Doctoral Dissertation

Optimal Design Of System Resource Management for Distributed Networks

Akiko Nakaniwa

Department of Electronics Engineering
Graduate School of Engineering
Kansai University

March 2002

Preface

The work presented in this thesis is a study on the optimal design of system resource allocation for distributed networks. An explosive increase in the number of network users has caused network servers and files to become overloaded. Due to this, we are faced with several essential issues, such as the decline of reliability, the increase of the access delay, etc. In finding a solution for these issues, distributed allocation of files and the mirroring of network servers have been considered and a number of studies concerning this issue have been done widely. In constructing effective networks and providing emerging multi-media services, one of the most important problems today lied in the allocation of mirror servers and copies of files on the networks. We first focus on constructing cost-effective networks and consider the optimal file allocation which minimize the cost. Secondly, we focus on constructing high reliability networks and consider the optimal allocation of mirror servers and files which maximizes the system reliability. Using these optimizations, we quantitatively evaluate general characteristics in distributed networks. We also consider the more important case of mobile database systems in which the system reliability is more likely to decline and develop the dynamic file allocation method in mobile environments. This thesis consists of the following 5 chapters.

Chapter 1 introduces the basic concepts and the background for the study.

Chapter 2 presents optimal design of distributed systems. A distributed system is a collection of autonomous computers linked by a network, with software designed to produce an integrated computing. In recent years the availability of high-performance personal computers, workstations and server computers and high-speed computer networks make it possible to shift towards distributed systems and away from centralized systems. In this chapter, we first present the characteristics of distributed systems in contrast with centralized systems. To develop distributed database systems, there are many complex and interrelated design issues. Among those issues, we particularly elaborate the concepts of data allocation and query optimization which are especially important problems in design of distributed database systems.

Chapter 3 proposes the optimal file allocation model such that the total cost is minimized subject to the total delay restriction, in view of a trade-off between costs and delays. In this model, we introduce a 0-1 integer programming formulation by setting the 0-1 variables on the allocation of files that determine whether to exist on each server. Our optimal file allocation model belongs to the class of NP hard problems. Therefore, we use an approximate method when solving this optimization problems in order to deal with practical large-scale systems. Using this approximate method, we quantitatively show the general characteristics on the file allocation in distributed systems. In this model, we introduce many cost and delay parameters to evaluate the total cost and delay in the system more exactly. In constructing practical systems, it is necessary to investigate the weight and the contribution of each parameter to the total cost. It is very useful to show how to estimate cost and delay parameters, on the basis of this analysis. In this chapter, we also analyze the sensitivity of these parameters on our optimal file allocation model and make clear the influence between principal parameters.

Chapter 4 focuses on constructing high reliability networks. This chapter mainly includes the following two topics. First, we propose the optimal mirror server allocation model such that the system reliability is maximized subject to the cost and delay restrictions. In this model, we consider not only the replicating of files that exist in each network server but also the mirroring of network servers themselves. Several mirror servers, which store copies of the same files and have the same function as network servers, are set in networks for the purpose of load balancing. Distributed allocation of mirror servers results in prompt access of users and improvement of system reliability. However, this causes the cost of setting up mirror servers and managing the whole system to increase. It is obvious that there is also a trade-off between the reliability and cost. This is why we consider the system reliability as well as the cost and delay in this optimization model. This optimization model is capable of dealing with various kinds of network topologies. As in the case with the optimal file allocation model, we formulate this optimal mirror allocation model into a 0-1 integer programming model, and we use an approximate method for numerical analysis in order to analyze more large-scale systems. Our objective is to find the optimal mirror allocation by solving this model, and to show quantitatively the general characteristics of the load balancing and the improvement of the system reliability by the distributed mirror allocation.

Secondly, in this chapter, we also consider dynamic file allocation in mobile database systems, which has become increasingly important. As we may easily imagine, the system reliability is more likely to decline in mobile environments. In particular, we consider providing reliable and time critical database information services in a more dynamic and critical environment where the need for high reliability is greater. We develop the innovative strategy for dynamic file allocation for this purpose. This strategy consists of two phases: 1) Periodically generating optimal file allocation via a heuristic algorithm based on the complete system state, and 2) Using knowledge-based policy to reallocate resources based on the partial system state changes. In the first phase, we periodically use a heuristic algorithm to find the allocation of files which minimize the operating cost subject to a set of constraints, such as the reliability and response time. Because of critical time requirement, we do not apply 0-1 integer programming formulation to this case. In the second phase, we use the knowledge-based reallocation policy to dynamically reallocate files when the system state changes significantly. Simulation experiments are needed to validate and refine this reallocation policy. We expect this strategy will contribute to consider various design issues concerning dynamic file allocation in mobile networks.

Chapter 5 summarizes the overall conclusions in this thesis.

The work summarized in this thesis is conducted by the author during her Ph.D. course at the Department of Electronics Engineering, Graduate School of Engineering, Kansai University.

Acknowledgements

This research described here has been carried out during the author's tenure of doctoral course at the Graduate School of Engineering, Kansai University, under the guidance of Professor Hiromi Okada at the Department of Electronics Engineering, Faculty of Engineering, Kansai University, Japan.

The author would like to express her sincere appreciation to Professor Hiromi Okada for his supervising, continuing encouragement, and valuable discussions throughout this research.

The author would also like to thank Professor Yasuo Nomura, Professor Shigeru Imanishi, Professor Yasuhisa Omura, Professor Yukio Iida, Professor Toshitaka Kojima, Professor Katsuhiro Yokota and Nobuhisa Ohigashi of the Faculty of Engineering, Kansai University, for their kind encouragement and guidance.

The author is much indebted to Associate Professor Hiroyuki Ebara for his helpful support and directions during the course of this research.

The author would also like to appreciate Professor Wesley W. Chu of the Department of Computer Science, University of California, Los Angeles for his considerate support and guidance.

Gratitude is also owed to past and present members of Information Network Laboratory, especially Mr. Masaki Onishi, Mr. Hiromichi Nishiyama, Mr. Jun Takahashi and Mr. Takashi Kato for their continuous support.

The author would also like to express her appreciation to Dr. Kiyoko Aoki for her English advice. Finally, the author thanks her family for their patience and support during the whole period of her education.

Contents

List of Figures

Chapter 1

Introduction

In the past decade the computer industry has been revolutionized. Users are demanding ever more complex applications to meet the need to provide information to support business functions in an increasingly competitive business environment. This has resulted in increasingly diverse requirements for data manipulation and presentation where often the information required is not in a predictable form. Today, applications are being constructed using a wide range of third-party, packaged software to satisfy the need for rapid application development and to exploit the new levels of functionality and ease of use that they provide. Tens of thousands of packaged software are available for the most popular computers and operating systems, thus providing an increasingly wide choice. End-users are consequently demanding the integration of a variety of discrete applications available at the desktop to accommodate greater levels of manipulation, integration and ease of use. The combination of powerful packaged software, the availability of advanced communication technologies and rapidly advancing PC architectures has been the stimulus for higher user expectation in terms of what levels of integration and ease of use are achievable.

These forces have created the need for a more flexible IT infrastructure giving greater freedom of design to build information systems. The infrastructure is required to facilitate integration of potentially diverse data sources, a range of media types (text, audio, video, etc.) and in a user environment which allows flexibility of access in a way that hides complexity. Furthermore, the infrastructure must exhibit levels of security, robustness and maintainability at least equivalent to that of the traditional centralized mainframe environment.

To date, many organizations have successfully evolved (some over twenty years) core information systems based on the centralized mainframe computing environment. These (largely transaction processing) systems are gradually being revamped as senior managers see opportunities to create more cost-effective, flexible and easy-to-modify systems. Evidence suggests that there is a large-scale move away from centralized mainframe systems towards distributed systems which is perceived as the key approach to infrastructure design for the foreseeable future.

The history of distributed systems has its origins in the development of multi-user computers and computer networks in the 1960s and was stimulated by the development of low-cost personal workstations, local area networks and the UNIX operating systems in the 1970s [2].

The potential for distributed systems first became apparent in the early 1970s, a few years after the emergence of minicomputers. The use of minicomputers as single-user graphical workstations and their effectiveness for software development and other interactive tasks were important influences, pointing the way for the provision of a more effective environment for interactive computing than could be achieved with multi-user timesharing systems. But the necessary hardware and software to make single-user computers fully effective and the communication facilities to enable them to be used in a cooperative fashion were absent.

The earliest and some of the most significant developments aimed at filling those gaps were made by a team of researchers working at the Xerox Palo Alto Research Center (known as Xerox PARC) in the period 1971 - 80. These included the development of the first single-user workstations, file and print servers, the first high-speed local network (the Ethernet) and several experimental distributed systems. The first workstation developed at Xerox PARC was the *Alto*. It had a high-resolution monochrome display, a mouse, 128 kilobytes of main memory, a 2.5 megabyte hard disk and a microprogrammed central processor that executed machine-level instructions at speeds of 2-6 microseconds. The Altos were linked by the Ethernet. They were exploited both as personal workstations and as server platforms and were used in a wide range of experimental services and applications including the Xerox Distributed File Server, and the Grapevine and Cedar systems.

Other research done before 1980 that had an important impact on subsequent developments include the Cambridge Distributed Computing System (CDCS), the Apollo Domain system, the Newcastle Connection and the Locus distributed operating system. From the start of the 1980s there was a rapid expansion of research and development in distributed systems, for example the Accent, Mach, Amoeba, Argus, V-system and Chorus research projects and the Sun NFS-based product development.

Nowadays, distributed systems are implemented on hardware platforms that vary in a size from a few workstations interconnected by a single local area network to the Internet – a world-wide interconnected collection of local and wide area networks linking thousands or even millions of computers.

Applications of distributed systems range from the provision of general-purpose computing facilities for groups of users to automated banking and multimedia communication systems, and they embrace almost all commercial and technical applications of computers. Increasingly, organizations are interconnecting computers for cooperative processing, and utilizing distributed database systems to control access to their decentralized information resources.

In chapter 2, we present the characteristics of distributed systems from several perspectives and elucidate the advantage of a distributed systems. The advantage of a distributed database system compared with a centralized one are increased reliability and availability, decreased access time, easy expansion, etc. The acceptance and widespread usage of distributed databases highly depends on their efficiency. However, for the development of a distributed database system we need to find effective solutions to many complex and interrelated design issues, including network topology, hardware allocation, data partitioning, data allocation, query optimization, data replication, concurrency control, reliability, and recovery. In this chapter, among those issues, we consider data allocation and query optimization which are especially important problems in design of distributed database systems. Considering these issues, it is important to supply a database management system with efficient tools to determine allocation of system resources such that the availability is increased, the access time and the cost is decreased. Main purpose of this thesis is to develop the optimal design tool of system resource allocation for distributed systems.

In chapter 3, we first focus on constructing cost-effective networks. Distributed systems consist of several servers with databases, that separate to each other geographically, and each server constitutes a subnetwork with a number of users. In distributed database systems, files must be stored in at least one of servers, and are shared by several users connected with different servers. Users can access any files stored in any servers. Replicated copies of the files accessed frequently are to be stored in several servers. Each server should store the files that it accesses frequently to reduce the communication cost and delay. However, the replicated storage of the same files in the system causes the storage cost required to store them to increase. It is obvious that there is a trade-off between costs and delays. Some systems that optimizes only the cost is not useful, because the system may require a great delay.

We propose the optimal file allocation model such that the total cost is minimized subject to the total delay in distributed systems [103][104][109][110]. In this model, we introduce a 0-1 integer programming formulation by setting the 0-1 variables, in which way we can obtain the concrete assignment of files to servers. However, our optimal file allocation model belongs to the class of NP hard problems and it is difficult to find an exact solution efficiently. We use an approximate method as well as exact method such as the exhaustive search and the branch and bound, when solving this optimization problems in order to deal with practical large-scale systems. We compare the performance of the both exact method and the approximate method and show the usefulness of the approximate method. Using this approximate method, we quantitatively show the general characteristics on the file allocation in distributed systems. In our optimal file allocation model, we introduce many cost and delay parameters to evaluate the total cost and delay in a system more exactly. In constructing practical systems, it is also useful to investigate the weight and the contribution of each parameter to the total cost and show how to estimate cost and delay parameters, on the basis of this analysis. In this chapter, we also analyze the sensitivity of these parameters on our optimal file allocation model and make clear the influence between principal parameters.

In chapter 4, we focus on constructing high-reliability networks. In light of the tremendous potential of reliable distributed computing systems, it is remarkable that relatively little attention has been paid to developing better reliability technologies and marketing them. There have been a great amount of research work done on file allocation problems. However, Most of them hardly consider the reliability issue. If a computing system cannot be relied upon, or compels a trade-off between reliability and performance, organizations will not use computers in settings where reliability is an issue.

This chapter mainly includes the following two topics. First, we propose the optimal mirror server allocation model such that the system reliability is maximized subject to the cost and delay restrictions [107][108][113][114]. In this model, we consider the mirroring of network servers as well as the replicating of files. Recent explosive increase in the number of network users causes network servers to become overloaded, and thus decreasing the system reliability. The mirroring of network servers has been considered as an effective solution to this serious issue and various studies have been done widely. Distributed allocation of mirror servers results in prompt access of users and improvement of system reliability. However, it is obvious that there is also a trade-off between the reliability and cost. As in the case with the optimal file allocation model, we formulate this optimal mirror allocation model into a 0-1 integer programming model, and using an approximate method, quantitatively show the general characteristics of the load balancing and the improvement of the system reliability by the distributed mirror allocation.

Secondly, in this chapter, we also consider dynamic file allocation in mobile database systems, where the system reliability is more likely to decline. Recent rapidly-advances in wireless communication technology have made dynamic file allocation in mobile networks an important issue. Distributed database management will be changed due to mobility. Resources will have to be dynamically reconfigured in response to the mobility of the users. In particular, we consider providing reliable and time critical database information services in a more dynamic and critical environment where the need for high reliability is greater. To meet this critical requirement, we propose an innovative strategy for dynamic file allocation. This strategy consists of two phases: 1) Periodically generating optimal file allocation via a heuristic algorithm based on the complete system state and 2) Using knowledge-based policy to reallocate resources based on the partial system state changes. In this strategy, we first find the initial file allocation that minimizes the operating cost subject to reliability and response time, using a heuristic algorithm. This heuristic algorithm is based on A$*$ algorithm and we apply several rules for pruning the original search space to meet critical time requirement. In the second phase, we use the knowledge-based reallocation policy to dynamically reallocate files according to changes in the state of systems. Repeatedly applying the knowledge-based reallocation policy will yield file allocation away from the optimal. Thus, we periodically use the heuristic algorithm to re-optimize the file allocation for the latest system state. We expect this strategy makes it possible to consider various design issues concerning dynamic file allocation in mobile database systems.

Chapter 5 we summarize the overall conclusions obtained in this thesis.

Chapter 2

Optimal Design of Distributed Systems

2.1 Introduction

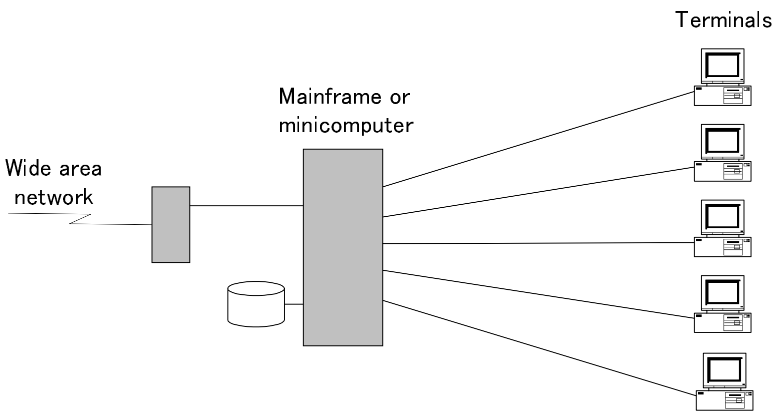


Fig.2.1. A centralized multi-user system

A **distributed system** consists of a collection of autonomous computers linked by a computer network and equipped with distributed system software. Distributed

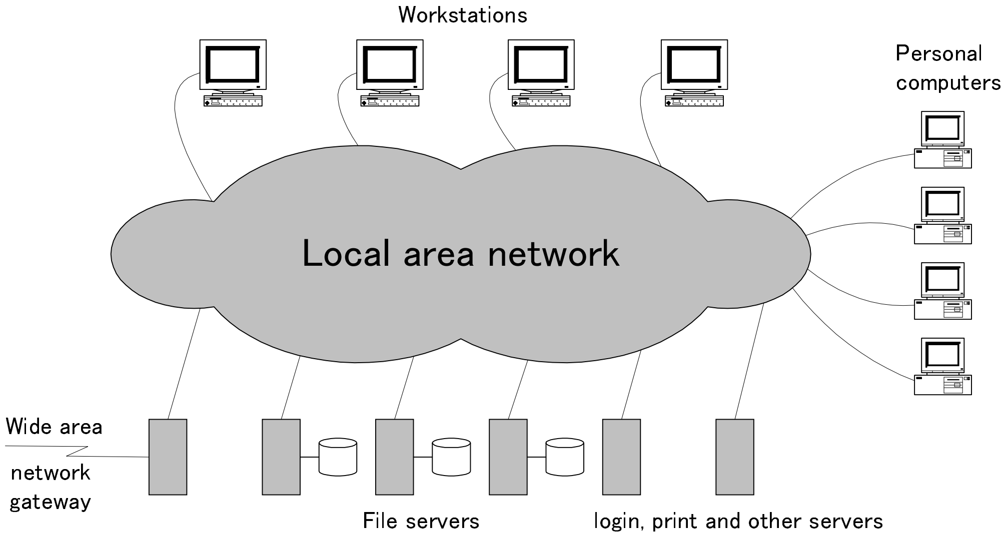


Fig.2.2. A simple distributed system

system software enables computers to coordinate their activities and to share the resources of the system, that is hardware, software and data. Users of a well-designed distributed system should perceive a single, integrated computing facility even though it may be implemented by many computers in different locations (see Fig.2.1 and 2.2).

The development of distributed systems followed the emergence of high-speed local area computer networks at the beginning of the 1970s. More recently, the availability of high-performance personal computers, workstations and server computers has resulted in a major shift towards distributed systems and away from centralized and multi-user computers. This trend has been accelerated by the development of distributed system software, designed to support the development of distributed applications.

Much research and development work has been done on the design of distributed systems and their underlying principles. The work continues, but there are already many effective practical implementations of distributed systems and a substantial body of theoretical and empirical knowledge about their design.

In this chapter, we present the current state of knowledge about the design and construction of distributed systems and to elucidate the concepts and techniques that underlie them. First in section 2.2, we present general structures of models of distributed systems and characteristics of distributed systems in contrast with centralized systems. In the following sections, we focus on the design issue of distributed database systems. To develop distributed database systems, there are a number of complex and interrelated design issues. Among those issues, we particularly elaborate the concepts of data allocation and query optimization. Data allocation and query optimization are especially important problems in design of distributed database systems.

As in many other fields of computer science, the development of distributed systems has progressed by the formulation of abstract models for systems that meet a range of general requirements, followed by the design and implementation of systems that support those models. Our method of description follows the same pattern. For each of the major topics, we define the requirements and describe the most important models that have been developed to meet them.

2.2 Characterization of Distributed Systems

2.2.1 Key Characteristics

Six key characteristics are primarily responsible for the usefulness of distributed systems. These are *resource sharing*, *openness*, *concurrency*, *scalability*, *fault tolerance* and *distribution transparency* [2]. In this section we define and elaborate on each of them. It should be noted that they are not automatic consequence of distribution; system and application software must be carefully designed in order to ensure that they are attained.

Resource Sharing

The term 'resource' is a rather abstract one, but it best characterizes the range of things that can be shared usefully in a distributed system. The range extends from hardware components such as disks and printers to software-defined entities such as files, windows, databases and other data objects. Users of both centralized and distributed computer systems are so accustomed to the benefits of resource sharing that they may easily overlook their significance. The benefits of shared access to a single filing system containing databases, programs, documentation and other

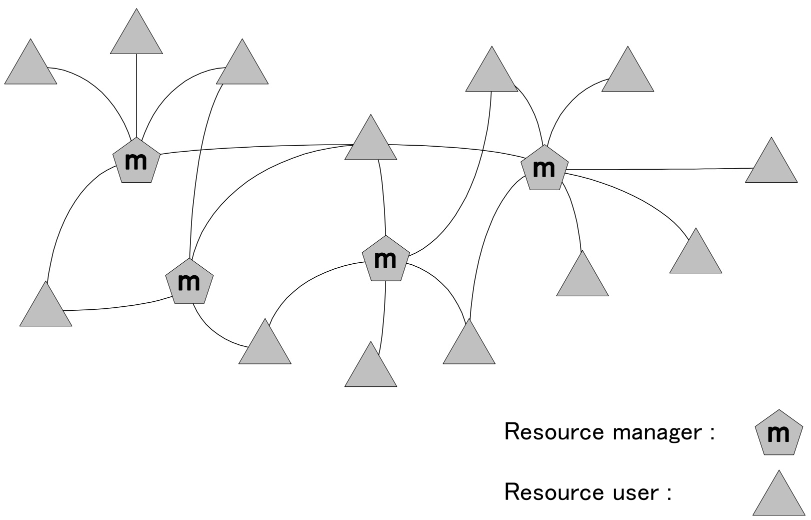


Fig.2.3. Resource managers and resource users

information were first recognized with the emergence of multi-user or time-sharing systems in the early 1960s, and multi-user UNIX system in the 1970s.

- Hardware devices such as printers, large disk drives and other peripherals are shared for convenience and to reduce costs.

- Data sharing is an essential requirement in many computer applications.

  – Software developers working as a team may need access to each other's work and can share the same development tools, requiring only a single copy of compilers, procedure libraries, editors and debugging aids; whenever a new development tool or a new release of a compiler is installed, all users immediately obtain access to it.

  – Many commercial applications, such as the systems used by airlines for seat reservation and ticketing, the networks operated by banks to support their branches and to operate their automatic teller (cash dispensing) machines and the systems operated by supermarket and other retail chains to coordinate their stock control, delivery and sales check-out, enable users to access shared data objects in a single active database.

- A rapidly-expanding area of application for networks and distributed systems is the use of computers in support of groups of users working cooperatively – project work, team management, remote teaching and many other cooperative tasks can benefit from such support. *Computer supported cooperative working* (known as *groupware*) depends heavily on the sharing of data objects between programs running on different workstations.

The resources of a multi-user computer are normally shared between all of its users, but the users of networked single-user workstations and personal computers do not automatically obtain the benefits of resource sharing. Resources in a distributed system are physically encapsulated within one of the computers and can only be accessed from other computers by communication. For effective sharing each resource must be managed by a program that offers a communication interface enabling the resource to be accessed, manipulated and updated reliably and consistently.

The generic term **resource manager** is sometimes used to denote a software module that manages a set of resources of a particular type. Each type of resource requires some separate management policies and methods, but there are also common requirements. These include the provision of a naming scheme for each class of resource, enabling individual resources to be accessed from any location; the mapping of resource names to communication addresses and the coordination of concurrent accesses that change the state of shared resources in order to ensure their consistency.

Fig. 2.3 illustrates an abstract views of a distributed system composed of a set of resource managers and a set of resource using programs. The resource users communicate with the resource managers to access the shared resources of the system.

Openness

The openness of a computer system is the characteristics that determines whether the system can be extended in various ways. A system can be open or closed with respect to hardware extensions – for example, the addition of operating system features, communication protocols and resource-sharing services. The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added without disruption to or duplication of existing services.

Openness is achieved by specifying and documenting the key software interfaces of a system and making them available to software developers. In a word, the key interfaces are *published*. This process is akin to the standardization of interfaces, but it often bypasses official standardization procedures which are cumbersome and slow-moving.

Historically, computer systems were largely closed. They executed programs in a range of programming languages but they did not allow application developers to extend the semantics of the languages to exploit new hardware or operating system features. UNIX was an early example of a more open system design. It includes a programming language, C, that allows programmers to access all of the facilities (resources) managed by the operating system. C is a high-level language, enabling programs to be compiled and executed on a wide range of different computers.

The availability of interprocess communication in BSD UNIX and other operating systems and of standard protocols for communication dramatically increased the scope for achieving openness in computer system design. It became possible for server processes running on any computer and any operating system to offer facilities for accessing their resources to other processes (clients) running in computers on the same network. The range of system resources and facilities available to an application program was no longer restricted by the system calls supported in the local operating system. This is the essential foundation on which current distributed systems have been developed.

Systems that are designed to support resource sharing in this way are termed *open distributed systems* to emphasize the fact that they are extensible. They may be extended at the hardware level by the addition of computers to the network and at the software level by the introduction of new services, enabling application programs to share resources. A further benefit that is often cited for open systems is their independence from individual vendors.

To summarize:

- Open systems are characterized by the fact that their key interfaces are published.

- Open distributed systems are based on the provision of a uniform interprocess communication mechanism and published interfaces for access to shared resources.

- Open distributed systems can be constructed from heterogeneous hardware and software, possibly from different vendors. But the conformance of each component to the published standard must be carefully tested and certified if users are to be protected from responsibility for resolving system integration problems.

Concurrency

When several processes exits in a single computer we say that they are executed concurrently. (Since each process only exists for the duration of a program execution, coexistence implies concurrency of execution.) If the computer is equipped with only a single central processor, this is achieved by interleaving the execution of portions of each process. If the computer has N processors, then up to N processes can be executed simultaneously (that is, in parallel), achieving an N-fold improvement in computational throughput.

In distributed systems there are many computers, each with one or more central processors. If there are M computers in a distributed system with one central processor each, then up to M processes can run in parallel, provided that the processes are located in different computers.

In a distributed systems that is based on the resource-sharing model described above, opportunities for parallel execution occur for two reasons (Fig. 2.4):

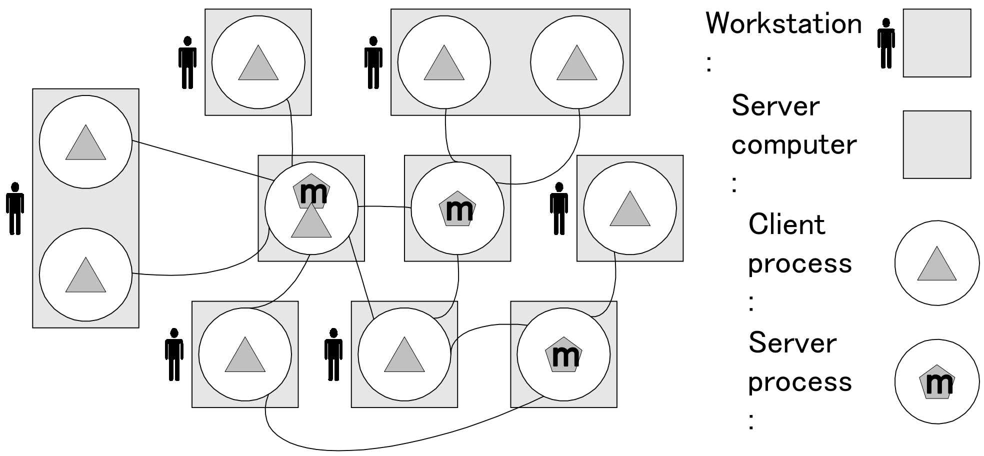


Fig.2.4. Opportunities for concurrency in a distributed system

1. Many users simultaneously invoke commands or interact with application programs.

2. Many server processes run concurrently, each responding to different requests from client processes.

Case 1 arises from one or more application processes running on behalf of each active user. In most distributed system architectures, application processes run in a user's computer and they do not conflict for processing resources with the application processes of other users. If the computer has only a single processor and there is more than one application process running on the computer, they are executed in an interleaved manner. Computers with multiple processors enable users to perform several computations in parallel or to execute application programs that exploit more than one processor.

Case 2 arises from the existence of one or more server processes for each type of resource. These normally run in additional computers, enabling each server process to proceed in parallel with other servers and with the processes running on computers. Multi-processor computers are particularly effective for the implementation of heavily-used servers, since an N processor server has the potential to handle up to N requests from clients without delay. Requests for access to resources in a given server are queued at the server and processed in sequence or several may be processed concurrently by multiple instances of the resource manager process. When several resource-using programs access the same resource concurrently, the server process must synchronize their actions to ensure that they do not conflict. The synchronization must be carefully planned to ensure that the benefits of concurrency are not lost.

Scalability

Distributed systems operate effectively and efficiently at many different scales. The smallest practicable distributed system probably consists of two computers and a file server, whereas a distributed system constructed around a single local-area networks may contain several hundred computers and many file servers, print servers and other special-purpose servers. Several local-area networks are often interconnected to form **internetworks**, and these may contain many thousands of computers that form a single distributed system, enabling resources to be shared between all of them.

The system and application software should not need to change when the scale of the system increases. This characteristics is achieved to a significant extent in most current distributed systems and components, but it is an area in which further research is required and in progress to accommodate the very large-scale systems and applications that are likely to emerge as internetworking increases and high-performance networks appear.

An example of lack of scalability was recently experienced by users of the London telephone system. Dialled locally, London numbers have seven digits, and these used to be prefixed by the code 01- when dialled from outside London. However, even these ten million numbers were not enough and the telephone company decided to split London into two regions, with prefixes 071- and 081-, with telephones retaining their seven-digit phone numbers inside the regions. This of course double the capacity. If eight-digit phone numbers had been assigned in the first place, then no disruption need have occurred and the numbering system would have been more coherent. However, to be fair to he telephone company, there is no correct solution to this problem. It is difficult to predict the demand that will be placed upon a system years ahead (note that an accurate projection would have had to predict the advent and large-scale usage of facsimile machines). Moreover, overcompensating for future growth may be worse than adapting to a change when we are forced to – after all, we could have been assigned twenty-digit numbers, just to be on the safe side.

The need for scalability is not just a problem of hardware or network performance. We shall see that the issue permeates almost every aspect of distributed system design. In centralized computer systems certain shared resources – memory, processors, input-output channels – are in limited supply and cannot be replicated indefinitely. In distributed systems, the limitation on the supply of some of those resources is automatically removed – we have already noted that there can be a potentially unlimited number of computers, each with memory, one or more central processors and input-output channels. But other limitations may remain if the design of the system does not explicitly recognize the need for scalability.

The demand for scalability in distributed systems has led to a design philosophy in which no single resource – hardware or software – is assumed to be in restricted supply. Rather, as the demand for a resource grows, it should be possible to extend the system to meet it. For example, the frequency with which files are accessed is likely to grow as the number of users and computers in a distributed system increases. It must be possible to add server computers to avoid the performance bottleneck that would arise if a single file server had to handle all file access requests. Some files may be accessed so frequently that the processing of requests for access to a single file or a small group of files becomes a performance bottleneck. In that case, those files must be replicated in several servers and the system must be designed so that when replicated files are updated the updates are applied to all of the replicas.

As the size and complexity of computer networks grows, it is a major challenge to design distributed system software that will remain effective for such network configurations. The Internet already embraces more than one million computers. The requirement is to design distributed services that continue to operate effectively with thousands or millions of clients. The work involved in processing any single request to access a shared resource should be nearly independent of the size of the network.

Fault Tolerance

Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or they may stop before they have completed the intended computation.

The design of fault-tolerant computer systems is based on two approaches, both of which must be deployed to handle each fault:

hardware redundancy: the use of redundant components;

software recovery: the design of programs to recover from faults.

To produce systems that are tolerant of hardware failures, two interconnected computers are often employed for a single application, one of them acting as a standby machine for the other. This is a costly solution – the cost of the hardware is doubled to produce two-fold redundancy in hardware provision.

In distributed systems, redundancy can be planned at a finer grain – individual servers that are essential to the continued operation of critical applications can be replicated.

The allocation of redundant hardware required for fault tolerance can be designed so that the hardware is exploited for non-critical activities when no faults are present. for example, a database may be replicated in several servers to ensure that the data remains accessible after the failure of any single server. The servers can be designed to detect faults in their peers; when a fault is detected in one server, clients are redirected to the remaining servers. By techniques such of this, tolerance to some types of hardware fault can be provided in distributed systems at a relatively low cost.

Software recovery involves the design of software so that the state of permanent data can be recovered or 'rolled back' when a fault is detected. In general, the computations performed by some programs will be incomplete when a fault occurs, and the permanent data that they update (files and other material stored in permanent storage) may not be in a consistent state. Mechanisms are developed that enable files and other persistent data to be restored to the state they were in before the failed program began its execution.

Distributed systems also provide a high degree of **availability** in the face of hardware faults. The availability of a system is a measure of the proportion of time that it is available for use. A single failure in a multi-user computer almost always results in the non-availability of the system to all of its users. When one of the components in a distributed system fails, only the work that was using the failed component is affected. A user may move to another workstation if the one that they were using fails; a server process can be restarted on another computer.

The networks on which distributed systems are based are not normally redundant. Failure of the network causes programs that use the network to hang until communication is restored, and the service to users will be interrupted. Overloading of the network degrades the performance and responsiveness of the system to users. Much effort has gone into the design of reliable and fault-tolerant networks.

Other software organizations are often needed to exploit concurrency fully and to obtain a higher degree of tolerance to faults than can be achieved with client-server systems. A useful paradigm for the construction of concurrent programs involving replication that has been developed as a result of recent research is the process group. This is a construct that enables a number of concurrent processes to cooperate by sharing a single communication mechanism – a sort of 'virtual network of computers' composed of a collection of processes that can be created, extended or diminished in size on demand.

Distribution Transparency

Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components. The implications of transparency are a major influence on the design of the system software.

The separation of components is an inherent property of distributed systems. Its consequences include the need for communication and for explicit system management and integration techniques. Separation allow the truly parallel execution of programs, the containment of component faults and recovery from faults without disruption of the whole system, the use of isolation and control of communication channels as a method for enforcing security and protection properties, and the incremental growth or contraction of the system through the addition or subtraction of components.

The ANSA Reference Manual and the International Standards Organization's Reference Model for Open Distributed Processing (RM-ODP) identify eight forms of transparency. These provide a useful summary of the motivation and goals for distributed systems. We have paraphrased the original ANSA definitions, employing the term 'information object' to denote the entities to which distribution transparency is applied:

Access transparency enables local and remote information objects to be accessed using identical operations.

Location transparency enables information objects to be accessed without knowledge of their location.

Concurrency transparency enables several processes to operate concurrently using shared information objects without interference between them.

Replication transparency enables multiple instances of information objects to be used to increase reliability and performance without knowledge of the replicas by users or application programs.

Failure transparency enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

Migration transparency allows the movement of information objects within a system without affecting the operation of users or application programs.

Performance transparency allows the system to be reconfigured to improve performance as loads vary.

Scaling transparency allows the system and applications to expand in scale without change to the system structure or the application algorithms.

The two most important transparencies are access and location transparency; their presence or absence most strongly affects the utilization of distributed resources.

They are sometimes referred to together as *network transparency*. Network transparency provides a similar degree of anonymity for resources to that found in centralized computer systems.

Transparency hides and renders anonymous the resources that are not of direct relevance to the task-in-hand from users and application programmers. This may not always be what is required. For example, it is generally desirable for similar hardware resources to be interchangeably allocated to perform a task – the identity of a processor used to execute a process is generally hidden from the user and remains anonymous. But when a document is printed, it is normal to arrange for it to be printed at a particular, named printer: usually one that is near to the user. Also, to a programmer developing parallel programs, not all processors are anonymous. He or she may be interested in the processors used to execute a parallel program, at least to the extent of their number and interconnection topology.

2.2.2 Structuring a Distributed Systems

In this section, we present several main paradigms are commonly used to structure a distributed system [4]:

- the master-slave model;
- the client/server model;
- the peer-to-peer model;
- the group model;
- the distributed object model;

Each of these models is examined in turn.

The Master-slave Model

A master-slave model may be an appropriate model for structuring a distributed system. In this model, a master process initiates and controls any dialogue with other (slave) processes. Slave processes exhibit very little intelligence, responding to commands from a single master process and exchange messages only when invited to by the master process. The master process defines the command set and appropriate responses associated with the dialogue. The slave process merely complies with the

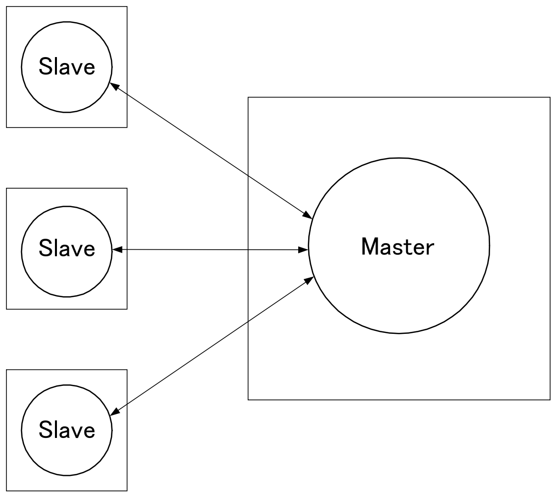


Fig.2.5. Master-slave model

dialogue rules. An example configuration is given in Fig. 2.5. This was the model on which online centralized computers running time-sharing information systems were based. The model has limited application in a distributed systems because it does not make best use of distributed resources and the master process represents a single point of failure.

The Client/server Model

The client/server model is the most widely used paradigm for structuring distributed systems. A client requests a particular service. One or more processes called *servers* are responsible for the provision of a service to clients. Services are accessed via a well-defined interface which is made known to the client community. On receipt of a valid request, a server executes the appropriate operation and sends a reply back to the client. The type of interaction is known as a **request/reply** or **interrogation**. Alternatively, an interaction could be initiated by a client resulting in the conveyance of information to the server requesting a function to be performed by the server process. This type of interaction is known as an **announcement** and is clearly one-way communication as no reply is sent back to the client. A client can potentially generate a request for service at any time. A server can have multiple interfaces

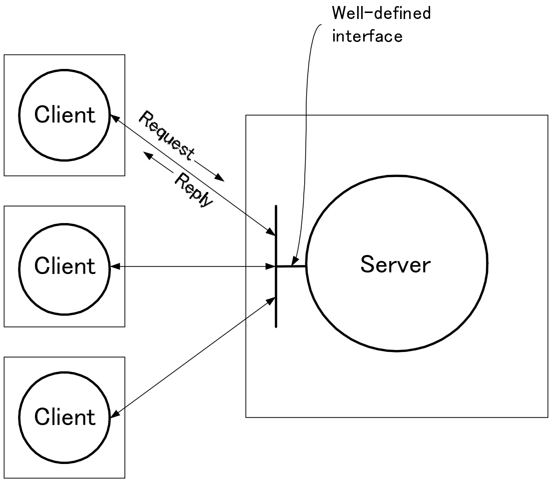


Fig.2.6. Client/server model

(e.g. a management interface and a separate interface through which client requests are handled).

Both clients and servers normally run as user processes. A single computer may run a single client or server process or may run multiple client or server processes (or both). A server process is normally persistent (non-terminating) and provides services to more than one client process. Clearly, client/server interaction is based on a request/reply protocol as illustrated in Fig. 2.6.

The main distinction between master-slave and client/server models lies in the fact that client and server processes are on an equal footing but with distinct roles. The use of a (small) manageable number of servers (i.e. increased centralization of resources) improves systems management compared with the case where potentially every computer can be configured as client and server. This model, known as a **peer-to-peer** model, is so-named because every process has the same functionality as a peer process as illustrated in Fig. 2.7

The Group Model

Client/server interaction involves two communicating parties: a client and a server. Another model which is appropriate to some types of distributed systems is the

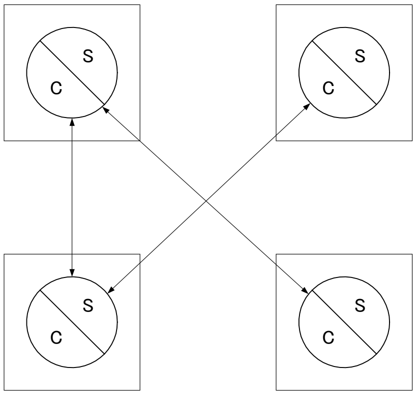


Fig.2.7. Peer to peer model: C, client function; S, server function

group model. In many circumstances, a set of processes need to co-operate in such a way that one process may need to send a message to all other processes in the group and receive responses from one or more members. For example, in a video conference involving multiple participants and a whiteboard facility, when someone writes to the whiteboard, every other participant must receive the new image. A second example is a computer conferencing system (e.g. the Internet USENET system). When a subscriber sends a message to a new item, all other subscribers receive it. This is modelled conveniently as set of group members which behaves as a single unit called a group. The group model is illustrated in Fig. 2.8. When a message is sent to the *group interface*, all members of the group receive it. How does a 'group' message get routed to every member? There are three main approaches, as shown in Fig. 2.9:

1. Send a separate copy of the message to be individually routed to each member. This is known as *unicasting*. An implicit assumption is that the sender knows the address of every member in the group. This may not be possible in some systems. In the absence of more sophisticated mechanisms, a system can resort to unicasting if member addresses are known. The number of network transmissions is proportional to the number of members in the group.

2. Send a single message with a group address which can be used for routing purposes. This is known as *multicasting* and relies on an underlying multicasting

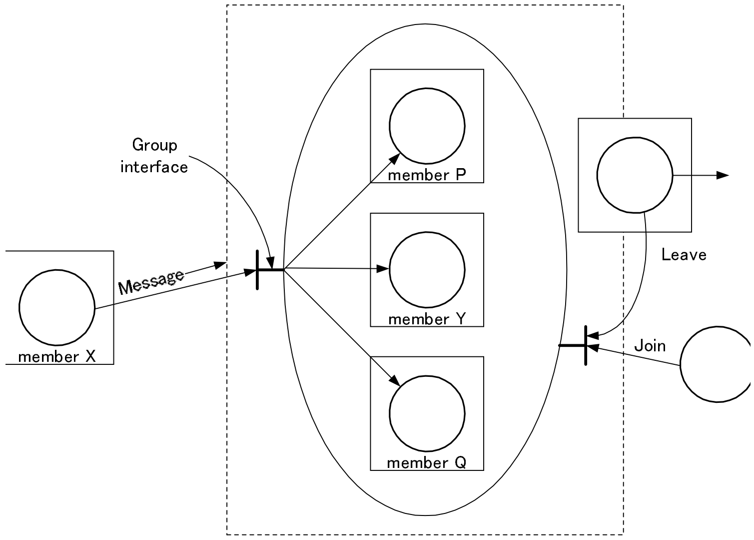


Fig.2.8. Group model

network facility. For example, multicasting is supported in most LAN protocols and the TCP/IP protocol. When a group is first created, it is assigned a unique group address. When a member is added to a group, it is instructed to listen for messages stamped with the group address as well as for its own unique address. This is an efficient mechanism since the number of network transmissions is significantly less than unicasting.

3. *Broadcast* the message by sending a single message with a broadcast address. The message is sent to every possible entity on the network. Every entity must read the message and determine whether they should take action or discard it. This may be an appropriate mechanism in the case when the address of members is not known since most network protocols implement a broadcast facility. If message are broadcast frequently, however, and there is no efficient network broadcast mechanism, the network soon becomes saturated as broadcasts are propagated over an internet (known as a broadcast storm). Therefore broadcast-based group communications do not scale well unless an efficient low-level broadcast facility is available.

In some cases, a group message (e.g. an update request to a group of replica servers) must be received by all group members or none at all. Group communication, in this case, is said to be **atomic** or **all-or-nothing**. Achieving atomicity in the presence of failures is difficult, resulting in many more messages being exchanged. Another aspect of group communications is the ordering of group messages. For example, in a computer conferencing system a user would expect to receive the original news item before any response to that item is received. This is known as **ordered multicast** and the requirement to ensure that all multicasts are received in the same order for all group members is common in distributed systems. Atomic multicasting does not guarantee that all messages will be received by group members in the order in which they were sent. Clearly, implementing group communications protocols is non-trivial. An example of a group communication system which address ordered multicasts and, to some extent, atomic multicast, is the ISIS system developed at Cornell University. In ISIS, broadcast primitives are defined to ensure that all messages arrive in the same order to all parties (e.g. group members) with fault tolerant features.

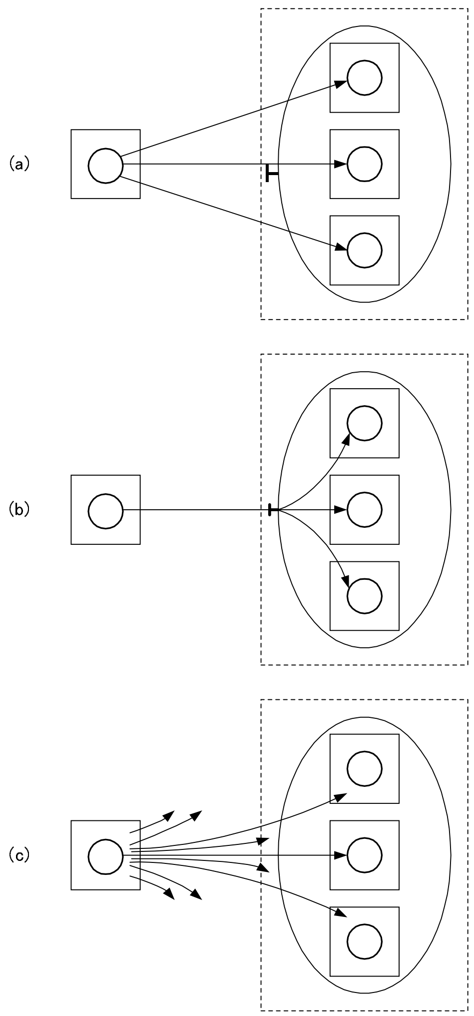


Fig.2.9. Routing group messages: (a)unicast; (b)multicast; (c)broadcast

The Distributed Object Model

The terms *object* and *object-oriented programming* are used ambiguously in the field of computer science which causes much confusion if not appropriately defined. Object technology offers a different programming model from traditional structured programming which is based on the separation of data and the processing of data through functions and procedures. An object is a computational entity which encapsulates both private data describing its state and a set of associated operations (which defines the object's interface to the outside world) as a representation of a real-world object. An object is described in terms of its *attributes* (internal state data) and methods (procedures which operate on the state data).

An object's state is visible only within the object and is completely protected and hidden from other objects as a way of hiding complexity and as a protection from misuse. The only way to examine or modify an object's state data is by sending a *message* to the object which has the effect of invoking one of a well-defined set of methods that define the object's interface to the outside world. Thus without knowing the detail of how an object is implemented, it can be used by simply knowing the methods that define the object's interface. An object may invoke other objects, allowing the creation of a potentially complex web of object invocations, as illustrated in Fig. 2.10.

An object-based or object-oriented model is recognized as offering important advantages in the development and maintenance of distributed information systems for the following reasons.

- Implementation detail is hidden and the use (and re-use) of objects emphasized through the definition of well-defined object interfaces. This can lead to greater development productivity, and reduced software maintenance costs.
- Objects can be written to handle a wide range of media types (including text, voice, animation and video) using essentially the same object interface which provides a consistent approach to handling multimedia objects.
- It provides the foundation for distributed information systems integration into specific line-of-business solutions by re-packaging existing functionality.
- An object is a natural unit of distribution. The message-passing nature of inter-object communications maps easily to a distributed systems environment.

A *class* is a template from which objects may be created (every object is therefore an instance of some class in as much as a class exists at program compile time, but

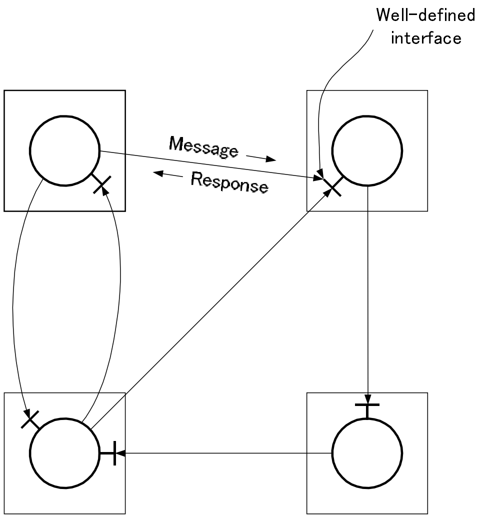


Fig.2.10. Distributed object model

objects exist at execution time with state representations and associated operations as defined by the class). *Inheritance* is a mechanism that permits new classes to be defined as an extension of another (previously defined) class which provides support for creating new objects by extending existing templates. *Polymorphism* (which means 'many forms') provides the ability for different (but related) objects to share the same method names, thus allowing different objects to be accessed through the same interface. A technique called *overloading* implements polymorphism by ensuring that, even though there may be multiple methods with the same name but different method definitions, the method definition used is one which is appropriate to the type of object involved. The overall effect is, for example, of multiple object types each representing a different media type which can be hidden behind a common object interface (e.g. all objects define open, create, display print and delete methods but with different method definitions).

Programming language that support objects as a language feature are known as *object based* and *object oriented* if they also support inheritance. The *sequential* object model is based on the notion of active and passive objects. An object is activated when it receives a message from another object. The object that sent the message becomes passive as control is transferred, and waits for a result. After servicing the message and sending the result, control is transferred to the waiting object which now becomes active and the sender passive. At any instant, only one object in the system is active. This is similar to process activation and passivation. Current widely-used object-oriented systems usually assume the sequential object model with all objects running in a single name space on a single computer. The distributed object model allows objects to persist in an active state irrespective of whether they have sent a message or received a result. In fact the model allows an object to send messages to multiple objects concurrently and receive results asynchronously.

Programming languages that support distributed objects define an object as the unit of distribution. The distributed object interaction model differs from the client/server model in that an object can both request and provide services (i.e. it can act both as a client and server). In this sense, objects interactions resemble peer-to-peer interactions except that objects can exhibit different functionality. Since each object-to-object interaction is request/reply dialogue, for the purpose of object modelling it can be useful to view an object as a composite client and server assemblage. A popular standard for implementing distributed objects is the Object Management Group's common object request broker architecture (CORBA).

2.2.3 Centralized versus Distributed Systems

A historical perspective on the evolution of distributed systems reveals a number of advantages and disadvantages [4].

Advantages of Distributed Systems

The main advantages of distributed systems is their ability to allow the sharing of information and resources over a wide geographic area, giving a systems designer freedom to optimize the placement of distributed system components such as data and processing. This then supports:

- **Improved flexibility**. Computers and other system components can be flexibly located at points within the organization where they can be utilized most effectively. Components can be added, upgraded, moved and removed (usually in small increments over a wide range of capacities), without impacting upon other components, to meet present and future needs.

- **Local autonomy**, by allowing domains of control to be defined where decisions are made relating to purchasing, ownership, IT budgets, operating priorities, information systems development and management, etc. Each domain decides where resources (including manpower and system components) under its control are located. This autonomy is recognition of the distributed nature of many organizational activities. Domains can be federated when necessary for mutual benefit.

- **Increased reliability and availability**. In a centralized system, a component (hardware or software) failure can mean that the whole system is down, stopping all users from getting work done. In a distributed system, multiple components of the same type can be configured to fail independently and (e.g. through replication) provide a level of fault tolerance. Thus, failure of one component may isolate a group of users, but does not necessarily prevent others from operating.

- **Improved performance**. Large centralized systems can be slow performers due mainly to the sheer volume of data and transactions being handled. A service which is partitioned over many server computers, each supporting a smaller set of applications and users with access to local data and resources, results in faster access (response times). Another performance advantage is the support for parallel access (updates and retrievals) to distributed data across the organization.

- **Security breaches are localized**. In distributed system with multiple security control domains, a security breach in one domain does not compromise the whole system. Each security domain has varying degrees of security authentication, access control and auditing.

Disadvantages of Distributed Systems

Against the above advantages are a number of disadvantages, most of which correspond to the advantages of centralized systems:

- **It is more difficult to manage and secure**. Centralized systems are inherently easier to secure (offering a single security domain, controlled by a single authority) and easier to manage. Distributed systems require more complex procedures for security, administration, maintenance and user support due to greater levels of co-ordination and control required. This usually results in higher costs associated with managing and securing a distributed systems environment.

- **Reduced reliability and availability**. In contrast to the potential improvements in reliability and availability discussed above, a centralized system can often offer more controlled physical, operational and environmental conditions borne out of years of development and improvement. Moreover, a distributed system consists of many more components which can potentially fail, causing loss of availability.

- **A shortage of skilled support and development staff**. In a decentralized operation, scarce support and development can be dispersed, resulting in a loss of economies of scale which, in turn, leads to higher costs. Another issue is the level of support offered by vendors. The commitment and level of support offered by vendors of distributed software and hardware to 'corporate' organizations grappling with the issues of building large-scale distributed systems is not yet comparable to the traditional large mainframe-based vendors. This is due in part to the fact that many vendors are supplying only a small part of the overall system. In practice, whereas in a centralized system the mainframe supplier provided the equipment and integration support as a single package, distributed system implementation commonly involves multiple vendors and therefore requires the skills of a systems integrator. Problem isolation and resolution can be much more difficult and costly when multiple vendors are involved. This is exacerbated by the need for staff to develop a wide range of knowledge and skills.

Thus, distributed systems are not a panacea for all the world's computing problems. Like many innovations, they provide a solution to many problems, but also introduce new ones. With the potentially significant disadvantages highlighted above, why should any organization take the risk of implementing a distributed system? The answer is that, although overcoming the disadvantages represents a significant challenge to any organization, it is often perceived that the advantages (which are generally user-centred) significantly outweigh the disadvantages (which are generally centred around staffing, management and administration). Implementing a distributed system gives an organization the flexibility to choose an appropriate balance between the high levels of discipline and co-ordination associated with centralized control, and the flexibility and local autonomy offered by decentralization.

A distributed system supports tight management of corporate data by locating it on a limited number of tightly managed server computers, easily accessible to users via a desktop computer. Data can be duplicated in a controlled manner to improve performance and availability. Thus an appropriate balance between control, efficiency, flexibility and local autonomy can be realized.

2.3 Distributed Databases

2.3.1 Distributed Databases: An Overview

It is well known that during the 1970s computers were extensively used for building powerful, integrated database systems. The technology of database systems has built its theoretical foundations and has been experienced in a large number of applications. At the same time, computer networks have been extensively developed, allowing the connection of different computers and the exchange of data and other resources between them.

The availability of databases and of computer networks has given rise to a new field: distributed databases. A distributed database is, in brief, an integrated database which is built on top of a computer network rather than on a single computer. The data which constitute the database are stored at the different sites of the computer networks, and the application programs which are run by the computers access data at different sites.

Completely new problems are faced for building and implementing a distributed database, and a great amount of research work has been done in order to solve them. This research work constitutes a new discipline having its own principles. In order to understand distributed databases, we must know more than the principles of traditional databases and computer networks: we must also integrate this knowledge with the study of the peculiar aspects of the new technology.

A typical, but rather vague, definition of a distributed database is the following: A distributed database is a collection of data which belong logically to the same system but are spread over the sites of a computer network [1]. This definition emphasizes two equally important aspects of a distributed database:

1. **Distribution**, i.e., the fact that the data are not resident at the same site (processor), so that we can distinguish a distributed database from a single, centralized database

2. **Logical correlation**, i.e., the fact that the data have some properties which tie them together, so that we can distinguish a distributed database from a set of local databases or files which are resident at different sites of a computer network

The problem with the above definition is that both properties, distribution and logical correlation, are too vaguely defined to always discriminate between those cases which really belong to distributed databases and those which do not. To elucidate the difference between them, we can add the following descriptions to the above definition of a distributed database. In a distributed database, each site of the network has autonomous processing capability and can perform local applications. Each site also participates in the execution of at least one global application, which requires accessing data at several sites using a communication subsystem.

In the rest of this section, we briefly present the relational model and algebra which are used in the following sections. The relational model is one of the most suitable data models for the description of distributed database concepts.

The development of a distributed database system requires effective solutions to many complex and interrelated design issues, including network topology, hardware allocation, data partitioning, data allocation, query optimization, data replication, concurrency control, reliability, and recovery. In the following sections, we focus on data allocation and query optimization among those issues. Data allocation and query optimization are especially important distributed systems design problems that are more closely interrelated. There are a few works that considered the itegration of query processing and data allocation [23][24]. However, researchers studying data allocation typically assume a fixed query optimization method to generate processing schedules, while researchers on query optimization assume a fixed data allocation. By assuming a solution to one problem and solving the other, researchers control the complexity of these two problems.

2.3.2 Relational Model

The relational model is more suitable data model than other models for the description of distributed database concepts. the relational model allows the use of powerful, set-oriented, associative expressions instead of the one-record-at-a-time primitives of more procedural models, like the Codasyl model [1].

In relational databases, data are stored in tables, called **relations**. Each relation has a (fixed) number of columns, called **attributes**, and a (dynamic, time-varying)

(a) Relation EMP

EMPNUM	NAME	AGE	DEPTNUM
3	David	31	1
7	Gordon	22	2
11	Janice	21	1
15	Mario	34	3
18	Sophia	28	1

(b) Relation EMP'

EMPNUM	AGE	DEPTNUM	NAME
3	31	1	David
7	22	2	Gordon
11	21	1	Janice
15	34	3	Mario
18	28	1	Sophia

Fig.2.11. Example of relation

number of rows, called **tuples**. There is an obvious correspondence between fields of traditional file organizations and attributes of the relational model, and between records of files and tuples of relations. The number of attributes of a relation is called its **grade**; the number of tuples is called its **cardinality**.

Fig. 2.11(a) shows a relation EMP (employee) consisting of four attributes: $EMPNUM$, $NAME$, AGE, and $DEPTNUM$. The relation has five tuples; for example, $\langle 3, Jones, 27, 1\rangle$ is a tuple of the EMP relation. The grade of relation EMP is 4 and the cardinality is 5.

The relation name and the names of attributes appearing in it are called the **relation schema** of the relation; for instance,

$$EMP(EMPNUM, NAME, AGE, DEPTNUM)$$

is the relation schema of relation EMP in the above example.

The set of possible values for a given attribute is called its **domain**. Thus, for instance, $EMPNUM$ takes its values from the domain of employee numbers, constituted by the positive integers, and AGE takes its values from the domain of ages, constituted by integers between 0 and 100. Though these two domains have some values in common, they are regarded as different domains, since the meaning given to common values is different in the two contexts; typically, it makes no sense to compare values from different domains, such as ages and employee numbers.

An important aspect of the definition of the relational model is to regard relations as **sets** of tuples; as a consequence:

1. There cannot be two identical tuples in the same relation.

2. There is no defined order of the tuples of a relation.

While considering relations as sets is important for deriving elegant and useful mathematical properties of relations, most relational database systems do not strictly implement relations as pure sets. Thus, in some systems replicated tuples are allowed (relations are regarded as multisets), and tuples are stored according to some internal ordering which is useful for processing.

A more formal definition of relations is based on the notion of domains; in this case, a collection of domains $D_1, D_2, \ldots, D_n$ is given a priori, and a relation of grade n over these domains is seen as any subset of the cartesian product of them. More precisely, a relation R is a set of ordered tuples $\langle d_1, d_2, \ldots, d_n \rangle$ such that d_1 belongs to D_1, d_2 belongs to $D_2, \ldots$, and d_n belongs to D_n. This definition is useful for analyzing the properties of the relational model and algebra, and this is the current definition in the research work.

Here, we consider the order of columns of a relation as immaterial. This corresponds to considering a relation as a set of mappings from a set of attribute names to a set of values; more simply, this corresponds to disregarding the "position" of each column in a relation, as long as we can recognize columns by means of attribute names. Thus, the relation EMP' in Fig. 2.11(b), in which the same attributes of EMP appear in the order: $EMPNUM, AGE, DEPTNUM, NAME$, is exactly the same relation as that of Fig. 2.11(a).

An important property of some attributes of relations is that they are a **key** of the relation; keys are subsets of the attributes of a relation schema whose values are unique within the relation, and thus can be used to uniquely identify the tuple of the relation. Given that relations are sets of tuples, a key must exist; at least, the set of all attributes of the relation constitutes the key. The property of being a key is a semantic one; we do not consider an attribute as a key only because it occasionally identifies all the tuples. Thus, in the examples of Fig. 2.11, $EMPNUM$ is an appropriate key of EMP (because identification numbers of employees are unique), while AGE is not a key (because by simply adding more tuples, we might find two of them with the same age value). We might assume instead that identification numbers are given to employees within departments; i.e., each department has its own numeration. In this case, the appropriate key of this relation would be the pair of attributes: $EMPNUM, DEPTNUM$. Notice that a relation can have several keys; typically, one of them is selected as the **primary key**.

The **relational algebra** is a collection of operations on relations, each of which takes one or two relations as operations and produces one relation as result. Operations of relational algebra can be composed into arbitrarily complex **expressions**; expressions allow specifying the manipulations which are required on relations in order to retrieve information from them.

Several versions of the relational algebra have appeared in the literature, differ in the choice of operators and in the formalism used to denote them. Here, we follow the same approach as reference, in which five basic operations are defined: selection, projection, cartesian product, union, and difference. These operations are complete, in the sense that they allow the same expressiveness as first-order calculus. From these operations, some other operations are derived, such as intersection, division, join, and semi-join. We particularly highlight those operations which have an important application in distributed databases, such as joins and semi-joins, while we do not consider other operations, like intersection and division. The meaning of each operation is described referring to the examples of Fig. 2.12.

Unary operations take only one relation as operand; they include selection and projection.

The **selectionSL**$_F R$ (Fig. 2.12(b)), where R is the operand to which the selection is applied and F is a formula which expresses a selection predicate, produces a result relation with the same relation schema as the operand which satisfy the predicate. the formula involves attribute names or constants as operands, arithmetic comparison operators, and logical operators. For example, given a relation schema $R(A, B, C)$, the following is a valid formula: $(A = B\ OR\ A < C) AND\ NOT\ A > 7$.

The **projectionPJ**$_{Attr} R$ (Fig. 2.12(c)), where $Attr$ denotes a subset of the relations schema. The tuples of the result are derived from the tuples of the operand relation by suppressing the values of attributes which do not appear in $Attr$. Moreover, replicated tuples which might result from this operation are eliminated; thus, the cardinality of the result might be less than the cardinality of the operand.

Binary operations take two relations as operands; we review union, difference, cartesian product, join, and semi-join.

The **union** R **UN** S (Fig. 2.12(d)) is meaningful only between two relation R and S with the same relation schema; it produces a relation with the same relation schema as its operands and the union of the tuples of R and S (i.e., all tuples appearing either in R or in S or in both).

The **difference** R **DF** S (Fig. 2.12(e)) is meaningful only between two relation R and S with the same relation schema; it produces a relation with the same relation schema as its operands and the difference between the tuples of R and S (i.e., all tuples appearing in R but not in S).

The **cartesian product** R **CP** S (Fig. 2.12(f)) produces a relations whose relation schema includes all the attributes of R and S. If two attributes with the same name appear in R and S, they are nevertheless considered different attributes; in order to avoid ambiguity, the name of each attribute is prefixed with the name of its "original" relation. every tuple of R is combined with every tuple of S to form one tuple of the result.

The **join** (Fig. 2.12(g)) of two relations R and S is denoted as $R\mathbf{JN}_F S$, where F is a formula which specifies the join predicate. The formula of a join specification is given by conjunctions of comparisons between attributes ordinately taken from the two operands; $A = C\ AND\ B > D$. If only equality appears in the formula, we denote the operation as an equi-join. A join is derived from selection and cartesian product as follows:

$$R\mathbf{JN}_F S = \mathbf{SL}_F(R\mathbf{CP}S)$$

Thus, the relation schema of the result of the join includes all the attributes of R and S, and all the tuples from R and S which satisfy the join predicate are included in the result.

The **natural join** R **NJN** S (Fig. 2.12(h)) of two relations R and S is an equi-join in which all attributes have both the same name and the same values in all the tuples of the result, one of the two attributes is omitted from the result.

The **semi-join** (Fig. 2.12(i)) of two relations R and S is denoted as $R\mathbf{SJ}_F S$, where F is a formula which specifies a join predicate. A semi-join is derived from projection and join as follows:

$$R\mathbf{SJ}_F S = \mathbf{PJ}_{Attr(R)}(R\mathbf{JN}_F S)$$

where $Attr(R)$ denotes the set of all attributes of R. Thus, the result of the semi-join is a subset of the tuples of R, constituted by those tuples which give a contribution to the join of R with S.

The **natural semi-join** (Fig. 2.12(j)) of two relations R and S, denoted R **NSJ** S, is obtained by considering a semi-join with the same join predicate as in the natural join.

Finally, we introduce a synthetic notation for the n-ary union, which we will find very useful. We denote as $\mathbf{UN}(R_1, R_2, \ldots, R_n)$ the union of n relations; this is perfectly equivalent to the expression: $R_1 \mathbf{UN} R_2 \mathbf{UN} \ldots \mathbf{UN} R_n$, which returns the n-ary union as a result of several binary unions (since union is commutative and associative).

Relational algebra is not very "user friendly," and understanding the meaning of an algebric expression can be difficult. Therefore, when we describe how users should program a distributed database, we can use a more user-oriented language, which can be understood more easily. The SQL language is widely used for this purpose.

A simple statement in SQL has the following structure:

Select $\langle$*attribute list*$\rangle$

from $\langle$*relation name*$\rangle$

where $\langle$*predicates*$\rangle$

The interpretation of this statement is equivalent to performing a selection operation using the predicates of the "where clause" on the relation specified in the "from clause" and then projecting the result on the attributes of the "select clause." For example, if we consider the relation of Fig. 2.11(a), the statement;

Select $NAME, AGE$

from EMP

where $AGE > 25$ *and* $DEPTNUM = 1$

returns the following relation:

NAME	AGE
David	31
Sophia	28

R

A	*B*	*C*
a	1	a
b	1	b
a	1	d
b	2	f

S

A	*B*	*C*
a	1	a
a	3	f

T

B	*C*	*D*
1	a	1
3	b	1
3	c	2
1	d	4
2	a	3

(a) Operand relations.

A	*B*	*C*
a	1	a
a	1	d

(b) Selection $\mathbf{SL}_{A=a}$ *R*

A	*B*
a	1
b	1
b	2

(c) Projection $\mathbf{PJ}_{A,\,B}$ *R*

A	*B*	*C*
a	1	a
b	1	b
a	1	d
b	2	f
a	3	f

(d) Union *R* **UN** *S*

R.A	*R.B*	*R.C*	*S.A*	*S.B*	*S.C*
a	1	a	a	1	a
b	1	b	a	1	a
a	1	d	a	1	a
b	2	f	a	1	a
a	1	a	a	3	f
b	1	b	a	3	f
a	1	d	a	3	f
b	2	f	a	3	f

(f) Cartesian Product *R* **CP** *S*

A	*B*	*C*
b	1	b
a	1	d
b	2	f

(e) Difference *R* **DF** *S*

A	*R.B*	*R.C*	*T.B*	*T.C*	*D*
a	1	a	1	a	1
a	1	a	2	a	3
b	1	b	3	b	1
a	1	d	1	d	4

(g) Join *R* $\mathbf{JN}_{R.C=T.C}$ *T*

A	*B*	*C*	*D*
a	1	a	1
a	1	d	4

(h) Natural join *R* **NJN** *T*

A	*B*	*C*
a	1	a
b	1	b
a	1	d

(i) Semi-join *R* $\mathbf{SJ}_{R.C=T.C}$ *T*

A	*B*	*C*
a	1	a
a	1	d

(j) Natural join *R* **NSJ** *T*

Fig.2.12. Operations of relational algebra

2.3.3 Reference Architecture for Distributed Databases

Fig. 2.13 shows a reference architecture for a distributed database. This reference architecture is not explicitly implemented in all distributed databases; however, its levels are conceptually relevant in order to understand the organization of any distributed database. We will therefore analyze and understand all the components of this reference architecture.

At the top level of Fig. 2.13 is the **global schema**. The global schema defines all the data which are contained in the distributed database as if the database were not distributed at all. For this reason, the global schema can be defined exactly in the same way as in a nondistributed database. However, the data model which is used for the definition of a global schema should be convenient for the definition of the mapping to the other levels of the distributed database. We will use the relational model for this purpose. Using this model, the global schema consists of the definition of a set of **global relations**.

Each global relation can be split into several nonoverlapping portions which are called **fragments**. There are several different ways in which to perform the splitting operation; they are described in the next section. The mapping between global relations and fragments is defined in the **fragmentation schema**. This mapping is one to many; i.e., several fragments correspond to one global relation, but only one global relation corresponds to one fragment. Fragments are indicated by a global relation name with an index (fragment index); for example, R_i indicates the ith fragment of global relation R.

Fragments are logical portions of global relations which are physically located at one or several sites of the network. The **allocation schema** defines at which site(s) a fragment is located. Note that the type of mapping defined in the allocation schema determines whether the distributed database is **redundant** or **nonredundant**; in the former case the mapping is one to many, while in the latter case the mapping is one to one. All the fragments which correspond to the same global relation R and are located at the same site j constitute the **physical image** of global relation R at the site j. There is therefore a one to one mapping between a physical image and a pair $\langle$ global relation, site $\rangle$; physical images can be indicated by a global relation name and a site index. To distinguish them from fragments, we will use a superscript; for example, R^j indicates the physical image of the global relation R at site j.

An example of the relationship between the object types defined above is shown in Fig. 2.14. A global relation R is split into four fragments R_1, R_2, R_3, and R_4.

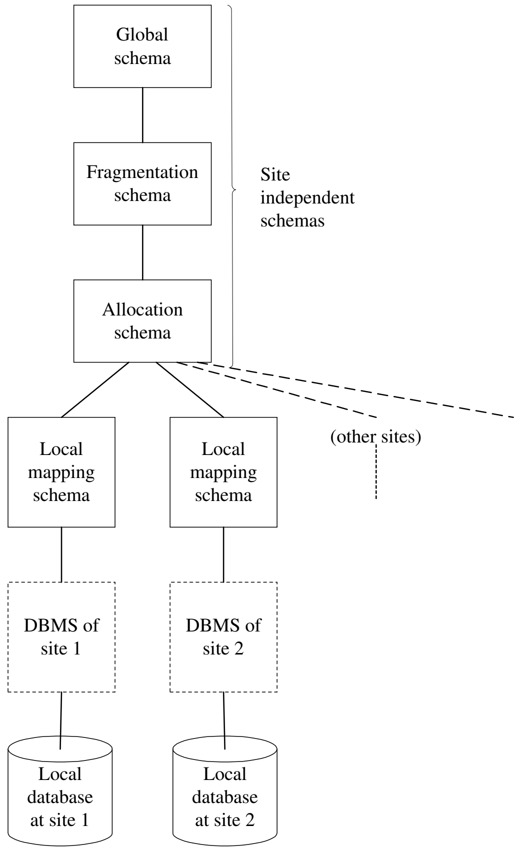


Fig.2.13. A reference architecture for distributed databases

These four fragments are allocated redundantly at the three sites of a computer network, thus building three physical images R^1, R^2, and R^3.

To complete the terminology, we will refer to a **copy of a fragment** at a given site, and denote it using the global relation name and two indexes (a fragment index and a site index). For example, in Fig. 2.14, the notation R_2^3 indicates the copy of fragment R_2 which is located at site 3.

Finally, note that two physical images can be identical. In this case we will say that a physical image is a copy of another physical image. For example, in Fig. 2.14, R^1 is a copy of R^2.

Let us now go back to the reference architecture of Fig. 2.13. We have already described the relationships between the objects at the three top levels of this architecture. These three levels are site independent; therefore, they do not depend on the data model of the local database management systems (DBMSs). At a lower level, it is necessary to map the physical images to the objects which are manipulated by the local DBMSs. this mapping is called a **local mapping schema** and depends on the type of local DBMS; therefore in a heterogeneous system we have different types of local mappings at different sites.

This architecture provides a very general conceptual framework for understanding distributed databases. The three most important objectives which motivate the features of this architecture are the separation of data fragmentation and allocation, the control of redundancy, and the independence from local DBMSs.

1. *Separating the concept of data fragmentation from the concept of data allocation.* This separation is very convenient in distributed database design, because the determination of relevant portions of the data is thus distinguished from the problem of optimal allocation, as will be shown in the next section.

2. *Explicit control of redundancy.* The reference architecture provides explicit control of redundancy at the fragment level. For example, in Fig. 2.14 the two physical images R^2 and R^3 are overlapping; i.e., they contain common data. The definition of disjoint fragments as building blocks of physical images allows us to refer explicitly to this overlapping part: the replicated fragment R^2. The explicit control over redundancy is useful in several aspects of distributed database management.

3. *Independence from local DBMSs.* This feature allows us to study several problems of distributed database management without having to take into account the specific data models of local DBMSs. Clearly, in a homogeneous system it is possible that the site-independent schemata are defined using the same data model as the local DBMSs, thus reducing the complexity of this mapping.

As a final remark on the reference architecture, we can say that it performs the same function as the ANSI/SPARK architecture in traditional DBMSs: it is not necessarily implemented by the existing systems, but it shows which levels and schemata are conceptually relevant.

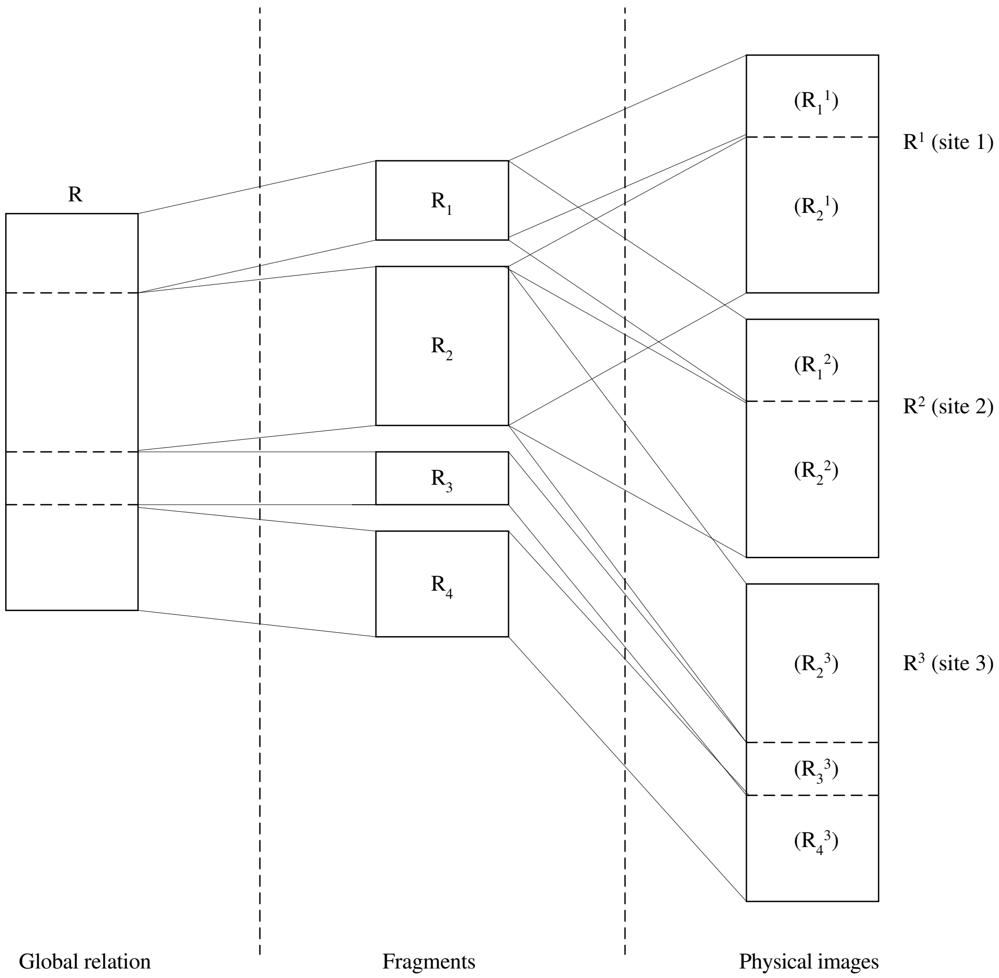


Fig.2.14. Fragments and physical images for a global relation

2.4 The Design of Data Distribution

The term of "distributed database design" has a very broad and unprecise meaning. In this section, we concentrate on those aspects with are peculiar to distributed databases, while we do not present design problems which are faced also in the design of centralized databases. The design of a centralized database amounts to:

- Designing the "conceptual schema" which describes the integrated database (i.e., all the data which are used by the database applications).

- Designing the "physical database," i.e., mapping the conceptual schema to storage areas and determining appropriate access methods.

In a distributed database these two problems become the design of the global schema and the design of the local physical databases at each site; the techniques which can be applied to these problems are the same as in centralized databases. The distribution of the database adds to the above problems two new ones:

- Designing the fragmentation, i.e., determining how global relations are subdivided into horizontal, vertical, or mixed fragments.

- Designing the allocation of fragments, i.e., determining how fragments are mapped to physical images; in this way, also the replication of fragments is determined.

These two problems fully characterize the design of data distribution. Fragmentation design has been studied only in recent times, since fragmentation has been established as a distinguishing feature of distributed databases; it had, however, been partially analyzed in the context of centralized systems with multiple storage devices. The allocation problem has instead been studied extensively since the development of distributed file systems, and is typically regarded in the literature as the "file allocation problem".

The distinction between these two problems is conceptually relevant, since the first one deals with the "local criteria" which motivate the fragmentation of a global relation, while the second one deals with the "physical" placement of data at the various sites. However, this distinction must be introduced with extreme care; in general, it is not possible to determine the optimal fragmentation and allocation by solving the two problems independently, since they are interrelated.

2.4.1 Objectives of the Design of Data Distribution

In the design of data distribution, the following objectives should be taken into account:

Processing locality Distributing data to maximize processing locality corresponds to the simple principle of placing data as close as possible to the applications which use them. The simplest way of characterizing processing locality is to consider two types of references to data: "local"references and "remote" references. Clearly, once the sites of origin of applications are known, locality and remoteness of references depend only on data distribution.

Designing data distribution for maximizing processing locality (or, conversely, for minimizing remote references) can be done by adding the number of local and remote references corresponding to each candidate fragmentation and fragment allocation, and selecting the best solution among them.

An extension to this simple optimization criterion is to take into account when an application has **complete locality**. We use this term to designate those applications which can be completely executed at their sites of origin. The advantage of complete locality is not only the reduction of remote accesses, but also the increased simplicity in controlling the execution of the application.

Availability and reliability of distributed data In section 2.2.3, we have indicated availability and reliability as advantages of distributed systems versus nondistributed ones. A high degree of availability for read-only applications is achieved by storing multiple copies of the same information; the system must be able to switch to an alternative copy when the one that should be accessed under normal conditions is not available.

Reliability is also achieved by storing multiple copies of the same information, since it is possible to recover from crashes or from the physical destruction of one of the copies by using the other, still available copies. Since physical destruction can be caused by events which have nothing to do with computer crashes (such as fire, earthquake, or sabotage), it is relevant to store replicated copies in geographically dispersed locations.

Workload distribution Distributing the workload over the sites the sites is an important feature of distributed computer systems. Workload distribution is done in order to take advantage of the different powers or utilizations of computers at each site, and to maximize the degree of parallelism of execution of applications. Since workload distribution might negatively affect processing locality, it is necessary to consider the trade-off between them in the design of data distribution.

Storage costs and availability Database distribution should reflect the cost and availability of storage at the different sites. It is possible to have specialized sites in the network for data storage, or conversely to have sites which do not support mass storage at all. Typically, the cost of data storage is not relevant if compared with CPU,I/O,and transmission costs of applications, but the limitation of available storage at each site must be considered.

Using all the above criteria at the same time is extremely difficult, since this leads to complex optimization models. It is possible to consider some of the above features as constraints, rather than objectives (for instance, it is possible to formulate constraints on the maximum amount of workload or the maximum available storage at each site). Alternatively, it is possible to consider the most important criterion in the initial design and to introduce other criteria in the postoptimization.

2.4.2 Types of Data Fragmentation

The decomposition of global relation into fragments can be performed by applying two different types of fragmentation: **horizontal fragmentation** and **vertical fragmentation**. We will first consider these two types of fragmentation separately and then consider the more complex fragmentation which can be obtained by applying a composition of both.

In all types of fragmentation, a fragment can be defined by an expression in a relational language (we will use relational algebra) which takes global relations as operands and produces the fragment as result. For example, if a global relation contains data about employees, a fragment which contains only data about employees who work at department D_1 can be obviously defined by a selection operation on the global relation.

There are, however, some rules which must be followed when defining fragments:

Completeness condition All the data of the global relation must be mapped into the fragments; i.e., it must not happen that a data item which belongs to a global relation does not belong to any fragment.

Reconstruction condition It must always be possible to reconstruct each global relation from its fragments. The necessity of this condition is obvious: in fact, only fragments are stored in the distributed database, and global relation have to be built through this reconstruction operation if necessary.

Disjointness condition It is convenient that fragments be disjoint, so that the replication of data can be controlled explicitly at the allocation level. However, this condition is useful mainly with horizontal fragmentation, while for vertical fragmentation we will sometimes allow this condition to be violated. The reason for this exception will be discussed when dealing with vertical fragmentation.

We can now consider the fragmentation rules.

Horizontal Fragmentation

Horizontal fragmentation consists of partitioning the tuples of a global relation into subsets; this is clearly useful in a distributed databases, where each subset can contain data which have common geographical properties. It can be defined by expressing each fragment as a selection operation on the global relation. For example, let a global relation be

$$SUPPLIER(SNUM, NAME, CITY)$$

Then the horizontal fragmentation can be defined in the following way:

$$SUPPLIER_1 = \mathbf{SL}_{CITY=\text{``}SF\text{''}} SUPPLIER$$

$$SUPPLIER_2 = \mathbf{SL}_{CITY=\text{``}LA\text{''}} SUPPLIER$$

The above fragmentation satisfies the completeness condition if "SF" and "LA" are the only possible values of the $CITY$ attribute; otherwise we would not know to which fragment the tuples with other $CITY$ values belong.

The reconstruction condition is easily verified, because it is always possible to reconstruct the $SUPPLIER$ global relation through the following operation:

$$SUPPLIER = SUPPLIER_1 \mathbf{UN} SUPPLIER_2$$

The disjointness condition is clearly verified.

We will call the predicate which is used in the selection operation which defines a fragment its **qualification**. For instance, in the above example the qualifications

$$q_1 : CITY = \text{``}\mathbf{SF}\text{''}$$

$$q_2 : CITY = \text{``}\mathbf{LA}\text{''}$$

We can generalize from the above example that in order to satisfy the completeness condition, the set of qualifications of all fragments must be complete, at least with respect to the set of allowed values. The reconstruction condition is always satisfied through the union operation, and the disjointness condition requires that qualifications be mutually exclusive.

Derived Horizontal Fragmentation

In some cases, the horizontal fragmentation of a relation cannot be based on a property of its own attributes, but is derived from the horizontal fragmentation of another relation. Consider, for example, a global relation

$$SUPPLY(SNUM, PNUM, DEPTNUM, QUAN)$$

where $SNUM$ is a supplier number. It is meaningful to partition this relation so that a fragment contains the tuples for suppliers which are in a given city. However, city is not an attribute of the $SUPPLY$ relation, it is an attribute of the $SUPPLIER$ relation considered in the above example. Therefore we need a semi-join operation in order to determine the tuples of $SUPPLY$ which correspond to the suppliers in a given city. The derived fragmentation of $SUPPLY$ can be therefore defined as follows:

$$SUPPLY_1 = SUPPLY \mathbf{SJ}_{SNUM=SNUM} SUPPLIER_1$$

$$SUPPLY_2 = SUPPLY \mathbf{SJ}_{SNUM=SNUM} SUPPLIER_2$$

The effect of the semi-join operations is to select from $SUPPLY$ the tuples which satisfy the join condition between $SUPPLIER_1$ or $SUPPLIER_2$ and $SUPPLY$, thus determining those tuples of $SUPPLY$ which refer to suppliers in San Francisco or Los Angeles, respectively.

The reconstruction of the global relation $SUPPLY$ can be performed through the union operation as was shown for $SUPPLIER$.

When a global relation R has a derived fragmentation, the qualifications of its fragments cannot be expressed as predicates which use attributes from R; rather, the condition for a tuple t to belong to a given fragment R_i of R is the existence, in some other fragment S_i of S of a tuple t' such that t and t' satisfy the semi-join specification of the derived fragmentation. Considering the above example, we give a simple representation of this condition (which should in fact include an existential quantifier) as follows:

$$q_1 : SUPPLY.SNUM = SUPPLIER.SNUM \ AND \ SUPPLIER.CITY = \text{“}SF\text{”}$$

$$q_2 : SUPPLY.SNUM = SUPPLIER.SNUM \ AND \ SUPPLIER.CITY = \text{“}LA\text{”}$$

The meaning of the above qualifications is to state, for any tuple of $SUPPLY_1$ ($SUPPLY_2$), the existence of a supplier from San Francisco (Los Angeles) with the same supplier number. The completeness of the above fragmentation requires that there be no supplier numbers in the $SUPPLY$ relation which are not contained also in the $SUPPLIER$ relation. This is a typical, and reasonable, integrity constraint for this database, and usually is called the **referential integrity** constraint.

The disjointness condition is satisfied if a tuple of the $SUPPLY$ relation which belong to two different fragments. In this case this condition is easily verified, because the supplier numbers are unique keys of the $SUPPLIER$ relation; however, in general it can be difficult to prove that this condition holds.

Vertical Fragmentation

The vertical fragmentation of a global relation is the subdivision of its attributes into groups; fragments are obtained by projecting the global relation over each group. This can be useful in distributed databases where each group of attributes can contain data which have common geographical properties. The fragmentation is correct if each attribute is mapped into at least one attribute of the fragments; moreover, it must be possible to reconstruct the original relation by joining the fragments together. Consider, for example, a global relation

$$EMP(EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)$$

A vertical fragmentation of this relation can be defined as

$$EMP_1 = \mathbf{PJ}_{EMPNUM,NAME,MGRNUM,DEPTNUM}EMP$$

$$EMP_2 = \mathbf{PJ}_{EMPNUM,SAL,TAX}EMP$$

This fragmentation could, for instance, reflect an organization in which salaries and taxes are managed separately. The reconstruction of relation EMP can be obtained as

$$EMP = EMP_1\mathbf{JN}_{EMPNUM=EMPNUM}EMP_2$$

because $EMPNUM$ is a key of EMP. In general, the inclusion of a key of the global relation into each fragment is the most straightforward way to guarantee that the reconstruction through a join operation is possible. An alternative way to provide the reconstruction property is to generate **tuple identifiers** which are used as system-controlled keys. This can be convenient in order to avoid the replication of large keys; moreover, tuple identifiers cannot be modified by users.

Notice that the above formula for the reconstruction of the global relation EMP is not complete, because the result of join EMP_1 and EMP_2 contains the column $EMPNUM$ twice. This undesired replication can be eliminated by a projection operation.

Let us finally consider the problem of fragment disjointness. First, we have seen that at least the key should be replicated in all fragments in order to allow reconstruction. In general, we can say that in vertical fragmentation the main motivation for having disjoint fragments (i.e., to control replication) is not as important as in horizontal fragmentation. In fact, if we include the same attribute in two different vertical fragments, we know exactly that the replicated data is constituted by the column which corresponds to this attribute; if instead we allow two horizontal fragments to overlap, by permitting nonmutually exclusive qualifications, then we cannot refer directly to the overlapping portion. For example, consider the following vertical fragmentation of relation EMP:

$$EMP_1 = \mathbf{PJ}_{EMPNUM,NAME,MGRNUM,DEPTNUM} EMP$$

$$EMP_2 = \mathbf{PJ}_{EMPNUM,SAL,TAX} EMP$$

The attribute $NAME$ is replicated in both fragments. We can explicitly eliminate this attribute when we reconstruct relation EMP through an additional projection operation:

$$EMP = EMP_1 \mathbf{JN}_{EMPNUM=EMPNUM} \mathbf{PJ}_{EMPNUM,SAL,TAX} EMP_2$$

Mixed Fragmentation

The fragments which are obtained by the above fragmentation operations are relations themselves, so that it is possible to apply the fragmentation operations recursively, provided that the correctness conditions are satisfied each time. The reconstruction can be obtained by applying the reconstruction rules in this case; consider, for example, the same global relation

$$EMP(EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)$$

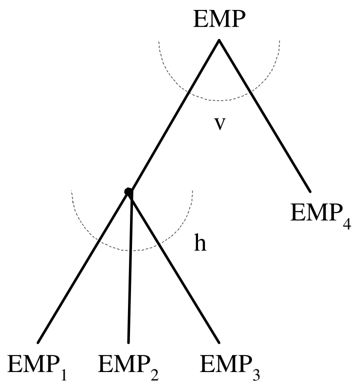


Fig.2.15. The fragmentation tree of relation EMP

The following is a mixed fragmentation which is obtained by applying the vertical fragmentation of the previous example, followed by a horizontal fragmentation on $DEPTNUM$:

$$EMP_1 = \mathbf{SL}_{DEPTNUM \leq 10}\mathbf{PJ}_{EMPNUM,NAME,MGRNUM,DEPTNUM}EMP$$
$$EMP_2 = \mathbf{SL}_{10 < DEPTNUM \leq 20}\mathbf{PJ}_{EMPNUM,NAME,MGRNUM,DEPTNUM}EMP$$
$$EMP_3 = \mathbf{SL}_{DEPTNUM > 20}\mathbf{PJ}_{EMPNUM,NAME,MGRNUM,DEPTNUM}EMP$$
$$EMP_4 = \mathbf{PJ}_{EMPNUM,NAME,SAL,TAX}EMP$$

The reconstruction of relation EMP is defined by the following expression:

$$EMP = \mathbf{UN}(EMP_1, EMP_2, EMP_3)\mathbf{JN}_{EMPNUM=EMPNUM} \\ \mathbf{PJ}_{EMPNUM,SAL,TAX}EMP_4$$

Mixed fragmentation can be conveniently represented by a **fragmentation tree**. In a fragmentation tree, the root corresponds to a global relation, the leaves correspond to the fragments, and the intermediate nodes correspond to the intermediate results of the fragment-defining expressions. The set of nodes which are sons of a given node represent the decomposition of this node by a fragmentation operation (vertical or horizontal). For example, Fig. 2.15 shows the fragmentation tree of relation EMP. The root (relation EMP) is vertically fragmented into two portions: one portion corresponds to a leaf node of the tree (EMP_4); the other portion is horizontally partitioned, thus generating the other three leaves, corresponding to fragments EMP_1, EMP_2, EMP_3.

2.4.3 The Design of Database Fragmentation

The purpose of fragmentation design is to determine nonoverlapping fragments which are "logical units of allocation," i.e., that are appropriate startpoints for the following data allocation problem.

Clearly, tuples or attributes of relations cannot be considered as individual "units of allocation," since the allocation problem would become unmanageable. Designing fragments consists in grouping tuples (in the case of horizontal fragmentation) or attributes (in the case of vertical fragmentation) which have the "same properties" from the viewpoint of their allocation. Each group of tuples or attributes having the "same properties" will constitute a fragment. The basic idea is that if any two elements of the same fragment have the "same properties" from the viewpoint of their allocation, any method used for allocating data will put them together; thus, the fragments which are obtained in this way are appropriate units of allocation. We must now make clear what we mean by "same properties" of the elements of fragments.

Example 2.1

Consider the determination of horizontal fragments for a global relation EMP. Assume that the important applications of this distributed database require information from relation EMP about employees who are members of projects. Let each department be a site of the distributed database.

Applications can be issued at any department; however, when they are issued at a given department, they refer to the tuples of employees of that department with higher probability than the tuples of other employees. This happens because existing employees are distributed in departments, because each project is made in one department, and because information about projects is typically requested by employees who work in the project. Then, in this simple case, fragments are designed by collecting in each fragment the tuples of employees who work at the same department; these tuples have the same properties from the viewpoint of their allocation.

Let us introduce also a simple example of a vertical fragmentation for relation EMP. Assume that attribute SAL and TAX are used only by administrative applications, which always use these attributes together. Then, the pair SAL, TAX is an appropriate vertical fragment.

Notice that the site of origin of each application is relevant for determining the locality properties for an appropriate definition of fragments; we therefore regard applications issued at different sites as different applications, even if they perform the same function.

In the following, we examine horizontal, vertical, and mixed fragmentation independently. As we have already observed, it is conceptually correct first to determine fragmentation and then to allocate fragments; however, we will be able to follow this approach "neatly" only in horizontal fragmentation, while we will need to take into account also the final allocation in performing vertical fragmentation.

Horizontal Fragmentation

Determining the horizontal fragmentation of a database amounts to determining both "logical" properties of data, such as the fragmentation predicates, and "statistical" properties of data, such as the number of references of applications to fragments; this coordination of logical and statistical aspects is rather difficult.

Recall that horizontal fragmentation are defined using selections on global relations; the correctness of horizontal fragmentation requires that each tuple of the global relation be selected in one and only one fragments. Thus, determining the horizontal fragmentation of a global relation requires determining a set of disjoint and complete selection predicates. The property that we require for each fragment is that the elements of them must be referenced homogeneously by all the applications.

Let R be the global relation for which we want to produce a horizontal fragmentation. We introduce the following definitions:

1. A **simple predicate** is a predicate of the type:

   Attribute = value

2. A **minterm predicate** y for a set P of simple predicates is the conjuction of all predicates appearing in P, either taken in natural form or negated, provided that this expression is not a contradiction. Thus:

$$y = \bigwedge_{p_i \in P} p_i^*$$

   where ($p_i^* = p_i$ or $p_i^* = \text{NOT } p_i$) and $y \neq$ false

3. A **fragment** is the set of all tuples for which a minterm predicate holds.

4. A simple predicate p_i is **relevant** with respect to a set P of simple predicates if there exist at least two minterm predicates of P whose expression differs only in the predicate p_i itself (which appears in the natural form in one case and negated in the other one) such that the corresponding fragments are referenced in a different way by at least one application.

Example 2.2

Consider Example 2.1 on horizontal fragmentation. Assume that some important applications require, as in Example 2.1, information about employees who are members of projects; assume that there are also other important applications which require only the data of employees who are programmers; these last applications can be issued at any site of the distributed database, and reference all programmers with the same probability. Let us assume for simplicity that there are only two departments, 1 and 2; thus $DEPT = 1$ is equivalent to $DEPT \neq 2$, and vice versa. Two simple predicates for this example are $DEPT = 1$ and $JOB =$ "P" (programmer). The minterm predicates for these two predicates are

$$DEPT = 1 \text{AND} JOB = \text{“P”}$$

$$DEPT = 1 \text{AND} JOB \neq \text{“P”}$$

$$DEPT =\neq \text{AND} JOB = \text{“P”}$$

$$DEPT =\neq \text{AND} JOB \neq \text{“P”}$$

All the above simple predicates are relevant, while, for instance, $SAL > 50$ is not a relevant predicate; in fact, if we partition any of the fragments corresponding to one of the above minterm predicates in two parts having either $SAL > 50$ or $SAL \leq 50$, then each one of the two fragments obtained in this way is referenced in the same way by applications.

The above definitions are not constructive; we still don't know how to build fragmentation. Unfortunately, the selection of predicates cannot be helped too much by precise rules, since understanding that a predicate is useful for describing fragmentation relies mostly on the intuition of the database designer. However, we can define two properties which characterize an appropriate fragmentation.

Let $P = p_1, p_2, \ldots, p_n$ be a set of simple predicates. in order for P to represent fragmentation correctly and efficiently, P must be complete and minimal.

1. We say that a set P of predicates is **complete** if and only if any two tuples belonging to the same fragment are referenced with the same probability by any application.

2. We say that the set P is **minimal** if all its predicates are relevant.

Fragmentation can be produced with the following method:

Basis Consider a predicate p_1 which partitions the tuples of R into two parts which are referenced differently by at least one application. Let $P = p_1$.

Method Consider a new simple predicate p_i which partitions at least one fragment of P into two parts which are referenced in a different way by at least one application. Set $P \leftarrow P \cup p_i$. Eliminate nonrelevant predicates from P. Repeat this step until the set of the minterm fragments of P is complete.

Vertical Fragmentation

Determining the vertical fragmentation of a global relation R requires grouping into sets attributes which are referenced "in the same way" by applications. We distinguish the **vertical pertitioning** problem, in which sets must be disjoint, from the **vertical clustering** problem, in which sets can overlap. The correctness conditions for vertical partitioning require that each attributed of R belong to at least one set and that each set include either a key of R or a "tuple indentifier".

The purpose of vertical fragmentation is to identify fragments R_i such that many applications can be executed using just one fragment. Let us consider a global relation R which is vertically partitioned into R_1 and R_2. An application takes advantage of this vertical partitioning if it can be executed using one of R_1 or R_2, since in this case the access to the (larger) relation R is avoided; however, if an application requires both R_1 and R_2, then the fragmentation is not beneficial, since an additional join is required for reconstructing R. This criterion applies also to nondistributed databases; in distributed databases, the advantage of vertical partitioning is greater when many applications which use R_1 and many applications which use R_2 are issued at different sites. In this case, placing R at one of the sites would penalize the applications at the other site, while fragmentating R allows satisfying both.

Determining a fragmentation for a global relation R is not easy, since the number of possible partitionings grows combinatorially with the number of attributes of R, and the number of possible clusters is even larger. Thus, in the presence of a large relation, heuristic approaches are necessary to determine the partitions or clusters; we briefly indicate how such methods operate.

Two alternative approaches are possible for attribute partitioning:

1. The **split approach** in which global relations are progressively split into fragments

2. the **grouping approach** in which attributes are progressively aggregated to constitute fragments

Both approaches can be classified as $greedyheuristics$, since they proceed by making at each iteration the "best" possible choice. Some form of back tracking can be made, by trying to move attributes from one set to another on the final partitioning.

Vertical clustering introduces **replication** within fragments, since values of overlapping attributes are replicated. Replication has a different effect on read-only and update applications. Read-only applications take advantage of replication, because it is more likely that they can reference data locally. For update applications replication is not convenient, since they must update all copies in order to preserve consistency.

Let us consider what happens when two fragments R_1 and R_2 are overlapping; i.e., there exists a set of attributes I which belong to both R_1 and R_2. Assume that R_1 and R_2 are at sites 1 and 2. Then read applications at site 1, using attributes of I together with other attributes of R_1, are local to site 1; likewise, read applications at site 2, using attributes of I together with other attributes of R_2, are local to site 2. However, update applications which change the value of attributes of I must reference them at both sites.

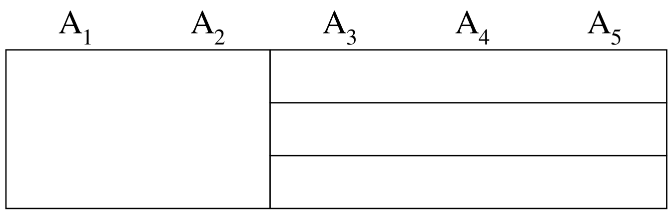
Therefore, in evaluating the convenience of vertical clustering, it is important that overlapping attributes are not heavily updated.

Mixed Fragmentation

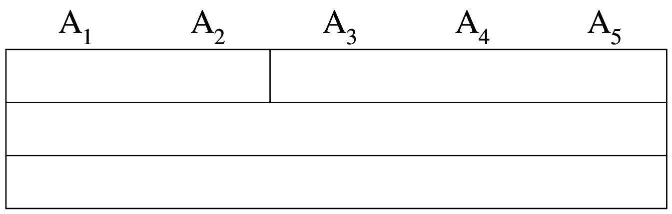
Finally, we examine mixed fragmentation. The simplest ways for building mixed fragmentation consist of:

1. Applying horizontal fragmentation to vertical fragments

2. Applying vertical fragmentation to horizontal fragments



(a) Vertical fragmentation followed by horizontal fragmentation



(b) Horizontal fragmentation followed by vertical fragmentation

Fig.2.16. Mixed fragmentation of relation $R(A_1, A_2, A_3, A_4, A_5)$

Although these operations can be recursively repeated, generating fragmentation trees of any complexity, it seems that having more than two levels of fragmentation is not practical interest. Solutions 1 and 2 allow both fragmentations to be considered for each relation, and thus to be taken advantage of. Note that the second fragmentation can be applied to subsets of the fragments produced by the first one; for instance, Fig. 2.16a and b show examples in which:

1. Horizontal fragmentation is applied just to one fragment produced by vertical fragmentation.

2. Vertical fragmentation is applied just to one fragment produced by horizontal fragmentation.

The order in which horizontal and vertical fragmentations are applied can affect the final fragmentation.

2.4.4 The Allocation of Fragments

The data allocation problem has been widely analyzed in the context of the "file allocation problem"; and there have been a number of works on the file allocation problem. The easiest way to apply these works to the fragment allocation problem is to consider each fragment as a separate file; however, this approach is not convenient for the following reasons;

1. Fragments are not properly modeled as individual files, since in this way we do not take into account the fact that they have the same structure or behavior.

2. There are many more fragments than original global relations, and many analytic models cannot compute the solution of problems involving too many variables.

3. Modeling application behavior in file systems is very simple (typically, applications make a "remote file access"), while in distributed databases applications can make a sophisticated use of data.

Some of these problems have not yet been solved satisfactorily; for instance, problem 3 is particularly difficult, since the correct approach would be to evaluate data distribution measuring how "optimized" applications should behave with it. This, however, requires optimizing all the important applications for each possible data allocation.

We do not present in this section any of the models for file allocation. There are, however, some general criteria that can be used for allocating fragments; we concentrate on them here.

General Criteria for Fragment Allocation

In determining the allocation of fragments, it is important to distinguish whether we design a final **nonredundant** or **redundant** allocation.

Determining a nonredundant final allocation is easier. The simplest method is a "best-fit" approach; a measure is associated with each possible allocation, and the site with the best measure is selected. This approach gives a solution which disregards the "mutual" effect of placing a fragment at a given site if a related fragment is also at that site.

Replication introduces further complexity in the design, because:

1. The degree of replication of each fragment becomes a variable of the problem.

2. Modeling read application is complicated by the fact that the applications can now select among several alternative sites for accessing fragments.

For determining the redundant allocation of fragments, either of the following two methods can be used:

1. Determine the set of all sites where the benefit of allocating one copy of the fragment is higher than the cost, and allocate a copy of the fragment to each element of this set; this method selects "all beneficial sites."

2. Determine first the solution of the nonreplicated problem, and then progressively introduce replicated copies starting from the most beneficial; the process is terminated when no "additional replication" is beneficial.

Both methods have some disadvantages. In the "all beneficial sites" method, quantifying costs and benefits for each individual fragment allocation is more critical than the nonredundant case, because it disregards the "mutual" effects of allocating different copies of the same fragment. The "additional replication" method is a typical heuristic approach; with this method, it is possible to take into account that the increase in the degree of redundancy is progressively less beneficial. Both the reliability and availability of the system increase if there are two or three copies of the fragment, but further copies give a less than proportional increase.

2.4.5 Allocation Problems in Distributed Databases

The problem of where to allocate a file and its copies, given a known set of retrievals and updates and their execution frequencies such that a cost function is minimized, is known as the *file allocation problem.* However, Apers [22] observes the file allocation problem does not adequately describe the allocation problem in distributed databases and state the data and operation allocation problem.

According to [22], The solutions for the file allocation problem do not characterize solutions to the allocation problem in a distributed database for the following reasons:

- The objects to be allocated are known prior to allocation. Relations, which describe logical relationships between data, are not suited as units of allocation because users at different sites might be interested in different fragments of a relation.

- The way the data are accessed is far more complex. In the file allocation problem the only transmission required to combine data from different files are transmissions from sites containing files to the result site, where the result is computed. In current research on distributed query processing, It is observed that to process a query, data transmissions between sites where fragments are allocated are also needed. This means that the fragments cannot be allocated independently.

To capture these aspects, the file allocation problem is generalized into the *data and operation problem*:

> *given the queries and updates, the frequencies of their usage, and the sites where the results have to be sent, determine (1) the fragments to be allocated, and (2) allocate these fragments, possibly redundant, and the operations on them to the sites of the computer network such that a certain cost function is minimized.*

Namely, when solving the file allocation problem, it is useful to consider simultaneously the allocation of the operations, i.e., how we can optimize query processing. In the next section, we will also present the basic concepts of distributed query optimization.

Data allocation and query optimization are two important distributed systems design problems that are more closely interrelated. Distributed query optimization depends on how the data are allocated, since processing schedules often include operations on different sites and data transmissions between them. On the other hand, the optimal method of allocating data depends on the processing strategies used for solving queries.

However, solving query optimization as a subproblem of data allocation is probably too hard, since it requires either an excessive amount of simulation or an excessively complex analytic computations. Typically, researchers studying data allocation assume a fixed query optimization method to generate processing schedules, while researchers on query optimization assume a fixed data allocation. By assuming a solution to one problem and solving the other, researchers control the complexity of these two problems.

2.5 Distributed Query Optimization

In this section, we give a classification of query processing problems, of the assumptions which are required for modeling and solving them, and of the criteria that are used in the optimization.

2.5.1 Problems in Query Optimization

The selection of a query processing strategy involves:

1. Determining the physical copies of the fragments upon which to execute the query, given a query expression over fragments. The term **materialization** is typically used in the literature to denote a nonredundant copy of the entire distributed database upon which the query is executed. In terms of the reference model of section 2.3, a materialization corresponds to the selection, for each fragment, of one of the fragment's copies. Notice that different queries will use, in general, different materializations.

   The selection of a materialization for a query is by itself a limitation on the general formulation of the problem, as there is no reason, in principle, to select different copy of each fragment for executing the same query. For instance, different subexpressions requiring the same fragment could be executed over different copies of it. However, most query processing algorithms operate in the context of a materialization.

2. Selecting the order of execution of operations; this involves the determination of a "good" sequence of join, semi-join, and union operations, since determining the order of execution of the other operations is not difficult.

3. Selecting the method for executing each operation; this involves the choice of performing some algebraic operations together within the same database access (for instance, executing selections and projections on the same operand at the same time), and the selection of a method for executing each database problem is to determine the best method for evaluating joins.

The above problems are not independent: for instance, the choice of the best materialization for a query depends on the order in which operations are executed; therefore, proceeding by solving them independently introduces errors. However, a typical simplification which is made by optimization methods consists exactly in considering the three problems as independent; thus:

1. A materialization is assumed for a given query.

2. The order of execution of operations is optimized.

3. Operations are clustered into local programs.

In practice, the first problem is often bypassed (by taking a materialization as granted), the third problem is also disregarded (because it is system-dependent), and the emphasis is placed on the second problem.

Query optimization is the process of ensuring that either the total cost or the total response time for a query are minimized. When distributed query processing techniques are compared, it is usually on the basis of cost of query execution (execution cost) or on the basis of time taken between the submission of the query and the receipt of the reply (response time). At the most general level, response time can be taken to represent a cost to an organization.

Execution cost optimizers aim to minimize the use of the (total) system resources for a query and hence reduce its cost. The value of the objective function for the optimization process is therefore obtained from the sum of expressions representing the consumption of the individual system resources.

Response time optimizers aim to minimize the response time for a query by exploiting parallelism, in particular that natural parallelism provided by having multiple processors in the network. Clearly sub-queries on the 'critical path', namely those determining the minimum response time for the complete query, merit most attention.

The value of the objective function for optimization in this case is obtained from the totality of expressions representing sub-queries on the critical path. It is easy to see that there could well be a difference between the 'fastest' and the 'cheapest' execution strategies in particular cases.

Optimization of query execution is an intractable problem and perfect data on the system's behavior is unlikely to be available, but satisfactory sub-optimal results, based on heuristic methods can be obtained.

2.5.2 Methods of Performing Joins

Of the basic relational operations (SELECT, JOIN, PROJECT) the most expensive in terms of both time and money is JOIN. It causes more page swaps than the other relational operations. The number of page swap is an important contribution to response time and cost. In a paging environment the storage devices form a hierarchy based on speed (such as primary storage, fast disk, slow disk and archiving storage), so swapping between levels is required. The frequency of page swaps is often used as the cost measure when modelling these systems.

Procedures can be devised for reducing the cost of JOINs when all possible combinations of pages of two or more relations are matched. However, in many cases the number of page pairs to be considered can be quite dramatically reduced. Some of these cases occur when at least one of the relations being joined is affected by either a preceding JOIN or a preceding SELECT. Alternatively a directory holding information on the values stored on each page can be used in order to ignore page pairs with no matching values.

Graph methods are often used in the study of JOIN. Object graphs and operator graphs have already been met. In **query graphs**, in the present context, for a given query the nodes represent relations (the query variables). An edge exists between two nodes if there is a JOIN predicate (usually equality in which case we have an EQUI-JOIN) in the query linking attributes in the corresponding relations.

A **tree query** is defined as one with no cycles in the graph. Otherwise it is a **cyclic query**. Efficient procedures to test whether or not a given query is a tree query are available. Tree queries can be processed efficiently but cyclic query processing is difficult and any procedure for processing cyclic queries must convert them into tree queries.

In the rest of this section we look at the two major approaches to the handling of JOINs in distributed databases, namely using SEMI-JOINs or not.

SEMI-JOIN Methods

A SEMI-JOIN involves an increase in local processing of data in comparison to a JOIN, but saves on the cost of data movement between sites.

Informally, it is the projection of the tuples of the relation resulting from the JOIN on to one (named) relation, say, R_1, of the pair (R_1, R_2) of relations being joined.

In what follows we use the notation $\lhd$ for the SEMI-JOIN operation with an appended condition as in the JOIN operation.

Suppose π_i is the PROJECTION over the attributes of R_i and $\bowtie$ is the natural JOIN with the same appended JOIN conditions as $\lhd$. Then the SEMI-JOIN can be defined as follows

$$R_2 \lhd R_1 = \pi_1 (R_1 \bowtie R_2)$$

This formula can clearly be replaced by the equivalent form

$$R_2 \lhd R_1 = R_1 \bowtie (\pi R_2) \tag{2.1}$$

where π is over the JOIN attributes only.

This second representation has potential advantages over the first if R_1 and R_2 are at different sites A and B and the execution of a full JOIN between R_1 and R_2 is to be done at a third site C. In the SDD-1 system [102] the execution of the πR_2, transmission and local JOIN at A (giving the result of the SEMI-JOIN) is referred to as the **reduction phase** of a distributed JOIN. It is followed by a simpler **processing phase**, where the two reduced relations at A and B are shipped to a strategic site such as C and finally JOINed. This method can be used to reduce communication costs when a full JOIN is to be executed at C. First R_1 and R_2 are both 'SEMI-JOIN reduced' as above, and then both sub-results are sent to C for execution of the JOIN.

Suppose R_1 is the SEMI-JOIN relation (i.e. as in Equation (2.1) above) and that the result of the SEMI-JOIN is required at a third site C. The procedure is as follows:

Procedure SEMI-JOIN
Begin

Step 1. Project the JOIN attributes from R_2 at B ($= \pi R_2$), after applying any required SELECTIONs.

Step 2. Transmit πR_2 to A.

Step 3. Compute the SEMI-JOIN of R_1 at A.

Step 4. Move the result to C.

end

The advantages of the SEMI-JOIN algorithm are clearly greatest when πR_2 is relatively small (and so step 3 reduces the number of tuples from both local sites to be transmitted to the result site), and local processing costs are much smaller than trasmission costs. The selectivities of SMI-JOINs are what makes it worthwhile to use them. JOINs can increase the size of the database; SEMI-JOINs never do.

When the JOIN is done at A or B, let R_i^1 be the projection of R_i onto the JOIN columns, and let $|R_i \lhd R_j|$ be the size of the SEMI-JOIN result. A SEMI-JOIN will have a lower transmission cost than a conventional JOIN if the cost of initiating a message (remember that extra incurrence of this cost has to be included for SEMI-JOINs) added to $\min(|R_2^1| + |R_2 \lhd R_1|, |R_1^1| + |R_1 \lhd R_2|)$ is less than $\min(|R_1|, |R_2|)$.

We can generalize this discussion by considering cases where there are more than two relations. Here the reduction phase of query evaluation has the objective of translating a query into a cost effective series of SEMI-JOINs.

Non-SEMI-JOIN Methods

In System R* [101] the query execution strategy is generated at compilation time and it therefore uses a *static* optimization algorithm. Queries can be compiled during times when the system is slack and the optimal access plans for their execution are determined then. Frequently-used SQL queries clearly benefit, but since bindings are determined prior to execution time, compilation might have to be repeated from time to time. When a JOIN over several sites is required, a choice can be made of sites at which to carry out the JOINing, as for SEMI-JOINs. There is also a choice between two techniques for transmitting relations between sites. We refer to these as 'fetch once' and 'fetch as required'.

The former method involves sending the *completerelation* as a stream of tuples to the other site, where it is either temporarily stored (at some I/O cost) for later execution of JOIN, or a JOIN is executed tuple by tuple on arrival. In 'fetch as required' shipment the transfer of *individualtuples* is coordinated by the node executing the JOIN. For example, a message can be transmitted to the other node (the one sending the data) to indicate that tuples with a particular JOIN column value are expected, and actions are suspended at the receiving site until matching tuples are returned. This could mean re-fetching some tuples which are required more than once.

The nested loop and merge scan methods are used for the JOIN in R*. The nested-loop method scans one relation called the **outer relation** for each tuple of the other relation, called the **inner relation** with a matching JOIN value. One simple way of doing this is to scan the entire inner relation for each tuple of the outer relation. The merge-scan method depends on the two operand relations being sorted on the JOIN attribute, and merging the two sorted relations produces the JOIN result. The choices of JOIN method, JOIN order, and access methods are all carried out as in the centralized System R optimizer. The efficiency of JOINing of several sub-query results usually depends on the order of access to the relations. (This is not true of the UNION of sub-query results, where each sub-query can be optimized optimally.)

Preceding query execution, four stages of 'query preparation' are completed in R*. They are

- analyze the SQL query and perform object and name resolution;
- look up the catalog to check authorization (privilege) and view integration;
- plan the minimum cost access strategy (a string of relation accesses, PROJECTIONs, SELECTIONs, data communication transfers, SORTs, and JOINs) for this global query;
- generate the access modules and store them at a master site.

The optimizer is invoked at the third stage.

A search tree is traversed to estimate the cost of each possible way of servicing the particular SQL query.

The published cost formulae include message, I/O and CPU contributions, calculated as follows:

$$\text{I/O} = \text{pages fetched} \times \text{weight}C_p$$

$$\text{CPU} = \text{tuples fetched} \times \text{weight}C_{cpu}$$

$$\text{message} = \text{messages sent} \times \text{weight}C_m$$

where the three weights C_p, C_{cpu} and C_m represent relative importance values distinguishing between I/O, CPU and messages respectively. The ratio vary with the system of course.

The criteria used for the partitioning of any relation are taken into account so that the costs of reconstructing it are kept low. The JOIN order and presentation order are considered in addition to the I/O, CPU and message factors above. This makes the fan-out of the search tree exceedingly large. Pruning heuristics for the centralized DBMS System R are used to reduce the space to be searched for an optimal plan.

To find the optimal execution for a JOIN-based query, System R builds a decision tree of the strategies for executing the query. This tree has branches corresponding to operations such as sorts or index accesses on the relations and each leaf of the tree identifies a different way of executing the query expressed as the root of the tree.

Heuristic rules are used to constrain the size of the tree. Some of the rules inspect the individual relations to discard unpromising strategies early (i.e. even before consideration of the JOIN method to be used). Exhaustive searching is then used to find the most efficient JOIN sequence, methods and sites. A well-known operations research technique called Dynamic Programming is used to speed up the search.

2.5.3 Distributed Query Optimization Algorithms

In this section, we illustrate the use of the techniques in the following four query optimization algorithms: the reduction algorithm of Distributed INGRES [100], System R* algorithm [101], SDD-1 algorithm [102], and family of algorithms of Apers, Hevner, and Yao [33], which we will refer to as AHY. We describe them because they are representative of different classes of algorithms and are therefore often used as paradigms.

1. The optimization timing is dynamic for distributed INGRES, while it is static for the others.

2. The objective function of SDD-1 and R* is to minimize total cost, while distributed INGRES and AHY aim at decreasing a combination of response time and total cost.

3. The optimization factors of the cost function are the message size for SDD-1. AHY uses message size and number of messages. System R*, which takes into account local processing cost, uses message size and local processing time (I/O + CPU cost).

4. The network topology is assumed to be a wide area point-to-point network by SDD-1 and AHY. The algorithms of distributed INGRES and R* can work in both local and wide area networks.

5. The use of SEMI-JOINs as a query optimization technique is employed by SDD-1 and AHY. Distributed INGRES and R* perform JOINs in a fashion similar to that of the centralized query optimization algorithms of their counterparts: INGRES and System R.

6. Each algorithm assumes statistical information about the data. As shown in Fig. 2.17, SEMI-JOIN algorithms typically use more information.

7. INGRES can handle fragments.

The difference between these four algorithms are summarized in Fig. 2.17.

Algorithms	Optm. Timing	Objective Function	Optm. Factors	Network Topology	Semi Joins	Stats[†]	Fragments
Distributed INGRES	Dynamic	Response time or total cost	Msg size, proc. cost	General or broadcast	No	1	Horizontal
R*	Static	Total cost	#Msg, Msg size, IO, CPU	General or local	No	1,2	No
SDD-1	Static	Total cost	Msg size	General	Yes	1,3,4,5	No
AHY	Static	Response time or total cost	#Msg, Msg size	General	Yes	1,3,5	No

[†] 1 = relation cardinality, 2 = number of unique values per attribute, 3 = join selectivity factor, 4 = size of projection on each join attribute, 5 = attribute size and tuple size.

Fig.2.17. Comparison of query optimization algorithms

2.6 Summary

In this chapter, we present concepts and the background of optimal design in distributed systems. Distributed systems have become the norm for the organization of computing facilities. They can be used to implement general-purpose interactive computing systems and to support a wide range of commercial and industrial applications of computers. They are increasingly being used as the basis for new applications in areas such as networked information services and multimedia applications, where communication is a basic requirement.

In all of those roles, distributed systems are capable of offering substantial benefits to their users. We have indicated their key characteristics under the following headings: resource sharing, openness, concurrency, scalability, fault tolerance and distribution transparency. The benefits that they bring are not achieved automatically, they depend upon the careful design of system components.

We have presented data allocation and query optimization which are especially important problems in the design of distributed systems. The objectives of the design of data distribution and the way of the data fragmentation for the distribution were presented as well as several algorithms of distributed query optimization It is very significant to find the effective solution for system resource allocation in view of many design issues of distributed systems. In the following chapters, we propose the optimization models for the system resource allocation in distributed systems.

It is useful to consider the allocation of operations as well as the file allocation, such as query scheduling and query processing issues. In this study, however, as it is too complicated to solve the both problems together, we basically focus on the file allocation and the mirror server allocation based on it.

Chapter 3

Cost-Based Optimal File Allocation

3.1 Introduction

There has been a great trend to shift from centralized systems to distributed systems. We may easily imagine that in distributed systems, it will be preferable to allocate important and popular files in several servers replicately in view of the delay, system scalability, reliability, and so on.

These distributed systems consist of several servers, with multimedia databases, that separate to each other geographically, and each server constitutes a subnetwork with a number of users. Each server is connected to one another by a high-speed communication network, and the whole system works as a distributed database system. In these distributed database systems, files must be stored in at least one of servers, and are shared by several users connected with different servers. Users can access any files stored in any servers. Replicated copies of the files accessed frequently are to be stored in several servers. Then, in distributed systems, it becomes very important problems which server stores the files required by users, because there are remarkable differences in the distance between to a local database and to the other databases.

Thus, we should consider if it is accessed frequently or not, and find optimal allocation of files. Each server should store the files that it accesses frequently to reduce the communication cost and delay. However, the replicated storage of the same files in the system cause to increasing in the storage cost required to store them. Also, there is necessity to keep the data consistency, and then the rewriting cost is increasing. Moreover, we should take account of the difference between the local communication costs and the global communication costs described above, which is an important element for distributing files. We should consider the optimal file allocation problem in view of these problems.

[22][34][42][43][46] presented the optimal file allocation to minimize costs in distributed database systems. However, they did not consider delays. It is obvious that there is a trade-off between costs and delays. Our objective is to find the optimal file allocation such that the total cost is minimized subject to the total delay in distributed systems.

It is necessary to introduce a new useful model, which may be applied to heterogeneous network environments, to find the optimal file allocation. In this study, we propose the cost-based optimal file allocation model which is capable of evaluating the optimization in considering the problems described above. We may apply this model to various network environments by controlling several parameters. Since we consider the consistency among the same files, this model covers wide range of multimedia network applications such as Video on Demand (VoD) systems, corporate intranets, etc.

In our optimal file allocation model, We introduce a 0-1 integer programming formulation that makes this model belongs to the class of NP hard problems. Therefore, it is very difficult to develop an efficient algorithm. We use an approximate method when solving this optimization problems in order to deal with practical large-scale systems. Through comparison results, we show that this approximate method is practical enough in view of both calculation time and accuracy. Using this approximate method, we quantitatively show the general characteristics on file allocation in distributed systems.

In this model, we introduce many cost and delay parameters to evaluate the total cost and delay in the system more exactly. In constructing practical systems, it is necessary to investigate the weight and the contribution of each parameter to the total cost. It is very useful to show how to estimate cost and delay parameters, on the basis of this analysis. In this chapter, we also show the results of the sensitivity analysis of these parameters on our optimal file allocation model and make clear the influence between principal parameters.

3.2 Optimal File Allocation Model

In this section, we propose the cost-based *Optimal File Allocation Model* as a 0-1 integer programming model. First, we introduce a system model of a distributed networks, that is followed by the formulation of the objective function and the restrictive conditions in the optimal file allocation problem.

3.2.1 System Model

We introduce a system model of a distributed system for the optimal file allocation that may be applied to wide range of multimedia network applications.

In Fig. 3.1, we show some typical organization of a distributed information network system. As shown in Fig. 3.1 , this system consists of servers, users, files, communication lines, and a high-speed communication network.

There exist s servers in the system. Each server is denoted by S_i $(1 \leq i \leq s)$, and has the storage capacity of B_i. Each server constitutes a subnetwork with a number of users. It is assumed that each user of system is connected with one of

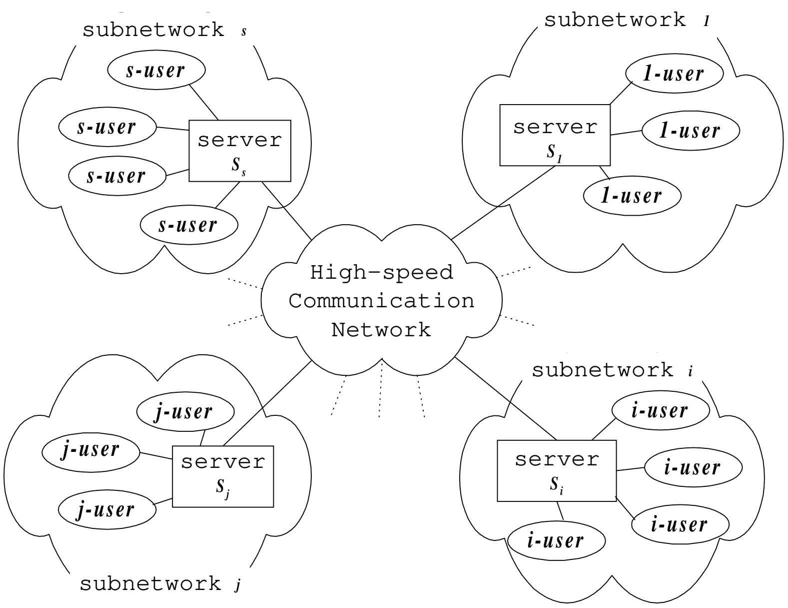


Fig.3.1. A model of a distributed network system

servers, which is called a local server. The users connected with the server S_i is called *i-users*. In this system, each server is connected by the high-speed communication network, and the whole system works as a distributed database system. In addition, we consider the selection of communication lines between servers and the communication network, since quality and costs of communication are mainly dependent on what classes of lines we use. There are u classes of lines that we can select, which is classified according to line quality and communication rates. Each server selects and uses a line of class n $(1 \leq n \leq u)$. Each subnetwork can take individual communication rates by selection of lines.

Also, the number of distinct files is m, and each file is denoted by M_k $(1 \leq k \leq m)$. It is possible to allocate multiple copies of each file in several servers, and the maximum duplicate number of the file M_k is r_k. The number of copies of the file M_k in the whole system should be less than or equal to r_k. It is assumed that the size of the file M_k is denoted by F_k. Each server can store these files as long as the total size of files does not exceed the capacity of the server.

i-users generate file access flow, where the average is λ_i per unit time. This amount of flow generates queuing time at servers. We should assume the queuing process as M/G/1. In analyzing of M/G/1 systems, we have to measure service distributions in the real systems and decide several parameters, averages, variances, etc. However, we do not have any practical environments to evaluate them. Here, for analytical simplicity, we take M/D/1 instead of M/G/1. The access size from *i-users* for the file M_k is denoted by L_{ik}. This access size is less than or equal to the file size F_k. Also, let P_{ik} denote the access probability from *i-users* for the file M_k $(\sum_k P_{ik} = 1)$. The parameter P_{ik} determines the traffic matrix in the system. In particular, if the access probability has the same value for the same k, the system is file symmetric. Moreover, the probability that *i-users* update the file M_k is denoted by Pw_{ik}. We may discuss various kinds of multimedia network applications according to the value of Pw_{ik}. If we consider VoD(Video on Demand) systems, we must determine that all Pw_{ik} is equal to zero. If Pw_{ik} is large, we can consider the system that there frequently happens to rewrite files, such as corporate intranets. In this study, we apply the write-through consistency protocol to keep the consistency among the same files.

3.2.2 Formulation

We formulate* the problem to minimize the total cost per unit time required in the whole system, subject to the total delay in the whole system and the initial facility cost.

Our objective is to find the optimal file allocation such that the total cost is minimized in distributed systems. The cost is very important criterion to evaluate distributed database systems. However, some systems that optimizes only the cost is not so useful, because the system may require a great total delay. This is why we take into account the total delay as the restrictive condition.

To formulate *the Optimal File Allocation Problem*, we define two kinds of 0-1 variables. We formulate this optimization problem as 0-1 integer programming problem using the following 0-1 variables.

1. The variables on the allocation of files X_{ik} [12].

$$X_{ik} = \begin{cases} 1 & \text{(the file } M_k \text{ is stored in the server } S_i\text{)} \\ 0 & \text{(otherwise)} \end{cases} \tag{3.1}$$

2. The variables on the selection of lines classes Y_{in}.

$$Y_{in} = \begin{cases} 1 & \text{(the server } S_i \text{ uses the line of class } n\text{)} \\ 0 & \text{(otherwise)} \end{cases} \tag{3.2}$$

Initial Conditions

Under the assumptions presented in the previous section, we add the following three conditions as initial conditions.

1. The number of copies of the file M_k in the system is less than or equal to r_k, that is

$$1 \leq \sum_i X_{ik} \leq r_k \tag{3.3}$$

*In the following formulation, it is assumed that $\sum_i$ denotes the summation for servers in the system, $\sum_k$ denotes the summation for files in the system, and $\sum_n$ denotes the selection of line classes.

2. Server S_i can store files as long as the total size of files does not exceed the capacity of the server, that is

$$\sum_k X_{ik} F_k \leq B_i \tag{3.4}$$

3. Each server selects and uses one of u classes of lines between servers and the communication network, that is

$$\sum_n Y_{in} = 1 \tag{3.5}$$

Objective Function

We formulate the total cost per unit time in managing the whole system as the objective function. It is assumed that the total cost C consists of four elements. Those are the communication cost Ct_{ik}, the server cost Crs_{ik}, the storage cost Cst_i, and the line cost Crl_i. The communication cost and the server cost are the costs for each access from an *i-user* for a file M_k, and we make the communication cost and the server cost the cost per unit time by multiplying by the average access rate λ_i from *i-users* to files per unit time. Also, the server S_i is required the storage cost and the line cost per unit time. That is,

$$C = \sum_i \left(\lambda_i \sum_k (Ct_{ik} + Crs_{ik}) + Cst_i + Crl_i \right) \tag{3.6}$$

We formulate these four elements of the cost using the 0-1 variables X_{ik} and Y_{in}. We may find the optimal file allocation and the optimal selection of lines by minimizing this equation subject to several restrictive conditions.

We give full details of the communication cost, the server cost, the storage cost, and the line cost in the following.

1. Communication Cost

   We assume that the communication cost is required to communicate between a user site and his/her request file site. For example, when an *i-user* accesses a file M_k, the communication cost Ct_{ik} will be required to communicate between an *i-user* and a server that stores the file M_k. In this study, this communication cost is the sum of the cost required to communicate between an *i-user* and his/her local server S_i, between the server S_i and the communication network, and between the communication network and the server stored the file M_k. If a local server S_i stores the file M_k, the communication cost Ct_{ik} becomes the cost required to communicate only between an *i-user* and the local server.

Also, in this study, if a file required from a user is not stored in the local server and several remote servers store the file, the user accesses the server that has the minimum sum of the communication cost and the server cost. This assumption may cause congestion on a specific server that has the less server cost, if the server cost has more influence than the communication cost. In this case, we might need to assign more copies of files to more servers.

We have the communication cost Ct_{ik} in an access from an *i-user* to a file M_k. Let Ctl_i denote the local communication cost coefficient between an *i-user* and a local server S_i. Let Ctr_{in} denote the remote communication cost coefficient between a server S_i connected to a line of the class n and the communication network, which includes the cost to communicate through the communication network. These coefficients have fixed values for individual lines.

$$Ct_{ik} = (X_{ik}\mathsf{Ctl}_i + \mathbf{Xf}_{ij}\mathbf{Cy}_{ij})\, L_{ik}P_{ik} \tag{3.7}$$

Here, it is assumed that index j equals to the value that minimize the following function on the positive side $(1 \le j \le s)$.

$$h1_{\min}(j) = \mathbf{Xf}_{ij}\,(\mathbf{Cy}_{ij} + \mathsf{Csa}_j) \tag{3.8}$$

Csa_j is the access cost coefficient for a service at the server S_j as described in the next paragraph. Also, we define $\mathbf{Xf}_{ij}$ and $\mathbf{Cy}_{ij}$ as follows.

$$\mathbf{Xf}_{ij} = (1 - X_{ik})\, X_{jk} \tag{3.9}$$

$$\mathbf{Cy}_{ij} = \mathsf{Ctl}_i + \sum_n \mathsf{Ctr}_{in} Y_{in} + \sum_n \mathsf{Ctr}_{jn} Y_{jn} \tag{3.10}$$

2. Server Cost

   We assume that the server cost is the cost of a service for an access from a user for a file at the server. Similarly, if a file required from a user is not stored in the local server and several remote servers store the file, the user accesses the server that has the minimum sum of the communication cost and the server cost.

We have the server cost Crs_{ik} in an access from an *i-user* to a file M_k. We consider three kinds of costs.

(i) The access cost for a service at the server

(ii) The update cost to update all of copies of the file when the file is updated. The update cost includes the cost for the communication in updating them.

(iii) The relay cost to relay in a local server when the server does not have a request file from local users

Let Csa_i, Csw_i, and Csr_i denote respectively the access cost coefficient, the update cost coefficient, and the relay cost coefficient at the server S_i. They depend on the coefficient Crs_i, that is, $\mathsf{Csa}_i = a\mathsf{Crs}_i$, $\mathsf{Csw}_i = w\mathsf{Crs}_i$, and $\mathsf{Csr}_i = r\mathsf{Crs}_i$. We can select values of the coefficient a, w, r according to system environments. Here, let Crs_i be the server cost coefficient of the server S_i, and this coefficient has a fixed value for an individual server.

$$
\begin{aligned}
Crs_{ik} &= (X_{ik}\mathsf{Csa}_i + (1 - X_{ik})\mathsf{Csr}_i + \mathbf{Xf}_{ij}\mathsf{Csa}_j)\, L_{ik}P_{ik} \\
&+ \left(\sum_{l \neq i} (X_{lk}\mathsf{Csw}_l) \right) RCnf\, P_{ik}Pw_{ik} \qquad (3.11)
\end{aligned}
$$

Here, similarly to the case of the communication cost, it is assumed that the index j equals to the value that minimize the function $h1_{\min}(j)$ (eq.(3.8)) on the positive side$(1 \leq j \leq s)$. Also, $RCnf$ denotes the local-global difference of the communication cost and we will mention it later.

3. Storage Cost

We assume that the storage cost is required for each server to store multimedia files. It is charged on unit time basis, and not dependent on the frequency of accesses. In this study, the storage cost depends on the size of all files stored at a server. We have the storage cost Cst_i of server S_i. Let Cst be the storage cost coefficient, which has a fixed value in common for all servers.

$$
Cst_i = \mathsf{Cst} \sum_k X_{ik}F_k \qquad (3.12)
$$

4. Line Cost

   We assume that the line cost is required to utilize a communication line between each server and the high-speed communication network. Similarly to the storage cost, it is charged on unit time basis and not dependent on the frequency of accesses. Let Crl_{in} denote the line cost coefficient between a server S_i connected to a line of the class n and the communication network. We have the line cost Crl_i between a server S_i and the communication network.

$$Crl_i = \sum_n \mathsf{Crl}_{in} Y_{in} \tag{3.13}$$

Restrictive Conditions

We formulate the total delay in working the whole system and the initial facility cost as the restrictive conditions for the optimal file allocation problem.

1. Total Delay

   It is assumed that the total delay consists of the communication delay Dt_{ik}, the service delay Dx_i, and the update delay Dw_i. For each access from an *i-user* for a file M_k, an *i-user* has to spend these delays. Let $D_{\max}$ be defined as the maximum total delay, and the total delay is formulated as follows.

$$D = \sum_i \left(\lambda_i \left(\sum_k Dt_{ik} + Dw_i \right) + \lambda_i^* Dx_i \right) \leq D_{\max} \tag{3.14}$$

   The total delay is the sum of the communication delay, the service delay, and the update delay for each access from an *i-user* multiplied by the average access rate of *i-users*. Here, we do not use λ_i but λ_i^* as the mean arrival rate. When we consider service in each server, we should consider not only λ_i accesses on average from local users but also accesses from users connected with other servers. The server can be accessed by some of them for their request file which is not stored in their local server. Thus, we formulate λ_i^*.

$$\lambda_i^* = \lambda_i + \sum_{j'} \delta\left(j - i\right) \sum_k \left(\mathbf{Xf}_{j'i} \lambda_{j'} P_{j'k}\right) \tag{3.15}$$

   Here, it is assumed that j equals to the value that minimize the following function $h2_{\min}(j)$ on the positive side $(1 \leq j \leq s)$.

$$h2_{\min}(j) = \mathbf{Xf}_{j'j} \left(\mathbf{Cy}_{j'j} + \mathsf{Csa}_j\right) \tag{3.16}$$

$$\delta(t) = \begin{cases} 1 & (t = 0) \\ 0 & (t \neq 0) \end{cases} \tag{3.17}$$

[Communication Delay]

The communication delay is spent to communicate from a user to a file per an access. Let Dtl_i denote as the local communication delay coefficient between an *i-user* and a local server S_i. Let Dtr_{in} denote the remote communication delay coefficient between a server S_i connected to a line of the class n and the communication network. We have the communication delay Dt_{ik} in an access from an *i-user* for a file M_k as follows. Here, it is assumed that index j equals to the value that minimize the function $h1_{\min}(j)$ (eq.(3.8)) on the positive side, too $(1 \leq j \leq s)$.

$$Dt_{ik} = (X_{ik}\mathsf{Dtl}_i + \mathbf{Xf}_{ij}\mathbf{Dy}_{ij})\, L_{ik}P_{ik} \tag{3.18}$$

Also, we define $\mathbf{Dy}_{ij}$ as follows.

$$\mathbf{Dy}_{ij} = \mathsf{Dtl}_i + \sum_n \mathsf{Dtr}_{in}Y_{in} + \sum_n \mathsf{Dtr}_{jn}Y_{jn} \tag{3.19}$$

[Service Delay]

The service delay is the delay of a service to a file in a server. We may assume this delay behavior as M/G/1, and here we take M/D/1 for analytical simplicity. We can introduce the service delay Dx_i in a server S_i by the *Pollaczek-Khinchin mean-value formula* and *Little's result* [10]. A service in a server S_i is assumed to take the service time x_i on average.

$$Dx_i = x_i + \frac{\lambda_i^* x_i^2}{2\,(1 - \lambda_i^* x_i)} \tag{3.20}$$

Here, we do not use the flow λ_i but λ_i^* as the mean arrival rate.

[Update Delay]

The update delay is spent to keep the data consistency when the same file is allocated in more than or equal to two servers. This update delay is evaluated per an updating from a user to a file, and includes a delay required to communication in updating. In this study, we apply the write-through consistency protocol, in which all of the same files in a system should be updated simultaneously when one of them is rewritten. The update delay coefficient is denoted by Dw. We have the update delay Dw_i in updating from *i-users*.

$$Dw_i = \sum_k \mathsf{Dw} RDnf \left(W_k - 1 + \delta\left(W_k\right)\right) P_{ik} Pw_{ik} \tag{3.21}$$

Here, $RDnf$ denotes the local-global difference of the communication cost and we will mention it later. Also, W_k is the number of the same files in the whole system, and defined as follows.

$$W_k = \sum_i X_{ik} \tag{3.22}$$

2. Facility Cost

The facility cost is required to invest in setting up servers and lines initially. It depends on a capacity of a server and the class of lines. Let $Cf_{\max}$ be defined as the maximum facility cost. Here, Cf_n denotes the facility cost coefficient of a line of class n, and Cb denotes the capacity cost coefficient. The facility cost is as follows.

$$Cf = \sum_i \left(\sum_n \mathsf{Cf}_n Y_{in} + \mathsf{Cb} B_i\right) \leq Cf_{\max} \tag{3.23}$$

3.3 Optimization and Algorithm

In solving optimization problems, it is very significant which algorithm we use. Using the exhaustive search, we may solve only unrealistic small models and have to take a long time to solve even the small problems. In this study, in order to apply our optimal file allocation model to practical large-scale systems, we mainly use the branch and bound method and an approximate method.

3.3.1 Optimization and Algorithm

Developing efficient algorithm is the most important issue in solving mathematical programming problems. Mathematical programming models tend to be of considerable size. A lot of models have hundreds or thousands of variables and constraints, some even have tens of thousands. In the study of our optimal file allocation model, the number of variables increases explosively according to the scale of the systems. For instance, in the case of the system in which the number of servers $s = 5$, the number of distinct line classes $u = 1$, the number of distinct files $m = 10$, the number of variables is 55, while in the case of the system in which the number of servers $s = 10$, the number of distinct line classes $u = 1$, the number of distinct files $m = 1000$, we have 10010 variables. Therefore, we need to use as efficient an algorithm as possible when we solve this kind of large-scale mathematical programming models.

In general, the computational complexity of an algorithm is determined by the total number of basic operations that is executed until algorithm ends, such as addition, subtraction, multiplication, division, size comparison, etc. Naturally, computational complexity is dependent on the size of the problems. However, some problems can be solved easier than other problems even if they are of the same size; and conversely, there are some problems that need more time to solve them. Therefore, when we evaluate theoretically computational complexity, if we determine the size of the problems, we consider how many times we need to execute operations in the worst case in order to solve arbitrary problems of that size. When we are certain that we can solve problems of size N with $f(N)$ numbers of operations, the computational complexity of the algorithm can be calculated to be $O(f(N))$ (order $f(N)$). In $f(N)$, we consider only the term of the highest degree of the variable N and ignore its coefficient (constant). This is because when the scale of problems N is large enough, the computational complexity is dominated by the term of the highest degree.

When the computational complexity of a certain algorithm is given by $O(f(N))$ and $f(N)$ is represented as a polynomial function such as N^2 and N^5+N^2 (including $\log n$), the algorithm is considered a polynomial time algorithm. In contrast, when $f(N)$ cannot be represented as a polynomial function but can be represented as an exponential function such as 2^N and $N!$, the algorithm is considered an exponential time algorithm. Although there are problems that take even more time to solve, most of the problems that are currently dealt with practically are the problems that can be solved by polynomial or exponential time algorithms. In a polynomial time algorithm, the computational complexity increases more moderately than in exponential algorithm as the scale of problems becomes larger. Therefore, it would be efficient even for large-scale problems. Conversely, in an exponential time algorithm, the computational complexity increases explosively when the scale of problems becomes even a little larger, and thus, the problems would not solvable practically.

In general, the problems that require polynomial time algorithms to solve them belong to the class of P problems, and are regarded as the problems that are relatively easy to solve. On the other hand, there are a number of problems that belong to the class of NP hard problems in which it is considered that no polynomial time algorithm exit to solve them. These problems are regarded as the difficult problems that cannot be solved easily.

The optimal file allocation model we proposed in this study belongs to the class of NP hard problems, and thus, it is very difficult to develop an efficient algorithm to solve it. However, if the scale of problems which is meaningful practically is not so large, there is still the possibility that we can manage it, even though it belongs to the class of NP hard problems. The branch and bound method is useful for this purpose. It is also useful to apply an approximate method to achieve our purpose, if its accuracy is good enough.

In this study, we use the exhaustive search and the branch and bound method as the exact methods and in order to deal with more practical large-scale systems, mainly use the approximate method in numerical calculations. In the following, we describe algorithm of the exact methods and the approximate method in detail.

3.3.2 Exact Method

Here, we present the exact methods that we use when we need to obtain exact solutions in this study, that is, the exhaustive search and the branch and bound method.

Exhaustive Search

The exhaustive search calculate values of the objective function for all solutions that satisfy restrictive conditions, compare all those values, and determine the optimal solution. That is, the exhaustive search in this study calculate the total costs for all file allocation that satisfy restrictive conditions, compare all those total costs, and determine the optimal file allocation that minimize the total cost. As mentioned above, it is very difficult to find efficient algorithm since our optimal file allocation model belongs to the class of NP hard problems. Therefore, first, we use the exhaustive search in numerical calculations. However, the exhaustive search has the following two serious shortcomings.

1. The scale of problems that can be solved is limited to small one.

2. It takes enormous time to solve even small-scale problems.

In our model, since the number of variables increases according to scale of systems, the largest scale of systems that we can solve is the number of servers $s = 4$, the number of distinct line classes $u = 1$, and the number of distinct files $m = 6$ (in this case, the number of variables is 28). Even this kind of scale of systems have to spend enormous time to obtain the optimal solution. This is why we also use the branch and bound as another exact method in order to deal with more practical scale of systems.

Branch and Bound Method

The branch and bound [8][9] is one of the best-known algorithm among a number of algorithms that have been proposed for the integer linear programming problem. It is an algorithm which finds the optimal solution by keeping the best solution found so far. If a partial solution cannot do better than the best, work on it is abandoned. The branch and bound has proven to be reasonably efficient on practical problems, and it has the added advantage that it solves continuous linear programs as subproblems, that is, linear programming problems without integer restrictions (0-1 restrictions in this study).

The optimal solution of a continuous linear relaxation problem is not the optimal solution of a original problem. However, the optimal solution of a continuous linear relaxation problem give us a lower bound (an upper bound if the problem is a maximization problem) of the original optimal solution. This nature of continuous linear relaxation problems plays an important role in the branch and bound.

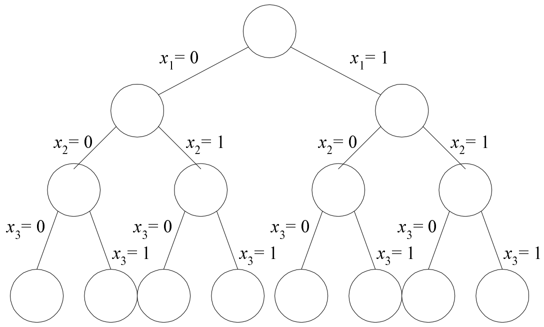


Fig.3.2. Example of an enumeration tree (the number of variables is 3)

In the case of 0-1 integer programming problems, the optimal solution of the problem that will be solved is one of 2^n solutions that is obtained by fixing each variable as 0 or 1. To enumerate all those solutions, we use an enumeration tree. We show an example of an enumeration tree in the case the number of variables is 3 in Fig. 3.2. In Fig. 3.2, the root node of the tree corresponds to the state in which no variables are not fixed yet, and the leaf nodes of the tree corresponds to the states in which all variables are already fixed as 0 or 1. The other nodes indicates the problems that some variables are already fixed and the others are not yet. Those problems is called subproblems.

In the branch and bound, instead of enumerating all subproblems, we examine subproblems generated gradually in whether there is possibility that solutions of subproblems give the optimal solution of the original problem. If it turns out that there is no possibility, we abandon solving the subproblem. Besides, in the case that we can get the optimal solution of subproblems in which all variables are 0 or 1, and that it turns out subproblems do not have feasible solutions, we can abandon solving those subproblems. This is exactly a bounding operation. Once it is abandoned, we don't have to investigate subproblems that is supposed to be generated from the subproblem abandoned, and thus, the number of subproblems generated decreases. This is the way we can shorten the calculation time in the branch and bound by limiting the search space. Also, for subproblems that we cannot abandon, we generate a new subproblem by choosing one of unfixed variables and fixing it to 0 or 1. This is a branching operation.

The essence of the algorithm of the branch and bound is as follows:

1. Solve the continuous linear relaxation of the problem. If the solution is 0-1 integer, then compare the objective value (the total cost) to the objective value of the best solution found so far and make it the new best solution only if the solution gives less value than the best. Otherwise create two new subproblems by branching on a fractional variable.

2. A subproblem is not active when any of the following occurs:

   (a) You used the subproblem to branch on.

   (b) All variables in the solution are 0 - 1 integer.

   (c) The subproblem is infeasible.

   (d) The lower bound of the subproblem is greater than the objective value of the best feasible solution so far.

3. Choose an active subproblem and branch on a fractional variable. Repeat until there are no active subproblems.

The preferred strategy for solving the node problems in the enumeration tree is of the depth-first type: When two child nodes are generated from the current node, we choose one of these two children as the next problem to solve. One reason for using this strategy is that on practical problems the optimal solution usually occurs deep in the tree. There is also a computational advantage: If the dual simplex algorithm is used to solve the linear program at each node, the solution of the child problem can be found by a simple update to the basis matrix factorization obtained at the parent node. The linear algebra costs are trivial.

In this study, we use the branch and bound method with the depth-first search for the numerical calculations. In solving continuous linear relaxation problems, we use the simplex algorithms [8][9][11].

In the following, we give full details of the continuous linear relaxation problem for our optimal file allocation model. What we ought to note here is that our model is a nonlinear 0-1 integer programming problem and thus, we need to relax not only 0-1 integer restriction but also nonlinear restriction. That is, we need to transform the objective function and restrictive conditions into linear functions. Here, it is very important to transform them so that the feasible region of the continuous linear relaxation problem includes the feasible region of the original problem. In our model, in order to relax the delay restriction, we need to make the one in the relaxation problem less than or equal to the delay in the original problem. It is also necessary to make the total cost less than or equal to the one in the original problem. We formulate the continuous linear relaxation problem in view of the above points.

[Continuous linear relaxation problem]

1. Objective function: Total cost C

$$C = \sum_i \left(\lambda_i \sum_k (Ct_{ik} + Crs_{ik}) + Cst_i + Crl_i \right) \tag{3.24}$$

Communication cost Ct_{ik}

$$Ct_{ik} = (X_{ik}\mathsf{Ctl}_i + (1 - X_{ik})(\mathsf{Ctl}_i + \mathsf{Ctr}_{ie} + \mathsf{Ctr}_{je}))\, L_{ik}P_{ik} \tag{3.25}$$

e**:** The line class that minimize the remote communication cost coefficient Ctr_{in} from the same server S_i.

j**:** The index of the server S_j which has the minimum local communication cost coefficient Ctl_j.

Server cost Crs_{ik}

$$\begin{aligned} Crs_{ik} &= (X_{ik}\mathsf{Csa}_i + (1 - X_{ik})\mathsf{Csr}_i + (1 - X_{ik})\,\mathsf{Csa}_g)\, L_{ik}P_{ik} \\ &+ \left(\sum_{l \neq i} (X_{lk}\mathsf{Csw}_l) \right) RCnfP_{ik}Pw_{ik} \end{aligned} \tag{3.26}$$

g**:** The index of the server S_g which has the minimum access cost coefficient Csa_g.

Storage cost Cst_i

$$Cst_i = \mathsf{Cst} \sum_k X_{ik}F_k \tag{3.27}$$

Line cost Crl_i

$$Crl_i = \sum_n \mathsf{Crl}_{in}Y_{in} \tag{3.28}$$

2. Restrictive conditions

(1) Initial conditions

$$0 \leq X_{ik} \leq 1, \quad 0 \leq Y_{in} \leq 1 \tag{3.29}$$

$$1 \leq \sum_i X_{ik} \leq r_k \tag{3.30}$$

$$\sum_k X_{ik} F_k \leq B_i \tag{3.31}$$

$$\sum_n Y_{in} = 1 \tag{3.32}$$

(2) Total delay

$$D = \sum_i \left(\lambda_i \left(\sum_k Dt_{ik} + Dw_i + Dx_i \right) \right) \leq D_{\max} \tag{3.33}$$

Communication delay Dt_{ik}

$$Dt_{ik} = \left(X_{ik} \mathsf{Dtl}_i + (1 - X_{ik}) \left(\mathsf{Dtl}_i + \mathsf{Dtr}_{ie} + \mathsf{Dtr}_{je} \right) \right) L_{ik} P_{ik} \tag{3.34}$$

e**:** The line class that minimize the remote communication delay coefficient Dtr_{in} from the same server S_i.

j**:** The index of the server S_j which has the minimum local communication delay coefficient Dtl_j.

Update delay Dw_i

$$Dw_i = \sum_k \mathsf{Dw} RDnf \left(W_k - 1 \right) P_{ik} Pw_{ik} \tag{3.35}$$

Service delay Dx_i

$$Dx_i = x_i + \frac{\lambda_i x_i^2}{2 \left(1 - \lambda_i x_i \right)} \tag{3.36}$$

(3) Facility cost

$$Cf = \sum_i \left(\sum_n \mathsf{Cf}_n Y_{in} + \mathsf{Cb} B_i \right) \leq Cf_{\max} \tag{3.37}$$

3.3.3 Approximate Method

In the case of difficult problems that belongs to the class of NP hard problems, the scale of the problems which is able to be solved practically is limited. For the purpose of dealing with more practical larg-scale systems, we often apply approximate algorithms. Approximate algorithms include a greedy method, a stingy method, a random method, a near search etc. In this study, we apply the greedy method, which is the most typical algorithm, to our optimal file allocation model

The greedy method finds feasible solutions directly fixing each variable on the basis of local critical values for contribution to an objective function. Here, it is considered that when determining the optimal allocation in our model, the traffic pattern from each user to each file would be an important factor. For instance, we can easily imagine that in VoD Systems, more popular files should be distributed to more servers in order to minimize the total cost. In this study, we take the access frequency from a server S_i to a file M_{ik}, that is $\lambda_i \times P_{ik}$, as the local critical values, which is denoted by Af_{ik} hereafter. We fix the variables $X_{ik} = 1$ one by one in the order of the value of the access frequency Af_{ik}, so long as all restrictive conditions are satisfied. Eventually, we consider the file allocation that has the minimum cost among all allocations obtained in this way as an approximate solution. In the following, we describe this algorithm using the greedy method.

[**Approximate Algorithm**]

(1) Initialize the value of the objective function, the total cost $C = \infty$, and the evaluation variable of the relay restriction $FLAG = 0$. Select the lowest class of lines.

(2) Calculate the access frequency Af_{ik} from each server to each file.

(3) For each file, seek the variable X_{ik} which is corresponding to the largest access frequency Af_{ik} among ones with the same index k, and fix the variable $X_{ik} = 1$. This procedure makes the restrictive condition $1 \leq \sum_i X_{ik}$, that is there is each kind of files in at least one server, satisfied. Then, set the unfixed variables to 0.

(4) Sort the variables X_{ik}, except the variables which are fixed in the previous procedure, in the decreasing order of corresponding Af_{ik}.

(5) Repeat the procedures from 5.1 to 5.4 until the last variable of the sorted sequence in the previous procedure.

5.1 Fix a variable $X_{ik} = 1$ in a sorted order.

5.2 Check whether the rest of initial conditions are satisfied or not. If they are not satisfied, return the value to 0, and go back to the procedure 5.1.

5.3 Check whether the delay restrictive condition are satisfied or not. If it is not, update $FLAG = 1$ and go back to the procedure 5.1.

5.4 Calculate the total cost for the current file allocation determined in the previous procedures, and commit the file allocation with the total cost to the memory. If the total cost is less than C, update C so that we can keep the minimum value.

(6) If $FLAG = 1$, change the line classes so that the delay restriction is satisfied and return $FLAG$ to 0. Repeat the procedures from (3) to (5).

(7) Select the file allocation such that the total cost is minimized of all file allocation calculated in the procedure 5.4, and consider it as the approximate solution.

3.3.4 Performance Evaluation of Approximate Method

We show the performance of the approximate method by the comparison results of the approximate method and the two exact methods, the exhaustive search and the branch and bound method.

Cost Comparison

First, we compare the approximate method to the exact method in terms of the total cost and investigate how accurate solutions the approximate method can get. Here, we use the branch and bound as the exact method in order to compare in as large systems as possible.

We define the relative error as the performance measure of the approximate method.

$$\text{Relative Error}(\%) = \frac{\text{approximate total cost} - \text{optimal total cost}}{\text{optimal total cost}} \times 100 \qquad (3.38)$$

In numerical calculations, we consider the system parameters as follows. There exist 5 servers ($s = 5$), and the capacity of each server is 30[Gbyte] ($B_i = 30000, 1 \leq i \leq 5$). There is 1 class of lines ($u = 1$), which is used between servers and the communication network. Also, there exist 15 kinds of files ($m = 15$), and the size of each file is 600[Mbyte] ($F_k = 600, 1 \leq k \leq 15$). The maximum number of copies of a file M_k is 10 ($r_k = 10$).

In this approximate method, it is considered that its behavior is greatly dependent on the user access pattern, since we use the access frequency from each server to each file ($\lambda_i \times P_{ik}$) as the local critical values, as mentioned previously. Therefore, we assume the following 8 types of traffic models and make a comparison between the approximate method and the exact method for each of them.

(A) Mean access rate λ_i from each server is uniform and

- (A-a) access probability P_{ik} is uniform.
- (A-b) access probability P_{ik} is random.
- (A-c) access probability P_{ik} serves as a model for corporate intranets.
- (A-d) access probability P_{ik} servers as a model for VoD.

(B) Mean access rate λ_i from each server is ununiform and

- (B-a) access probability P_{ik} is uniform.
- (B-b) access probability P_{ik} is random.
- (B-c) access probability P_{ik} serves as a model for corporate intranets.
- (B-d) access probability P_{ik} serves as a model for VoD.

Here, in the traffic model A, in which the mean access rate λ_i is uniform, we set $\lambda_i = 200$ regardless of which server. On the other hand, in the traffic model B, in which the mean access rate λ_i is ununiform, we set $\lambda_1 = 100, \lambda_2 = 150, \lambda_3 = 200, \lambda_4 = 250, \lambda_5 = 300$. Also, in the model (A-a) and (B-a), all values of the access probabilities are set as $P_{ik} = 0.0666\ldots$ and we set the values randomly in the model(A-b) and (B-b). In the model(A-c) and (B-c), in which the access probability P_{ik} serves as a model for corporate intranets, we consider that there is the locality in file accesses of users connected with each server. Here, we assume that *1-users*, the local users of the server S_1, tend to access most frequently the file M_1, M_2, M_3, *2-users* tend to access the file M_4, M_5, M_6, $\ldots$, *5-users* tend to access

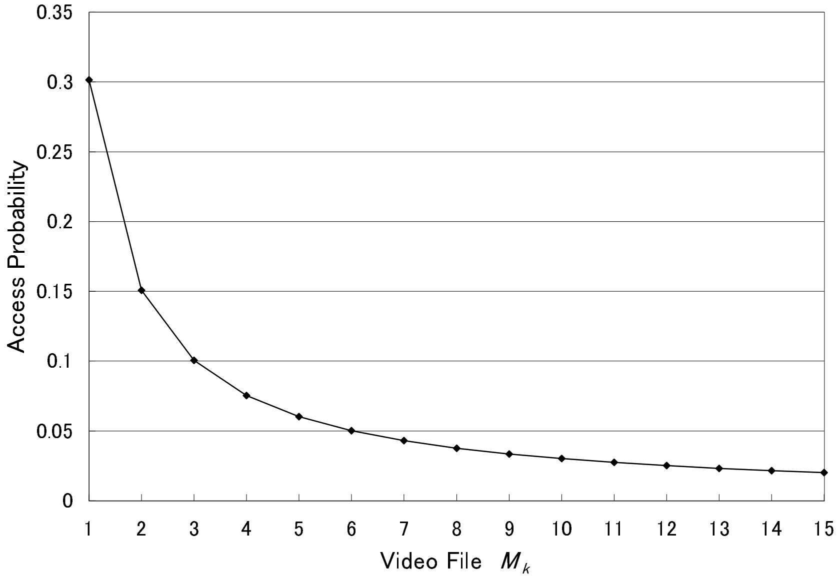


Fig.3.3. Zipf's law (in the case the number of distinct files is 15)

the file M_{13}, M_{14}, M_{15}. The ratio of the access probabilities to the file frequently requested to the access probabilities to the other files is set to approximately 1/60. In the model(A-d) and (B-d), in which the access probability P_{ik} serves as a model for VoD, popularities differ with files. We apply the zipf's law which is shown in Fig. 3.3 to these VoD traffic models. In the case that we apply the zipf's law, the access probability P_{ik} is represented as the next equation using a constant α.

$$P_{ik} = \frac{\alpha}{k} \tag{3.39}$$

In Fig. 3.4 to 3.11, we show the comparison results of the both methods for each traffic model respectively. These methods show the total cost characteristics vs. the local-global differences of the communication cost $RCnf$ for the both case of using the approximate method and the branch and bound method. We also show the average relative error in each figure in Table 3.1.

From each figure and Table 3.1, the average relative error is at most 3.38 % and there are even several models for which the average relative error is 0 %. Namely, we see the approximate solutions by the approximate method is hardly different from

Table 3.1: Relative error of the approximate method for different traffic models (%)

Access Probability P_{ik}	uniform	random	corporate intranets	VoD
λ_i uniform	2.21	0.611	0	3.38
λ_i ununifrom	0	0.247	0	0.385

the exact solutions. In particular, for the traffic models B, in which the mean access rate is ununiform, we can get extremely good results. Even for the traffic models A, the approximate solutions is considered to be useful enough.

From these results, we can conclude that the approximate method is practical enough regardless of which traffic model.

Time Comparison

Next, we compare the approximate method with the two exact methods in the average time it takes to solve the optimization problem and investigate how useful this approximate method is in terms of the calculation time.

In the following tables, the notation which denotes scale of systems means "the number of servers - the number of distinct line classes - the number of distinct files". For example, 4-1-2 means the number of servers $s = 4$, the number of distinct line classes $u = 1$, and the number of distinct files $m = 2$. Here, we used Sun UltraSPARCII$_i$ (334MHz) for the numerical calculations.

In Table 3.2, we show the average time [second] when solving the optimal file allocation problems by the exhaustive search, the branch and bound, and the approximate method for various scale of systems. In the case of using the exhaustive search, the scale of systems that we can solve with practical time is limited to the very small-scale system in which the number of servers $s = 4$, the number of distinct line classes $u = 1$, and the number of distinct of files $m = 6$. This is why we selected the scale of systems shown in Table 3.2. There are remarkable differences between the exhaustive search and the other two algorithms. In the case of using the exhaustive search, the average calculation time is increasing explosively as scale of systems becomes larger, while in the other two cases, the increase in the average calculation time is moderate. In particular, in the case of using the approximate method, there is hardly increase in the average calculation time.

Next, in Table 3.3, we show the average calculation time by the branch and

Table 3.2: Time comparison between three algorithms (sec.)

Scale of Systems	4-1-2	4-1-3	4-1-4	4-1-5	4-1-6
Exhaustive Search	0.88	18.41	360.15	6688.18	121641.4
Branch and Bound Method	0.08	0.20	0.64	1.95	4.87
Approximate Method	0.03	0.05	0.09	0.12	0.16

Table 3.3: Time comparison between the branch and bound and the approximate method (sec.)

Scale of Systems	5-1-5	5-1-10	5-1-15	5-1-20
Branch and Bound Method	0.88	77.66	4427.07	213279.71
Approximate Method	0.20	0.79	1.75	3.08

bound and the approximate method. Here, we make scale of systems relatively large. From this table, we see that even in the case of using the branch and bound, it takes considerable time to solve larger-scale systems and the scale of systems that can be solved with practical time has limitation. In contrast, in the case of using the approximate method, the increase in the calculation time according to change of scale of systems is still extremely moderate.

In summary, compared to the exhaustive search, the branch and bound is capable of dealing with more practical scale of systems with practical time. However, the scale of systems that can be solved is still limited. On the other hand, in the case of using the approximate method, we can significantly decrease the calculation time compared to the exact methods. We can apply this approximate method to considerably large-scale systems.

The both comparison results show that this approximate method is very useful in terms of the accuracy and the calculation time, when considering practical large-scale systems. In next section, using this approximate method, we show numerical results of the optimal file allocation problem for the practical large-scale system models.

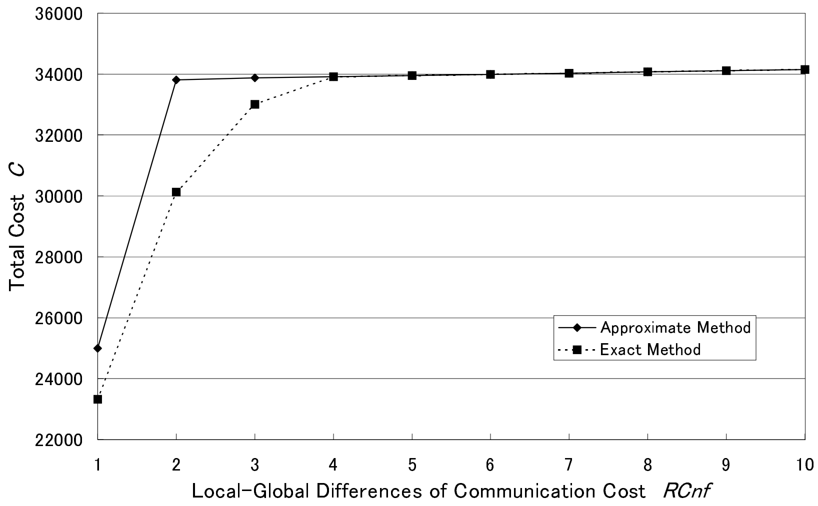


Fig.3.4. Cost comparison between the both algorithms for the traffic model (A-a) ($RDnf = 5.0, \mathsf{Cst} = 0.5$)

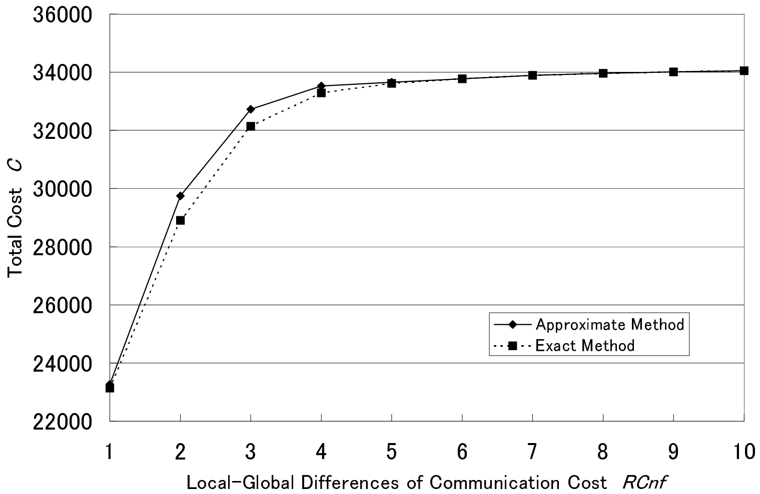


Fig.3.5. Cost comparison between the both algorithms for the traffic model (A-b) ($RDnf = 5.0, \mathsf{Cst} = 0.5$)

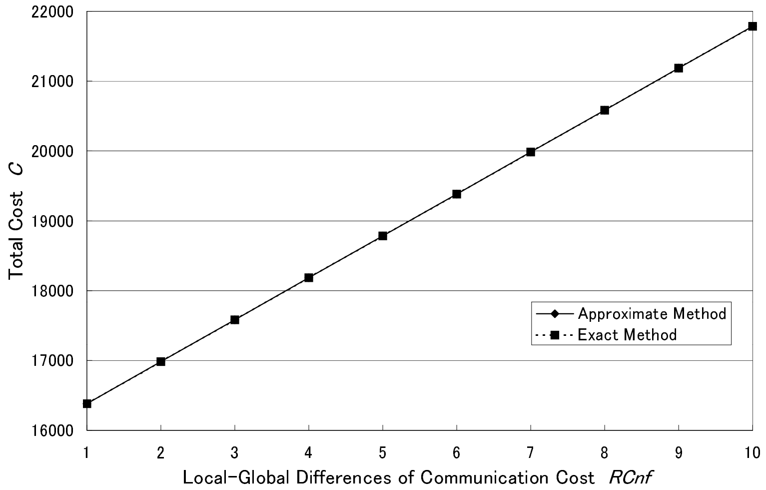


Fig.3.6. Cost comparison between the both algorithms for the traffic model (A-c) ($RDnf = 5.0, \mathsf{Cst} = 0.5$)

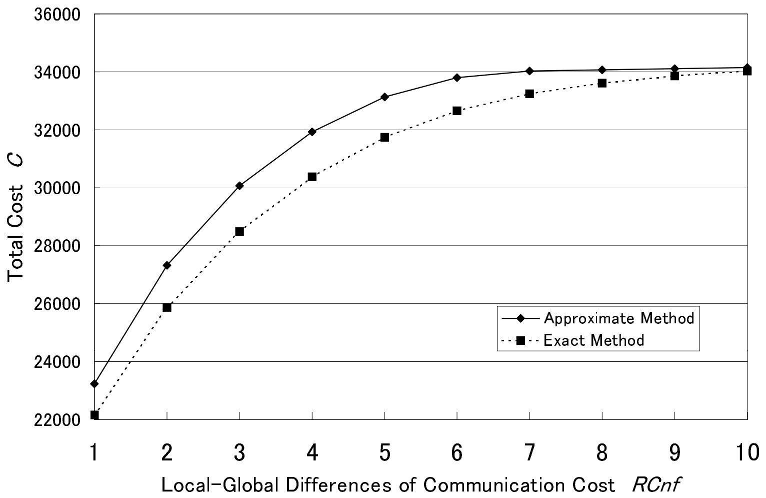


Fig.3.7. Cost comparison between the both algorithms for the traffic model (A-d) ($RDnf = 5.0, \mathsf{Cst} = 0.5$)

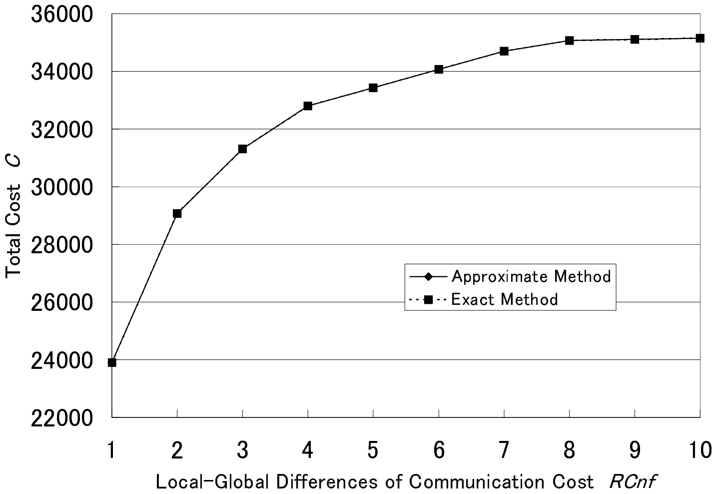


Fig.3.8. Cost comparison between the both algorithms for the traffic model (B-a) ($RDnf = 5.0, \mathsf{Cst} = 0.5$)

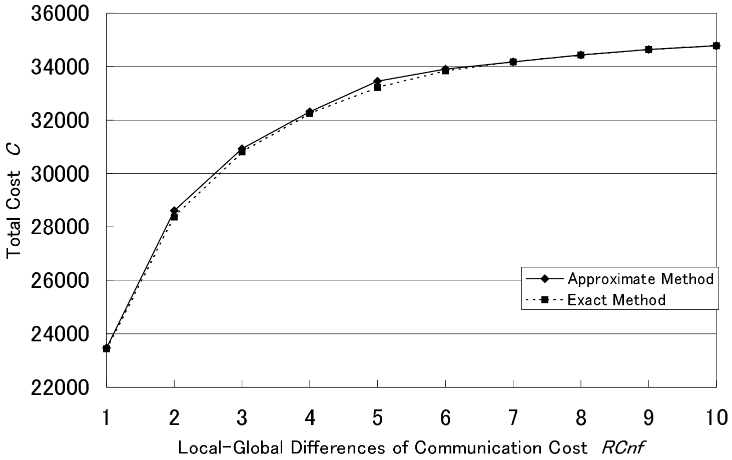


Fig.3.9. Cost comparison between the both algorithms for the traffic model (B-b) ($RDnf = 5.0, \mathsf{Cst} = 0.5$)

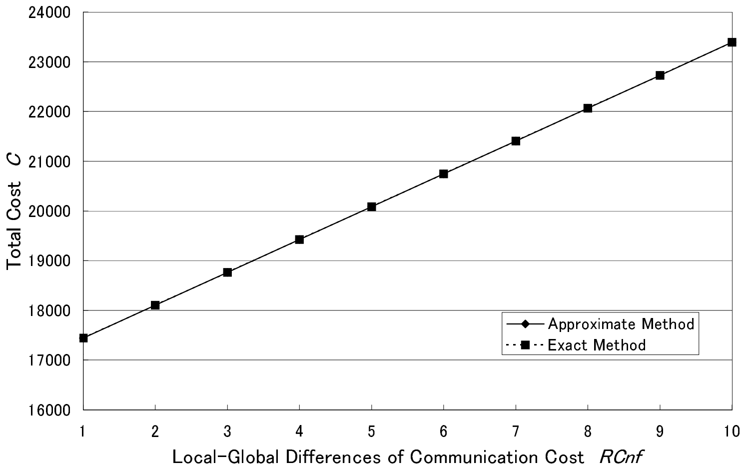


Fig.3.10. Cost comparison between the both algorithms for the traffic model (B-c) ($RDnf = 5.0$, Cst $= 0.5$)

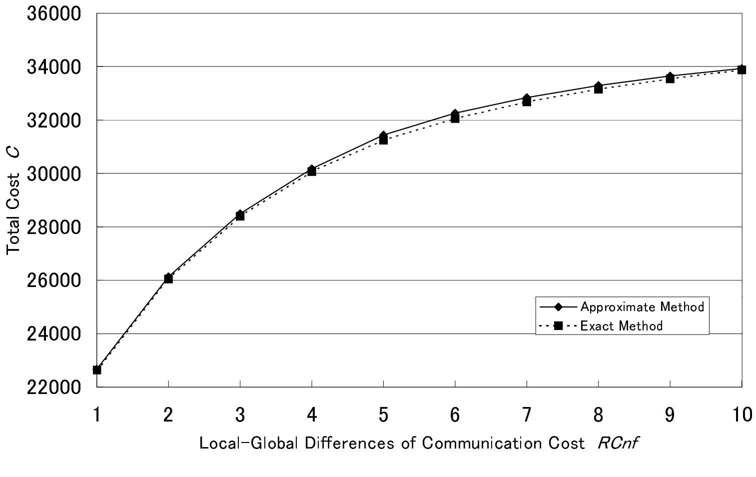


Fig.3.11. Cost comparison between the both algorithms for the traffic model (B-d) ($RDnf = 5.0$, Cst $= 0.5$)

3.4 General Characteristics on Distributed File Allocation

Here, we show the optimization results using the optimal file allocation model. We use the approximate method presented in the previous section in solving the optimization problem.

3.4.1 System Parameters for Numerical Results

We consider the system parameters for numerical calculations as follows. There exist 10 servers ($s = 10$), and the capacity of each server is 100[Gbyte] ($B_i = 100000, 1 \leq i \leq 10$). There is 1 class of lines ($u = 1$), which is used between servers and the communication network. This means we do not consider the line selection here. Also, there exist 1000 kinds of files ($m = 1000$), and the size of each file is 100[Mbyte] ($F_k = 100, 1 \leq k \leq 1000$). The maximum number of copies of a file M_k is less than 10 ($r_k = 10$). In the traffic model in section 3.4.2, 3.4.3 and 3.4.4, the mean access rate λ_i is ununiform and varies from 50 to 500, and the access probability P_{ik} takes random values. In this section, for analytical simplicity, we consider the capacity of servers, the size of files, etc. are identical. However, we may deal with many kinds of multimedia networks by controlling these system parameters in our model.

In considering the optimization results, we define the distribution rate Rd for performance measure as follows.

$$Rd = \frac{\sum_i \sum_k X_{ik}}{m} \tag{3.40}$$

Rd is the ratio of the number of all files allocated in the system to the number of distinct files.

We obtain the concrete assignment of files to servers as the optimal solution. On the basis of this assignment, we calculate distribution rates, the total cost, the total delay, and so on. We describe our considerations about optimization results according to these criterions.

Moreover, we define the other parameters that determine the economic environments of the distributed systems as follows.

$RCnf$ The local-global difference of the communication cost. This parameter means the difference between the local communication cost from a user to its local server and the global communication cost from a local server to a remote server which stores a requested file. The larger a local-global difference $RCnf$ is, the more a difference of the communication cost between to a local server and to remote servers is.

$RDnf$ The local-global difference of the communication delay. This parameter means the difference between the local communication delay from a user to its local server and the global communication delay from a local server to a remote server which stores a requested file.

C_rate_n The cost rate of nth line class. This parameter means the ratio of the communication cost and the line cost, which is dependent on line classes.

v_rate_n The speed rate of nth line class. This parameter means the ratio of the communication delay, which is dependent on line classes.

Using the above parameters, we define the coefficients which is introduced in section of 3.2.

- The remote communication cost coefficient between a server S_i connected to a line of the class n and the communication network Ctr_{in}

$$\mathsf{Ctr}_{in} = RCnf \cdot C\_rate_n \mathsf{Ctl}_i \tag{3.41}$$

- The line cost coefficient of the line of the class n between a server S_i and the communication network Crl_{in}

$$\mathsf{Crl}_{in} = C\_rate_n \cdot \mathsf{Cl}_i \tag{3.42}$$

  Here, let Cl_i be the line cost coefficient between a server S_i and the communication network.

- The remote communication delay coefficient between a server S_i connected to a line of the class n and the communication network Dtr_{in}

$$\mathsf{Dtr}_{in} = RDnf \cdot \mathsf{Dtl}_i / v\_rate_n \tag{3.43}$$

3.4.2 Trade-off between the Communication Cost and the Storage Cost

In Fig.3.12, we show a trade-off between the storage cost and the communication cost. This figure shows the distribution characteristics vs. the local-global differences of the communication cost $RCnf$ for different values of the storage cost coefficient Cst, i.e., 0.01, 0.05, 0.1, 0.3, 0.5, 1.0. As mentioned before, Rd is the ratio of the number of all files allocated in the system to the number of distinct files. That is, Rd means that there exists the same kinds of files at Rd servers on the average in the whole system. As shown in Fig.3.12, when the storage cost coefficient Cst is larger, the distribution rates Rd become smaller. When the storage costs are small and the communication costs are larger than the storage costs, files should be distributed in many servers to minimize the total cost. However, when the storage costs are very large, files should be centralized to minimize the total cost.

Moreover, as you can see in Fig.3.12, when $RCnf$ becomes larger, more files tend to be distributed. As $RCnf$ grows larger, the communication costs to remote

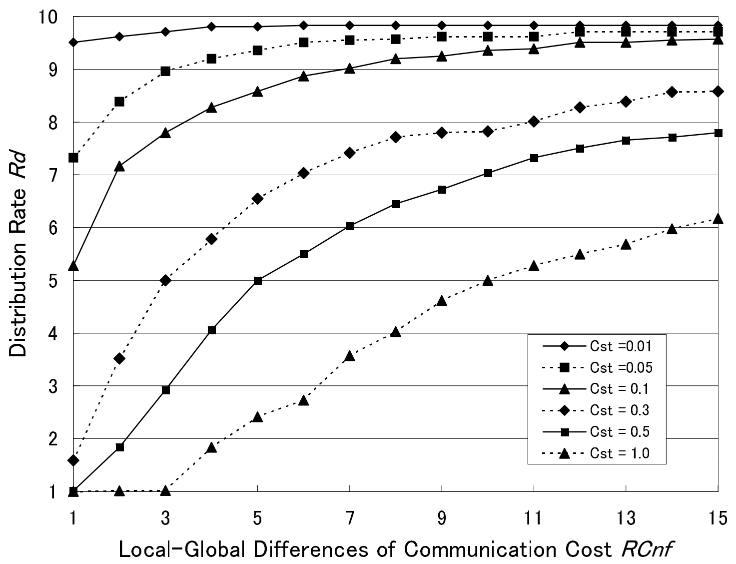


Fig.3.12. A trade-off between the storage cost and the communication cost $(RDnf = 5.0, Pw = 0.1)$

servers increase. Thus, files become to be distributed and the distribution rates become increasing, so that files will be declined necessity to communicate and the communication costs will be reduced. This tendency is also generally evident.

3.4.3 Characteristics on File Updating

Next, we show the effect of file updating. In Fig. 3.13, we show the distribution characteristics for different values of the update probabilities Pw_{ik} i.e., 0, 0.1, 0.3, 0.5. As shown in this figure, the distribution rates Rd take much larger values as Pw_{ik} is small. In the case of $Pw_{ik} = 0$, file updating does not need to be considered such as VoD. The update costs required to update at remote servers become larger as Pw_{ik} has larger values. To reduce the update costs, files tend to be centralized at servers that have more accesses. In this paper, we apply the write-through consistency protocol to the updating protocol as described before. We can guess that this is the reason for the increasing in the update costs. When the update

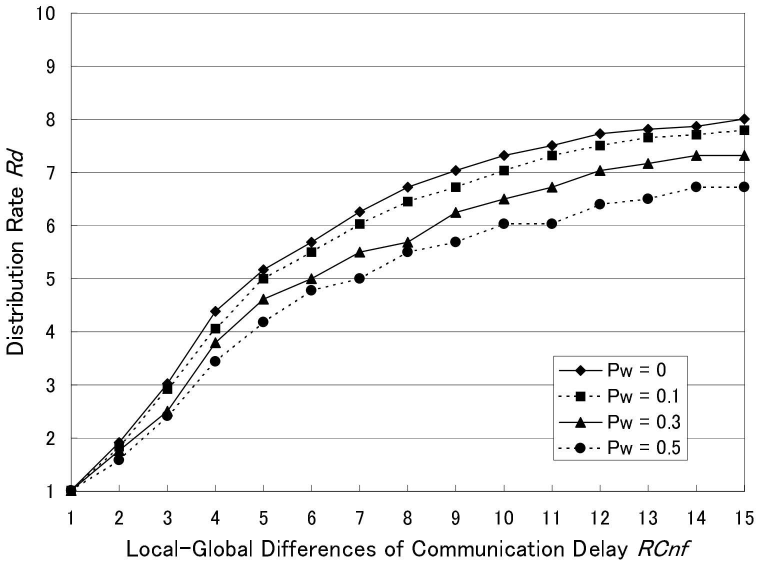


Fig.3.13. A distribution characteristics about $RCnf$ for different values of Pw ($RDnf = 5.0, \mathsf{Cst} = 0.2$)

probabilities are large, we should consider other protocols.

3.4.4 Characteristics on Delay Restriction

We show the effect of the delay restriction on the optimization of file allocation. In Fig.3.14, we show the characteristics of the total delay vs. the local-global differences of the communication delay $RDnf$ for different values of the maximum total delay D_{max}, i.e., 3000, 4000, 5000, 6000, 7000. We see the total delay are almost not affected by the restriction when $D_{\text{max}} = 7000$. However, when $D_{\text{max}} = 3000, 4000, 5000, 6000$, the restriction becomes to have influence on the total delay in the case that $RDnf$ becomes larger. In this case, the total delay in the optimal file allocation that minimize the total cost may exceed the maximum (D_{max}). We show this effect of the restriction on the file allocation in Fig.3.15. In the case $RDnf$ is larger value, files are more distributed to reduce the communication delays. This figure shows that files should be distributed, when the maximum total delay is small.

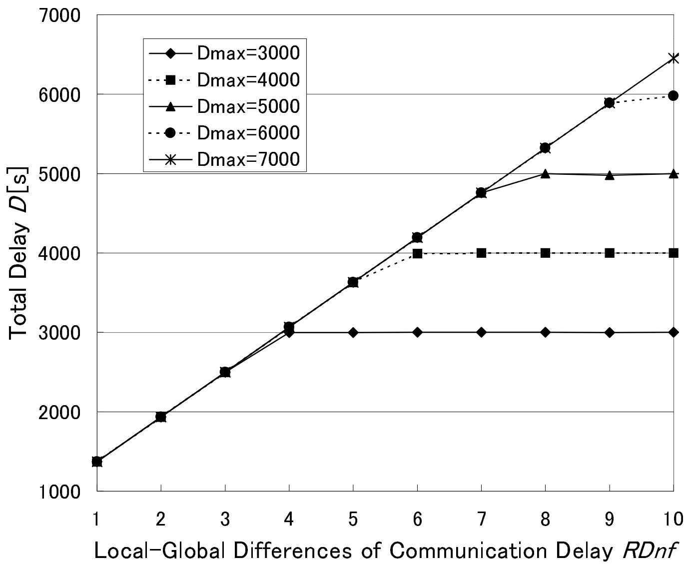


Fig.3.14. A delay characteristics about $RDnf$ ($RCnf = 3.0, Pw = 0.1, \mathsf{Cst} = 0.5$)

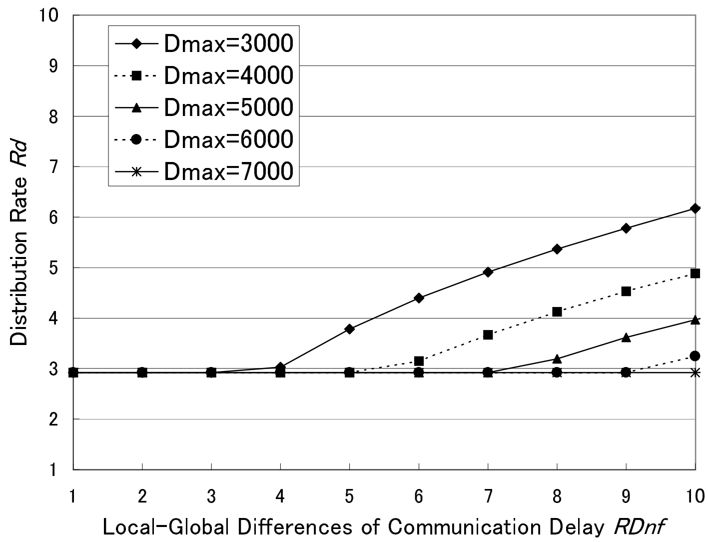


Fig.3.15. A distribution characteristics about $RDnf$ ($RCnf = 3.0, Pw = 0.1, \mathsf{Cst} = 0.5$)

3.4.5 Characteristics on Traffic Matrix

It is considered that the general characteristics that we have shown so far are dependent on traffic matrices from users to files. Therefore, we show the distribution characteristics of files about different traffic matrices.

we assume that the mean access rate λ_i is ununiform and varies from 50 to 500 and the access probability P_{ik} from an *i-users* to a file M_k takes the following 4 traffic models similarly to the section 3.3.4.

(a) Access probability P_{ik} is uniform.

(b) Access probability P_{ik} is random.

(c) Access probability P_{ik} serves as a model for corporate intranets.

(d) Access probability P_{ik} serves as a model for VoD.

We set a value of P_{ik} to 0.001 in the model(a) and select the values randomly in the model(b). In the model(c), we assume that *1-users*, the local users of the server

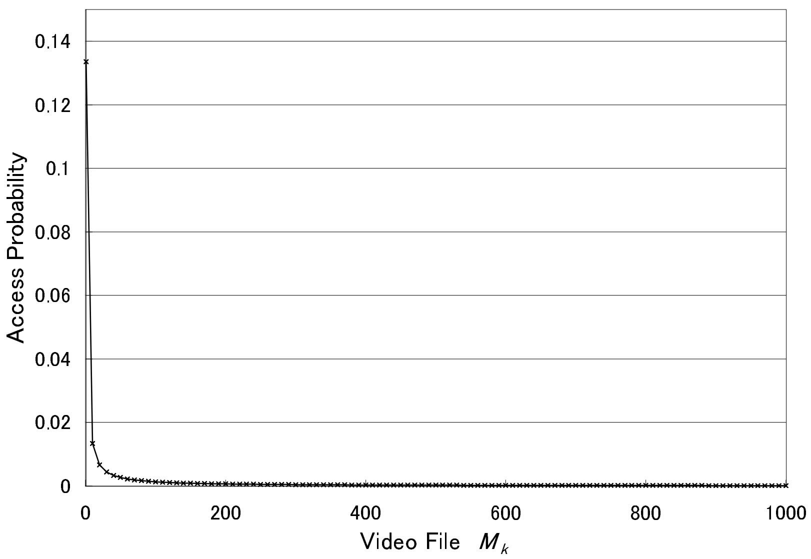


Fig.3.16. Zipf's law (in the case the number of distinct files is 1000)

S_1, tend to access most frequently the file $M_1, \ldots, M_{100}$, *2-users* tend to access the file $M_{101}, \ldots, M_{200}$, $\ldots$, *10-users* tend to access the file $M_{901}, \ldots, M_{1000}$. The ratio of the access probabilities to the file frequently requested to the access probabilities to the other files is set to approximately 1/1000. In the model(d), we apply the zipf's law and we indicate the distribution of the zipf's law in this case in Fig. 3.16. In addition, in order to make these traffic models as practical as possible, we set the values of the update probabilities Pw_{ik} to 0.3 in the model(c), which corresponds to corporate intranets, 0 in the model(d), which is corresponds to VoD systems, and 0.1 in the other models. In the model(d), we also set the access size L_{ik} to larger values than the other models.

Fig. 3.17 shows the distribution characteristics of files about the different traffic models. In this figure, we show the distribution rates Rd vs. the local-global differences of the communication cost $RCnf$ for the above traffic models. As you may see in this figure, it is obvious that there are significant differences in the ways files are distributed among these traffic models. In the model(a) and (c), since there are many files that have the same access probabilities, the change of the file allocation according to the change of the local-global differences is drastic. On the other hand,

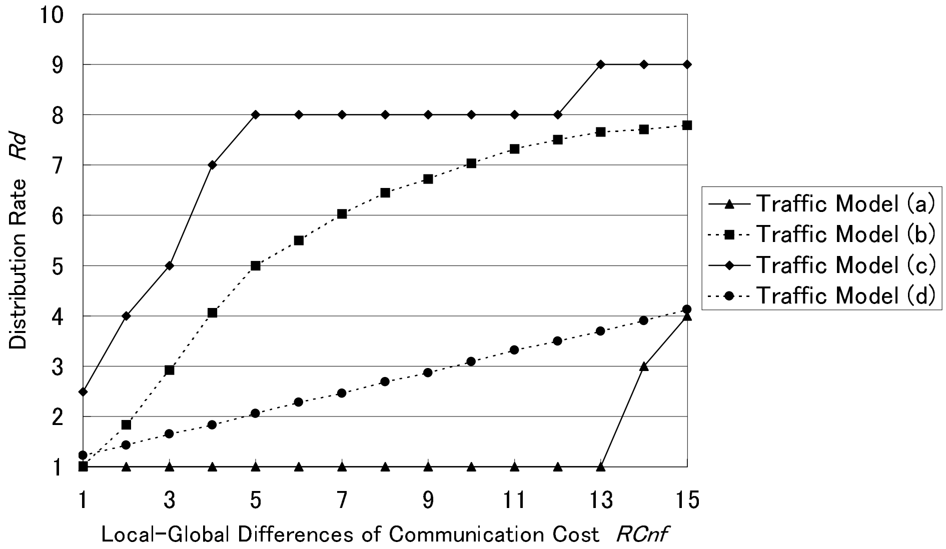


Fig.3.17. Distribution characteristics about different traffic models ($RDnf = 5.0, \mathsf{Cst} = 0.5$)

in the model(b) and (d), the file allocation changes gently.

There are also remarkable differences in the values of the distribution rates Rd among the different traffic models. In particular, files are more distributed in the model(b) despite the higher update probabilities, while in the model(d), files are not distributed as much as the model(b) and (c) though the update probability is 0. We may guess this is because in this model, due to the characteristics of VoD, most of the files which is not that popular do not tend to be distributed, while several kinds of files which is more popular than the others are more likely to be distributed. Because of the different environmental settings that are required for different types of traffic models, it might be difficult to make consistent comparisons between them in terms of the values of the distribution rates.

It is more remarkable that in any traffic models, there are the same characteristics that more files are distributed to more servers as the local-global differences of the communication costs is larger and the remote communication cost becomes larger. Therefore, it is considered to be possible to evaluate the general characteristics of distributed networks with different traffic models, similarly to what we have shown previously. However, when we evaluate it, we will need to fully understand the behavior of each traffic model and set the environmental parameters accordingly.

Finally, in order to demonstrate more clearly the differences in the ways files are distributed among different traffic models, we show the concrete file assignment that we can obtain as the optimization results for the 2 traffic models which assume corporate intranets and VoD networks. Here, the system is limited to a small scale and we use the branch and bound method for the numerical calculation. Also, we consider the system parameters as follows. There exist 5 servers ($s = 5$), and the capacity of each server is 30[Gbyte] ($B_i = 30000, 1 \leq i \leq 5$). There is 1 class of lines ($u = 1$), which is used between servers and the communication network. Also, there exist 10 kinds of files ($m = 10$), and the size of each file is 100[Mbyte] ($F_k = 100, 1 \leq k \leq 10$). The maximum number of copies of a file M_k is 5 ($r_k = 5$). The mean access rate λ_i is ununiform and we set $\lambda_1 = 100, \lambda_2 = 150, \lambda_3 = 200, \lambda_4 = 250, \lambda_5 = 300$. In the VoD traffic model, we apply the zipf's law again. The file M_1 is the most popular and the smaller index of the files, the more popular. In the corporate intranets traffic model, we assume that *1-users* tend to access most frequently the file M_1, M_2, *2-users* tend to access the file $M_3, M_4, \ldots$, *5-users* tend to access the file $M_9, \ldots, M_{10}$. We have 2 cases in which the ratio of the access probabilities to the file frequently requested to the access probabilities to the other files is set to approximately 1/40 and 1/200 respectively.

We show the optimal file allocation for the VoD traffic model in Fig. 3.18, 3.19, 3.20. These figure show respectively the file allocation in the case that the local-global differences $RCnf$ are equal to 1, 5, and 10. In VoD systems, as $RCnf$ becomes larger, the files get allocated to more servers in the order of their popularity, thus increasing their distribution rates. Eventually, all files are distributed to all servers when $RCnf$ is equal to 10. Also, in terms of servers, the server S_5 is most likely to store more files, since its mean access rate is the largest. This is also true for the corporate intranets traffic model as shown in the following.

In Fig. 3.21, 3.22, 3.23. we show the optimal file allocation for the corporate intranets traffic model(I), in which the ratio of the access probabilities is 1/40. Likewise, Fig. 3.24, 3.25, 3.26, show the case of the corporate intranets traffic model(II), in which the ratio is 1/200. We assume again that the local-global differences $RCnf$ are equal to 1, 5, and 10 in the respective figures. In the both cases, as $RCnf$ becomes larger, the servers that have more frequent accesses start to store more files. Also, compared to the case that the ratio is 1/40, files are less distributed in the case that the ratio is 1/200. It is considered that because the locality of file accesses will become more significant, the servers that are accessed less are less likely to store files. In the case of the corporate intranets traffic model, this kind of differences in the locality have influence on the file allocation.

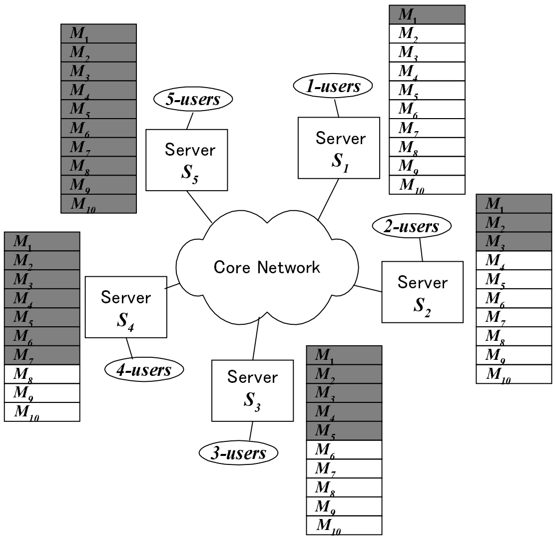


Fig.3.18. Optimal file allocation for the VoD traffic model ($Rd = 2.6, RCnf = 1.0$)

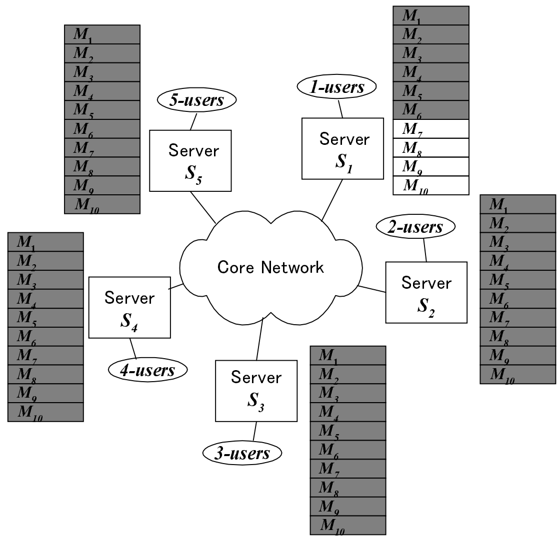


Fig.3.19. Optimal file allocation for the VoD traffic model ($Rd = 4.6, RCnf = 5.0$)

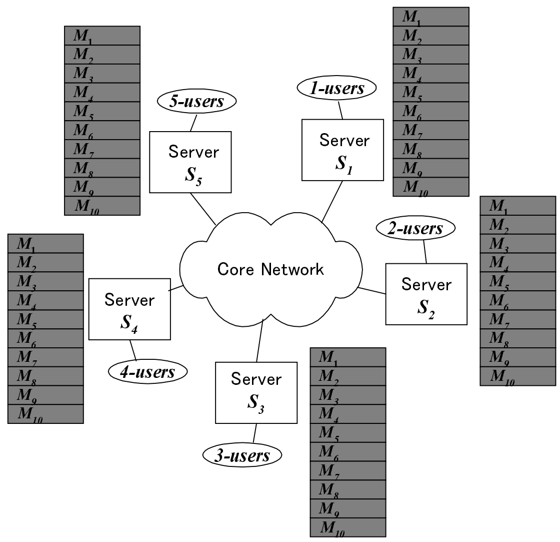


Fig.3.20. Optimal file allocation for the VoD traffic model ($Rd = 5.0, RCnf = 10.0$)

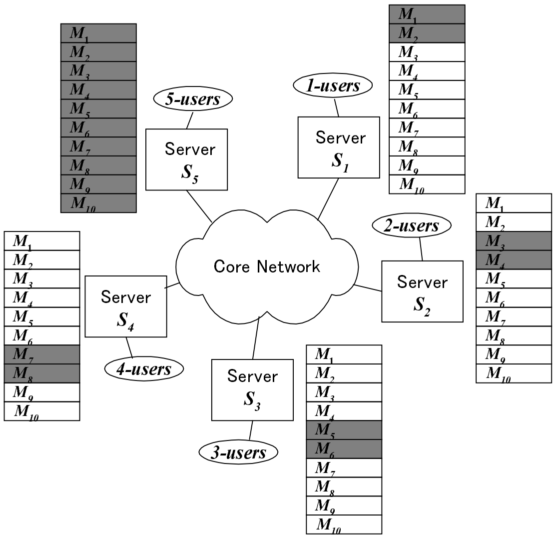


Fig.3.21. Optimal file allocation for the corporate intranets traffic model(I) ($Rd = 1.8, RCnf = 1.0$)

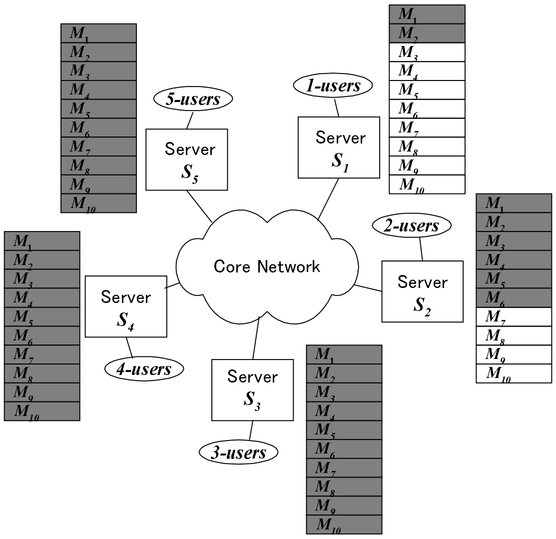


Fig.3.22. Optimal file allocation for the corporate intranets traffic model(I) ($Rd = 3.8, RCnf = 5.0$)

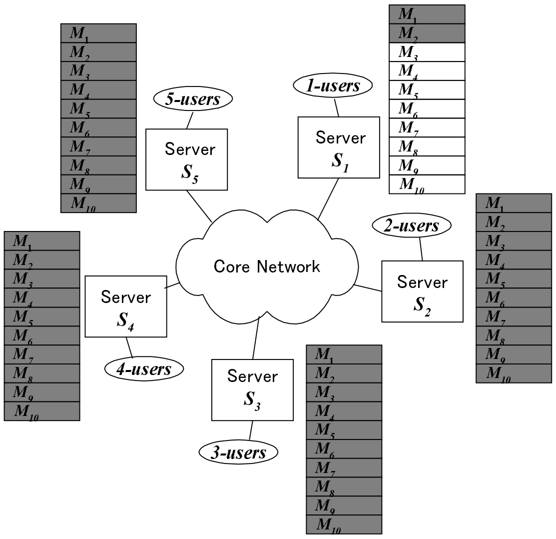


Fig.3.23. Optimal file allocation for the corporate intranets traffic model(I) ($Rd = 4.2, RCnf = 10.0$)

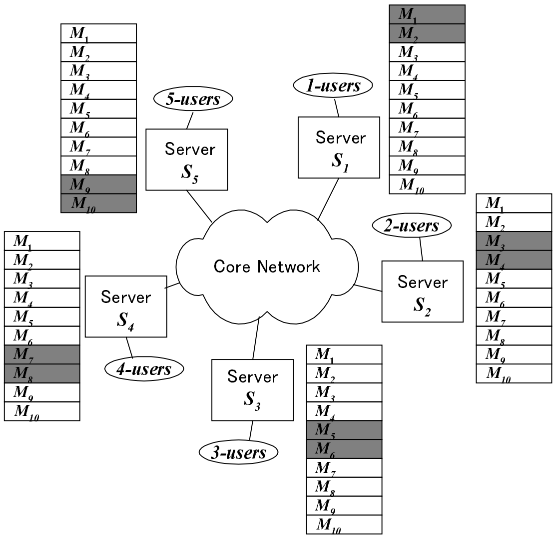


Fig.3.24. Optimal file allocation for the corporate intranets traffic model(II) ($Rd = 1.0, RCnf = 1.0$)

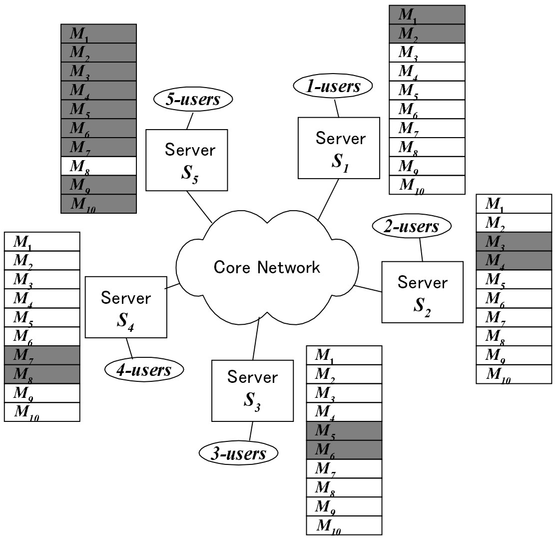


Fig.3.25. Optimal file allocation for the corporate intranets traffic model(II) ($Rd = 1.7, RCnf = 5.0$)

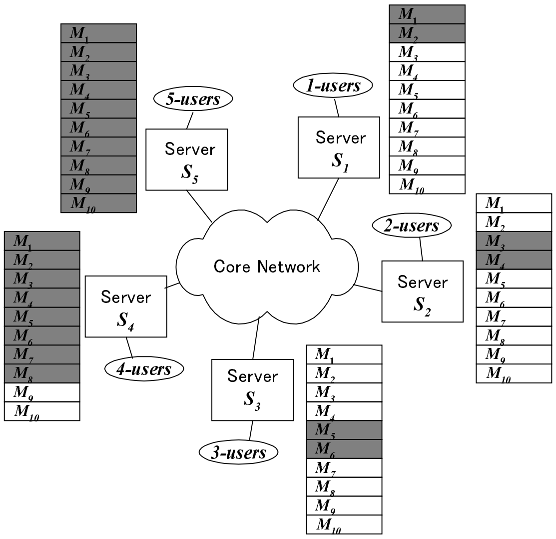


Fig.3.26. Optimal file allocation for the corporate intranets traffic model(II) ($Rd = 2.4, RCnf = 10.0$)

3.5 Sensitivity Analysis of System Parameters

In this section, we analyze the sensitivity of cost and delay parameters in our optimal file allocation model. We believe that this analysis makes our optimization model even more useful.

3.5.1 Purpose of Sensitivity Analysis

In the previous section, we show that using our optimal file allocation model, we can evaluate quantitatively the general characteristics in distributed networks from various points of view. Our optimal file allocation model is very useful in considering different issues in constructing practical distributed networks.

However, in this model, there are many cost and delay parameters affecting the total cost and the total delay, in order to evaluate them in systems more exactly. When we consider different issues in constructing practical systems, it is very important to show how to estimate these cost and delay parameters according to different situations. Otherwise, there is still the possibility that it might not be applicable to some specific environments, although this model is generally capable of dealing with those kinds of networks.

For the purpose of estimating these parameters correctly, we need to examine the weight and the contribution to the total cost of each parameter. Namely, it is very meaningful that we investigate which parameters have the most significant influence on the systems, and what extent the total cost is influenced by the variation of each parameter. In this study, we analyze the sensitivity of these parameters on the optimal file allocation model we have proposed, in order to make clear the weight and the contribution of each parameter and estimate values of these parameters. We consider that this analysis makes it possible to evaluate the optimal systems for all kinds of situations.

In this section, first, we make a comparison among all the cost and delay parameters to know which parameters are the most sensitive. And subsequently, we examine deeply about the behavior of a few parameters which have the most significant influence.

3.5.2 Sensitivity of Cost and Delay Parameters

First, we show the comparison results of sensitivity of all the cost and delay parameters defined in section 3.2. Fig. 3.27 shows the total cost in the case of both 50 %

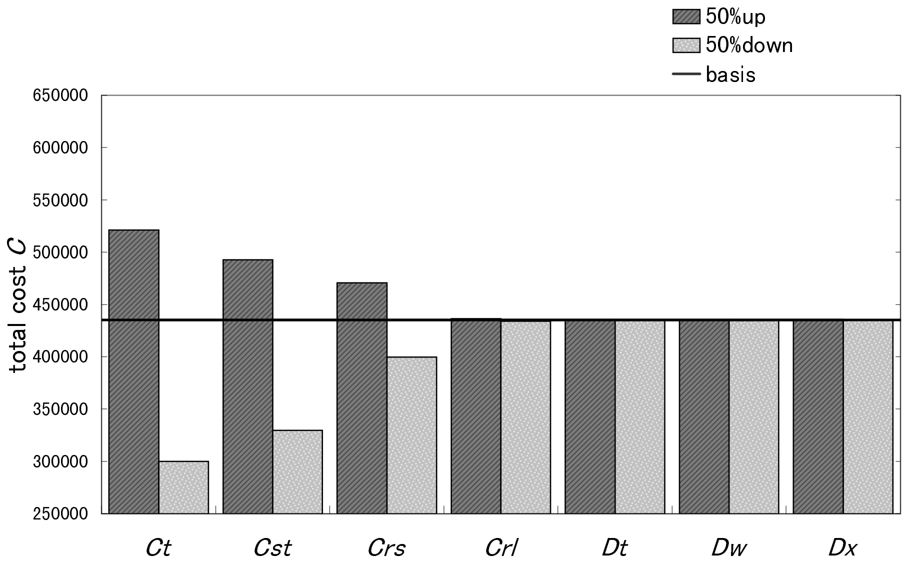


Fig.3.27. Sensitivity of cost and delay parameters ($D_{\max} = 8000$)

increasing and 50 % decreasing from the basic value in each parameter to investigate its influence. In this figure, the line of basis is the total cost in the case that all the cost and delay parameters have the basic values. The sensitivity of each parameter is given by the difference between the basis and the case of the variation of 50 % in each parameter. As shown in this figure, the variations in the communication cost Ct and the storage cost Cst have significant influence on the objective function, that is the total cost. It is obvious that the delay parameters hardly have influence on the total cost. We can easily imagine there is hardly effect of the delay restriction on the file allocation, since we set the maximum total delay $D_{\max} = 8000$ in this figure.

We show the same comparison results in the case the delay restrictions have more influence in Fig. 3.28 and Fig. 3.29. In these figures, we set $D_{\max} = 4000, 2000$ respectively. As shown in Fig. 3.28, when $D_{\max} = 4000$, the communication delay Dt becomes to have a little influence on the objective function in the case of 50 % increasing. Moreover, in the case of $D_{\max} = 2000$, when the restrictive condition is tighter, the communication delay Dt becomes to have much more influence. Besides, the rewriting delay Dw and the service delay Dx become to have some influence.

There is also a remarkable result. In Fig. 3.29, the line of basis itself becomes higher than the other 2 cases. These results show the trade-off between the cost and the delay that the total cost is increasing when the delay is constrained to be lower. We may guess that because of this tendency that the line of basis also becomes higher, the influences of delay parameters do not become even more sensitive according as the delay restriction becomes tighter.

There is another remarkable results in these figures. As the delay restriction becomes tighter, the storage cost Cst becomes to have much more influence on the total cost than Ct. We give full detail of results for the change of the delay restriction in section 3.5.4.

As you can see in these results, the sensitivity of cost parameters is more significant than delay parameters. From this point on, we concentrate on investigating the sensitivity of cost parameters.

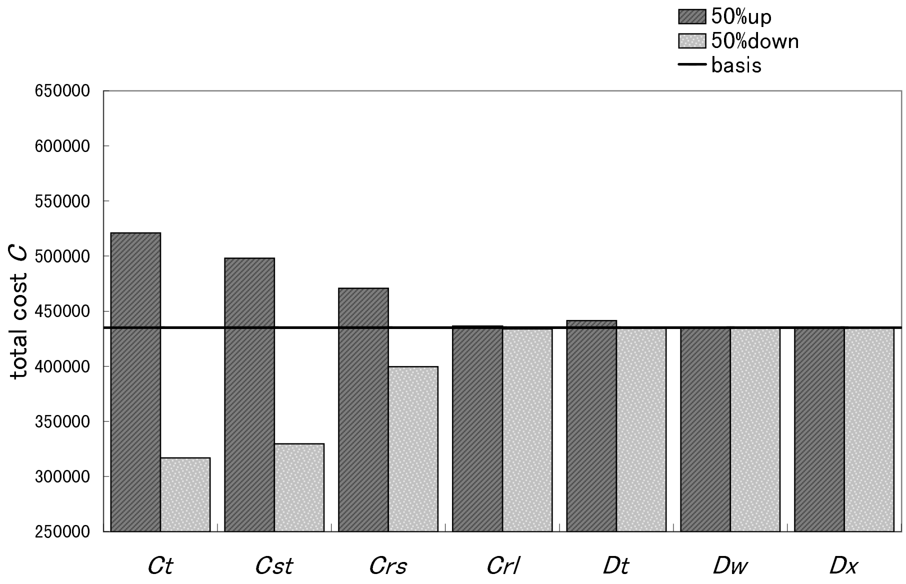


Fig.3.28. Sensitivity of cost and delay parameters ($D_{\max} = 4000$)

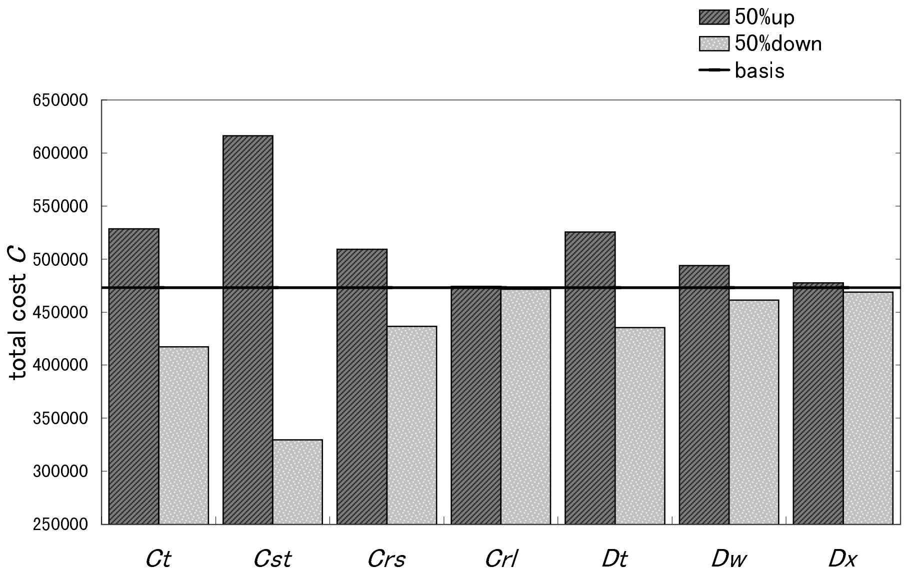


Fig.3.29. Sensitivity of cost and delay parameters ($D_{\max} = 2000$)

3.5.3 Sensitivity of Cost Parameters

Here, we show the sensitivity of the cost parameters in detail. Fig. 3.30 and Fig. 3.31 show the total cost characteristics vs. the variation of the cost parameters for $D_{\max} = 8000$ and $D_{\max} = 2000$ respectively. These figures are normalized at the point where the cost coefficients is equal to 5. When the cost coefficients is equal to 5, all lines cross one another in these figures. These intersections are corresponding to the line of basis in Fig.3.27 and Fig.3.29. The gradient of each line in these figures means the sensitivity of each cost parameter. As shown in Fig. 3.30, the order of sensitivity is Ct, Cst, Crs, Crl, in the case of $D_{\max} = 8000$. On the other hand, in the case of $D_{\max} = 2000$, the order is Cst, Ct, Crs, Crl. We also see in these figures that the storage cost Cst becomes to have more influence on the total cost than the communication cost Ct as the delay restriction becomes tighter (see the section 3.5.2). In addition, we can see the important results in these figures. The total cost grows almost linearly according to the increase in all the cost parameters. We can estimate the value of these cost parameters according to this linear characteristics.

From the both results, we can see the communication cost and the storage cost are the most sensitive parameters. Thus, we show the sensitivity of communication cost and the storage cost in detail.

In Fig.3.32 and Fig.3.33, we show the total cost characteristics for different values of the local-global differences of the communication cost $RCnf$, i.e., 1, 3, 5, vs. the communication cost and the storage cost respectively. As we may also see in these figures, the sensitivity of the communication cost and the storage cost have approximately linear characteristics. These characteristics become more evident when the both cost parameters take large values. Also, in Fig. 3.32, the gradients of the lines is almost identical for different values of the local-global differences $RCnf$. On the other hand, in Fig.3.33, the gradients of the lines becomes large when the local-global differences $RCnf$ is getting larger. This is because as $RCnf$ grows larger, the files should be distributed at many servers and the storage cost becomes to have more significant influence, that is, the storage cost becomes more sensitive.

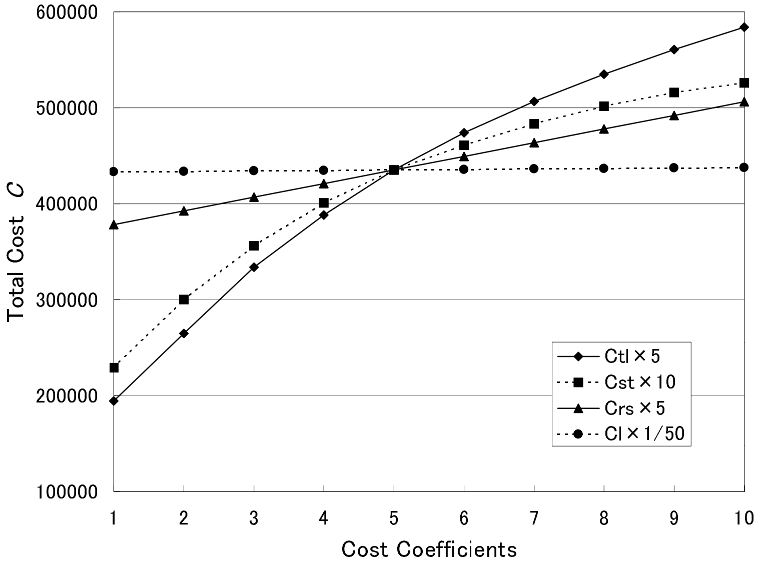


Fig.3.30. A total cost characteristics in the case of $D_{\max} = 8000$ vs. cost parameters, the local communication cost coefficient Ctl times 5, the storage cost coefficient Cst times 10, the server cost coefficient Crs times 5, the line cost coefficient Cl times 1/50 ($RCnf = 3.0, RDnf = 5.0, Pw = 0.1$)

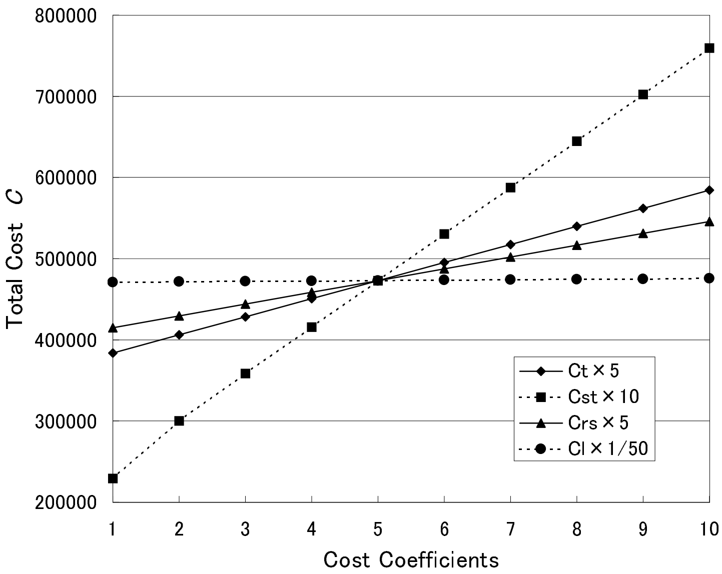


Fig.3.31. A total cost characteristics in the case of $D_{\max} = 2000$ vs. cost parameters, the storage cost coefficient Cst times 10, the local communication cost coefficient Ctl times 5, the server cost coefficient Crs times 5, the line cost coefficient Cl times 1/50 $(RCnf = 3.0, RDnf = 5.0, Pw = 0.1)$

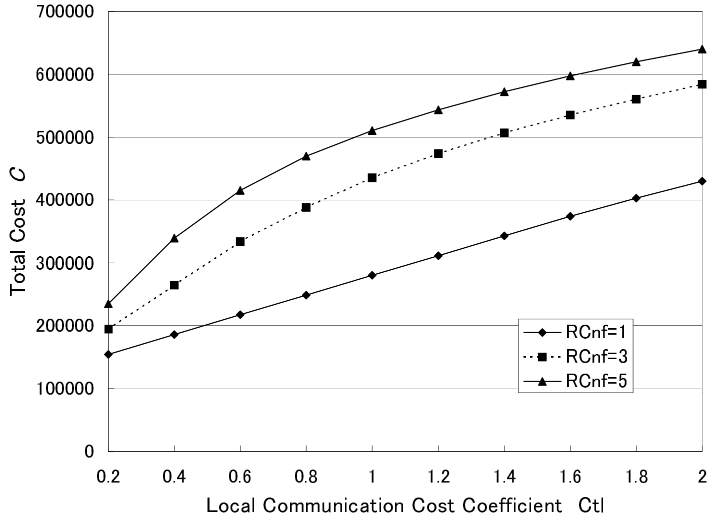


Fig.3.32. A total cost characteristics vs. local communication cost coefficient Ctl ($D_{\max} = 8000, RDnf = 5.0, Pw = 0.1$)

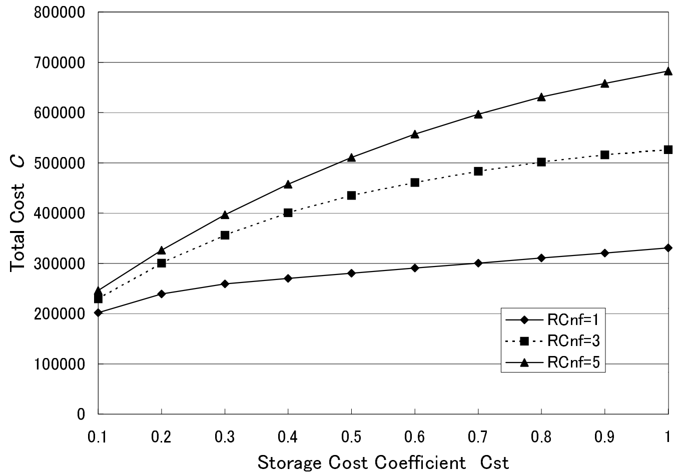


Fig.3.33. A total cost characteristics vs. storage cost coefficient Cst ($D_{\max} = 8000, RDnf = 5.0, Pw = 0.1$)

3.5.4 Influence of Delay Restrictions

Finally, we give full detail of the influence of the delay restrictions upon the sensitivity of each parameter, which appears in the above analysis. In Fig. 3.34 and Fig. 3.35, we show the sensitivity of the communication cost Ct and the storage cost Cst regarding the change of the delay restriction. These figures show the total cost characteristics for different values of the maximum total delay $D_{\max}$, i.e. 2000, 4000, and 8000, vs. the communication cost and the storage cost coefficient respectively. As shown in Fig. 3.34, the gradient of the line becomes smaller as the maximum total delay $D_{\max}$ is smaller, that is, the delay restriction becomes tighter. On the other hand, in Fig. 3.35, the gradient of line becomes larger as the delay restriction becomes tighter. Files are more distributed to reduce the communication delays when the restriction is tight as shown in Fig.3.15. Thus, when $D_{\max}$ is small, the storage cost becomes to have much more significant influence, while the communication cost becomes to have less influence. As was observed above, there is a reversal of the sensitivity of the cost parameters with respect to the changes in the delay restrictions.

Moreover, as we can see in these figures, when the local communication cost coefficient Ctl is more than 1.6, and the storage cost coefficient Cst is less than 0.3, the gradients of three lines are identical. When the communication cost is large and the storage cost is small, files are more distributed. We can imagine that the total delay does not exceed the maximum $D_{\max}$ due to this distribution of files in any cases, even in the case that $D_{\max} = 2000$. Thus, there are hardly differences in the gradients for different $D_{\max}$, when the communication cost is large and the storage cost is small.

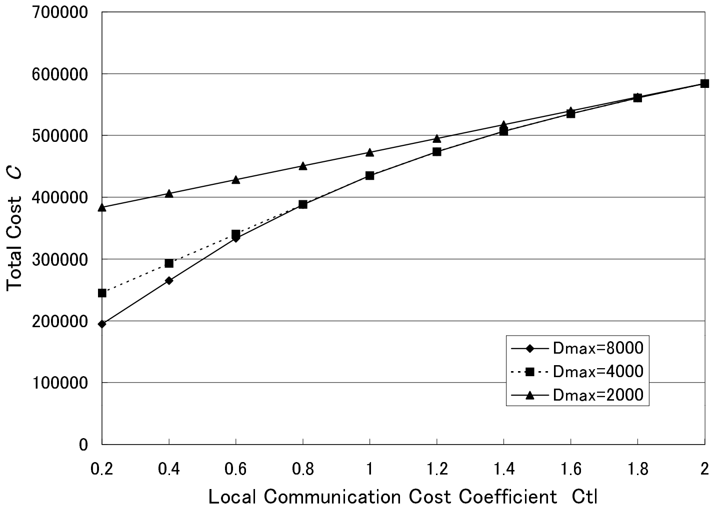


Fig.3.34. A total cost characteristics vs. the communication cost for different values of $D_{\max}$ $(RCnf = 3, RDnf = 5.0, Pw = 0.1)$

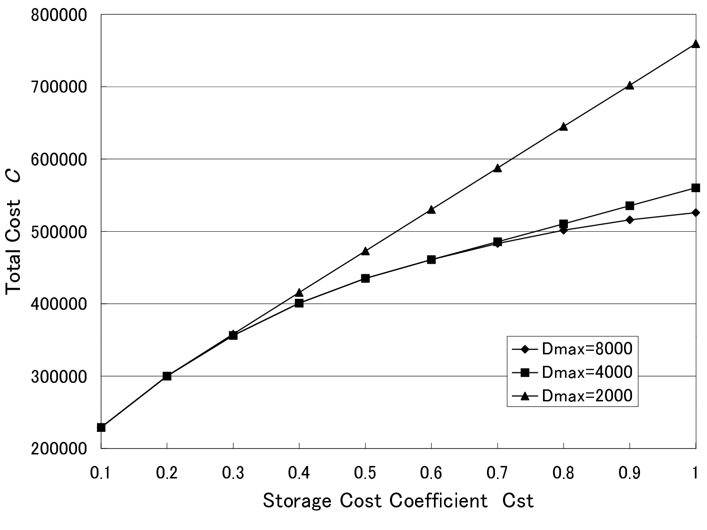


Fig.3.35. A total cost characteristics vs. the storage cost for different values of $D_{\max}$ $(RCnf = 3, RDnf = 5.0, Pw = 0.1)$

3.6 Summary

In this chapter, we have proposed the optimal file allocation model such that the total cost is minimized subject to the total delay in distributed information network system. This optimization model covers a wide range of multimedia network applications. We have formulated this problem as a 0-1 integer programming, and we have described the considerations of numerical results by solving this optimization problem.

In numerical calculations, we used the approximate method which is based on a greedy algorithm, since our optimization model belongs to the class of NP hard problems. As a result of performance evaluation, we have shown this approximate method is very useful in terms of calculation time and its approximate solution is also practical enough.

As the numerical results by the approximate method, we have quantitatively shown the general characteristics on file distribution in practical large scale of systems, such as a trade-off between the communication costs and the storage costs and the effect of the delay restriction. The system model presented here is very useful for evaluating various elements in constructing a distributed system.

However, in our model, there are many parameters affecting the total cost and the total delay, in order to deal with the system more exactly. It is very useful to examine the weight and the contribution of each parameter in constructing practical systems. Therefore, we also have analyzed the sensitivity of the cost and delay parameters. As a result, the variation in the communication cost and the storage cost have the most significant influence on the objective function mainly. Also, the sensitivity of these cost parameters have the linear characteristics. According to this characteristics, we can estimate the values of these parameters when we construct practical systems. In addition, we have shown the influence of the delay restrictions on the sensitivity of these parameters. We have made clear that we can grasp the influence among principal parameters in full detail by this sensitivity analysis.

Chapter 4

Reliability-Based Optimal Resource Allocation

4.1 Introduction

There has been an explosive increase in the number of users in multi-media networks, particularly over the Internet. Furthermore, the increasing traffic over these networks to enable users to connect to the Internet from mobile computers including cellular phones is spreading rapidly. This causes network servers to become overloaded, and we are faced with several essential issues, such as the decline of reliability, the increase of the access delay and so on [30][31].

Despite the tremendous potential of reliable distributed computing systems, relatively little attention has been paid to developing systems with better reliability technologies. In this study, we focus on constructing high-reliability networks.

In this chapter, we mainly present the two topics, which include the optimal mirror and file allocation in static distributed networks and the dynamic file allocation in mobile distributed networks.

Firstly, in finding a solution for the issues of load-balancing, the mirroring of network servers has been considered and various studies concerning this issue have been done widely. Several mirror servers, which store copies of the same files and have the same function as network servers, are set in networks for the purpose of load balancing.

Distributed allocation of mirror servers results in prompt access of users and improvement of system reliability. However, this causes the cost of setting up mirror servers and managing the whole system to increase. It is obvious that there is a trade-off between the reliability and cost. In constructing effective networks and providing the emerging multi-media services, one of the most important problems today lied in the allocation of mirror servers and copies of files on the networks. Though there have been many works on distributed file allocation for this purpose, most of them consider only costs and delays, and do not consider reliability [12]-[16][22][25][34][42]-[46][103]-[106][109]-[112].

In this study, We propose the reliability-based *Optimal Mirror Allocation Model* such that the system reliability is maximized subject to the cost and the delay, in view of the trade-offs described above. In this model, we introduce a 0-1 integer programming formulation by setting the 0-1 variables on the allocation of mirror servers and files that exist on each server. We can find where each mirror server should be, and the concrete assignment of files to mirror servers.

Also, we give a network topology by an adjacency matrix, whose elements have weights relating to the distance between a pair of nodes. Most of the studies concerning the mirroring in networks consider only the tree structure as a network topology [98][99]. However, the topology of current multi-media networks such as the Internet is not limited to the tree structure. We need to be able to consider other network structures. Our model is capable of dealing with various kinds of network structures, including tree structures, mesh structures, etc. In addition, we may take into account any problems with regards to communication costs and delays by using the weight of the distance.

Our objective is to find the optimal mirror allocation by solving this model, and demonstrate quantitatively the general characteristics on the improvement of the system reliability using the distributed allocation of mirror servers. This work on reliability-based optimal mirror allocation is described in section 4.2 and 4.3.

Secondly, we consider dynamic file allocation in mobile database systems, where the system reliability is more likely to decline. Recent rapidly-advances in wireless communication technology have made dynamic file allocation in mobile networks a critical issue. It is very important to consider reliability-based dynamic file allocation in mobile networks as well as to consider file allocation in static networks that we have done.

It is natural that there are many differences between static networks and dynamic networks. The main difference in dynamic networks is clearly its mobility. Distributed data management will be changed due to mobility. Resources will have to be dynamically reconfigured in response to the mobility of the users.

In particular, we consider providing time critical database information services in a more dynamic and critical environment where the need for high reliability is greater. In this study, we develop the innovative strategy for dynamic file allocation to meet this critical requirement. This strategy consists of two phases: 1) Periodically generating optimal file allocation via a heuristic algorithm based on the complete system state and 2) Using knowledge-based policy to reallocate resources based on the partial system state changes.

In this strategy, we first find the initial file allocation that minimizes the operating cost subject to reliability and response time, using a heuristic algorithm. This heuristic algorithm is based on A$*$ algorithm and we apply several rules for pruning the original search space to meet critical time requirement. In the second phase, we use the knowledge-based reallocation policy to dynamically reallocate files according to changes in the state of systems. Over time, repeatedly applying the knowledge-based reallocation policy only gets us farther from the optimal file allocation. Thus, we periodically use the heuristic algorithm to optimize the file allocation for the latest system state. This dynamic file allocation strategy is described in section 4.4.

4.2 Reliability-Based Optimal Mirror Allocation Model

4.2.1 System Model

We introduce a system model of a distributed system for the optimal mirror allocation.

In the optimal file allocation model that we proposed in the previous chapter, we have not considered a topology of networks. In this model, we give a topology of a network by an adjacency matrix **A** so that we can consider more accurately network structures. The elements of the adjacency matrix have weights relating to the distance between a pair of nodes. We calculate the shortest path matrix **Q**, and formulate the reliability, the cost, and the delay using the matrix **Q**.

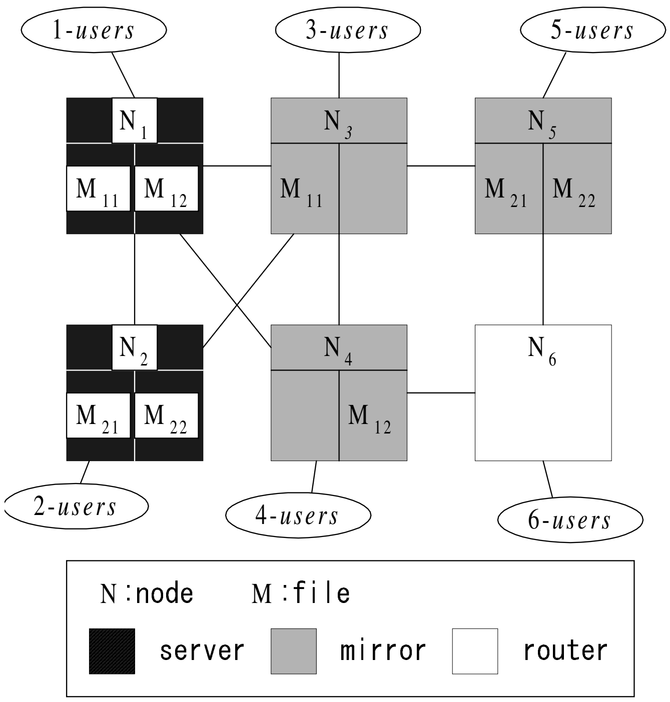


Fig.4.1. An example of distributed network model

There exist n nodes in the system, and each node is denoted by N_i $(1 \leq i \leq n)$. It is assumed that all nodes originally function as routers. s nodes $(s \leq n)$ work as servers, which hold individual files, and the rest of the nodes, say $(n-s)$ nodes, may be mirror servers of the s servers by storing all or a part of their files. In the case that the node N_i is a server or a mirror server, it has the storage capacity of B_i. It is assumed that each user in the system is connected to one of the nodes, called a local node. The users connected with the node N_i are called *i-users*.

The number of distinct files at each server N_i is m_i $(1 \leq i \leq s)$, and each file is denoted by M_{ik} $(1 \leq k \leq m_i)$. It is assumed that the size of the file M_{ik} is denoted by F_{ik}. Each mirror server can store these files as long as the total size of files does not exceed the storage capacity.

i-users generate file access flow, where the average is α_i per unit time. Also, let P_{ijk} denote the access probability from *i-users* for the files M_{jk} $(\sum_j \sum_k P_{ijk} = 1)$. The parameter P_{ijk} determines the traffic matrix in the system. It is assumed in this model that all file accesses are just to read files, and we do not consider the updating of files.

Moreover, we define λ_i as the failure rate of a node N_i, μ_i as the service rate, and t_{ij} as the communication rate of a link between the nodes N_i and N_j. In this study, it is assumed for analytical simplicity that the communication rate of each link has the identical value t. We also do not consider the case of the failure of each link in networks.

4.2.2 Formulation

Our objective is to find the optimal allocation of mirror servers such that the system reliability is maximized. We also take into account the cost to set up mirror servers, the total cost of managing the whole system, and the communication delay as the restrictive condition. In addition, we would like to consider the MTTR (mean time to repair), the detour of the path, etc. in the future to analyze more accurately system behavior where faults occur.

To formulate the optimal mirror allocation model, we define two kinds of 0-1 variables. We formulate this optimization problem as 0-1 integer programming problem using the following 0-1 variables.

1. The variables on the function of nodes S_{i1},S_{i2}.

$$S_{i1} = \begin{cases} 1 & \text{(the node } N_i \text{ is a server)} \\ 0 & \text{(otherwise)} \end{cases} \tag{4.1}$$

$$S_{i2} = \begin{cases} 1 & \text{(the node } N_i \text{ is a mirror server)} \\ 0 & \text{(otherwise)} \end{cases} \tag{4.2}$$

2. The variables on the allocation of files X_{ijk} [12].

$$X_{ijk} = \begin{cases} 1 & \text{(the file } M_{jk} \text{ is stored in the node } N_i) \\ 0 & \text{(otherwise)} \end{cases} \tag{4.3}$$

Initial Conditions

Under the assumptions presented in the previous section, we add the following capacity condition as a initial condition. Each node N_i can store files as long as the total size of files does not exceed the capacity of the node, that is

$$\sum_j \sum_k X_{ijk} F_{jk} \leq B_i \tag{4.4}$$

Objective Function

We formulate the reliability per unit time in the whole system as the objective function. In this study, we define the system reliability as the mean of the success rate of each access from each user to each file. We consider the required nodes to complete the access from each user to each file, and calculate the reliability of the combination of these nodes using the reliability of each node as the success rate of each access. In the case that there are several paths between nodes, for the sake of analytical simplicity, we take into account only the shortest path. We should consider the detour of the path in the case of path failure in the future.

In this study, we assume that the user accesses the node that has the shortest distance from the local node, if a file requested from a user is not stored in the local node and several remote nodes store the requested file. Using common probability formulas [36][39][41], the node reliability of a node N_i, $R_i(\mathbf{S}, \mathbf{X})$, is formulated as follows.

$$R_i(\mathbf{S}, \mathbf{X}) = S_{i1} \cdot \exp\left(-\lambda_i \cdot \sum_j \sum_k \frac{\alpha_j P_{jik} F_{ik}}{\mu_i} \left(\Delta_{ji} + (1 - \Delta_{ji})(1 - X_{jik}) \Delta_{jh_1} \right) \right)$$

$$+ S_{i2} \cdot \exp\left(-\lambda_i \cdot \frac{1}{\mu_i} \sum_j \sum_k \left(X_{ijk} \cdot \alpha_i P_{ijk} F_{jk} + \sum_g X_{ijk} (1 - X_{gjk}) \Delta_{ih_2} \cdot \alpha_g P_{gjk} F_{jk} \right) \right)$$

$$+ (1 - S_{i1})(1 - S_{i2}) \tag{4.5}$$

Here, Δ_{ji} is defined as follows, and index h_1 and h_2 are equal to the value that minimizes respectively the following functions $h1$ and $h2$ on the positive side.

$$\Delta_{ji} = \delta(j - i) = \begin{cases} 1 & (j = i) \\ 0 & (j \neq i) \end{cases} \tag{4.6}$$

$$h1_{\min}(h_1) = X_{h_1 ik} \cdot q_{jh_1} \tag{4.7}$$

$$h2_{\min}(h_2) = X_{h_2 jk} \cdot q_{gh_2} \tag{4.8}$$

In equation (4.5), the first term refers to the case where node N_i is a server, the second term refers to the case where the node is a mirror server, and the third term refers to the case the node only functions as a router. In the case that each node works as only a router, it is assumed that the failure rate of the node is equal to 0. This means that the reliability of the node is equal to 1.

Using this node reliability, we calculate the system reliability. In the following, we describe the procedure of the calculation.

1. We calculate *R_path* as the reliability of all nodes that are required to be available for the user to succeed in accessing each file as follows. In the case that there exist several nodes which store the requested file from the user, we calculate *R_path* for all paths from the user to these nodes.

$$R\_path = \prod_i R_i \tag{4.9}$$

2. In the case that there is only one node which stores the requested file, we calculate $R\_temp(j, M_{ik})$ as the success rate of the access from *j-user* to the file M_{ik} as follows.

$$R\_temp(j, M_{ik}) = R\_path \tag{4.10}$$

Otherwise, $R\_temp(j, M_{ik})$ is as follows.

$$R\_temp(j, M_{ik}) = 1 - \prod (1 - R\_path) \tag{4.11}$$

3. We calculate the mean of $R\_temp(j, M_{ik})$ as the system reliability R_s.

$$R_s = \frac{\sum_j \sum_i \sum_k R\_temp(j, M_{ik})}{n \cdot \sum_i m_i} \tag{4.12}$$

Restrictive Conditions

We formulate the mirror cost, the total cost, and the communication delay as the restrictive conditions for the optimal mirror allocation problem.

1. Mirror Cost Cf

   The mirror cost Cf is the initial cost to set up mirror servers. This depends on the storage capacity of files on mirror servers. Let $Cf_{\max}$ be defined as the maximum mirror set cost. Here, Cb denotes the capacity cost coefficient, and Cm denotes the administration cost coefficient. Then, the mirror cost is as follows.

$$Cf = \sum_i S_{i2} \cdot (B_i \cdot \mathsf{Cb} + \mathsf{Cm}) \leq Cf_{\max} \tag{4.13}$$

2. Total Cost C

   The total cost is the cost to manage the whole system per unit time. In the optimal file allocation model proposed in the previous chapter, we consider that the total cost includes the server cost and the line cost as well as the storage cost and the communication cost. However, according to the sensitivity analysis that we did, the storage cost and the communication cost have more significant influence than the other two costs. Therefore, in this model, it is assumed that the total cost C consists of the storage cost Cs and the communication cost Ct. Let $C_{\max}$ be defined as the maximum total cost, and the total cost is as follows.

$$C = Cs + Ct \leq C_{\max} \tag{4.14}$$

   Next, we give full details of the storage cost and the communication cost.

   - Storage Cost Cs

     We assume that the storage cost is required for each node to store files. It is charged on the basis of unit time, and it is not dependent on the frequency of accesses. In this model, the storage cost depends on the size of all the files stored at a node. Let Cs be the storage cost coefficient, which has a fixed value that is common for all nodes.

$$Cs = \mathsf{Cs} \cdot \sum_i \sum_j \sum_k X_{ijk} F_{jk} \tag{4.15}$$

- Communication Cost Ct

  We assume that the communication cost is required to communicate between a user site and his/her requested file site. For example, when an *i-user* accesses a file M_{jk}, the communication cost will be the cost to communicate between an *i-user* and a node that stores the file M_{jk}. If a local node N_i stores the file M_{jk}, the communication cost is the cost to communicate only between an *i-user* and the local node. Also, in this study, if a file requested from a user is not stored in the local node and several remote nodes store the file, the user accesses the nodes that have the shortest distance from the local node, as described before. We define Ct_{jk} as the communication cost coefficient of the file M_{jk}. It is assumed that the communication cost coefficient Ct_{jk} is dependent on the size of the file F_{jk}. Ctt is the communication cost coefficient which has a fixed value that is common for all files.

  $$\mathsf{Ct}_{jk} = \mathsf{Ctt} \cdot F_{jk} \tag{4.16}$$

  Using this coefficient and 0-1 variables X_{ijk}, we formulate the communication cost as follows.

  $$Ct = \sum_i \left(\sum_j \sum_k \left(X_{ijk} \mathsf{Ct}_{jk} + (1 - X_{ijk}) X_{h_3 jk} q_{ih_3} \mathsf{Ct}_{jk} \right) \alpha_i P_{ijk} \right) \tag{4.17}$$

  Here, it is assumed that the index h_3 equals the value that minimizes the following function on the positive side.

  $$h3_{\min}(h_3) = X_{h_3 jk} \cdot q_{ih_3} \tag{4.18}$$

3. Communication Delay Dt

   The communication delay is spent to communicate in accessing from a user to a file. If we let $Dt_{\max}$ be defined as the maximum communication delay, the communication delay is as follows. We define Dt_{jk} as the communication delay coefficient of the file M_{jk}. It is assumed that the communication delay coefficient Dt_{jk} is dependent on the size of the file F_{jk} and the communication rate t.

$$Dt = \sum_i \left(\sum_j \sum_k \left(X_{ijk} \mathsf{Dt}_{jk} + \left(1 - X_{ijk}\right) X_{h_3 jk} q_{ih_3} \mathsf{Dt}_{jk} \right) \alpha_i P_{ijk} \right)$$

$$\leq Dt_{\max} \tag{4.19}$$

$$\mathsf{Dt}_{jk} = \frac{F_{jk}}{t} \tag{4.20}$$

Here, it is also assumed that index h_3 equals the value that minimizes the function $h3_{\min}(h_3)$ (eq.(4.18)) on the positive side, too.

4.3 Reliability Characteristics on Distributed Mirror Allocation

In the previous section, we introduced the optimal mirror allocation model as a 0-1 integer programming model. Using this optimization model, likewise with the optimal file allocation model, it is possible for us to evaluate quantitatively various issues regarding improvement of the system reliability by distributed mirror allocation.

4.3.1 Approximate Method

We use an approximate algorithm for the optimization problem to deal with practical large-scale systems as well. As mentioned before, in solving optimization problems, the algorithm used is very significant. Our optimal mirror allocation model belongs to the class of NP hard problems. Using an exhaustive search, we may solve only a small model, and we may have to spend a long time to solve even the small problem. Here, we simply describe the approximate algorithm we use and show the performance comparison between the approximate method and the exhaustive search to prove that the approximate algorithm is useful enough for our optimal mirror allocation model.

Approximate Algorithm

The approximate algorithm we use in solving the optimal mirror allocation problem is also based on the Greedy Method. Greedy Method finds feasible solutions by directly fixing each variable on the basis of local critical values for contribution to an objective function. We can easily imagine that more popular files are likely to be distributed in distributed database systems. In this study, we take the access frequency to the files as the local critical values, that is $\alpha_j \times P_{jik}$, which is denoted by Af_{jik} hereafter. We fix the variable $X_{jik} = 1$ in the order of the value of the corresponding access frequency Af_{jik}, as long as all restrictive conditions are satisfied. In the following, we describe our algorithm based on the Greedy Method.

[**Approximate Algorithm**]

(1) First, we calculate the access frequency Af_{jik} to each file.

(2) We sort the variables X_{jik} in decreasing order of Af_{jik}.

(3) We repeat the procedures from 3.1 to 3.5 until the last variable of the sorted sequence in the previous procedure.

  3.1 Fix a variable $X_{jik} = 1$ in a sorted order.

  3.2 Check whether the initial condition is satisfied or not. If the initial condition is not satisfied, we return the value to 0, and go back to the step 3.1.

  3.3 Check whether the rest of restrictive conditions are satisfied or not. If the restrictive conditions are not satisfied, go back to the step 3.1.

  3.4 Fix variables S_{i1} and S_{i2} on the basis of the current values of X_{jik}.

  3.5 Calculate the system reliability for the current allocation determined in the previous procedures, and commit the mirror allocation with the system reliability to the memory.

(4) We select the mirror allocation such that the system reliability is maximized among all allocation calculated in step 3.5, and consider it as the approximate solution.

Performance Evaluation

We evaluate the usefulness of the approximate method in terms of the accuracy and the calculation time. In the comparison between the approximate solution and the optimal solution by the exhaustive search, we assume four kinds of traffic models as shown in Table 4.1.

Table 4.1: Traffic model

Traffic Model	Mean Access Times α_j	Access Probability P_{jik}
1	uniform	uniform
2	uniform	random
3	ununiform	uniform
4	ununiform	random

We define the relative error as the performance measure.

$$\text{RelativeError}(\%) = \frac{\text{optimal reliability} - \text{approximate reliability}}{\text{optimal reliability}} \times 100 \quad (4.21)$$

In Fig.4.2, We show the performance comparison between the approximate method and the exhaustive search for the above traffic models. Here, we consider the system parameters as follows. There exist 5 nodes ($n = 5$), and the capacity of each node

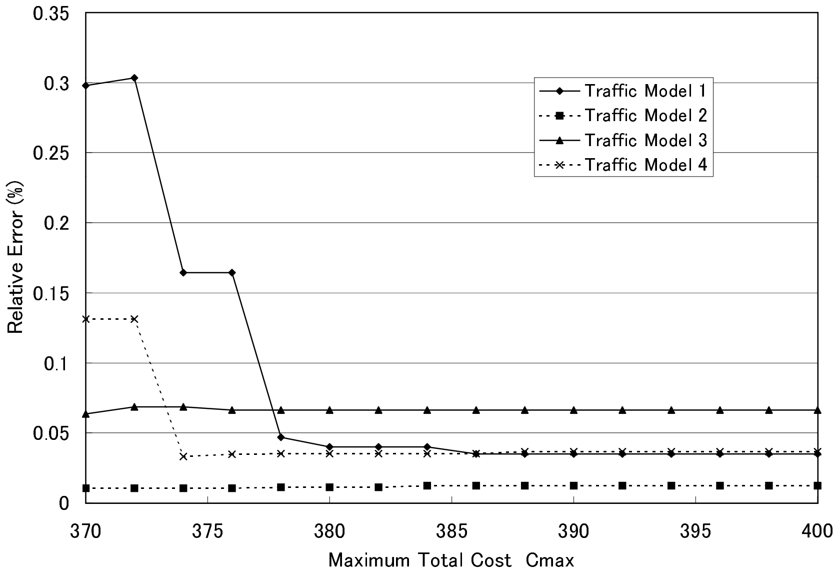


Fig.4.2. A comparison of two algorithms

Table 4.2: Average time comparison between two algorithms in traffic model 2 [sec.]

n	s	$\sum_i m_i$	# of combinations of variables	Exhaustive search	Approximate algorithm
4	2	4	2^8	0.13	0.00111
5	2	4	2^{12}	41.07	0.00144
4	2	6	2^{12}	38.5	0.00171
4	2	8	2^{16}	12411	0.00289
5	2	6	2^{18}	234792	0.00351

is 20[Gbyte] ($B_i = 20000, 1 \leq i \leq 5$). There are 2 servers ($s = 2$), and the rest of the nodes, say 3 nodes, can be mirror servers. Also, there exist 2 kinds of files on each server respectively ($m_i = 2, 1 \leq i \leq s$), and the size of each file is 30[Mbyte] ($F_{ik} = 30$). This figure shows the relative error vs. the maximum total cost $C_{\max}$. As illustrated, the value of the relative error of the approximate method is less than approximately 0.3% in all traffic models. In most cases, the relative error has a value less than 0.1%. From these results, we can see that the approximate method is almost exact, and that it is practical enough.

Next, we compare the average time it takes to solve the optimization problem in Table 4.2. In this table, we show the average time [second] each algorithm takes in solving several kinds of scales of systems. Here, we used Sun UltraSPARC II$_i$ (334MHz) for the numerical calculations. There is a remarkable difference between these two algorithms. In the case of the exhaustive search, the average time increases exponentially as systems scale. On the other hand, in the case of the approximate method, the increase in the calculation time is very little. We may guess it wouldn't take that much time even for large-scale systems. Therefore, it is obvious that the approximate method is useful in view of the calculation time.

In the case of the exhaustive search, the scale of the system model we can solve is basically limited to the scale $n = 5, s = 2, m_i = 3$. By using the approximate method, it is possible to get numerical results of the optimization problem for practical large-scale system models in the next section.

4.3.2 System Parameters for Numerical Results

We show quantitatively the general characteristics on the load balancing by the distributed allocation of mirror servers. In our model, we may consider various network environments such as the Internet, mobile network, etc., by setting system parameters. Here, for analytical simplicity, we consider the system parameters as follows. There exist 30 nodes ($n = 30$), and the capacity of each node is 20[Gbyte] ($B_i = 20000, 1 \leq i \leq 30$) in the case that the node is a mirror server. There are 10 servers ($s = 10$), and the rest of the nodes, say 20 nodes, can be mirror servers. Also, there exist 10 kinds of files on each server respectively ($m_i = 10, 1 \leq i \leq s$), and the size of each file is $10 \sim 100$ [Mbyte] ($F_{ik} = 10 \sim 100$). However, we set the file size differently in the paragraph of characteristics on the mirror restriction. Also, we assume the high rate of failure so that we can clearly see the difference in the system reliability due to the mirroring.

Considering the optimal mirror allocation problem, we define the number of mirrors in systems Nm and the distribution rate of files Rdf as performance measures.

$$Nm = \sum_i S_{i2} \tag{4.22}$$

$$Rdf = \frac{\sum_j \sum_i \sum_k X_{jik}}{\sum_i m_i} \tag{4.23}$$

Nm is the number of nodes which work as mirror servers in the system. Rdf is the ratio of the number of all files allocated in the system to the number of distinct files. We obtain the assignment of mirrors to networks, and files to nodes as the optimal solutions. On the basis of this assignment, we calculate the distribution rates, the system reliability, the total cost, the communication delay, etc. We describe our considerations regarding the optimization results according to these criterion.

4.3.3 Reliability Characteristics on Change of Traffics

Firstly, we show the characteristics of the system reliability on the change of the average number of accesses from users.

In Fig.4.3, we show the system reliability vs. the average number of accesses for the different scales of systems, i.e., the number of nodes n equal to 50, 30. We set the number of servers s as 10 here, as mentioned before. In this figure, we assume that the restrictive conditions do not affect the optimization at all, in order to investigate the influence of the change of traffic. As we may see from this figure, the system reliability decreases, as the number of accesses increase. This figure shows quantitatively the decline of the system reliability according to the increase of the traffic.

We may guess that the allocation of more mirror servers can constrain such a decline of the system reliability. Therefore, we show quantitatively the improvement of the system reliability by the mirror distribution in the following.

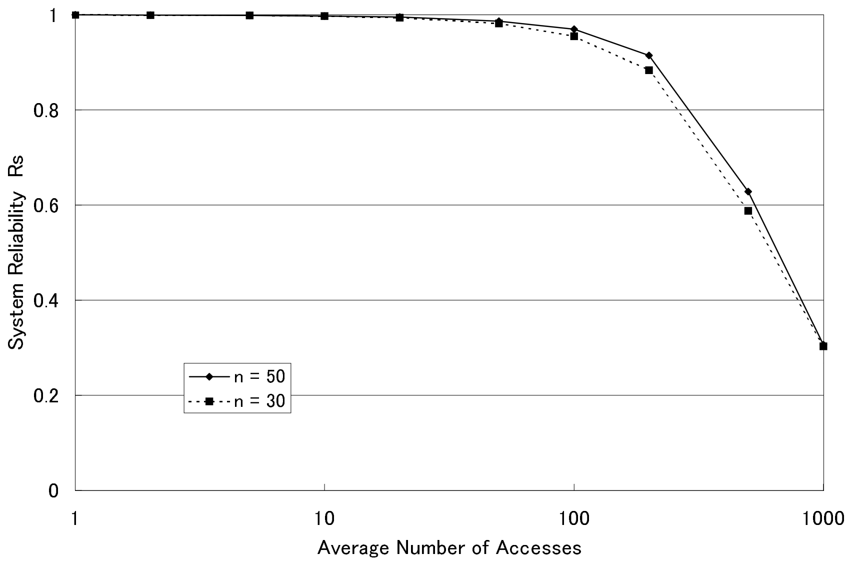


Fig.4.3. Reliability characteristics on change of traffics

4.3.4 Characteristics on Cost Restriction

We show the characteristics of the mirror distribution regarding the restrictive condition of the total cost.

In Fig.4.4, we show the system reliability and the number of mirrors vs. the maximum total cost $C_{\max}$ in the case that the average number of accesses is equal to 3. As shown in this figure, when $C_{\max}$ is small, files are not distributed in so many mirror servers and the system reliability is low. When $C_{\max}$ is large, that is, the total cost restriction is not so tight, all nodes, which can be mirror servers, work as mirror servers, and there is improvement in the system reliability. We can obtain the assignment of mirrors to networks, and files to nodes as the optimal solution. From this figure, we can see clearly that the system reliability improves using the mirror distribution. This result also illustrates quantitatively the trade-off between the system reliability and the total cost.

In Fig.4.5, we show the trade-off between the system reliability and the total cost in more detail. This figure shows the system reliability vs. the normalized value of the maximum total cost for the different numbers of accesses, that is, 1, 5,

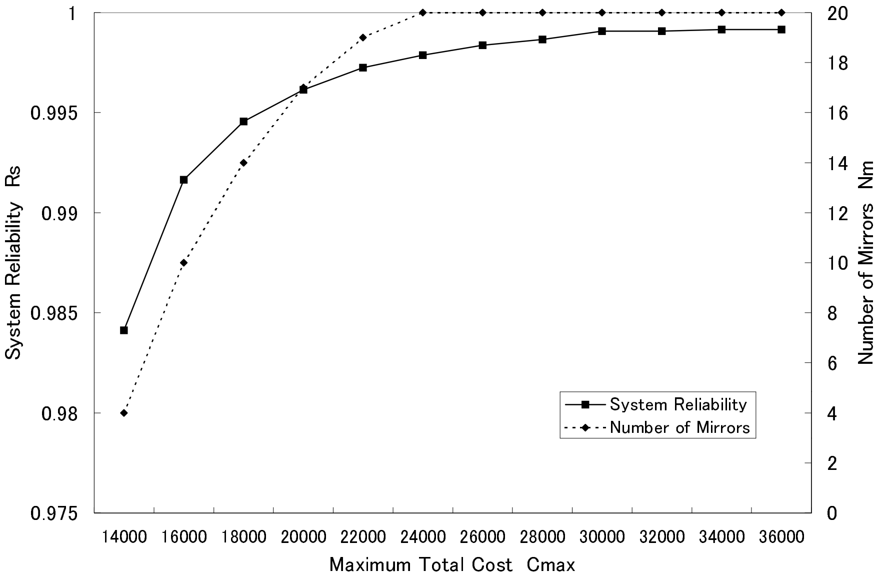


Fig.4.4. An improvement of the system reliability by the mirror distribution

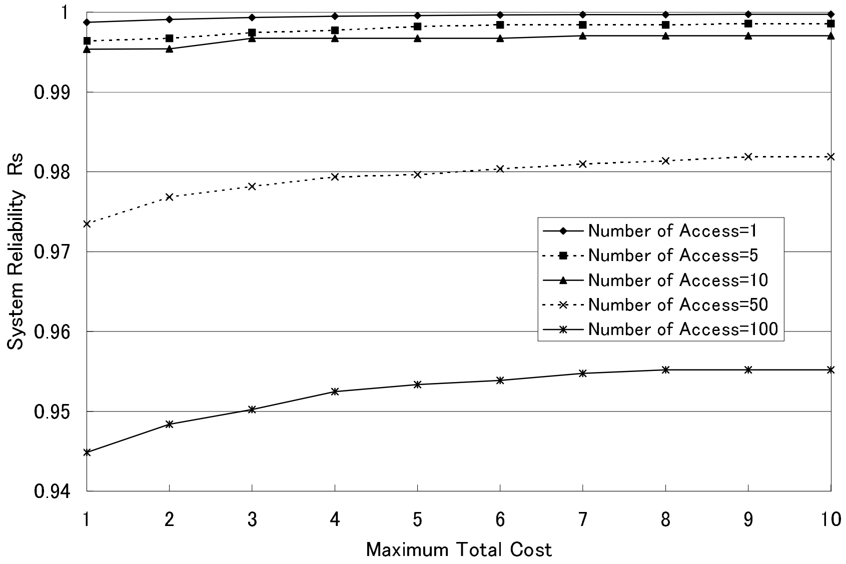


Fig.4.5. A trade-off between the reliability and the cost

10, 50, 100. As shown in this figure, the larger the maximum total cost, the higher the system reliability. Also, in the case that the average number of accesses has a larger value, the system reliability is lower as mentioned in Fig.4.3. In the following figures, we may see how mirrors and files are distributed in this case.

Next, in Fig.4.6 and 4.7, we show the distribution characteristics of mirrors and files regarding the total cost restriction for the different values of the average number of accesses. These figures show the number of mirrors Nm and the distribution rate Rdf vs. the normalized value of the maximum total cost. In the both figures, as the average number of accesses is larger, more files tend to be distributed on more mirror servers, so that it will be disinclined to forward requested files from remote servers, thus reducing the communication cost. Also, as shown in Fig.4.6, when the maximum total cost is larger than 6, in all cases, the number of mirrors is equal to 20, the maximum number of mirror servers. This result means that mirror servers should be distributed as much as possible, in order to maximize the system reliability. We may imagine that the number of mirrors will increase if there are more nodes which can work as mirror servers. On the other hand, as in Fig.4.7, as the total cost restriction becomes loose, the distribution rate Rdf tends to settle down to the specific value for each number of accesses individually. For example, in the case that the average number of accesses is equal to 1, 5 or 10, *Rdf* settles down to 5.58. This result shows that each file should be assigned to five or six nodes so that the system reliability is maximized, under the above system parameters.

In Fig.4.8, we show the trade-off between the storage cost and the communication cost. This figure shows each cost and the distribution rate of files vs. the maximum total cost. Here, we set the average number of accesses to 3. As the maximum total cost becomes larger, the distribution rate *Rdf* and the storage cost Cs become larger, while the communication cost Ct becomes smaller. This tendency is due to the fact that files are stored in more mirror servers to improve the system reliability.

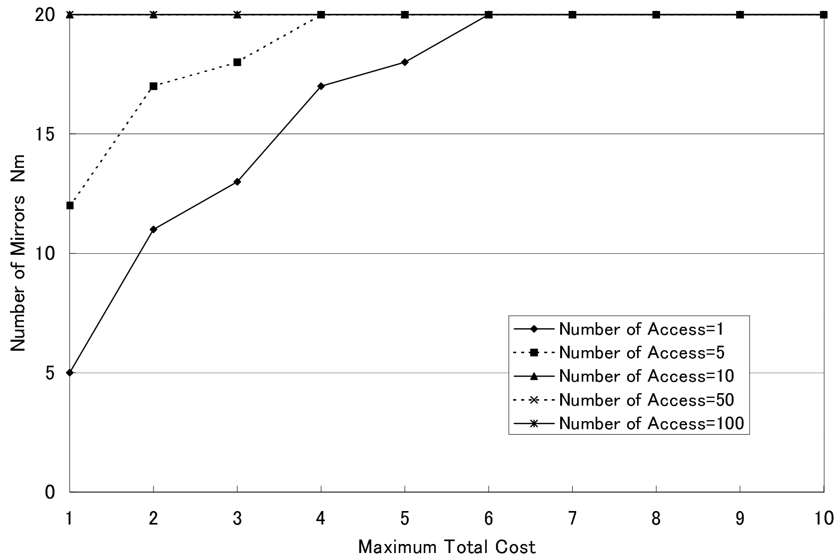


Fig.4.6. Distribution characteristics of mirrors based on the cost restriction

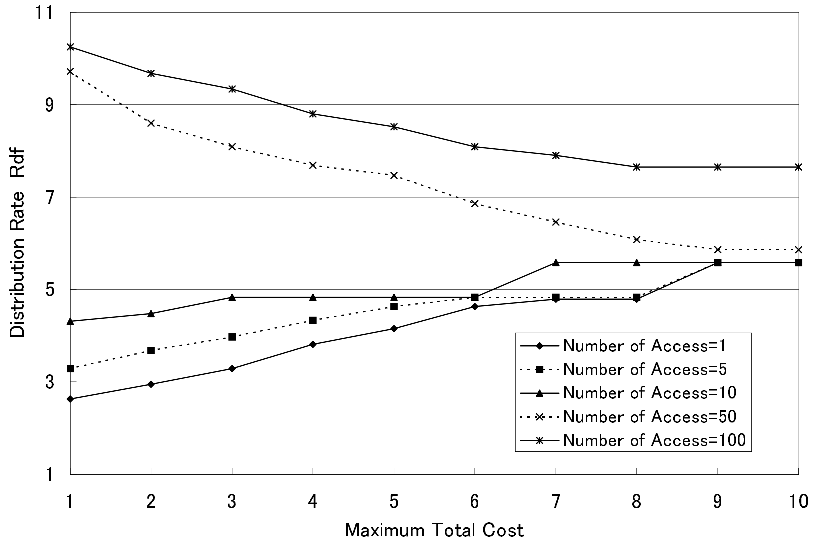


Fig.4.7. Distribution characteristics of files based on the cost restriction

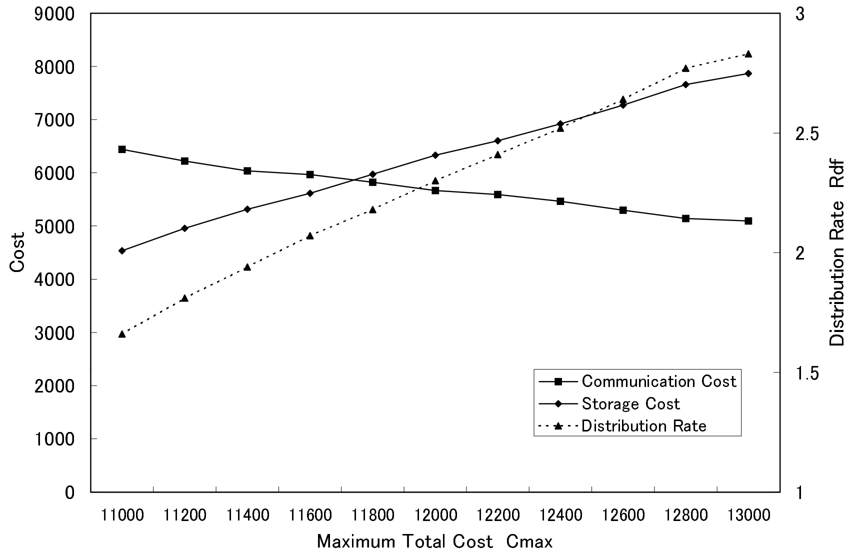


Fig.4.8. A trade-off between the storage cost and the communication cost

4.3.5 Characteristics on Delay Restriction

We show the characteristics of the mirror distribution regarding the communication delay restriction. In Fig.4.9, we show the system reliability and the number of mirrors vs. the maximum communication delay $Dt_{\max}$. Here, we set the average number of accesses to 10. As shown in this figure, when the maximum communication delay is larger, the system reliability improves. On the other hand, the number of mirrors Nm is not affected by the delay restriction and always has an identical value 20, which is the maximum number of mirror servers. This result shows that we should assign as many mirrors as possible to maximize the system reliability.

Next, in Fig.4.10, we show the distribution characteristics of files regarding the communication delay restriction for different numbers of accesses. As the communication delay restriction becomes looser, the distribution rate Rdf becomes smaller, while the system reliability is improved as you can see in Fig.4.9. As we can also see in Fig.4.7, the distribution rate settles down to the specific value for each number of accesses. More files should be distributed to more nodes to reduce the communication delay when the restriction is tighter. As the restriction becomes looser, fewer

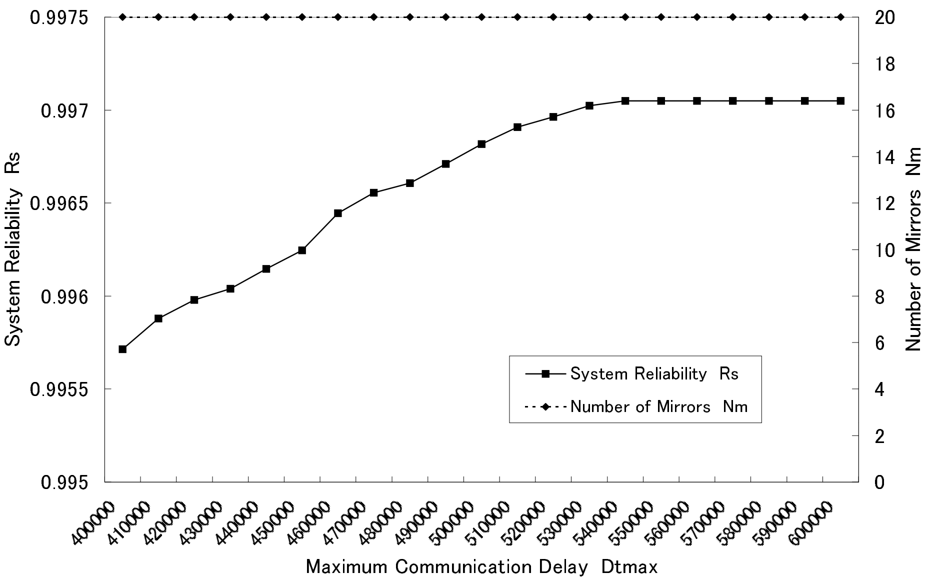


Fig.4.9. Distribution characteristics of mirrors based on the delay restriction

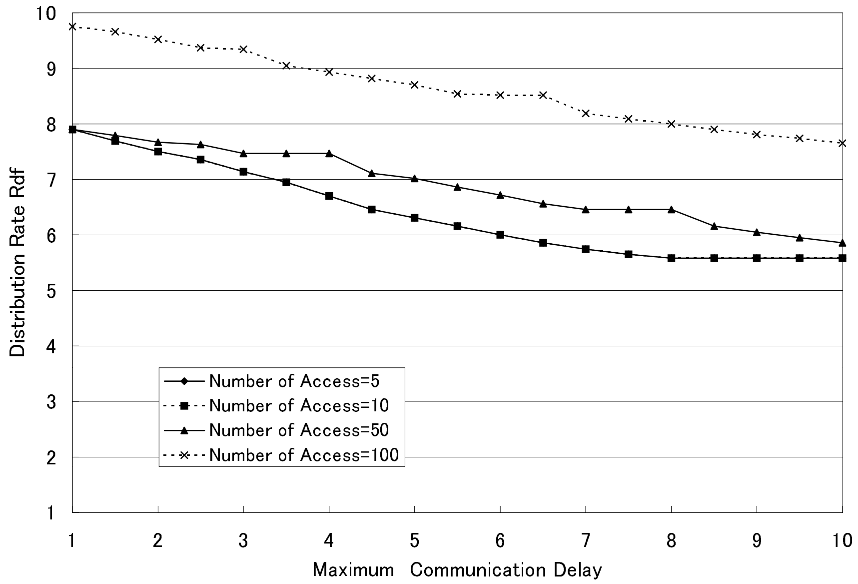


Fig.4.10. Distribution characteristics of files based on the delay restriction

files are distributed and the reliability of each node improves, since the number of files on each node decreases. It is considered this is why the system reliability improves when the communication delay restriction becomes looser.

From both figures in this section, we may see quantitatively the trade-off between the reliability and the delay.

4.3.6 Characteristics on Mirror Restriction

We show the characteristics of the mirror distribution regarding the restrictive conditions of the mirror cost and the capacity of the mirror servers.

In Fig.4.11, we show the distribution characteristics of mirrors and files vs. the maximum mirror cost $Cf_{\max}$. Here, we set the average number of accesses to 10. Also, in Fig.4.12, we show the system reliability vs. the maximum mirror cost $Cf_{\max}$ for the different values of the average number of accesses respectively. As the maximum mirror cost increases, the number of mirrors and the distribution rate increase and the system reliability improves. In the case that the mirror cost restriction is tighter, all nodes are not able to work as mirror servers and files cannot be distributed to more nodes. As the mirror cost restriction becomes looser, all nodes start working as mirror servers, which improves the system reliability. This result clearly shows the trade-off between the system reliability and the mirror cost.

Next, we show the effect of the mirror capacity restriction. In Fig.4.13 and 4.14, we show the distribution rate of files and the system reliability vs. the mirror capacity B_i for different values of the file size F_{ik}. Here, we set the average number of accesses to 3. As the mirror capacity becomes larger, more files tend to be distributed, and the system reliability improves. However, after the mirror capacity reaches a specific value, we cannot see the variation in the distribution rate and the system reliability. For example, in the case that the maximum size of files is 300 [Mbyte], 4.5 Gbyte is enough capacity for the optimal storage of files such that the system reliability is maximized. When the mirror capacity is larger than 4.5, the distribution rate of files has an identical value 5.58, as described in the paragraph on the characteristics using the cost restriction. For analytical simplicity, in this study, the number of distinct files is set to a value still smaller than that of real systems. In real systems, we must need more capacity on the nodes for the optimal storage of files.

From these numerical results, we see that there are benefits in assigning as many mirror servers as possible, while the distribution rate of files is dependent on various conditions. However, this result with regards to the number of mirror servers might be derived from our read-only assumption. The more general situation of updating files should be considered in the future.

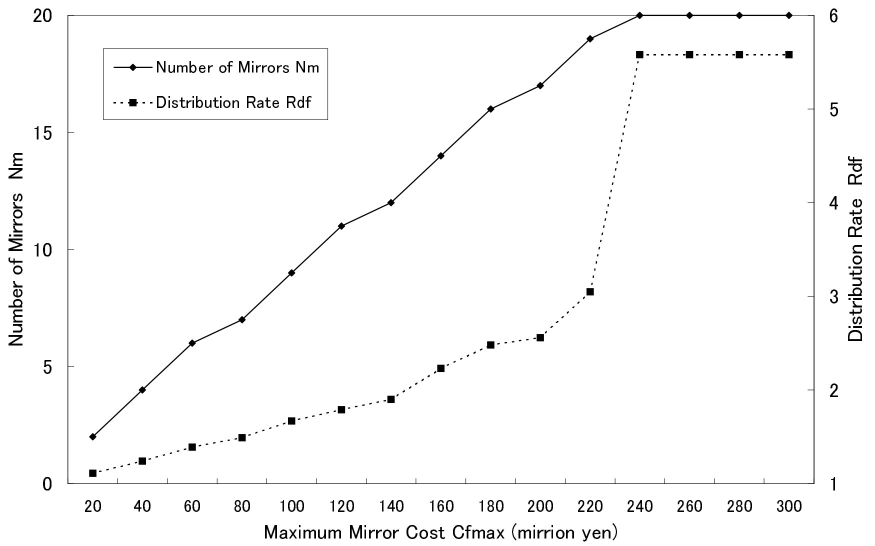


Fig.4.11. Distribution characteristics of mirrors and files based on the mirror cost restriction

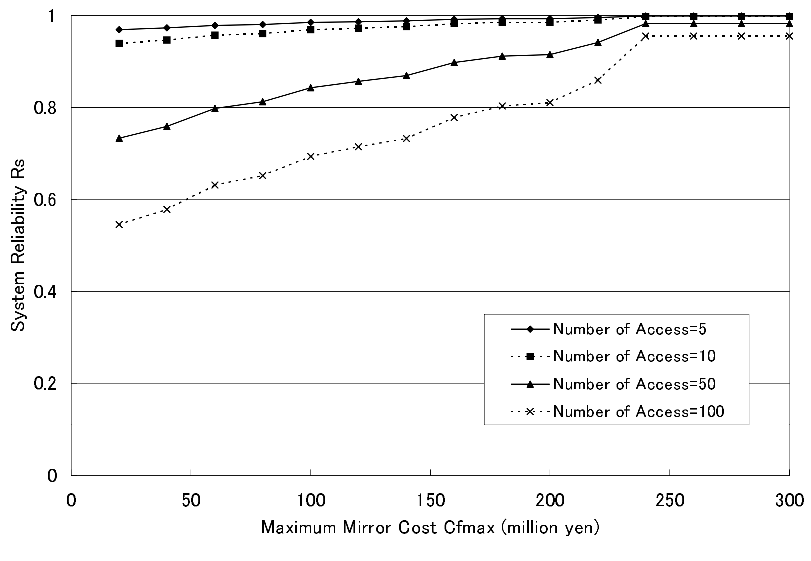


Fig.4.12. A trade-off between the system reliability and the mirror cost

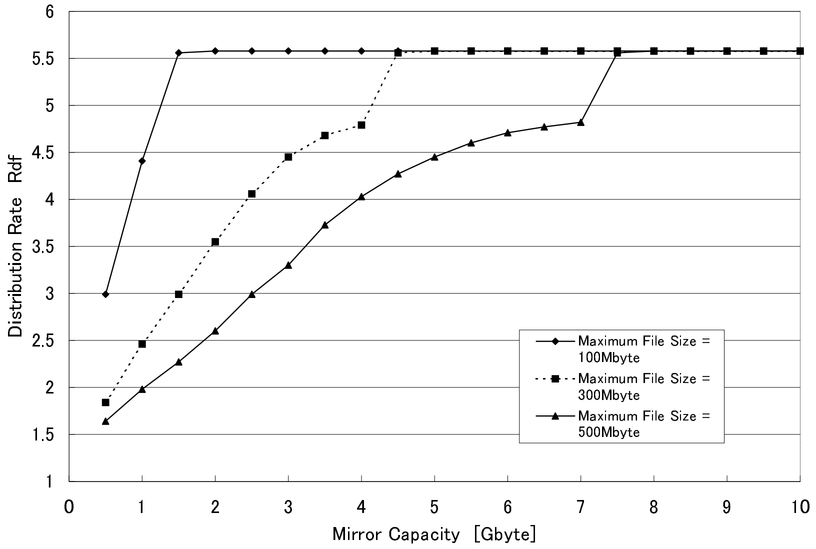


Fig.4.13. Distribution characteristics of files based on the capacity restriction

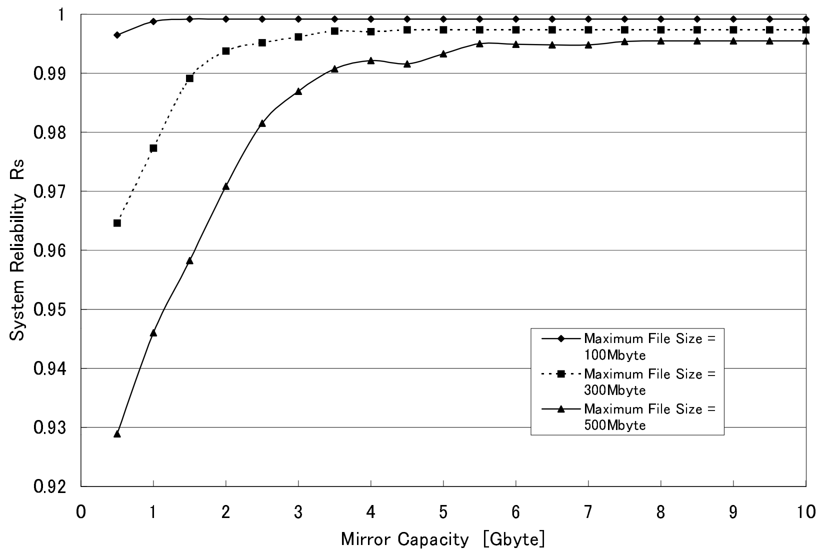


Fig.4.14. System reliability characteristics based on the capacity restriction

4.4 Reliability-Based Dynamic File Allocation in Mobile Networks

In this section, We now change our focus to dynamic networks, where we had only been considering static networks until now. System reliability is more likely to decline in mobile environments. Recent advances in wireless communication technology have made dynamic file allocation in mobile networks an important issue. It is very important to consider reliability-based dynamic file allocation in mobile networks as well as to consider file allocation in static networks that we have done.

4.4.1 Challenges for Dynamic Database Management

The rapidly expanding technology of wireless communications have made it possible for mobile users to access information services anywhere and at any time [77][78][79]. A number of novel mobile computing systems, such as tele-medicine systems, real-time traffic information and navigation systems, and mobile Internet stock trading systems, are emerging as mobile users require instant access to information using their palmtops, personal digital assistant(PDA), and notebook computers. For instance, an investor can monitor the recent situation of the stock market from his limousine, using a mobile computing environment that provides connectivity to the computer of his dealer. On the other hand, a sales manager of a mobile communication provider can monitor the work of his or her own sales persons through accessing their private data. After summarizing the obtained data, he or she can write the most recent sales report.

It is natural that there are many differences between static networks and dynamic networks. The main difference in dynamic networks is clearly its mobility. Some researchers believe that moving from static to mobile environments will have a similar scope of influence as migrating from centralized to distributed systems [77]. The normally asked question "centralized or distributed," will now be enhanced to "static or mobile".

Techniques for distributed database management have been based on the assumption that the location of hosts in the distributed system does not change and the connections among hosts do not change. However, in mobile computing environment, these assumptions are no longer valid. Hence, distributed database management will be changed due to mobility. Resources will have to be dynamically reconfigured in response to the mobility of the users. This involves dynamic

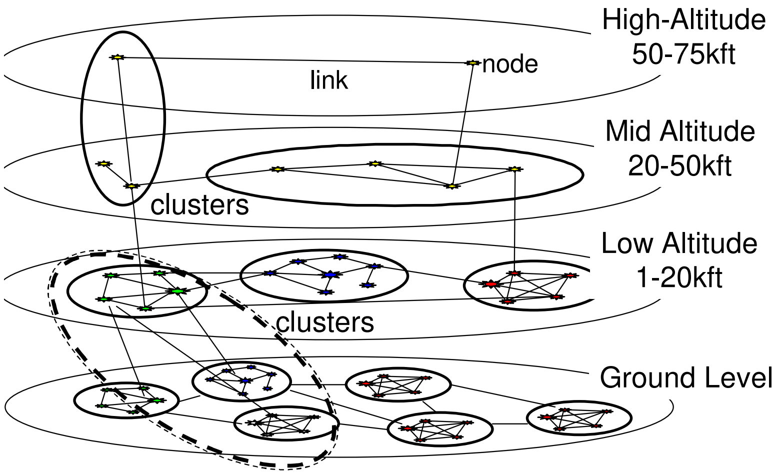


Fig.4.15. Hierarchical system architecture of of mobile networks

replication of data as well as services.

Mobile computing technology not just only improves the distribution and flow of information, but at the same time, it also greatly increases the functionality of real-time database applications. The realization of "instant" information access over mobile networks relies on real-time processing of transactions and it makes the timeliness of data accesses over an important issue.

In particular, in this study, Our objective is to provide reliable and time critical database information services in a more dynamic and critical environment. In Fig. 4.15, we show the system architecture of mobile networks that we basically consider. In this system, information is stored hierarchically. Each group(cluster) has a coordinator and a vice coordinator. The coordinator contain the brief information of each group member and the general information for the whole group, e.g. mission which have to be done as a whole group etc. The vice coordinator is a backup of the information stored in the coordinator.

Due to its mobility, there are many challenges peculiar to dynamic database systems, such as

- Decreased reliability

- Dynamic changes in data availability

  - Database nodes damaged, appearing and disappearing unpredictably
  - Communication link failures

- Dynamic changes in users access patterns to each file

- Time critical requirement for information access and services

In dynamic and critical environment, it is considered that database nodes has high probability of being appearing and disappearing from each cluster unpredictably because of being damaged, moving to other regions, fading in wireless communication, or some other reasons. Also, wireless communication link is more likely to fail. Consequently, reliability in mobile networks decreases and availability of each file from users in each cluster changes dynamically.

On the other hand, user nodes is also mobile as well as database nodes. They move to outside a cluster frequently and this causes the number of users that request each file to change. That is, the need of each file to be in each cluster also changes dynamically. Actually, users access patterns to each file changes even in static networks as time goes by. For instance, in VoD networks, although a file may be extremely popular when it first becomes available, it is natural that its popularity decreases over time. There have been many studies concerning dynamic file allocation according to the changes in users access pattern across time in static networks. In this study, in addition to similar perspective to these studies, we consider changes in users access patterns because of their mobility.

Moreover, in this study, we consider time critical situations in query response requirements. Here, the allocation of resources as well as the scheduling of query processing can play an important role in query response time. For example, replication of data sources will reduce the data access time.

Considering these issues, in this study, we propose a dynamic file allocation strategy to best satisfy the reliability and critical query response time requirements in mobile networks. Because of high potential of failure of database nodes, it is better that we keep a copy of files in more than one node to ensure high reliability. And if one node is not available, we may still get some information from other nodes where replication of file is stored. However, to keep all the replicated data consistent in a changing environment increases the communication cost and delay. Therefore, the degree of replication will be affected by the data update rate.

To remedy this issue, weak concurrency control techniques may be used for certain data type to reduce the communication cost. Similarly, conventional two-phase commit protocol will be too time-consuming to commit a transaction in a real-time system. Weaker commit protocol may be used to reduce time to commit a transaction in a dynamic distributed system. Though we do not consider the details of these kind of issues in this study, they should be considered later on. We also consider developing a query scheduling policy and combining it with the dynamic file allocation strategy as a future work.

4.4.2 Innovative Strategy for Dynamic File Allocation

We present the strategy of dynamic file allocation. This strategy consists of two phases: 1) Periodically generating optimal file allocation via a heuristic algorithm based on the complete system state and 2) Using knowledge-based policy to reallocate resources based on the partial system state changes. In the first phase, we periodically use a heuristic algorithm to find the allocation of files which minimize the operating cost subject to a set of constraints, such as reliability and response time. Because of critical time requirement, we do not apply 0-1 integer programming formulation to this heuristics. In the second phase, we use the knowledge-based reallocation policy to dynamically reallocate files when the state of the system changes significantly. The goal of the proposed dynamic file allocation strategy is to satisfy the reliability and real time requirements under the mobile operating environment.

System Model

A distributed mobile network (Fig. 4.16) consists of a number of clusters that is formed base on their location proximity. Each cluster may include several mobile nodes that can be database servers (nodes) as well as user nodes. These nodes may move from one cluster to another which are tracked by the resource registry wchich contains in the Resource Allocator(RA). The resource registry contains resource information for the entire network, such as nodes, link, files, data type, security level, and estimate retrieval time. Sensors will be used to monitor this information and report to the resource registry using pro-active and pattern matching technology. The resource registry directs the resource allocator to reallocate files as needed, according to the changes in the state of the system.

There exist n nodes in the system, and each node is denoted by N_i $(1 \leq i \leq n)$. While being user nodes, all nodes can also simultaneously be database nodes by storing files. The storage capacity of a node N_i is B_i. The number of hops between nodes N_i and N_j is h_{ij}.

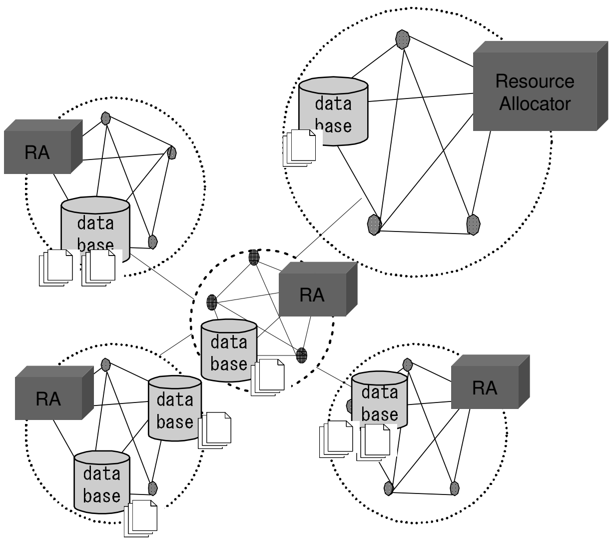


Fig.4.16. A model of mobile distributed networks

The number of distinct files is m, and each file is denoted by M_k $(1 \leq k \leq m)$. It is assumed that the size of a file M_k is denoted by F_k. Each database node can store these files as long as the total size of files does not exceed the storage capacity. The number of replicated copies of a file M_k is r_k.

The access frequency for a file M_k at a node N_i is denoted by a_{ik} and the update probability after each access is P_u. We also We give of them in the following. assume that the update file size of a file M_k is f_k. When users access each file, they have response time requirement for each query and time requirement for an access for a file M_k at a node N_i is denoted by TR_{ik}.

Moreover, the resource registry has to contain communication rate of each link, service rate of each node and failure rate of each node and link to estimate system reliability and response time.

Based on this information provided by the resource registry, the resource allocator uses a dynamic file allocation strategy to reconfigure the resources for satisfying such system performance as reliability and response time requirements, etc.

Strategy for Dynamic File Allocation

A dynamic file allocation strategy is developed for handling real time dynamic system state changes. One of the key consideration is that the time to generate the optimal allocation must be included in the response time requirement since file allocation is well known of its time complexity in optimization. For this reason, we propose a two phase allocation strategy: 1) Periodic optimal file allocation via a heuristic algorithm based on complete system state information, and 2) knowledge-based file reallocation based on the system state changes between periodic allocation periods.

First, the initial file allocation (including replicated copy) is carried out based on the resource information given by the resource registry via a heuristic algorithm. To satisfy the critical time requirement, we do not apply 0-1 integer programming formulation to this heuristic algorithm, since it would take much more time when using 0-1 integer programming. However, because of the dynamic changes in the system states (e.g. database nodes disappearing or appearing in a cluster due to node migration) and the critical response time requirement, even the heuristic technique

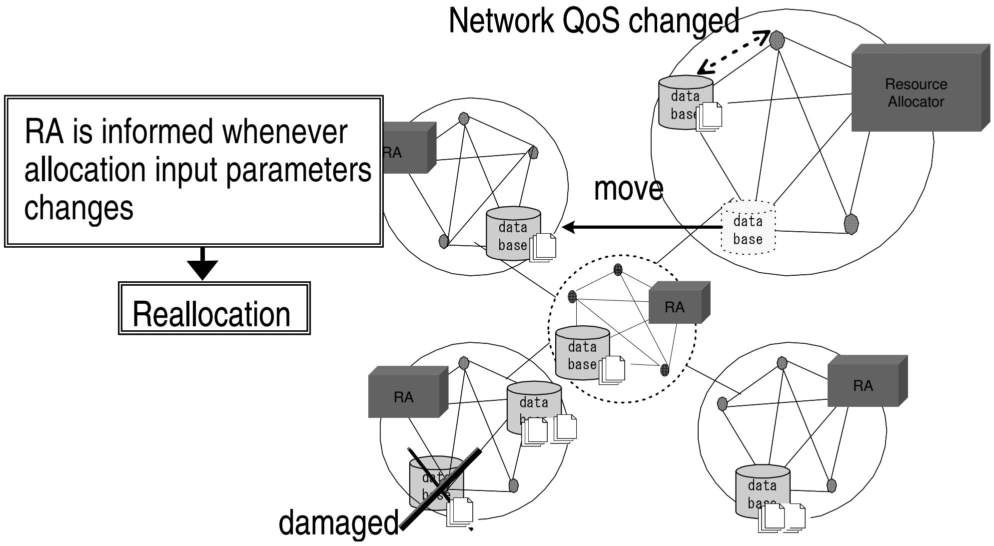


Fig.4.17. File reallocation process

may be insufficient in terms of time to run the algorithm.

As a remedy, a set of knowledge-based file reallocation policies established a prior will be used to reallocate the files based on the changing of the system state. Since the knowledge-based reallocation policy is only based on limited set of conditions, the knowledge-based reallocation althouigh is rapid in reallocation decision but can only be response to partial state changes.

Clearly, repeatedly applying such polichy will yield file allocation away from the optimal. Thus, we periodically use the heuristic algorithm to re-optimize the file allocation for the complete system state changes.

From the next section, we give full details of each phase.

4.4.3 Heuristic File Allocation Algorithm

The objective of optimal file allocation is to find the file allocation that minimize the operating cost (communication cost and update cost) for processing a task subject to a set of constraints, such as storage capacity of each node, the reliability and response time requirements of the processing task etc.

A* Algorithm

The heuristic algorithm we use is based on similar principles to the A* algorithm [34][35][82][83] that searches for the optimal solution in the state space search tree. The A* algorithm prunes the expanded tree more effectively than the Branch and Bound algorithm because at each step it uses also an estimation of the cost incurred in getting from the step to the final solution of the problem.

The optimal file allocation problem is formulated as a state-space search problem. In a state-space search problem, each state description is denoted by a node in the state-space search tree. A problem starts from the initial state s_0 and expands states one by one (expanding a state s means generating all immediate successors of each node). In this algorithm, a numerical value $f(s)$ is assigned to each state, consisting of two terms:

$$f(s) = g(s) + h(s)$$

where $g(s)$ is the known cost incurred in getting from s_0 to s, and $h(s)$ is a heuristic function that estimate the additional cost that will be incurred in getting from s to a final state, the solution of the problem. To choose which state to expand next, it always selects for node expansion the state s having the smallest value for $f(s)$. The performance of A* algorithm depends on the choice of $h(s)$. The better this function estimates the cost to a final state, the faster the algorithm. This brings up the issue of admissibility of heuristic functions. The heuristic function $h(s)$ is *admissible* if it always returns a lower bound on the additional cost that will be incurred in getting from s to a final state. If $h(s)$ is admissible, then the A* algorithm is guaranteed to find a final state (leaf node) s_F such that the cost of getting from s_0 to s_F is minimal among all final states.

State Space for Search

Each possible file allocation plan is defined as a state in the search space for the optimal file allocation problem. Let all file allocation plans for each each file M_k be $\mathbf{P_k} = \{P_{k1}, P_{k2}, \ldots, P_{kx_k}\}$, where P_{kj} is the jth allocation plan for the file M_k and x_k is the number of possible allocation plans for the file M_k. Each allocation plan P_{kj} is a set of nodes and includes all nodes that store an original or a copy of the file M_k in this plan, that is $P_{kj} = \{N_{j_1}, N_{j_2}, \ldots N_{j_{n(jk)}}\}$ where N_{j_e} is the eth node and $n(jk)$ is the number of the nodes in this allocation plan. The state space is constructed as follows:

(1) Every state s is an m-tuple $\langle P_{1j_1}, P_{2j_2}, \ldots, P_{mj_m}\rangle$, where for each k, either $P_{kj_k} \in \mathbf{P_k}$ (in which case s suggests that a plan P_{kj_k} is used to allocate a file M_k), or else $P_{kj_k} = \text{NULL}$ (in which case s does not suggest any plan for a file M_k yet). The cost of s, denoted by $g(s)$, is the operating cost required when allocating files according to the allocation plans in s.

(2) The initial state is the state $s_0 = \langle \text{NULL}, \text{NULL}, \ldots, \text{NULL}\rangle$, and the states $s_F = \langle P_{1j_1}, P_{2j_2}, \ldots, P_{mj_m}\rangle$ with $P_{kj_k} \neq \text{NULL}$, for all k, are the final states.

(3) Given a state $s = \langle P_{1j_1}, P_{2j_2}, \ldots, P_{mj_m}\rangle$, let

$$next(s) = \begin{cases} \min\{k|P_{kj_k} = \text{NULL}\} & \text{if } \{k|P_{kj_k} = \text{NULL}\} \neq 0 \\ k+1 & \text{otherwise.} \end{cases} \tag{4.24}$$

(4) Let the state s have at least one NULL entry and $a = next(s)$, then the immediate successors of s include every state $s' = \langle P_{1j'1}, P_{2j'_2}, \ldots, P_{mj'_m}\rangle$ satisfying the following properties:

$$P_{kj'_k} = P_{kj_k}, \quad \text{for } 1 \leq k < a \;;$$

$$P_{aj'_a} \in \mathbf{P_a} \;;$$

$$P_{kj'_k} = \text{NULL}, \quad \text{for } a < k \leq m \;.$$

The cost of the transition from s to s' is the additional cost needed to process the new plan $P_{aj'_a}$, given the (intermediate or final) results of processing the plans in s.

Evaluation Function

As in the A* algorithm, our heuristic file allocation algorithm use the evaluation function for a state s, $f(s) = g(s) + h(s)$. Since the objective function is the operating cost, we evaluate $g(s)$ and $h(s)$ using the operating cost function.

It is assumed that the operating cost consists of the communication cost C_t and the update cost C_u. Here, $C_t(h_{ij})$ is the communication cost coefficient which depends on the number of hops between nodes and C_u is the update cost coefficient which is common for all files. The cost function of a jth file allocation plan for a file M_k is formulated as follows.

$$\text{Cost } C = \text{Communication cost } C_t + \text{Update cost } C_u$$

$$C(P_{kj_k}) = \sum_{i=1}^{n} \Big(a_{ik} Ct(h_{ij_{\min(i)}}) F_k + a_{ik} P_u C_u f_k r_k \Big) \tag{4.25}$$

Here, $j_{\min(i)}$ is the index of the node $N_{j_{\min(i)}}$ which has the minimum number of hops from the node N_i among all the nodes included in the jth allocation plan P_{kj}.

Using this cost function, the evaluation function is formulated.

$$g(s) = \sum_{k=1}^{next(s)-1} C(P_{kj_k}) \tag{4.26}$$

$$h(s) = \sum_{k=next(s)}^{m} \min_{j_k} \left[C(P_{kj_k}) \right] \tag{4.27}$$

Restrictive Conditions

In the optimal file allocation problem, we also have a set of constraints, that is storage capacity, reliability, and query response time.

1. Storage capacity

   Each node N_i can store files as long as the total size of files does not exceed the capacity of the node, that is

$$\sum_{k:\text{All the files stored in the node } N_i} F_k \le B_i \tag{4.28}$$

2. Reliability(Availability)

   Let R_k be the reliability (availability) of the file M_k In mobile networks, the reliability of each file changes dynamically according to changing in the state of the system. Therefore, it is important issue to reallocate files so that we can maintain high reliability in the system. Here, R_n denotes the reliability of the node and R_c denotes the reliability of the communication channel. Based on the information of failure rate of each node and link given by the resource registry, we can estimate R_n and R_c. We estimate R_n to be the average of the reliability of all nodes which store the file M_k. On the other hand, R_c is estimated to be the product of the reliability of the channels which is used to access the file M_k. Let $R_{\min}$ be defined as the minimum reliability, and then the reliability R_k has to be greater than $R_{\min}$ for all k, i.e., for all the files.

$$R_k = R_n \left[1 - (1 - R_c R_n)\, r_k\right] \ge R_{\min} \tag{4.29}$$

3. Query response time

   We have the network information such as the network topology, the access rate, and the update rate. Based on this information, we can estimate the traffic at each node and offered traffic to the network. Here, let L denote the number of links, λ_l denote the total traffic on the lth channel, and γ denote the total offered traffic to the network. Moreover, we assume that $1/\mu'$ is the average transmission time of message, $1/\mu$ is the average transmission time of data, t_p is the propagation delay, and t_n denotes the nodal processing time, which is common for all nodes. This information enable us to estimate average network response delay. The query response time T consists of the network response delay T_n and the query processing delay t_q. Let $T_{\max}$ be defined as the maximum allowable time, and then the query response time T has to be less than $T_{\max}$. $T_{\max}$ is given based on the response time requirement for each query TR_{ik}. If the query response time is greater than $T_{\max}$, then we need to increase the replication numbers of certain files that causes traffic bottleneck to different nodes.

Query response time T

$= 2 \times$ network response delay T_n + query processing delay t_q

$$T = 2\sum_{l=1}^{L} \frac{\lambda_l}{\gamma} \left(\frac{\lambda_l/\mu'}{\mu' - \lambda_l} + \frac{1}{\mu} + t_p + t_n \right) + t_q \leq T_{\max} \tag{4.30}$$

Heuristic Algorithm with Rule-Based Search Space Reduction

Our heuristic algorithm is characterized by rule-based pruning of the original search space. This algorithm not only prunes down the search space by the estimating cost but also diminishes the original search space itself by using a set of rules. As a result, only a subset of file allocation plans, rather than the entire space, is searched. Thus, in our heuristic algorithm, we first create the state-space search tree which consists of the set of states selected by the rules, and then search this search tree in a similar way as the A* algorithm.

The following is a set of sample rules that can be used for reducing the original search space:

- Select the node to allocate the file such that it has the highest reliability
- Select the node to allocate the file such that it has the most critical time requirement for this file
- Select the node to allocate the file such that it has the highest access frequency for this file

We apply these rules to determine each file allocation plan P_{kj}. The candidate allocation plan P_{kj} are the set of nodes that satisfy one or more of these rules. We may select several nodes for each rule, for example, for the third rule, we can select not only the node that has the highest access frequency, but also the second highest one, the third highest one, ..., etc. In this case, we need to determine how many cases we allow for each rule. Each file allocation plan is the combination of the nodes that are selected based on these rules.

In Fig. 4.18, we show an example of the state space search tree. Each node in the search tree is corresponding to each file allocation plan determined by rules. Here is the concrete algorithm of our heuristic file allocation algorithm

1. Create an array OPEN and insert the initial state s_0. into OPEN as the first factor.

2. Remove from OPEN the state s that has the smallest value for $f(s)$. If there is more than one such node, then we select the one at the deepest level.

3. Generate the successor states of s, and insert them into OPEN.

4. Repeat the procedures from 2 to 4 until s is a final state.

5. Return s as the optimal solution.

We use this heuristic file allocation algorithm to determine the initial file allocation and then re-optimize the file allocation periodically based on changing of the system states. We need to study such behavior of this heuristic algorithm as the run time for selected problem sizes, the accuracy of this algorithm, the effectiveness of the rules used for pruning the search space.

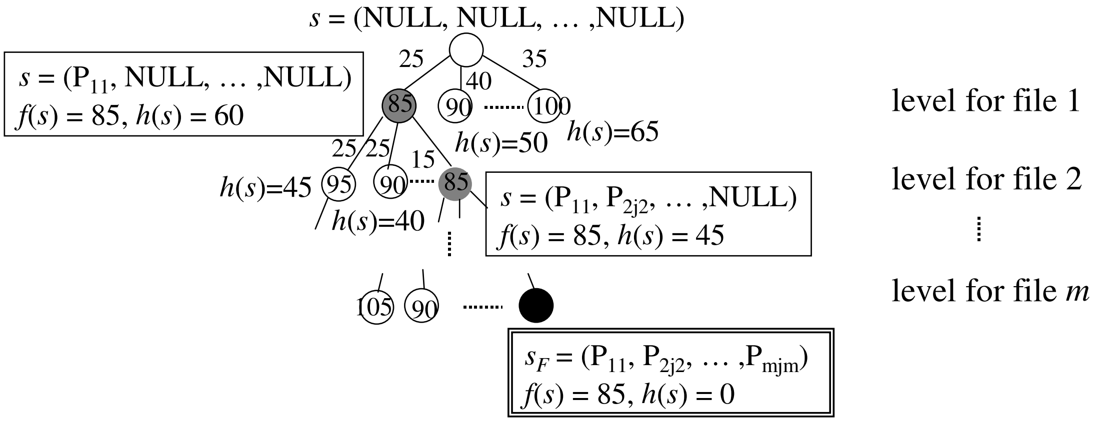


Fig.4.18. An example of the state space search tree

4.4.4 Knowledge-Based Dynamic File Reallocation

In this study, we consider providing reliable and time critical database information services in mobile networks. In mobile database systems, database and user nodes are migrating from one cluster to another continuously. It is difficult to keep satisfying the requirements for the reliability and the response time by only periodically using the heuristic algorithm. For the purpose of dealing with this issue, the knowledge-based file reallocation is considered to be efficient.

The knowledge-based reallocation policy is used to provide rapid responce in accordance to the continuous changes in the state of the system. We generate a set of rules a prior to determine whether to reallocate files for a specific situation. In this study, we assume that backup copies of files are available for access when needed. Therefore, to allocate or reallocate a file means to activate a backup copy of the file, thus provide rapid response to real time system needs. Files will be removed from the nodes when they are no longer needed which corresponds to deactivate a backup copy of the file. While files is deactivated, it is unnecessary to keep the data of the files consistent and update the files constantly. This greatly decreases the update cost. When the access rate of a file increases to satisfy the required response time, the policy will re-activate the file. The following is a set of knowledge-based reallocation policies that based on limited state change conditions:

- Whenever a database node disappears select a nearby backup node with similar database contents as a replacement and activate the backup copy if it is deactivated.
- Whenever a redundant database node appears, the system evaluates(e.g. update cost vs. response time) whether to deactivate a backup database copy.
- Whenever the response time degrades, to share the load, activate a backup database copy near to the users that request the file.
- Whenever the response time improves, to reduce update overhead, the system evaluates whether to deactivate redundant replicated copies.
- Whenever the time requirement becomes more critical, activate one or more backup database copies that near to the users that request the file.
- Whenever the time requirement become less critical, the system evaluate whether to deactivate redundant replicated copies.

- Whenever the access frequency increases over a certain percentage, activate a backup database copy that is near to the users that request the file which becomes more frequently accessed.

- Whenever the access frequency decreases, the system evaluate whether to deactivate redundant replicated copies.

We need to study the effectiveness of knowledge based file reallocation through simulation under dynamic environments, and also use simulation to determine the parameter values (the boundary to determine whether to deactivate a file) used in the policies under selected dynamic environments.

4.5 Summary

In this chapter, firstly, we have proposed the optimal mirror allocation model for the purpose of load balancing in network servers. In this model, we have formulated the optimal allocation problem of mirror servers such that the system reliability is maximized subject to the cost and the delay, as a 0-1 integer programming model. For numerical analysis, we used an approximate method, based on the Greedy Method in order to deal with more large-scale systems. From comparison results between this approximate method and exhaustive search, we have shown that this approximate method is very useful in terms of accuracy and calculation time.

As for the numerical results, we have analyzed the characteristics on load balancing using distributed mirror allocation. We have quantitatively shown the general characteristics, such as the improvement of the system reliability using the mirror distribution, the trade-off between the system reliability and the cost, the effects of the cost and the delay restriction, etc. In particular, we have obtained the followings results:

1. There are benefits in assigning as many mirror servers as possible.
2. The extent of the file distribution is considerably dependent on several conditions, such as the cost restriction, the delay restriction, etc.
3. When systems are released from the restrictive conditions, the distribution rate of files settles down to a specific value. That is, we may be able to find how many copies of each file are required to be stored in the whole systems so that the system reliability is optimized.

Consequently, we can conclude that our optimal mirror allocation model is very useful in evaluating various issues in the construction of multi-media networks.

Secondly, we have also presented the dynamic file allocation strategy for mobile environments. We have mentioned the challenges for dynamic database management in mobile networks where system reliability is more likely to decline. Since a number of novel mobile computing systems are rapidly emerging as mobile users require instant access to information, it is very important to consider reliability-based dynamic file allocation in mobile networks.

This strategy consists of two phases: 1) Periodically generating optimal file allocation via a heuristic algorithm based on the complete system state and 2) Using knowledge-based policy to reallocate resources based on the partial system state changes. We have presented the heuristic file allocation algorithm with rules for the pruning of the original search space for the sake of critical time requirement. Also we have listed the policies that are used for the dynamic file reallocation according to the changes in the state of systems.

We believe this dynamic file allocation strategy will contribute to solve design issues in constructing reliable and time critical mobile networks. Simulation experiments are needed for parameter estimation, knowledge-based rule refinement, as well as showing the effectiveness in satisfying user and system performance requirements of the proposed two phase dynamic file allocation strategy for operating in a mobile environment.

Chapter 5

Conclusions

The work presented in this thesis has been devoted to the optimal design of system resource allocation in distributed networks. To develop distributed systems, it is very important to find the effective method of the optimal resource allocation considering various design issues.

Mainly, we have proposed two types of optimization models from different perspectives, which includes the cost-based optimal file allocation model and the reliability-based optimal mirror allocation model. In the former, we aim to find the optimal file allocation such that the total cost is minimized subject to the total delay in view of a trade-off between costs and delays. On the other hand, in the latter, our objective is to find the optimal allocation of mirror servers themselves as well as files such that the reliability is maximized subject to the cost and the delay restriction. Both optimization models have been formulated as the 0-1 integer programming models and the similar approximate methods based on a greedy algorithm were used for numerical calculation in order to deal with more practical large-scale systems.

Our objectives in developing these optimization models are to provide the useful tools to evaluate various issues such as reliability, cost, response time, etc. in constructing practical systems. Consequently, we have been able to show quantitatively the general characteristics on distributed file allocation and mirror allocation, such as trade-offs between the cost and delay as well as between the reliability and cost, the improvement of system reliability due to mirror distribution, and so on. We also have analyzed the sensitivity of system parameters on the optimal file allocation model and made clear the influence between principal parameters. In addition, as the optimization results of distributed mirror allocation, we have obtained the meaningful results for practical systems;

- There are benefits in assigning as many mirror servers as possible.
- The extent of the file distribution is considerably dependent on several conditions.
- When systems are released from the constraints, the distribution rate of files settles down to a specific value.

These results indicate that when considering various design issues of distributed networks that are increasingly needed, our optimization models we have proposed in this study are very beneficial.

In this study, we have also considered dynamic file allocation in mobile environments, which has become increasingly needed. In particular, our objective is to provide reliable and time critical database information services in a more dynamic and critical environment. For this purpose, we have developed the innovative strategy for dynamic file allocation, which consists of two phases, i.e., 1) Periodically generating optimal file allocation via a heuristic algorithm based on the complete system state and 2) Using knowledge-based policy to reallocate resources based on the partial system state changes. Simulation experiments are needed for parameter estimation, knowledge-based rule refinement, as well as showing the effectiveness in satisfying user and system performance requirements of the proposed two phase dynamic file allocation strategy for operating in a mobile environment. We believe that this two phase dynamic file allocation strategy is capable of contributing to evaluate various issues in the design of mobile database systems.

Through this study, we have developed the useful design tools for the optimal system resource allocation in distributed systems, in view of several essential issues including the cost and the reliability. I hope that this study would make some contributions towards the design of distributed network systems.

Bibliography

[1] S.Ceri and G.Pelagatti, Distributed Databases Principles and Systems, McGraw-Hill Book Company, 1984.

[2] G.Coulouris, J.dollimore and T.Kindberg, Distributed Systems Concepts and Design Second Edition, Addison-Wesley Publishing Company, 1994.

[3] S.Mullender, Distributed Systems Second Edition, ACM Press New York, 1993.

[4] E.Simon, Distributed Information Systems From Client/Server to Distributed Multimedia, The McGraw-Hill Company, 1996.

[5] H.Lorin, Aspects of Distributed Computer Systems, John Wiley & Sons, 1980.

[6] H.Lorin, Aspects of Distributed Computer Systems Second Edition, John Wiley & Sons, 1988.

[7] N.Shiratori and M.Takizawa, Distributed Processing, Maruzen, 1996 (in Japanese).

[8] M.Fukushima, Introduction to Mathematical Programming, Asakura, 1996 (in Japanese).

[9] T.Ibaraki, Algorithms and Data Structure in C, Shokodo, 1999 (in Japanese).

[10] L.Kleinrock, Queueing Systems, A Wiley-Interscience Publication, 1975.

[11] H.P.Williams, Model Building in Mathematical Programming Third Edition Revised, John Wiley & Sons, 1993.

[12] W. W. Chu, "Optimal File Allocation in a Multiple Computer System," IEEE Trans. Comput., Vol.C-18, No. 10, pp.885–890, October 1969.

[13] W.W.Chu, "Optimal File Allocation in a Multiple-Computer Information System," Computer-Communication Networks, pp.83-94, 1973.

[14] R.G.Casey, "Allocation of Copies of Files in an Information Network," Proceedings of Spring Joint Comput. Conf., IFIPS, pp.617-625, 1972.

[15] S.Ceri, G.Martella and G. Pelagatti, "Optimal File Allocation for a Distributed Database on a Network of Minicomputers," Proceedings of International Conference on Databases, July 1980.

[16] S.Ceri, G.Martella and G. Pelagatti, "Optimal File Allocation in a Computer Network: A Solution Method Based in the Knapsack Problem," Computer Networks, Vol.6, pp.345-357, 1982.

[17] S.Ceri, S.Navathe and G.Wiederhold, "Distribution Design of Logical Database Schemas," IEEE Trans. Softw. Eng. Vol.SE-9, No.4, pp.487-563, July 1983.

[18] K.P.Eswaran, "Placement of Records in a File and File Allocation in a Computer Network," Proceedings of the IFIP Congress on Information Processing 1974, pp.304-307, 1974.

[19] K.D.Levin and H.L.Morgan, "Optimizaing Distributed Databases — A Framework for Research," Proceedings of 1975 AFIPS NCC, vol.44, pp.473-478, 1975.

[20] S.Mahmoud and J.S.Riordon, "Optimal Allocation of Resources in Distributed Information Networks," ACM Trans. Database Syst. Vol.1, No.1, pp.66-78, March 1976.

[21] C.V.Ramamoorthy and B.W.Wah, "The Placement of Relations on a Distributed Relational Database," Proceedings of the 1st International Conference on Distributed Computing Systems, IEEE, pp.642-650, October 1979.

[22] P.M.G.Apers, "Data Allocation in Distributed Database Systems," ACM Trans. on Database Systems, Vol.13, No.3, September 1988.

[23] P.M.G.Apers, "Data Allocation and Distributed Query Processing," Proceedings of ACM Pacific'80, pp.48-54, November 1980.

[24] P.M.G.Apers, "Redundant Allocation of Relations in a Communication Network," Proceedings of 5th Berkeley Workshop on Distributed Data Management and Computer Networks, pp.245-258, February 1981.

[25] P.M.G.Apers, "Centralized or Decentralized Data Allocation," Proceedings of 2nd Seminar on Distributed Data Sharing Systems, pp.101-116, June 1981.

[26] L.W.Dowdy and D.V.Foster, "Comparative Models of the File Assignment Problem," ACM Computing Surveys Vol.74, pp.287-313, 1982.

[27] B.W.Wah and Y.N.Lien, "Design of Distributed Databases on Local Computer Systems with a Multiaccess Network," IEEE Trans. Softw. Eng. Vol.SE-11, No.7, pp.606-619, July 1985.

[28] B.Gavish and H.Pirkul, "Computer and Database Location in Distributed Computer Systems," IEEE Trans. Comput., Vol.COMP-35, No.7, pp.583-590, 1986.

[29] X.M.Lin, M.E.Orlowska and Y.C.Zhang, "Database Placement in Communication Networks for Minimizing the Overall Transmission Cost," Mathematical Compt. Modeling, Vol.19, No.1, pp.7-19, January 1994.

[30] S.T.Cheng,"Topological Optimization of a Reliable Communication Network," IEEE Trans. on Reliability, Vol.47, No.3, September 1998.

[31] I.Shyu and S.Shieh, "Balancing Workload and Recovery Load on Distributed Fault-Tolerant VOD Systems," IEEE Communications Letters, Vol.2, No.10, October 1998.

[32] R.Blankinship, A.R.Hevner and S.B.Yao, "An Iterative Method for Distributed Database Optimization," Data & Knowledge Engineering 21, Elsevier Science, 1997.

[33] P.M.G.Apers, A.R.Hevner and S.B.Yao, "Optimization Algorithms for Distributed Queries," IEEE Trans. on Software Engineering, Vol.SE-9, No.1, January 1983.

[34] Y.K.Kwok, K.Karlapalem, I.Ahmad and N.M.Pun, "Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems," IEEE Journal on Selected Areas in Communications, Vol.14, No.7, September 1996.

[35] Spyros Ligoudistianos, Dimitri Theodoratos, Timos Sellis and Yiannis Vassiliou, "Heuristic algorithms for designing the Data Warehouse with SPJ Views," Proceedings of Dawak'99, August 1999.

[36] J.F.Lawless, Statistical Models and Methods for Lifetime Data, John Wiley & Sons, 1982.

[37] C.C.Shen and W.H.Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," IEEE Trans. Comput., Vol.COMP-34, No.3, pp.197-203, March 1985.

[38] V.M.Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," IEEE Trans. Comput., Vol.37, No.11, pp.1384-1397, November 1988.

[39] S.M.Shatz and J.Wang, "Models & Algorithm for Reliability -Oriented Task-Allocation in Redundant Distributed-Computer Systems," IEEE Trans. on Reliability, Vol.38, No.1, April 1989.

[40] G.Hwang, "A Heuristic Task Assignment Algorithm to Maximize Reliability of a Distributed System," IEEE Trans. on Reliability, Vol.42, Nol.3, pp.408-415, 1993.

[41] A.K.Verma and M.T.Tamhankar,"Reliability-Based Optimal Task-Allocation in Distributed-Database Management Systems," IEEE Trans. on Reliability, Vol.46, No.4, December 1997.

[42] C. C. Bisdikian and Baiju V. Patel, "Issues on Movie Allocation in Distributed Video-on-Demand Systems," IEEE Proc.of ICC '95, pp.250-255, 1995.

[43] C. C. Bisdikian and B. V. Patel, "Cost-Based Program Allocation for Distributed Multimedia-on-Demand Systems," IEEE Multimedia, pp.62-72, Fall 1996.

[44] D.Deloddere, W.Verbiest, and H.Verhille, "Interactive video on demand," IEEE Commun.Mag., pp.82-88, May 1994.

[45] V. O. K. Li, W. Liao, X. Qiu, and E. W. M. Wong, "Performance Model of Interactive Video-on-Demand Systems," IEEE Journal on Selected Areas in Communications, Vol.14, No.6, August 1996.

[46] S. A. Barnett and G. J. Anido, "A Cost Comparison of Distributed and Centralized Approaches to Video-on-Demand," IEEE Journal on Selected Areas in Communications, Vol.14, No.6, August. 1996.

[47] T.D.C.Little and D.Venkatesh, "Popularity-Based Assignment of Movies to Storage Devices in a Video-on-Demand System," Multimedia Syst., Vol.2, pp.280-287, January 1995.

[48] D.J.Chen, W.C.Hol, R.S.Chen and D.T.K.Chen, "A Heuristic Algorithm for the Reliability-Oriented File Assignment in a Distributed Computing System," Computers Math Applications, Pergamon Elsevier Science, Vol.29, No.10, 1995.

[49] S.I.McClean, D.A.Bell and F.J.McErlean, "Heuristic Methods for the Data Placement Problem," Journal of the Operational Research Society, Vol.42, No.9, 1991.

[50] A.M.Tamhankar and S.Ram, "Database Fragmentation and Allocation: An Integrated Methodology and Case Study," IEEE Transactions on Systems Man and Cybernetics Part A-Systems and Humans, Vol.28, No.3, pp.288-305, May 1998.

[51] K.Chari, "Resource Allocation and Capacity Assignment in Distributed Systems," Computers & Operations Research, Pergamon Elsevier Science, Vol.23, No.11, pp.1025-1041, November 1996.

[52] C.Y.Lee, "Application of a Cross Decomposition Algorithm to a Location and Allocation Problem in Distributed Systems," Butterworth-Heinemann, Computer Communications, Vol.18, No.5, pp.367-377, May 1995.

[53] C.H.E.Chow, "Resource-Allocation for Multiparty Connections," Elsevier Science Jounal of Systems and Software, Vol.28, No.3, pp.253-266, March 1995.

[54] E.P.Robinson and L.L.Gao, "Location of Computers and Multiple Files in Distributed Computer-Systems," Mathematical and Computer Modelling, Pergamon-Elsevier Science, Vol.20, No.7, pp.111-120, October 1994.

[55] S.Ram and S.Narasimhan, “Database Allocation in a Distributed Environment — Incorporating a Concurrency Control Mechanism and Queuing Costs,” Management Science, Vol.40, No.8, pp.969-983, August 1994.

[56] D.B.Johnson and L.Raab, “Complexity of Network Reliability and Optimal Resource Placement Problems,” SIAM Journal on Computing, Vol.23, No.3, pp.510-519, June 1994.

[57] B.P.Weems and L.C.Swayze, “Allocation Techniques for Distributed Reduction Data Elements,” Inromation Processing Letters, Elsevier Science, Vol.50, No.3, May 1994.

[58] H.Lee and O.R.L.Sheng, “A Multiple Criteria Model for the Allocation of Data Files in a Distributed Information-System,” Computers & Operations Research, Pergamon-Elsevier Science, Vol.19, No.1, pp.21-33, January 1992.

[59] H.S.Lee, Y.Shi and J.Stolen, “Allocating Data Files over a Wide Area Network — Goal-Setting and Compromise Design,” Information & Management, Elsevier Science, Vol.26, No.2, pp.85-93, February 1994.

[60] H.Lee and T.Park, “Allocating Data and Workload among Multiple Servers in a Local-Area Network,” Information Systems, Pergamon-Elsevier Science, Vol.20, No.3, pp.261-269, May 1995.

[61] G.S.Jang and H.Lee, “File and Workload Allocation Design for Distributed Database on a Two-Level Local Multi-Access Computer Network,” Computers & Industrial Engineering, Pergamon-Elsevier Science, Vol.33, No.3-4, pp.649-652, December 1997.

[62] H.S.Lee and G.S.Jang, “File and Workload Allocation on a Local Multi-Access Computer Network: Incorporating Local Processing and Communication Overhead,” International Journal of Systems Science, Vol.27, No.9, pp.831-837, September 1996.

[63] P.A.Tom and C.S.R.Murthy, “Algorithms for Reliability-Oriented Module Allocation in Distributed Computing Systems,” Journal of Systems and Software, Elsevier Science, Vol.40, No.2, pp.125-138, February 1998.

[64] R.Boutaba and B.Folliot, "Program and File Allocation Algorithm for Large-Scale Distributed Systems," Applications in Parallel and Distributed Computing IFIP Transactions A-Computer Science and Technology, Vol.44, pp.165-173, 1994.

[65] Y.Bartal, A.Fiat and Y.Rabani, "Competitive Algorithms for Distributed Data Management," Journal of Computer and System Sciences, Vol.51, No.3, pp.341-358, December 1995.

[66] J.L.Aguilar, "File Decomposition, Replication and Assignment Problems: Definition and Resolution Method," Engineering Intelligent Systems for Electrical Engineering and Communications, Vol.8, No.4, pp.245-254, December 2000.

[67] D.Ghosh, I.Murthy and A.Moffett, "File Allocation Problem — Compariosn of Models with Worst Case and Average Communication Delays," Operations Research, Vol.40, No.6, pp.1074-1985, November-December 1992.

[68] G.G.Tsegelik and A.V.Demidovich, "Construction of Mathematical Models of Optimal Allocation of File Copies in Distributed Databases," Automatic Control and Computer Sciences, Vol.32, No.1, pp.46-54, 1998.

[69] A.Kumar, R.M.Pathak and Y.P.Gupta, "Genetic Algorithm-Based Approach for File Allocation on Distributed Systems," Computers & Operations Research, Pergamon-Elsevier Science, Vol.22, No.1, pp.41-54, January 1995.

[70] A.R.Chaturvedi, A.K.Choubey and J.S.Roan, "Scheduling the Allocation of Data Fragments in a Distributed Database Environment — A Machine Learning Approach," IEEE Transactions on Engineering Management, Vol.41, No.2, pp.194-207, May 1994.

[71] P.Y.Chang, D.J.Chen and K.M.Kavi, "Multimedia File Allocation on VC Networks Using Multipath Routhing," IEEE Transactions on Computers, Vol.49, No.9, pp.971-977, September 2000.

[72] W.T.Cai and B.S.Lee, "File Allocation with Balanced Response Time in a Distributed Multi-Server Information System," Information and Software Technology, Elsevier Science, Vol.40, No.1, pp.27-35, April 1998.

[73] Y.W.Wang, J.C.L.Liu, D.H.C.Du and J.W.Hsieh, "Efficient Video File Allocation Schemes for Video-on-Demand Services," Multimedia Systems, Vol.5, No.5, pp.283-296, September 1997.

[74] S.T.March and S.Rho, "Allocating Data and Operations to Nodes in Distributed Database Design," IEEE Transactions on Knowledge and Data Engineering, Vol.7, No.2, pp.305-317, April 1995.

[75] P.Y.Chang, D.J.Chen and K.M.Kavi, "File Allocation Algorithm to Minimize Data Transmission Time in Distributed Computing Systems," Journal of Information Science and Engineering, Vol.17, No.4, pp.633-646, July 2001.

[76] K.Kalpakis, K.Dasgupta and O.Wolfson, "Optimal Placement of Replicas in Trees with Read, Write, and Storage Costs," IEEE Transactions on Parallel and Distributed Systems, Vol.12, No.6, pp.628-637, June 2001.

[77] T.Imielinski and B.R.Badrinath, "Mobile Wireless Computing," Communications of the ACM, Vol.37, No.10, October 1994.

[78] K.Y.Lam, T.W.Kuo, W.H.Tsang and G.C.K.Law, "Concurrency Control in Mobile Distributed Real-Time Database Systems," Information Systems, Pergamon Elsevier Science, Vol.25, No.4, 2000.

[79] Budiarto, K.Harumoto, M.Tsukamoto and S.Nishio, "On Strategies for Allocating Replicas of Mobile Databases," IEICE Trans. on Information & System, Vol.E81-D, No.1, January 1998.

[80] Budiarto, K.Harumoto, M.Tsukamoto and S.Nishio, "On Relocation Dicision Policies of Mobile Databases," IEICE Trans. on Information & System, Vol.E82-D, No.2, February 1999.

[81] E.Kayan and O.Ulusoy, "An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems," The Computer Journal, Vol.42, No.6, 1999.

[82] K.Shim, T.Sellis and D.Nau, "Improvements on a Heuristic Algorithm for Multiple-Query Optimization," Data & Knowledge Engineering 12, Elsevier Science, 1994.

[83] N.J.Nilsson, Principles of Artificial Intelligence, Morgan Kaurmann Publ., 1980.

[84] A.R.Hevner and S.B.Yao, "Query Processing in Distributed Database Systems," IEEE Trans. on Software Engineering, Vol.SE-5, No.3, May 1979.

[85] A.R.Hevner, O.Q.Wu and S.B.Yao, "Query Optimization on Local Area Networks," ACM Transactions on Office Information, Vol.3, No.1, January 1985.

[86] M.Steinbrunn, G.Moerkotte and A.Kemper, "Heuristic and Randomized Optimization for the Join Ordering Problem," The VLDB Journal 6, 1997.

[87] J.J.Sung and J.T.Park, "Semantic Query Processing in Object-Oriented Database Systems," Database Systems for Advanced Applications '91, World Scientific Publishing Co., 1991.

[88] B.Finance and G.Gardarin, "A Rule-Based Query Optimizer with Multiple Search Strategies," Data & Knowledge Engineering 13, Elsevier Science, 1994.

[89] W.W.Chu, C.M.Sit and K.K.Leung, "Task Response Time for Real-Time Distributed Systems with Resource Contentions," IEEE Trans. on Software Engineering, Vol.17, No.10, October 1991.

[90] W.W.Chu and C.M.Sit, "Estimating Task Response Time with Contentions for Real-Time Distributed Systems," The Proceedings of the Real Time Systems Symposium, December 1988.

[91] W.W.Chu and C.M.Sit, "A Batch Service Scheduling Algorithm with Time-Out for Real-Time Distributed Processing Systems," The Proceedings of the 7th International Conference on Distributed Computing Systems, September 1987.

[92] K.Yajima, H.Kitagawa, K.Yamaguchi, N.Ohbo and Y.Fujiwara, "Optimization of Queries Including ADT Functions," Database Systems for Advanced Applications '91, World Scientific Publishing Co., 1991.

[93] K.A.Ahn, H.Choi and W.O.Kim, "Architecture of a VOD System with Proxy Servers," IEICE Trans. on Communications, Vol.E830B, No.4, April 2000.

[94] T.Hara, K.Harumoto, M.Tsukamoto, S.Nishio and J.Okui, "A Dynamic Replica Allocation Method Based on Data Migration in Broadband Netowrks," IEICE Trans. D-I, Vol.J82-D-I, No.8, August 1998 (in Japanese).

[95] S.Makabe, K.Ohta, N.Kato, G.Mansfield and Y.Nemoto, "Dynamic Mirror Server Selection Method with Consideration about Fluctuation of Network Load," IEICE Trans. B, Vol.J84-B, No.3, March 2001 (in Japanese).

[96] K.Nishimura, M.Yamamoto and H.Ikeda, "Contents Distributing Algorithm for Replicated Web Servers Based on Economics Principles," IEICE Trans. B, Vol.J83-B, No.2, February 2000 (in Japanese).

[97] D.C.K.Siew and G.Feng, "Tree-Caching for Multicast Connections with End-to-End Delay Constraint," IEICE Trans. on Communications, Vol.E84-B, No.4, April 2001.

[98] T. Yamamoto, D. Nakajima, H. Ohsawa and Y. Yasuda, "A Study on Cache Layout in Hierarchical Network," Proceedings of the 1999 IEICE General Conference, March 1999 (in Japanese).

[99] M. Shiina, Y. Tajima, H. Morikawa and T. Aoyama, "An Adaptive Network Web Caching System," Proceedings of the 1999 IEICE General Conference, March 1999 (in Japanese).

[100] R.S.Epstein, M.Stonebraker and E.Wong, "Distributed Query Processing in a Relational Data Base System," ACM SIGMOD Conference 1978, June 1978.

[101] P.G.Selinger, M.M.Astrahan, D.D.Chamberlin, R.A.Lorie and T.G.Price, "Access Path Selection in a Relational Database Management System," ACM SIGMOD Conference 1979, June 1979.

[102] P.A.Bernstein, N.Goodman, E.Wong, C.L.Reeve and J.B.Rothnie, "Query Processing in a System for Distributed Databases (SDD-1)," ACM Transactions on Database Systems(TODS), Vol.6, No.4, December 1981.

[103] A. Nakaniwa, H. Ebara and H. Okada, "File Allocation Designs for Distributed Multimedia Information Networks," IEICE Trans. on Communications, Vol.E81-B, No.8, August 1998.

[104] A. Nakaniwa, M. Onishi, H. Ebara and H. Okada, “File Allocation in Distributed Multimedia Information Networks,” Proceedings of IEEE GLOBECOM ’98, November 1998.

[105] A. Nakaniwa, M. Onishi, H. Ebara and H. Okada, “Sensitivity Analysis of File Allocation for Distributed Information Networks,” Proceedings of IEEE ICC ’99, June 1999.

[106] A. Nakaniwa, M. Onishi, H. Ebara and H. Okada, “Sensitivity Analysis in Optimal Design for Distributed File Allocation Systems,” IEICE Trans. on Communications, Vol.E84-B, No.6, June 2001.

[107] A. Nakaniwa, J. Takahashi, H.Ebara and H. Okada, “Reliability-Based Optimal Allocation of Distributed Mirror Servers for Internet,” Proceedings of GLOBECOM2000, November 2000.

[108] A.Nakaniwa, J.Takahashi, H.Ebara and H.Okada, “Reliability-Based Mirroring of Servers in Distributed Networks,” to be published in IEICE Trans. on Communications.

[109] A.Nakaniwa, A.Akihiro, J.Mizutani, H.Ebara and H.Okada, “Optimal File Allocation in Multi-media Distributed Information Networks,” Proceedings of the 1997 IEICE General Conference, March 1997 (in Japanese).

[110] A.Nakaniwa, H.Ebara and H.Okada, “File Allocation in Distributed Information Networks,” Technical Report of IEICE, IN97-131, November 1997 (in Japanese).

[111] A.Nakaniwa, H.Ebara and H.Okada, “A Study on Sensitivity Analysis of Parameters on File Allocation in Distributed Information Networks,” Proceedings of the 1998 IEICE General Conference, March 1998 (in Japanese).

[112] A.Nakaniwa, H.Ebara and H.Okada, “Sensitivity Analysis of Parameters on File Allocation in Distributed Information Networks,” Technical Report of IEICE, TM98-21, July 1998 (in Japanese).

[113] A.Nakaniwa, J.Takahashi, M.Onishi, H.Nishiyama, H.Ebara and H.Okada, “Reliability-based Optimal Allocation of Distributed Mirror Servers,” Proceedings of the 2000 IEICE General Conference, March 2000 (in Japanese).

[114] A.Nakaniwa, J.Takahashi, H.Ebara and H.Okada, "Reliability-Based Optimal Allocation Model of Distributed Mirror Servers," Technical Report of IEICE, IN2000-9, April 2000 (in Japanese).

[115] A.Nakaniwa, J.Takahashi, H.Ebara and H.Okada, "Reliability Maximized Allocation of Distributed Mirror Servers for Mobile Computing," Proceedings of YRP Telecom Summit, July 2000 (in Japanese).