the information that the same massage is to be delivered to all processes in a group allows the implementation to be efficient in its utilization of bandwidth. it can take steps to send the massage no more than once over any communication link, by sending the massage over a distribution tree; and it can use network hardware support for multicast where this is available. The implementation can also

minimize the total time taken to deliver the massage over to all destinations, instead of transmitting it separately and serially.

- ***Delivery guarantee:*** if a process issues multiple independent *send* operations to individual processes, then there is no way for the implementation to provide delivery guarantees that affect the group of processes as a whole. If the sender fails half-way through sending, then some members of the group may receive the massage while others do not, and the relative ordering of two massages delivered to any two group members is undefined.

An ideal multicast routing algorithm has the following features [Pau02]:

- The load on network should be minimal. This essentially involves avoiding loops and avoiding traffic concentration on a link or a subnetwork.
- It should support reliable transmission.
- The routing algorithm should be able to select optimal routes, taking into consideration different cost functions, including available resource, bandwidth, number of links, node connectivity, price to be paid and end-to-end delay. It should further maintain optimality of the routes after any changes occur in the group or the network.
- It should minimize the amount of state that is stored in the routers, so that more groups can be supported in a network without any scalability issues.
- The data transmitted should reach only the members of the group.

3.3.4 Anycast

Anycast is a network addressing and routing scheme whereby data is routed to the "nearest" or "best" destination as viewed by the routing topology. The term is intended to echo the terms unicast, broadcast and multicast. In unicast, there is a one-to-one association between network address and network endpoint: each destination address uniquely identifies a single receiver endpoint. In broadcast and multicast, there is a one-to-many association between network addresses and network endpoints: each destination address identifies a set of receiver endpoints, to which all information is replicated. In anycast, there is also a one-to-many association between network addresses and network endpoints: each destination address identifies a set of receiver endpoints, but only one of them is chosen at any given time to receive information from any given sender. See figure (3.6).

On the Internet, anycast is usually implemented by using Border Gateway Protocol (BGP) to simultaneously announce the same destination IP address range from many different places on the Internet. These results in packets addressed to destination addresses in this range being routed to the "nearest" point on the net announcing the given destination IP address. Anycast is best suited to connectionless protocols (generally built on UDP [Wik07c].

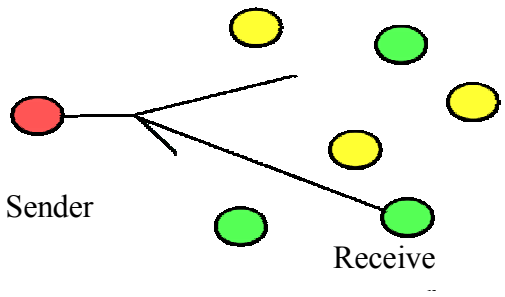

Figure (3.6) Anycast [Wik07c]

Chapter Four

System Implementation and Testing

Chapter Four

System Implementation and Testing

4.1 Introduction

Emerging high-speed networks allow machines to access remote data nearly as quickly as they can access local data. This trend motivates the use of *cooperative caching* which coordinate the file caches of many machines distributed on a LAN to form a more effective overall file cache.

This chapter demonstrates the proposed a Cooperative Caching for a Distributed System (CCDS). The CCDS system depends on the concept of multicast and unicast routing technique architecture and memory allocation mechanism. It stores the frequently used data in a fast memory either at a client or at a server which is connected to clients via a network to improve the performance of the distributed file system by reducing the number of disk accesses.

CCDS involves three logical entities: clients, servers and managers. Clients access the blocks stored on the remote servers, and the managers control the CCDS. The control provided by the managers includes locating the required blocks in the local and global caches and decide from which cache to get the block. Every machine in the LAN contains the CCDS with its three entities. Although it is enticing to think of cooperative caching as simply another layer in the storage hierarchy, the CCDS differs from the other levels of the storage hierarchy, because the cooperative cache can potentially involve every machine in the system since the cache is distributed across all the clients and it therefore shares the same physical memory as the local caches of the clients.

Figure (4.1) depicts the cooperative caching system communication.

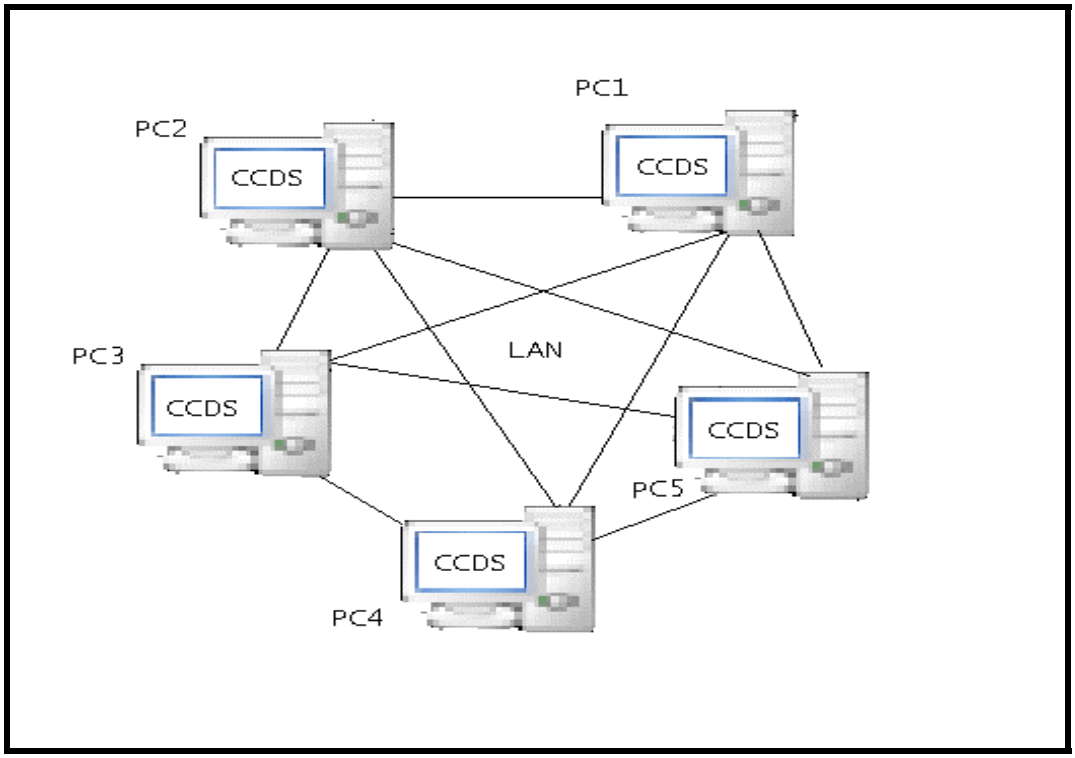

Figure 4.1 CCDS system communication group

The CCDS must be implemented using a language, which support the networking operations, in addition to the operating system API. The CCDS system was implemented using Java programming language (JBuilder Editor), which is a network operation support language. The system is running on a wired LAN at which all computers are connected to a network switch, and all computers are running under windows XP operating system. The java facilities that play a key role in the implementation of the CCDS system are

- Java networking:
- Java multithreading:
- Java Platform Independent

4.2 CCDS Architecture

In CCDS system, the datagram method is used to send messages to several receivers at a time. Instead of sending a message to a specific receiver, it will be send to a group of receivers and this group known as a *multicast group*. The sender of a multicast message is normally included in the multicast group. To send or receive messages in java, a socket is created as a MulticastSocket in which the used port is specified.

A multicast message is sent in the form of datagram where one of the things a datagram contains is the receiver's network address. When sending a multicast message, a network address for some particular receiver is not indicated. Instead, an imaginary *multicast address* is used that all the receivers in the group know and listen to. This multicast address is an IP address in the interval (224.0.0.1 to 239.255.255.255) in this work the used IP address were (230.0.0.1). Every receiver joins a particular group must report this by calling the method joinGroup in class MulticastSocket. Java provides many methods which are used in the CCDS to implement the communication of multicasting. They are:

getLocalHost()	Gives the IntAddress to one's own computer
isMulticastAddress()	indicates whether this IntAddress can be used to send multicast messages
DatagramSocket(port)	creates a socket through the port
setSoTimeout(ms)	indicates that receiver will wait at most ms millisecond
DatagramPacket(data, len, iaddr, port)	creates a datagram with the content data of length len will be sent to the port at iaddr
	Gives the data that has been sent or will sent

getData()	
getLength()	Gives the length of data that has been sent or will be sent
getAddres()	gives the sender's IntAddress
getPort()	gives the sender's port number
MulticastSocket(port)	creates a multicast socket through the port
Send(pack)	sends the datagram pack
receive(pack)	receives the datagram pack
joinGroup(iaddr)	Join the multicast group with the multicast address iaddr
leaveGroup(iaddr)	leaves the multicast group with the multicast address iaddr

The CCDS system uses the multicast and unicast routing techniques with multithreading technique. This approach offers architecture of two processes, one of them (client) which sends a request to all servers in the group, and the second process (server) serves the requested client, by establishing a connection between these two processes. The CCDS system read a block from a file (file name, block no.) using socket IPC. The operation allowed to be performed on the files is read operation only. Therefore, no needs to consider the consistency and coherency of the data, which are usually needed with the write operation. The CCDS system consists of three main classes. These are Manager, Client, and Server. Figure (4.2) illustrates the model of the CCDS system on one computer.

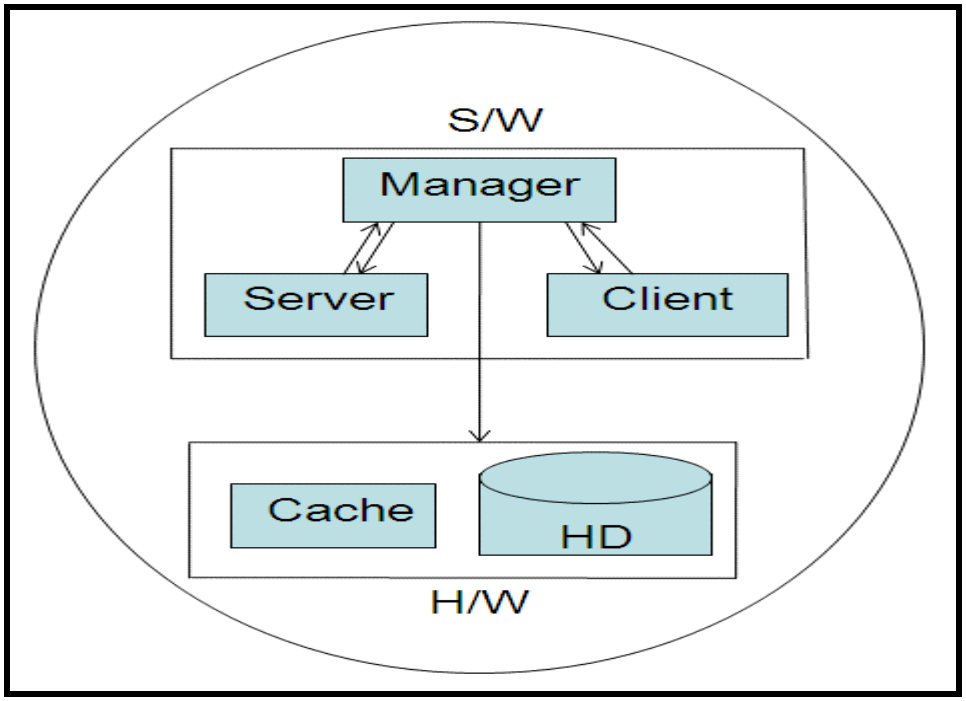

Figure 4.2 CCDS Model on one PC

At the beginning, the CCDS system starts the manager and the server classes on each computer in the distributed system. Then each computer waits for a request which is a data consists of a (file name and block number). When a request arrives, the CCDS manager searches the required block in its local cache or (when the block not available) sends a multicast request to all servers in the multicast group searching for the required block in their caches and wait for reply (maximum wait 1 second). If any server finds the requested block in its cache then it will open a unicast connection with the requested client and sends the required block. The client receives the block and gives it to the manager for use. If none of the servers fined the requested block, then no reply will be received by the client (time out of wait). In this case the manager will read requested block from the hard disk.

Figure (4.3) shows the CCDS mechanism in a LAN with 7 PC five of them contains the CCDS Model and one of them (PC4) is the sender of a multicast request to the other four computer in the multicast group while PC2 and PC7 out of the multicast group

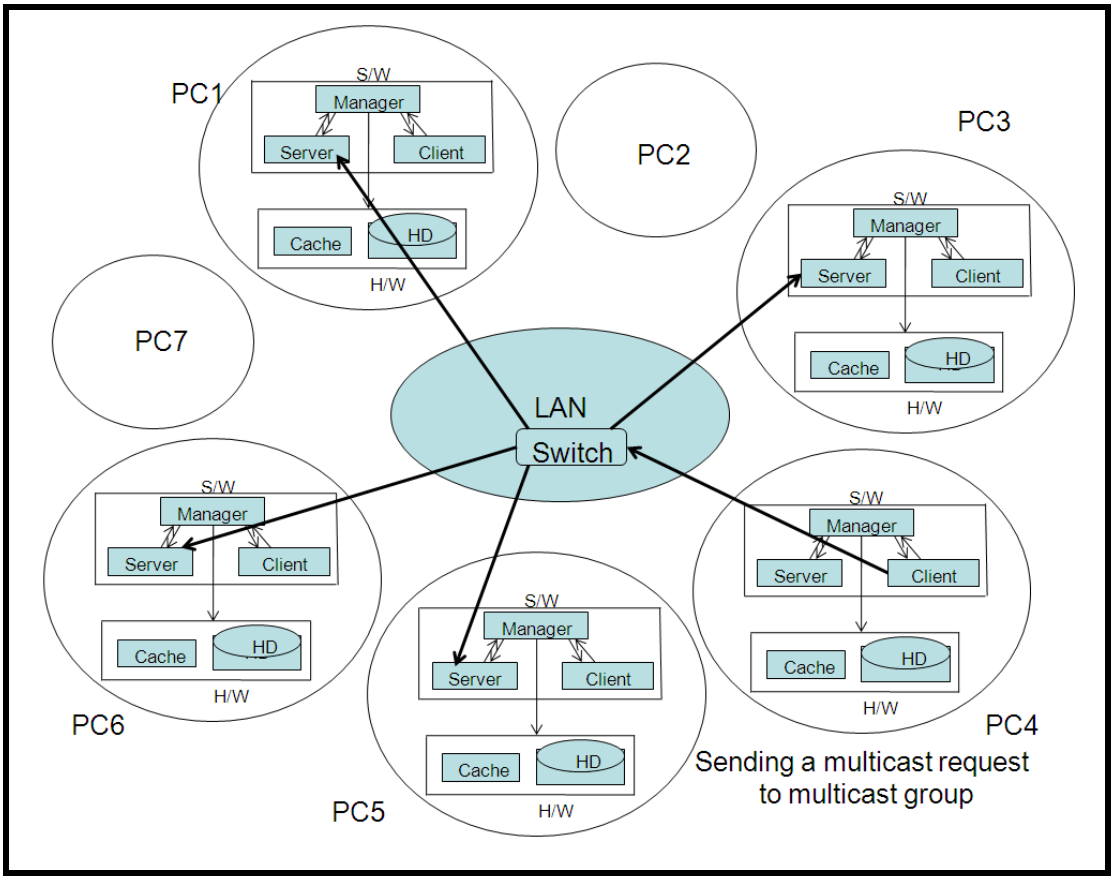

Figure (4.3) shows the CCDS sending a multicast request.

While figure (4.4) shows the CCDS mechanism in a LAN with 7 PC five of them contains the CCDS Model and one of them (PC1) is the sender of a unicast response to the sender of the request (PC4) in the multicast group.

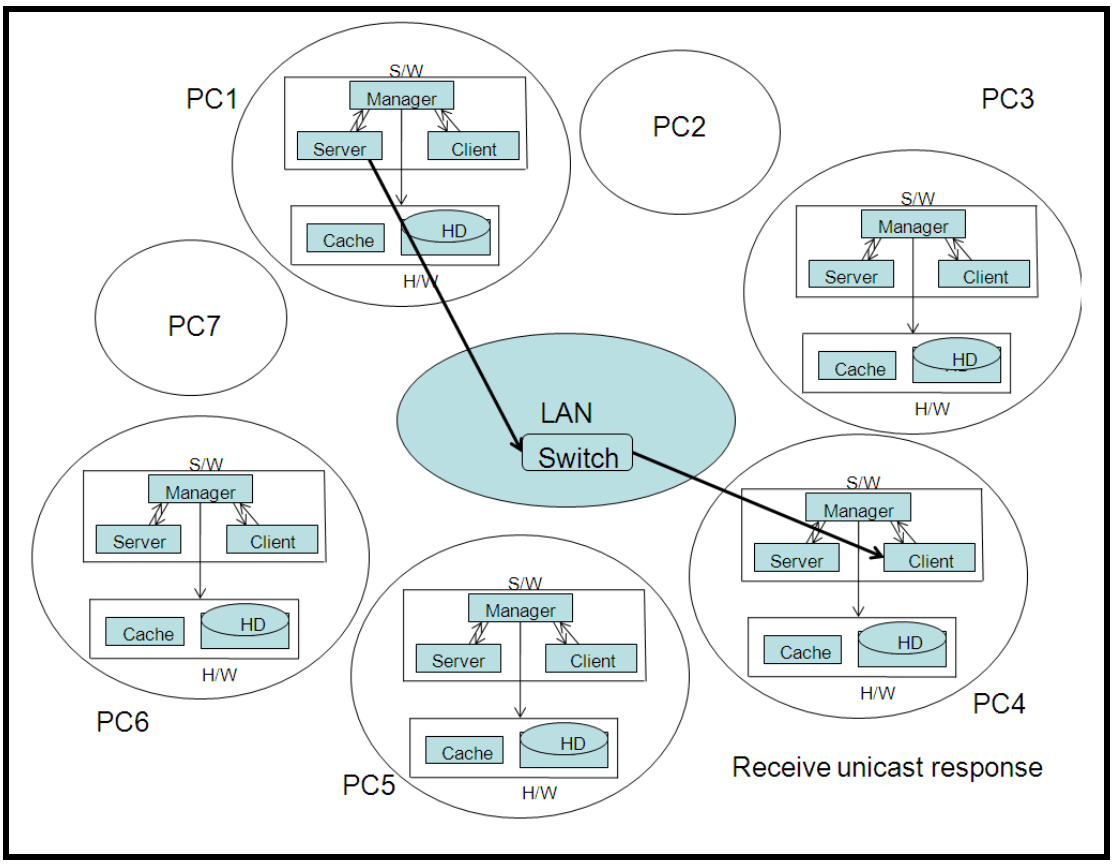

Figure (4.4) shows the CCDS receiving a unicast response.

If no one of the five PCs in the multicast group in the LAN contain the required block in its local cache the sender PC4 will wait for 1 second and assume that the block is not available in any cache of these PCs, therefore PC4 will read it from its local hard disk. When the required block founded on multiple PCs, the first PC that finds the block on its local cache and sends response, this response will the only unicast response is taken and the other are ignored. Figure (4.5) shows the CCDS architecture.

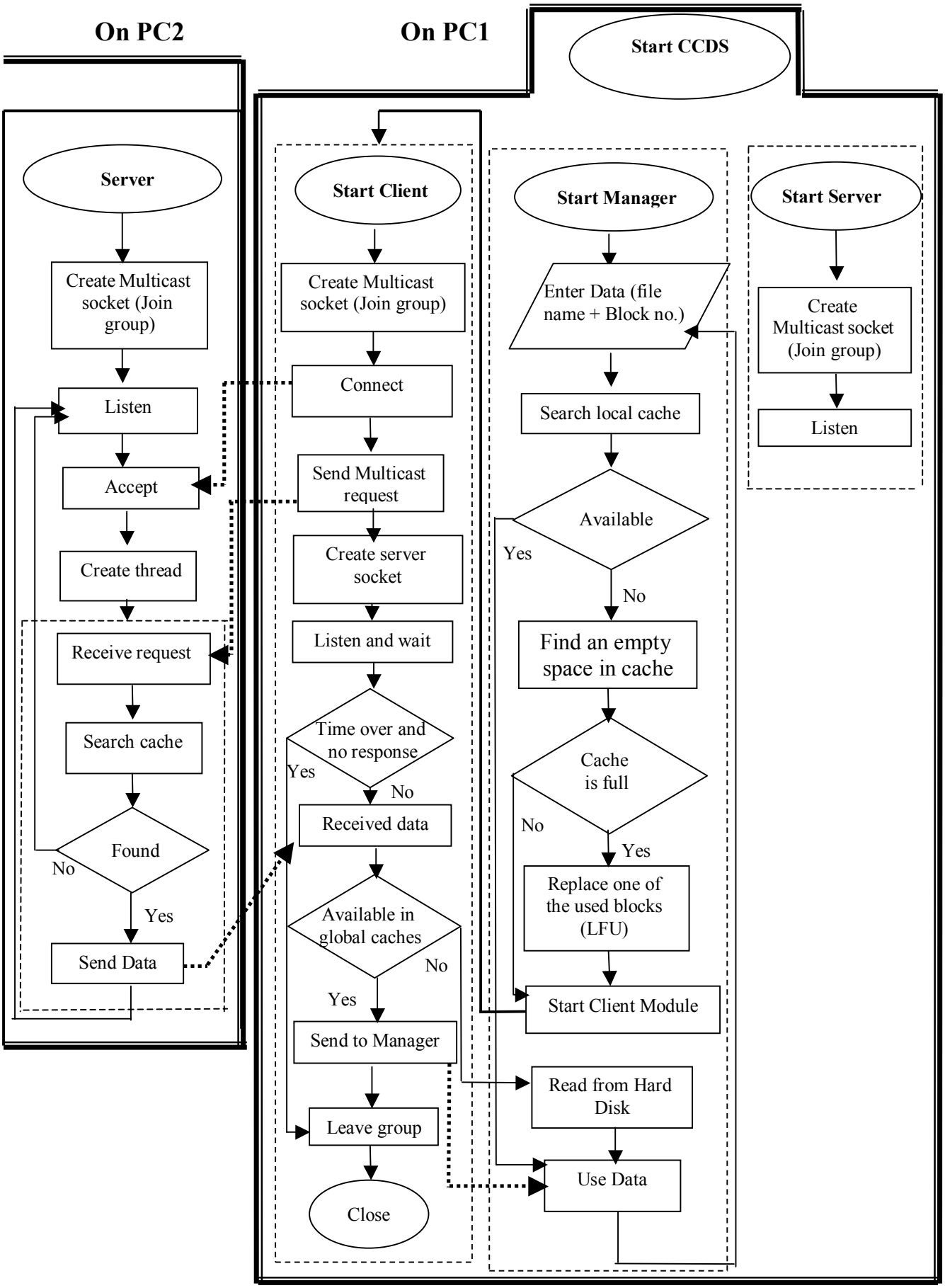

Figure (4.5) The CCDS architecture

4.2.1 The CCDS Manager Module

This module is the controller or the manager of the distributed system. It is responsible for managing the remote caches as well as the local cache. The cooperative cache manager has two major functions:

First, it performs cache replacement of its local cache. The replacement policy used in the CCDS system is LFU, this strategy chooses the block that is accessed least frequently. A reference count is associated with every cached block. The count is incremented when the block is reused. The objective of this algorithm is to find the most popular blocks and keep them in the cache. In order to do it, the system keeps counter with the number of times each block has been referenced. Whenever a block has to be discarded, the one with the smaller number of references is replaced with a new block when the cache is full.

The second function of is searching for the required blocks on the local and remote caches. The cooperative cache manager uses a static block size which is equal to 4KB, and has static data structure array of records where each record consists of the following fields:

```
String   FileName;
int   BlockNumber;
boolean Master;
int Counter;
```

The main functions of the manager module are:

- Read the file name and block number.
- Search for the specific block in the local cache. If the block is not available locally, it starts client module which multicasts the request to the other clients in the distributed system to check if it is available on a global cache. If the block is not found locally (local cache) and globally (distributed caches) then, the block will be read from the hard disk.
- Search for an empty space in the local cache to put the new block in it.
- If the local cache is full and the manager wants to put the requested block in the local cache, the manager must replace it with the least frequently used block in the local cache.

Algorithms (4.1), (4.2), (4.3), (4.4) and (4.5) are the distributed cache manager algorithms.

Algorithm (4.1) Manager ◄

Step1: Read data (FileName, BlockNumber).
Step2: Search local cache (algorithm 4.2)
Step3: Check available, if (yes) go to Step9
Step4: Find an empty space in cache (algorithm 4.3)
Step5: Check full cache, if (no) go to Step7
Step6: Replace one of the used blocks (using LFU) (algorithm 4.4)
Step7: Start Client class to search global caches (algorithm 4.6)
Step8: Read from Hard Disk (algorithm 4.5)
Step9: Use Data

Algorithm (4.2) Search local cache
Input: file name, block number Output: i or -1 (return the index of the block in local cache)
Step1: i=-1; x=false; Step2: Search the local cache for a specific block of the specified file Step3: if the block is found i = get place of record; and break; Step3: return (i);

Algorithm (4.3) Find empty space in cache
Output: index of the empty place in local cache or (-1) when cache is full
Step1: index=-1; Step2: if cache is not full then set index=address of empty place Step3: return (index)

Algorithm (4.4) LFU
Output: the place of item that have the minimum value of counter in local cache
Step1: i=1; min= 0; minv= localcache[min].counter; Step2: search in a local cache if localcache[i].counter<minv then Step3: min= i; minv= localcach[i].counter; i++; go to Step2; Step4: return min;

Algorithm (4.5) read from HD
Input: File name, block number Output: 0 or -1
Step1: Read a specific block (data) from the file name Step2: Store the information about this block in a record in the manager table, and read the block content in the local cache.

4.2.2 The CCDS Client Module

This module is for the sender of the request to the distributed caches. The main functions of the client module are:

- Join multicast group and open a connection.
- Send multicast request to all servers in the group of machines, which are connected by a LAN, asking for a specific block in a specific file.
- Create a server socket and Listen to receive the reply from the server which have the required block and sends the received information to its manager to use it.
- Wait the response from servers maximum 1 second if there is no responses assume the block is not found and terminate connection
- Store the received block and information (file name, block number) in its local cache.

- Leave multicast group and close the connection (only the client).

The client must use an IP address and port number of the computer which the server will listen to it. Then the clients connected to the network will be able to send and receive data by using get output stream and get input stream as data. The client module uses the following Java methods:

Created by a client *(create client)* new Socket(addr, port) : this create new socket which connect to the port in the server addr. *(send/receive data)* getOutputStream() : gives a stream to send data to other computer. getInputStream() : gives a stream to read data from the other computer. *(close connection)* Close() :breaks the connection.

In multicast client must identify an IP address (230.0.0.1) of a multicast group, then use a port number for sending datagram that differ from the port used in client-server. By using join method the client will be connected to multicast group and sending data to them. Algorithm (4.6) the distributed cache client algorithm.

Algorithm (4.6) Client
Step1: Create Multicast Socket (Join group). Step2: Connect to LAN Step3: Send Multicast request to all servers in LAN Step4: Create server socket Step5: Listen and wait (max 1 s) Step6: If wait time over and no response if (yes) then go to Step10 Step7: Receive data from remote server that reply the request Step8: If there is received data from LAN servers if (no) then go to Manager-Step8 (Read from Hard Disk) Step9: Send data to Manager-Step9 (Use Data) Step10: Leave group. Step11: Close connection.

4.2.3 The CCDS Server Module

This module is for the receiver of the requests from the distributed cache clients. The main functions of the server are:

- Join multicast group, and listen to a specific port waiting for a request from a client.
- If the received request comes from the same machine, (i.e., Client and Server have the same IP), then stop serving.
- Create a threaded server to serve the request.
- Search the local cache, if the required block is found locally, then the server open a socket connection with the sender client and send the required block.
- Wait for another request.

The server must use a port for listening to discover whether there is a client wishing to make a connection. If there is a client, then it gives the port

number to the client. The server waits for the client using accept() method and returns from this method when a client has been connected. The servers just like the client can receive/send data by using get input and output stream as data to be read and written. Algorithm (4.7) is the distributed cache server algorithm. The methods used by the server are:

Created by a server *(Create server)* new ServerSocket(port) : create new server socket which gives a socket that will listen in the port. *(accept request)* accept() : wait until a client has been connected; return the socket created by the client. *(receive/send data)* getInputStream() : gives a stream to read data from the other computer. getOutputStream() : gives a stream to send data to other computer. *(close connection)* close() :breaks the connection.

Algorithm (4.7) the distributed cache server algorithm.

Algorithm (4.7) Server (remote)
Step1: Create Multicast Socket (Join group). Step2: Listen to LAN Step3: Accept Step4: Create thread Step5: Receive request from any client (client-Step3) Step6: Search cache Step7: Check if the requested block is found if no go to Step2 Step8: Send data to (client-Step7) and go to Step2

4.3 Tests and Results

In a computer networking the term *bandwidth* also known as (throughput) refers to the data rate supported by a network connection or interface. Bandwidth is expresses in term of bit per second (bps). Where bandwidth represents the total distance or range between the highest and lowest signals on the communication channel (band). Also represents the capacity of the connection, the greater capacity is more likely that greater performance will follow.

The term *latency* to any of several kinds of delays incurred in processing of network data. A so-called *low latency* network connection is one that generally experiences small delay times, while a *high latency* network connection is one that suffers from long delays.

Although the theoretical peak bandwidth of a network connection is fixed according to the technology used, the actual bandwidth obtain varies over the time and is affected by high latencies. Excessive latency creates bottlenecks that prevent data from filling the network pipe, thus decreasing effective bandwidth.

The most common way to measure latency is by determining the time it takes for a given network packet to travel from source to destination and back this called *round trip time*, which is not the only way to specify latency.

The used LAN to implement CCDS system is wired LAN Ethernet (Fast Ethernet): IEEE 802.3 100Mbits/sec this transfer rate equal to 12.5MB/s.

To calculate the time of flight in the LAN by using the considered average LAN bandwidth which equal to (7.5MB/s) and the used file size by the following equation:

Time of flight= file size / LAN bandwidth

T= (2MB) / (7.5 MB/s)

T= 0.266 s this time is wasted in transferring the file (delay)

Therefore, in this work the speed of transferring data from the used HDD and the LAN were calculated using java.util.Date().getTime() method. This method is typically preferred over getTime() since it is more reliable.

If HDD speed is compared with the wired LAN Ethernet speed which is the comparison of average 37.5MB/s the speed of HDD and 12.5MB/s (or average 7.5MB/s) the speed of LAN. It is clear that HDD is faster than LAN speed. The differences which included with the delay network overload are the delays caused by the OS requirements. The OS requirements are: using device drivers, loading files time, time of using LAN card, and converting the sends message to signals, received signals to massage, and the time needed for the language used (Java) in implementing CCDS system.

In addition to that the using of datagram to send a multicast massage has the disadvantage is the lack of security. When sending a datagram, it is not certain that it will really reach the receiver; neither it will be certain that the datagram will arrive in the same order as they were sent. The average of lost datagram was: 1%.

Figure (4.6) shows the transfer rate to transfer (20) different files of the same size from HDD to the RAM of the same computer with disk fragmentation and without disk fragmentation, and through the LAN from the RAM of one computer to the RAM of another computer.

Transfer rate = Data size / total time.

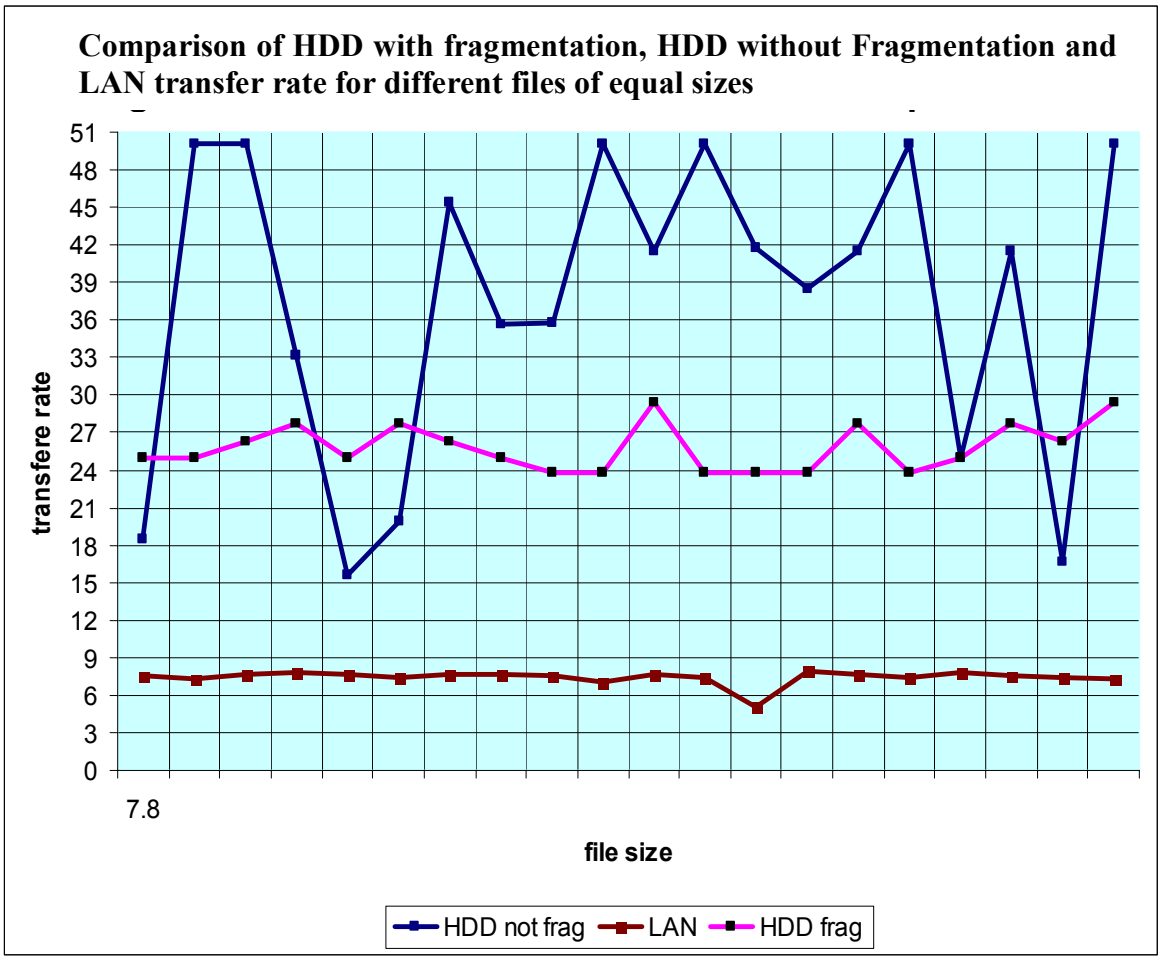

Figure (4.6) shows the transfer rate of transferring (20) different files of the same size

Figure (4.7) shows the calculated time to transfer (20) different files of different sizes from HDD to the RAM of the same computer with disk fragmentation and without disk fragmentation, and through the LAN from the RAM of one computer to the RAM of another computer.

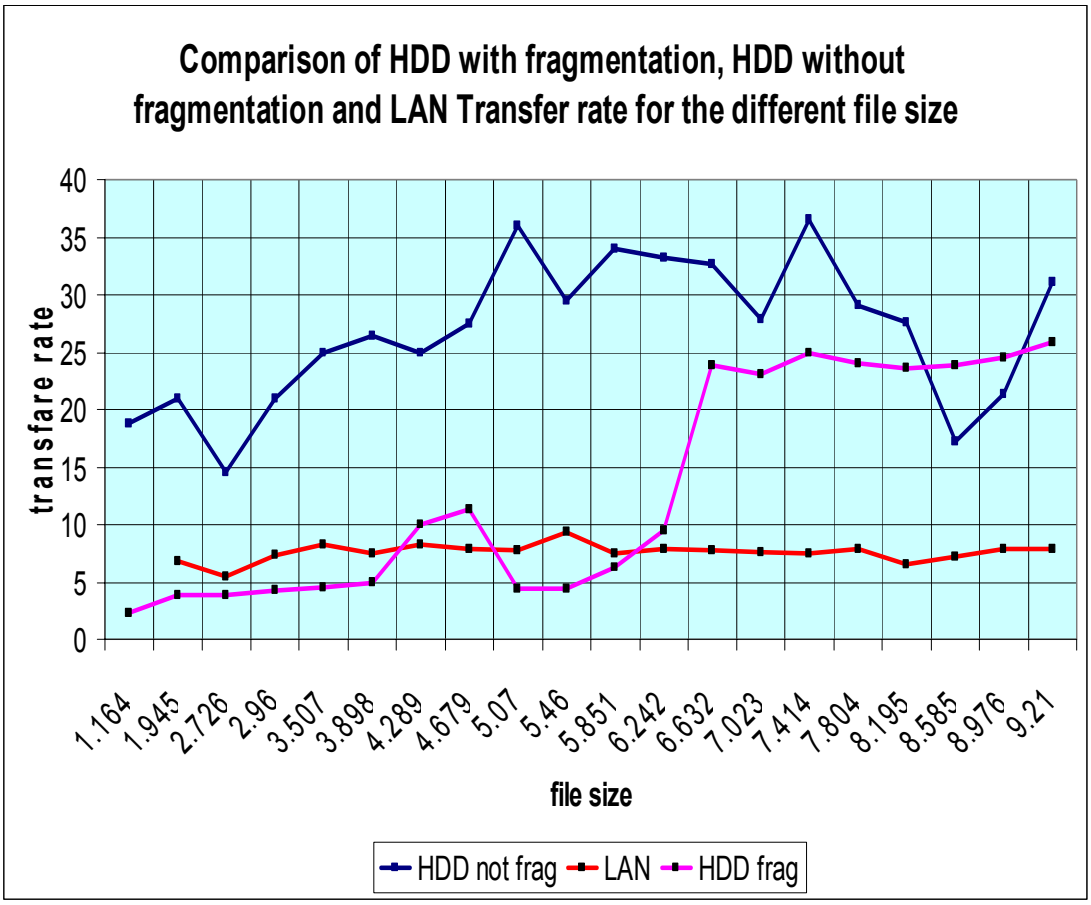

Figure (4.7) shows the transfer rate to transfer (20) different files of different sizes

4.4 Examples

There are many measures of software performance, split between size, speed, and resource use. In the CCDS system, the time- related performance is the main measurement.

In order to evaluate the performance and verify the basic functionality of the developed CCDS, a number of tests were conducted. The measurements are performed using different files with equal block size (4KB), all with and without load on the system. The LAN consists of (8) computers.

EX1: When PC1 requests block number (45) from file name (hhh.bin)

1- The CCDS system runs the manager and server modules, which joins the multicast group.
2- The manager searches for the block 45 in its local cache and the block was not available.
3- The client module joins a multicast group and sends a multicast request to all servers in the LAN (All PCs).
4- Each server (except the server of the sender PC1) then searches for the block in its local cache (LAN caches) then, there were no response from the servers.
5- The manager reads this block from the hard disk and save it in its local cache.

The time required to access local cache was 0 ms (less than 0.5 ms)

The time required to send a request on LAN was nearly 62 ms.

The time required to read block from HDD was nearly 16 ms.

The total time = 78 ms.

EX2:

When PC2 requests block number (45) from file name (hhh.bin)

1- The CCDS system runs the manager and server modules, which joins the multicast group.
2- The manager searches for block 45 in its local cache and the block was not available.
3- The client module joins a multicast group and sends a multicast request to all servers in the LAN (All PCs).
4- Each server (except the server of the sender PC2) then searches for the block in its local cache (LAN caches).
5- PC1 server finds this block in its local cache, opens socket connection with the requested client and sends reply to PC2 with the data block.
6- PC2 client receives this data block and stores it in local cache then it uses this data.

The time required to access local cache was 0 ms (less than 0.5 ms)
The time required to send a request on LAN was nearly 46 ms.
The total time = 46 ms.

EX3:

When PC1 requests block number (45) from file name (hhh.bin) again

1- The CCDS system runs the manager and server module, which joined the multicast group.
2- The manager searches for the block in its local cache and the block is available.
3- The manager will read this block and use it.

The needed time of local cache was 0 ms (less than 0.5 ms)
The total time = 0 ms.

After a lot of read process, either through LAN or through HDD many of the requested blocks are saved in the local cashes of the LAN and this had minimized the required time of disk accesses. Table (4.3) shows sequence of requests from different computers to different files and blocks over the LAN and the needed time for each one.

PC request	File name	Block no.	PC response (LAN)	Time	HDD Time	Copy type	Count
PC7	Aaa.bin	999	PC3	46 ms		Non-master	2
PC1	Kkk.bin	34	PC8	62 ms		Non-master	9
PC3	a.bin	88			15 ms	Master	1
PC6	dddd.bin	28			16 ms	Master	1
PC2	Wvv.bin	14	PC2	0 ms		Non-master	3
PC5	nn.bin	90	PC1	93 ms		Non-master	2
PC8	l.bin	156			31 ms	Master	1
PC4	Ooo.bin	287			15 ms	Master	1
PC9	cc . b i n	542	PC4	78 ms		Non-master	5
PC1	jjjj . b i n	159	PC5	94 ms		Non-master	8
PC5	b . b i n	478			16 ms	Master	1
PC9	O . b i n	435			32 ms	Master	1
PC2	a a a . b i n	99	PC7	78 ms		Non-master	3
PC1	a . b i n	123	PC3	62 ms		Non-master	2
PC6	Dd . b i n	45	PC2	46 ms		Non-master	6
PC4	w . b i n	12	PC8	78 ms		Non-master	3

Table (4.3) several CCDS request and time

EX4:

When there are 10 PCs each one requests 10 blocks of different files and all of these files are available in one PC cache. There is a load on this PC server, but because of the multithreading, it is able to reply to all of these requests. The server creates a thread to each request and serves it at a time as shown in table (4.4).

The average time needed to each request to be served = 67.28 ms

Table (4.4) Average of 100 different block of different files request from 10 PC on LAN to one PC (load on one pc cache)

PC request	**PC response**	**Average Time**	**Copy type**
PC0	PC4	49.2 ms	Non-master
PC1	PC4	78.0 ms	Non-master
PC2	PC4	58.8 ms	Non-master
PC3	PC4	78.0 ms	Non-master
PC5	PC4	70.0 ms	Non-master
PC6	PC4	78.0 ms	Non-master
PC7	PC4	62.5 ms	Non-master
PC8	PC4	50.3 ms	Non-master
PC9	PC4	70.0 ms	Non-master
PC10	PC4	78.0 ms	Non-master
Average of all time	**67.28**		

EX5:

When PC8 requests 100 different blocks of different files and all of these files available in PC3 cache there is a load on reading from the cache and load on network transmitting blocks.

The average time needed to each request to be served = 114.03 ms.

This is illustrated in table (4.5).

PC request	File name	Block no.	PC response (LAN)	Time	Copy type	Count
PC8	a . b i n	10	PC3	188 ms	Non-master	4
PC8	b . b i n	20	PC3	110 ms	Non-master	2
PC8	c. b i n	30	PC3	172 ms	Non-master	7
PC8	d . b i n	40	PC3	157 ms	Non-master	3
PC8	e . b i n	50	PC3	156 ms	Non-master	3
PC8	f . b i n	60	PC3	94 ms	Non-master	2
PC8	g . b i n	70	PC3	78 ms	Non-master	8
PC8	h . b i n	80	PC3	109 ms	Non-master	5
PC8	i . b i n	90	PC3	125 ms	Non-master	5
PC8	j . b i n	100	PC3	141 ms	Non-master	8
PC8	k . b i n	200	PC3	93 ms	Non-master	6
PC8	l . b i n	300	PC3	62 ms	Non-master	9
PC8	m . b i n	400	PC3	140 ms	Non-master	4
PC8	n . b i n	500	PC3	78 ms	Non-master	3
PC8	o . b i n	600	PC3	156 ms	Non-master	7
PC8	p . b i n	700	PC3	140 ms	Non-master	5
average on 100 request =114.03 ms						

Table (4.5) Load from PC8 to PC3 throug h LAN

Chapter Five

Conclusion and Future Work

Chapter Five

Conclusion and Future Work

5.1 Conclusion

From this research, many things were noticed and concluded. The following are the most important ones:

1. The disadvantage of using multicast is the possibility of losing the datagram by the average 1% (UDP not reliable) and lake of security.
2. There is a big difference in the transfer rate of fragmented hard disk and non-fragmented hard disk. The fragmented HD transfer rate reduced approximately to the half when equal file sizes are transferred. The fragmented HD transfer rate reduced than LAN transfer rate when small file sizes are transferred (see figure (4.6) and figure (4.7)).
3. When different clients requests different blocks available on one server, the serve time becomes high because of the load on the network and on the server.
4. When one client requests different blocks available on one server, the serve time becomes high because of the load on the client and on the server.
5. Each machine (node) in the multicast group has both the client and server processes. It could ask for a request or serves a request. When a client sends a request the server on this machine (node) not is allowed to perform the serving process.
6. By comparing the calculated HDD transfer rate and LAN transfer rate, we notice that: the used LAN is slower than the HDD speed which makes getting the data from the hard disk is better than cooperative caches. To solve this problem a fast LAN must be used.

5.2 Suggestion for future work

After building CCDS, several ideas come to mind that may improve the overall performance. These ideas have been left as recommendation for future work. These recommendations are:

1. Build Cooperative Caching inside file system layer and use it with the web.
2. Using another search technique in local cache.
3. When the required block is available on different machines a serving machine must be chosen in a way that balance the load on the machines.
4. Enhancement to the algorithm used (LFU) by adding to it specific timer to solve the problem of the blocks that entered for the first time, or just use another replacement algorithm.
5. Preftching is the next logical step after caching. It uses the same data structure and mechanisms but instead of remembering blocks requested in the past, its tries to predict the blocks that will be requested in the near future. So far, a cache only keeps the blocks already used by the applications.
6. It possible to increase the performance of cooperative caching system by implementing the write operation through the cache with coherency and consistency mechanisms to avoid writing problems.
7. Running the proposed system on a Metropolitan Area Network (MAN) and Wide Area Network (WAN).
8. It also seems interesting to study a way to increase the system security.

References

References

1. [And01] Anderson R., "Security Engineering", 2001, on site http://www.cl.cam.ac.uk/~rja14/book.html.

2. [And03] Andrade H., Kurc T., Sussman A., Borovikov E. and Saltz J., " On Cache Replacement Policies for Servicing Mixed Data Intensive Query Workloads" ,2003.

3. [Ann04] Annapureddy S., Freedman M.J., Mazieres D., " Shark: Scaling File Servers via Cooperative Caching" , University of New York, 2004.

4. [Are95] Arens Y. and Knoblock C. "Intelligent Caching: Selecting, Representing and Reusing Data in an Information Server", University of Southern California 1995.

5. [Arl99] Arlitt M., Friedrich R. and Jin T, "Performance Evaluation of Web Proxy Cache Replacement Policies" 1999.

6. [Bit02] Bitorika A., "Scalability Issues in Cluster Web Servers", M.SC. Thesis, University of Dublin, 2002.

7. [Boy06] Boydens J., Steegmans E., "Confrontation of an aspect -oriented solution with an object-oriented solution, a case study on caching", 2006.

8. [Cor97] Cortes T., "Cooperative Caching and prefetching in parallel/distributed file systems", PHD thesis, Dept. of computer architecture, Barcelona, 1997.

9. [Cou01] Coulouris G., Dollimore J., Kindberg T., "Distributed System", Addison-Wesly, Third edition, 2001.

10. [Dah95] Dahlin M. D., Wang R. Y., Anderson T. E. and Patterson D. A., "Cooperative Caching: Using Remote Client Memory to Improve File System Performance", University of California at Berkeley, 1995.

11. [Dei01] Deitel H. M. and Deitel P. J., "Java How to Program", Fourth edition, published by Prentice Hall, 2001.

12. [Gab01] Gabrick K., Weiss D.," J2EE and XML Development" , Bookpool.com, 2001. http://www.amazon.com/J2EE-XML-Development-David-Weiss/dp/1930110308.

13. [Grb03] Grbic A., "Assessment of Cache Coherence Protocols in Shared memory – Multiprocessors", PHD Thesis, Department of Electrical and Computer Engineering, University of Toronto, 2003.

14. [Gri02] Gridley D., Clow B. and Wilson D., "multicast (Protocols, Routing, Architectures, and Applications) ", 2002.

15. [Gwe97] Gwertzman J., Seltzer M., "World-Wide Web Cache Consistency", Harvard University, 1997.

16. [Has04] Hassoon I., "Exploiting Idle Memories in LAN Using Cooperative Caching Technique", MSC Thesis, Dept. Computer Science, Baghdad University, 2004.

17. [Kha00] Khattab T. M. S., "Performance Analysis of Wireless Local Area Networks (WLANs)", M.SC. thesis, Electronics and Communication Dept. Faculty of Engineering Cairo University Egypt, 2000.

18. [Kos95] Kostkova P., "Process allocation in tightly coupled multiprocessor", M.SC. thesis ,Dept. of Software Engineering, Mathematics and Physics Faculty, Charles University, Prague,1995.

19. [Lag02] Langston H., DeCoro C. and Weinberger J. " Cash: Distributed Cooperative Buffer Caching", Courant Institute of Mathematical Sciences, New York University, 2002.

20. [Lan02] Lancelotti R., Ciciani B. and Colajanni M.,"Distributed cooperation schemes for document lock up in multiple Cache Server", Dept. of Computer Engineering, University of Roma "Tor Vergata", 2002.

21. [Lix06] Li X., Plaxton C.G., Tiwari M., and Venkataramani A., " Online Hierarchical Cooperative Caching", University of Texas, 2006.

22. [Mir05] Miranda H., Leggio S., Rodrigues L., and Raatikainen K. " A Stateless Neighbour-Aware Cooperative Caching Protocol for Ad-Hoc Networks ", University of Lisboa, Portugal, 2005.

23. [Nar05] Narravula S., Balaji P., Vaidyanathan K., Jin H. W. and Panda D. K., "Architecture for Caching Responses with Multiple Dynamic Dependencies in Multi-Tier Data-Centers over InfiniBand", Dept. of Computer Science and Engineering, The Ohio State University, 2005.

24. [Nel90] Nelson M. N., "Virtual Memory vs. The File System", Western Research Laboratory, California, USA, 1990.

25. [Pau02] Paul P. and Raghavan S. V., "Survey of Multicast Routing Algorithms and Protocols", paper, Dept. of Computer Science and Engineering, Indian Institute of Technology Madras, 2002.

26. [Rus98] Russinovich M., "Inside the cache manager" from windows IT Pro book 1998. http://www.windowsitpro.com/Articles/Index.cfm?ArticleID=3864.

27. [Saf06] Al-Saffar R. B. J., "Fault and Accounting Components Monitor System Design and Implementation", MSC thesis, Dept. Computer Science, Al-Nahrain University, Baghdad, 2006.

28. [Sar96] Sarkar P. and Hartman J., "Efficient Cooperative Caching using Hints", Dept. of Computer Science, University of Arizona, 1996. http://www.cs.arizona.edu/swarm/papers/ccache.

29. [Ska00] Skansholm J., " Java From the Beginning", Addison-Wesly, , 2000.

30. [Sno01] Snoeyink J., Suri S. and Varghese G., "A Lower Bound for Multicast Key Distribution", paper IEEE 2001.

31. [Sun94] Sun microsystem, "Cache File System (CacheFS) White Paper ", Inc., California, U.S.A., 1994.

32. [Tan02] Tanenbaum A. S. and Steen M., "Distributed system principle and paradigms", 2002 on site. http://www.prenhall.com/divisions/esm/app/$author_tanenbaum/custom/dist_sys_1e.

33. [Voe98] Voelker G. M., Anderson E. J., Kimbrel T., Feeley M. J.y, Chase J. S., Karlin A. R., and Levy H. M., "Implementing Cooperative Prefetching and Caching in a Globally-Managed Memory System", Dept of Computer Science and Engineering University of Washington, 1998.

34. [Wan03] Wang C., Xiao L., Liu Y., Zheng P. "Distributed Caching and Adaptive Search in Multilayer P2P Networks", Michigan State University, 2003.

35. [Web06] Webopedia site, "Cache ", 2006. http://www.webopedia.com/TERM/c/cache.html.

36. [Wik07a] Wikipedia site, the free Encyclopedia, "File System", 2007. http://en.wikipedia.org/wiki/File_system#searchInput.

37. 37[Wik07b] Wikipedia site, the free Encyclopedia, "Cache coherency" , 2007. http://en.wikipedia.org/wiki/Cache_coherency.

38. [Wik07c] Wikipedia site, the free Encyclopedia, "Routing ", 2007. http://en.wikipedia.org/wiki/Routing.

39. [Wik07d] Wikipedia site, the free Encyclopedia, "Cache ",2007. http://en.wikipedia.org/wiki/Cache.

الخلاصة

Caching هي تقنية خزن البيانات المستخدمة بكثرة في ذاكرة سريعة, اما عند الزبون او عند الخادم, الذي يكون مربوط للزبائن بواسطة الشبكة. الذاكرة المتعاونة *Cooperative Caching* يبحث عن تحسين كفاءة أنظمة ملفات الشبكة عن طريق تحديث محتويات ذاكرة الزبائن والسماح للطلبات لا ان تلبى من قبل ملف الذاكرة المحلية للزبون وانما أن تلبى بواسطة ذاكرة زبون اخر.

هذه الأطروحة تطمح إلى بناء وإنجاز(الذاكرة المتعاونة للنظام الموزع) (Cooperative Caching for a Distributed System (CCDS), التي تدير الذاكرات البعيدة والمحلية في شبكة محلية(LAN) التي تعمل تحت نظام التشغيل (Windows). تم تطويره باستخدام لغة البرمجة (Java). الذاكرة المتعاونة للنظام الموزع تتكون من ثلاث أجزاء : الإدارة (Manager) , الزبون (Client), و الخادم (Server). الأدارة هو المسيطر على CCDS والذي يتظمن ايجاد البيانات المطلوبة في الذاكرت المحلية والبعيدة ويقررمن أي ذاكرة يحصل على البيانات. الأدارة يسيطر على كل الذاكرات المتعاونة. الزبون يصل للبيانات المخزونة عند الخادم. هو المرسل للطلب الى الذاكرات الموزعة. الزبون يسيطر على ذاكرة الزبون المحلية. الخادم يخدم الزبائن الطالبة. هو مستقبل الطلبات من ذاكرات الزبائن الموزعة. الخادم يسيطر على ذاكرة الخادم. كل حاسوب في شبكة محلية LAN يحتوي على CCDS مع مكوناته الثلاثة.

فائدة CCDS قدمت دعما فعالاً ل(Scalability) لنظام الذاكرة المتعاونة بسبب الاتصال وتوزيع البيانات يعتمد على تقنيات مسلك (Multicast) و(Unicast) و بالإضافة إلى دعم مشاركة المصادر للبيانات الموزعة.

جمهورية العراق
وزارة التعليم العالي و البحث العلمي
جامعة النهرين
كلية العلوم

الذاكرة المتعاونة

النظام الموزع

رسالة
مقدمة إلى كلية العلوم, جامعة النهرين
كجزء من متطلبات نيل شهادة الماجستير في علوم الحاسوب

من قبل
ورود سعد إبراهيم ألعبيدي

بكالوريوس
٢٠٠٤

المشرفون

د. لمـــياء حافظ خالد

كانون الأول ٢٠٠٧

د. بان نديم الخلاق

ذو الحجة ١٤٢٨