Supporting Cloud Computing in Thin-Client/Server Computing Model

Conference Paper · September 2010

DOI: 10.1109/ISPA.2010.42 · Source: DBLP

1 author:

Gwan-Hwan Hwang
National Taiwan Normal University
56 PUBLICATIONS **365** CITATIONS

Supporting Cloud Computing in Thin-client/server Computing model

Gwan-Hwan Hwang
Department of Computer Science and Information Engineering
National Taiwan Normal University, Taipei, Taiwan
ghhwang@csie.ntnu.edu.tw

***Abstract*- This paper addresses the issue of how to support cloud computing in thin-client/server computing model. We describe the design and implementation of the multple-application-server thin-client/server computing (MAS TC/S) model that allows users with thin-client devices to roam around a wide area network whilst experiencing transparent working environments. MAS TC/S can be applied to a wide variety of applications in wide area network, such as pervasive software rental, service-oriented infrastructure for an Internet service provider, and office automation for transnational corporations. The MAS TC/S architecture model comprises five major components: the display protocol, the multiple-application-server topology, the application-server discovery protocol, the distributed file system, and the network input/output protocol. We discuss problems and present solutions in the design of each constituent component. A prototype of the MAS TC/S that spans the campuses of two universities in Taiwan has been built – we also report our experiences in constructing this prototype.**

***Keywords*- Cloud computing, thin client, distributed file system, appointed prefetching.**

1. INTRODUCTION

Cloud computing is Internet-based computing, whereby shared resources, software and information are provided to computers and other devices on-demand. Cloud computing describes a new supplement, consumption and delivery model for IT services based on the Internet, and it typically involves the provision of dynamically scalable and often virtualized resources as a service over the Internet. In this paper, we address how to apply the thin-client/server computing model as a platform for cloud computing in wide area network.

The thin-client/server computing model is growing very rapidly due to its low cost and the rapid deployment of applications running at the server side, or so-called server-based computing. Server-based computing allows corporations to gain more control over applications by managing them at the server infrastructure instead of at the desktops. The multiuser thin-client/server computing model takes this one stage further, by mandating that applications run solely on a server: client devices merely monitor inputs from mouses and keyboards, pass them to the server, and wait for the displays returned by the server.

The thin-client/server computing model also benefits end users by providing them with transparent working environments regardless of what type of client devices they use, and where they use them. The traditional way of implementing a transparent working environment is to assume a single-application-server topology in which each client device always connects to the same application server [1,2,3,4,5], and that all the client's data and application software are stored on this server. With such a topology, a user could only roam within a restricted area (the local area network, or LAN) close to this application server within which efficient transfer of mouse clicks, keyboard inputs, and screen updates via some display protocol is possible [2,3,4,5].

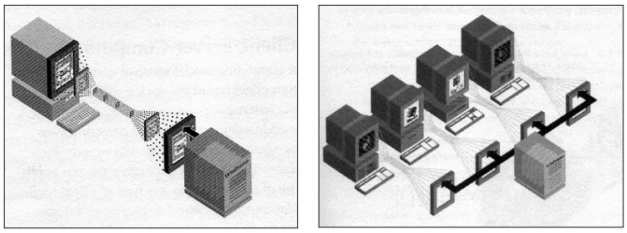
In this paper, we propose a multiple-application-server architecture model for the thin-client/server computing model, denoted MAS TC/S. In this architecture model, multiple application servers are installed over a wide area network (WAN), and each client device can freely connect to any application server that is close to it. A user's data are stored only in a limited number of servers since, as we will explain later, it is impossible to duplicate the entire users' data in all these servers. Before providing services to a user, an application server must fetch absent files that are required by the services. One of the key issues for obtaining reasonable response times in the MAS TC/S architecture model is how to reduce the delay for data or file fetching. We have achieved this by designing a distributed file system for MAS TC/S, which includes the functionality of traditional distributed file systems with some enhancements, including an intelligent prefetching mechanism, an appointed prefetching mechanism, and a cache coherency protocol. To perform the intelligent prefetching, the application server speculates about what files are going to be accessed by users and then prefetches those files in parallel with the user's work. Apart from the intelligent prefetching in which systems are trying to identify the frequent file-access patterns of a given user, the proposed distributed file system also provides an interface for users to specify when, where, and what files s/he may need to meet a specific schedule. The schedule is described as a workflow model with a friendly GUI for users to specify tasks as well as the order and branching of their invocation. The distributed file system will then fetch the required files according to this schedule.

In addition to the distributed file system, we also develop an application-server discovery protocol and a network input/output (I/O) protocol to enable MAS TC/S to support mobile applications. The application-server discovery protocol helps a thin-client device to identify the most appropriate

978-0-7695-4190-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ISPA.2010.42

application server to connect to. The network I/O protocol allows the connected application server to access the I/O devices either as specified by the user or as configured automatically. We have constructed a prototype of the MAS TC/S that allows users to roam across two universities in Taiwan. Preliminary experimental results show that MAS TC/S provides a practical infrastructure for service-oriented mobile applications.

The remainder of this paper is organized as follows: Section 2 gives an overview of the traditional thin-client/server computing model, Section 3 presents the architecture of MAS TC/S, Section 4 introduces the distributed file system that we developed for the MAS TC/S, Section 5 describes the network I/O protocol for MAS TC/S, Section 6 describes the prototype we have implemented, and Section 7 concludes this paper and describes ongoing and future work.



(A) (B)

Figure 1: Thin-client/server computing model

2. Overview Of The Traditional Thin-Client/Server Computing Model

The thin-client/server computing model consists of three key components: (1) thin-client hardware devices, (2) the application server, and (3) a display protocol, as shown in Figure 1A. All the applications and data are deployed, managed, and supported at the application server. In addition, applications are executed only on the server. Thin-client devices gather inputs from users in the form of mouse clicks and keystrokes, send them to the application server for processing, and collect screen updates as the response from the application server. All the interactions between thin-client devices and the application server occur via an efficient display protocol. The display protocol is highly optimized for specific software APIs to reduce their bandwidth requirements; e.g., X [1], independent computing architecture (ICA) [2,3], remote desktop protocol (RDP) [4], and stateless low-level interface machine (SLIM) [5]. The application server provides a centralized maintenance environment since all the applications are installed on it. The information-systems department of a corporation can deploy and update the applications instantly without ever needing to "touch" every desktop or PC, which thereby dramatically reduces the cost of upgrading and deploying applications. Users also have access to applications and data over a wider area, which increases their productivity; and security is also enhanced because all data are maintained on the application server. In addition, the thin-client/server computing model increases sharing of computing and memory resources on the application server. For example, consider Figure 1B where several users execute the same application on the application server. These users share a single copy of this application's code image and the computing power of the server.

The thin-client hardware devices can be realized using low-cost, diskless computers with the display protocol built into their ROMs. They need only the following hardware components: keyboards, monitors, serial or network interfaces, high-speed serial ports, and bidirectional parallel ports. Examples of proprietary thin-client devices include X terminal [1], SLIM console [5], and ICA's windows-based terminal [3]. An ordinary personal computer or workstation can also be used as a thin-client device by installing appropriate software that supports the display protocol [3].

3. The MAS TC/S Architecture Model

In this section, we present the architecture model of MAS TC/S. We first discuss the transparent working environment requirements of the MAS TC/S and then describe its constituent components.

3.1 Transparent Working Environment

A distributed system is commonly considered to be transparent when the constituent components are concealed from the user, so that the entire system is perceived as a whole rather than as a collection of independent components. The transparent working environment of the MAS TC/S should provide at least the following forms of transparency [6]:

- *Access transparency* enables local and remote resources to be accessed using identical operations.
- *Location transparency* enables resources to be accessed without knowledge of their location.
- *Mobility transparency* allows the movement of resources and clients within a system without affecting the operation of users.

Access transparency enables local and remote resources to be accessed using identical operations.If a user is allowed to log onto a system with the same account and password from different locations in the network, and work with the same files, applications, and desktop working interface without any environment settings and translations, we refer to this system as having a transparent working environment. Examples of systems that are able to maintain a transparent working environment in a LAN include Sun's NFS + NIS, Novell Netware, and Microsoft Windows NT. However, the transparency of their working environments does not apply to a WAN.

The traditional thin-client/server computing model offers a transparent working environment [1,2,3,4,5]. In this model, each thin-client device is always connected to a single application server, and all the applications and data are deployed, managed, and supported on the server. Therefore, users have transparent access to their proprietary files and applications. The response time of the system is determined by

the network bandwidth and the load on the application server. Therefore, the thin-client devices and the application server should be located in the same LAN, and the number of thin-client devices must be limited. Consider the following somewhat fanciful example:

> *Peter is the president of a transnational corporation. His office is located in California, but he often travels to departments of his company that are located in several different countries. During such trips he has to conduct some business such as checking e-mail, and making or editing documents and slices. For example, when he reaches a department in Japan, he logs onto the system with any thin-client device and then continues with the work he has not finished in California. He does not need to perform any environment settings and or file transformations, and the system's response is just as quick as it was in California. In other words, the working environment is the same as he had in his office back in California; i.e., Peter has transparent file and application access from any thin-client device to his personal account at a remote site.*

It is obvious that the traditional thin-client/server computing model with a single application server is not appropriate for Peter's transnational corporation: if his application server is located in California, he could hardly have an acceptable response time if he connects to it from Japan. Some organizations install multiple application servers under the traditional thin-client/server paradigm for load sharing, and each application server handles a limited number of thin clients. For example, a company with, say, 1,000 employees may install 10 application servers so that, on average, only 100 employees share one application server. However, a thin-client device is only allowed to log onto a particular application server. Therefore, unless all the users' data are fully replicated in all application servers, which would have a prohibitively high cost, a user is not allowed to roam into an area that use a different application server. This violates the mobility transparency.

3.2 Components of MAS TC/S

The MAS TC/S architecture model comprises the following major components:

- The display protocol that allows the application server and thin-client device to communicate.
- The multiple-application-server topology.
- The application-server discovery protocol.
- The distributed file system to maintain an efficient transparent working environment.
- The network I/O protocol for connecting the application server to the users' I/O devices such as printers, disk drivers, and scanners.

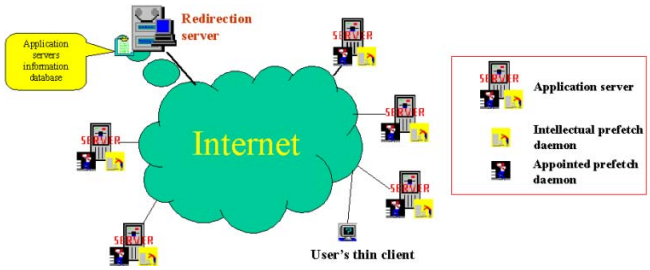


Figure 2: The multiple-application-server topology

The display protocol used in MAS TC/S is the same as that used in traditional thin-client/server computing model, such as X, ICA, RDP, and SLIM. Therefore, the thin-client devices designed for thin-client/server computing model can also be used for MAS TC/S. The multiple-application-server topology involves several application servers connected in a WAN (or the Internet), as shown in Figure 2. In the example given in Section 3.1 about Peter's corporation, at least two application servers should be installed: one in California and the other in Japan. Peter may log onto either of the two servers.

When a user tries to connect to an application server by using a thin-client device, the *application-server discovery protocol* is used to identify the most appropriate server. We propose and implement two kinds of application-server discovery protocols: (1) multicast discovery: the thin-client device can conduct a multicast broadcast for application servers with one or more specific group name or the default group name; and (2) unicast discovery: each lookup of application servers issued by the thin-client device is sent to one or several *redirection servers*. The redirection server is equipped with a database that contains information about all the application servers. It will guide the thin-client device to connect to the most appropriate application server according to the location of the thin-client device, and the layout and status of available application servers.

We construct a distributed file system for the MAS TC/S to achieve a working environment with access, location, and mobility transparencies. The potentially large number of application servers installed in a WAN environment for the MAS TC/S means that the storage and communication costs of replicating all users' data or files on all the application servers may be prohibitively high. An efficient distributed file system is indeed very important for the MAS TC/S. Ideally, it must possess the following two features:

1. A sophisticated file prefetching mechanism that predicts users' data demands with reasonable accuracy.
2. An efficient cache coherency mechanism to keep the cached copy consistent with the master copy.

We present the details of the proposed distributed file system in Section 4.

Furthermore, we design a network I/O protocol for the MAS TC/S. This protocol defines how the connected application server can access the user's I/O device. For

example, the connected application server can send a user's data to networked printers or obtain a file that is located in a disk driver of user's thin-client device. Details of the protocol are presented in Section 5.

4. A Distributed File System For MAS TC/S

The proposed distributed file system for MAS TC/S includes the functionality of traditional distributed file systems with enhanced mechanisms for file prefetching and coherency. The working environment for a user needs the following types of data:

1. *User's records or preferences*: This includes the window manager's records and the record files of various applications.
2. *User's files*: These are the personal files that belong to the user, such as files for e-mail, word processors, spread sheets, graphics, and multimedia.
3. *Application software*: These are the binary codes for various applications.

These three types of data exist in the application server in the form of *files*. In the remainder of this paper, we use the term *user's data* to refer collectively to both a user's records and a user's files. To provide the user with a transparent working environment, the required files should be ready after the user logs onto an application server. However, in MAS TC/S, a user could log onto any application server via a thin-client device, and thus an application server should prepare each user's data in order to provide a prompt service. A trivial solution is to replicate each user's data and application software on all the application servers. However, full replication of the user's data is usually too expensive: consider an enterprise with 10,000 users whose individual disk quota is 100 MB – to fully replicate user's data, each application server has to allocate 10,000 × 100 MB=1,000 GB of disk storage. In addition to the huge disk-space wastage, synchronizing a user's data may consume much of the available network bandwidth. For example, updating a 10 MB file would cause a total of 10 MB × 10,000=100 GB of data to be routed among all the application servers (by using the well-known read-one-write-all scheme [7]). Note that we assume that application software is fully duplicated on all the application servers, since the application software is usually shared among users and must be installed and set up properly beforehand.

The distributed file system proposed here does not require a user's data to be fully replicated on an individual application server. Therefore, it is possible that some of user's data may be absent when the user logs on some application server (which will be discovered by the application-server discovery protocol). If some absent file is needed by the user, this connected application server has to fetch it. In a traditional thin-client/service computing model, an acceptable response time could be obtained as long as the following two requirements are met: (1) the network bandwidth between the thin-client device and the connected application server is fast enough for the display protocol, and (2) the load on the application server is moderate. In the MAS TC/S, if the users' data are not duplicated on all the application servers, the delay associated with fetching absent files must be considered in addition to the above two factors when determining an acceptable response time.

The distributed file system proposed here does not require a user's data to be fully replicated on an individual application server. Therefore, it is possible that some of user's data may be absent when the user logs on some application server (which will be discovered by the application-server discovery protocol). If some absent file is needed by the user, this connected application server has to fetch it. In a traditional thin-client/service computing model, an acceptable response time could be obtained as long as the following two requirements are met: (1) the network bandwidth between the thin-client device and the connected application server is fast enough for the display protocol, and (2) the load on the application server is moderate. In the MAS TC/S, if the users' data are not duplicated on all the application servers, the delay associated with fetching absent files must be considered in addition to the above two factors when determining an acceptable response time.

There are two kinds of fetching strategy: demand fetching and prefetching. The demand fetch commences after a user has requested file access, whereas in prefetching the files are fetched beforehand. There are similar technologies for data fetching in the context of the CPU cache and the paging system of operating systems. The granularity of data fetched in CPU cache and the paging system are cache lines and memory pages, respectively, whereas application servers fetch files. According to previous work on prefetching technologies in the CPU cache and paging systems, prefetching is usually much better than demand fetching in terms of the miss ratio because memory access for ordinary programs is usually sequential. It is obvious that there are some differences between memory access by the processor and file access by users. First, cache lines and memory pages are usually of fixed size, whereas files are of variable size. Second, files possess some extra information such as creation date, updated time, owner, and type. The following considerations are important when designing an appropriate prefetching strategy for MAS TC/S: (1) the multiple-application-server topology, and (2) the requirements of a transparent working environment. In the following subsections, we will describe two prefetching strategies: intelligent prefetching and appointed prefetching.

4.1 Intelligent Prefetching

The intelligent prefetching mechanism proceeds in parallel with the processing of user's data requests by the application server. Its main mission is to predict a set of data that will be needed by the user after the current data request. We classify the user's data into the following three categories according to when it is needed in a connection session:

1. *System data*: This dictates the set of data required by the user's desktop working environment immediately after the user logs on, including record files for window

managers, the setups for various applications, and information about the home file directory. The system data of a given user is required by the application server for providing the user with his or her proprietary environment.

2. *Working data*: This refers to the set of files that the user needs to work with during the connection session.
3. *Unused data*: This represents all the other files not used during the connection session.

It is preferable that both system data and working data of a user session can be fetched before they are actually needed. However, while system data can be determined precisely, it is unlikely that the working data can be determined accurately. Moreover, the size of system data pertaining to a given user is much smaller compared to his working data (usually no more than 100 kB, whereas it is quite common that a user possesses files of hundreds or thousands of megabytes under his home directory).

To fetch the system data, a straightforward approach is for every constituent application server to cache the system data of every user, which normally constitutes a total of less than 100 MB storage for 1,000 users. In fact, an application server needs only to cache the system data of users from other application servers with longer communication delays, since for two application servers with a reasonable communication bandwidth between them there is no need to cache users' system data from each other's sites; on-demand data fetching is often sufficient since users can usually tolerate a longer delay when s/he first logs on.

Our focus in conducting intelligent prefetching is to develop techniques for predicting the set of working data based on the historical data obtained in previous connections, including access times, access operations, and sizes of files. Specifically, we consider prefetching based on access patterns: This approach dynamically prefetches files based on the user's current file request and the user's most frequent access patterns. For example, when the user places a request to open a file, we would like to predict with a high probability which files are subsequently needed. These files are candidates for prefetching [8].

4.2 Appointed Prefetching

In addition to the intelligent prefetching, in which the system attempts to determine the file access patterns that frequently appear in the user's routine schedules, we design another prefetching mechanism – called appointed prefetching – to catch a user's irregular schedules. User's schedules are described as a workflow model with a focus on their dataflows. Specifically, we model a schedule as a set of tasks, and each task is a triplet (D, L, Fs), where D and L specify respectively the durations and locations in which this task will be conducted, and Fs is a set of files needed by the task. Consider the following continuation of the example in Section 3.1:

Peter is going to make a presentation at the Japanese R&D department of his company. During his presentation, he will use pre1.doc, pre1.ppt, and pre1.scr. Instead of bringing them to Japan, he only specifies a task presentation {(2002-2-30:9:00 2002-2-30:12:00), Japan R&D, (pre1.doc, pre1.ppt, pre1.scr)} beforehand. The system will transfer these files to the application server in the Japanese R&D department prior to his presentation in Japan.

Just as in the intelligent prefetching, the schedules specified by appointed prefetching can be used for both prefeteching and replacement. By considering the sizes of the needed files, the time and place a task is planned to take place, and the bandwidth of the network, the system decides a schedule of lowest cost for transferring files under the constraint that the files must arrive at the correct location before the task is actually conducted. Once the time of the performance of a task has passed, these files may become candidates for replacement. See [9] for details.

4.3 Data Consistency

The fundamental architectural principle for data consistency in the distributed file system of MAS TC/S is the caching of the entire set of files from servers. To do so, we adopt the callback mechanism similar to that used by the so-called Andrew file system (AFS). According to the recent work by Spasojevic and Satyanarayanan [10] that examined the AFS as a provider of wide-area file services to over 100 organizations around the world, the AFS provides robust and efficient data access in its current configuration, thus confirming its viability as a design point for wide-area distributed file systems.

However, the AFS has to be further enhanced to support mechanisms used in MAS TC/S, such as intelligent prefeteching and appointed fetching. Thus, we are implementing a distributed file system which extends the AFS with the following reinforcements:

1. We modify the open and close system calls of the AFS to collect information about a user's file accesses in order to support data-mining-based prefetching and replacement.
2. The cache management mechanism is modified to conduct intelligent prefetching and appointed prefetching.

5. Network I/O Protocol For MAS TC/S

In the thin-client/server computing model, an application is executed solely on an application server. User access to certain I/O devices (e.g., printers, diskette drivers, and scanners) must occur via the connected application server. This does not cause much trouble in the traditional thin-client/server computing model because it operates in a small area (the LAN), and the static application server is aware of all the I/O devices available in that area. Users can access any I/O devices that the application server can access.

However, the application server still cannot access the built-in I/O devices of the user's thin-client device.

The problem becomes even more serious for MAS TC/S. For one thing, users may not be aware of what I/O devices are reachable by the connected application server. The location transparency requirement may be invalidated if the user has to specify a different I/O device each time s/he logs onto a different application server. Furthermore, the connected application server has no knowledge about the I/O devices that a thin-client device is connected to.

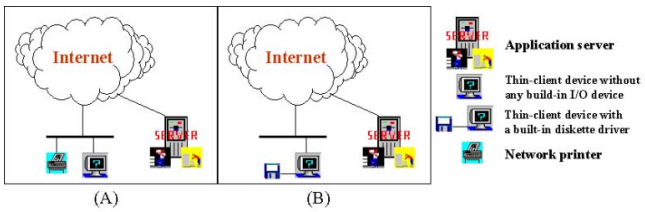


Figure 3: Network input/output protocol

In this paper, we propose a network I/O protocol for the MAS TC/S. With it, the connected application server is able to access I/O devices that are either specified by the user or configured automatically. We consider two types of I/O devices: network I/O devices (see Figure 3A), and I/O devices that are built into thin-client devices (see Figure 3B).

Consider the scenario shown in Figure 3A, where a thin-client device and a networked printer (or any other type of I/O device) are located in the same network segment but the connected application server resides at a different network segment. The user has to provide the application server with detailed information about the networked printer, including the IP address, and the user's ID and device type so that a connection to the I/O device can be set up properly. The same procedure is followed to set up a connection between a built-in device and the connected application server, as shown in Figure 4B where a thin-client device is equipped with a built-in diskette driver that is unknown to the connected application server. When a user uses a thin-client device to log onto an application server, the connections to all the build-in I/O devices are set up automatically.

We design a network I/O protocol to support the dynamic information exchange between I/O devices and application servers. This protocol is based on the so-called Jini technologies [11]. There are three types of players in a running Jini system: the services, such as printers and toasters; the clients that make use of the services; and lookup services (or so-called service locators) that act as brokers between services and clients. The service sends a proxy object to the lookup service. The proxy object contains information needed for clients to establish connections to the service. The client then discovers and retrieves the service's proxy object from the lookup service. For clarity, we refer to a service, a client, and a lookup server as a Jini service, a Jini client, and a Jini lookup service in the following discussion. The major goal of Jini technologies is to offer "network plug and play" capability for a distributed computing environment; we rely on Jini technologies in the design of our network I/O protocol. Referring to Figure 4, the networked I/O device (the network printer in the figure) is a Jini service, the application server is a Jini client, and the two Jini lookup service servers have to be established: one is near the network I/O device and the other is close to the application server. Since the cost of installing a Jini lookup service server is low, we suggest that such a service is established within each application server.

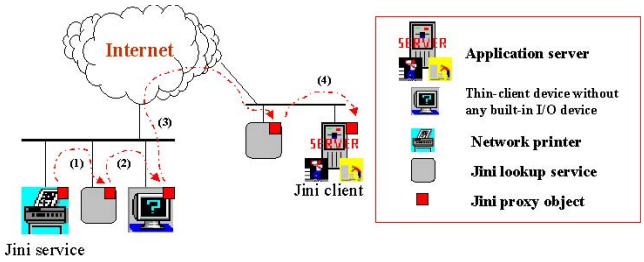


Figure 4: I/O devices initialization in an application server

The thin-client device is not itself a Jini service, client, or service, but rather it plays a particular role as the bridge between the local Jini lookup service of the I/O device and application server. The installation of I/O devices in the application server consists of four steps, as shown in Figure 4. Firstly, a network I/O device discovers the local Jini lookup service and sends its proxy object to it. Secondly, the thin-client device receives commands from the user about what network I/O device s/he would like the application server to use. Then it searches for the proxy object of the network I/O device according to the discovery protocol of Jini. This protocol starts by contacting the local Jini lookup service and eventually obtains the proxy object of the network I/O device. Thirdly, the thin-client device sends the proxy object to the Jini lookup service on the application server. Finally, the application server obtains the proxy object from its own Jini lookup service. Note that the first step is bypassed if a thin-client device wants to install its built-in I/O devices into the connected application server, and the proxy objects of its built-in I/O devices are sent to the Jini lookup service of the application server directly.

6. Implementation And Experiments

We are in the process of building the prototype of the MAS TC/S. This prototype spans the campuses of two universities located in different cities of Taiwan : National Taiwan Normal University in Taipei (in northern Taiwan) and National Tsing-Hua University in TsinChu (in central Taiwan). The networking systems of the two universities constitute a WAN.

The application servers are executed on Pentium PCs running Linux OS, and since each Linux workstation is equipped with a built-in X protocol, this was chosen as the display protocol. As for the thin-client devices, we consider three types of platforms:

1. *Thin-client device type 1*: General-purpose PCs or workstations with built-in X protocol and Web browsers.

2. *Thin-client device type 2*: PDAs with built-in X protocol and Web browsers.
3. *Thin-client device type 3*: Any other computers with a Web browser that is capable of executing Java applets. These do not need a built-in X protocol.

All three kinds of thin-client devices must be equipped with Web browsers since we use a Web server to implement the unicast application-server discovery protocol described in Section 3.2. Recall that a redirection server is associated with a database that contains information about all the installed application servers. Once a thin-client device is trying to make a connection to an application server, it first visits the home page of the redirection server with its Web browser. The user may either choose one application server on her/his own or let the system make a choice for him or her.

The connection process between a thin client and an application server is depicted in Figure 5. Figure 5A shows the connection process for a thin-client device of type 1 or 2 (i.e., which has a built-in X protocol). Firstly, the thin-client device visits the home page of the direction server. Secondly, the redirection server redirects the thin client to the login home page on the chosen application server, and the user clicks on the radio button to let the application server know that it has a built-in X protocol. Finally, the application server executes the appropriate window manager that uses the X protocol to communicate with the user's thin-client device. If a client device is of type 3 and hence does not install the X protocol, the user indicates this to the application server. The application server then sends an Java applet called Xweird [17] that emulates the X protocol.

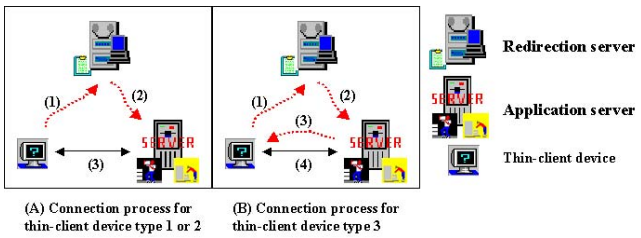


Figure 5: The connection process of a thin-client device and an application server

7. Conclusion And Future Work

We have proposed a thin-client/server computing model called MAS TC/S that provides transparent working environments to thin-client devices roaming a WAN. According to our implementation and experiment, the proposed MAS TC/S can be applied to the cloud computing architecture to fulfill a wide variety of applications, such as pervasive software rental, service-oriented infrastructure for Internet service providers, and office automation for transnational corporations. The MAS TC/S architecture model comprises five major components: the display protocol, the multiple-application-server topology, the application-server discovery protocol, the distributed file system, and the network I/O protocol. The display protocol follows the standard protocol used in traditional, LAN-based thin-client/server architectures such as X, ICA, RDP, and SLIM. The application-server discovery protocol helps the thin-client device to identify the most appropriate application server to connect to. The distributed file system includes the functionality of traditional distributed file systems with some reinforcements, including a data-mining-based intelligent prefetching mechanism, an appointed prefetching mechanism, and a cache coherency protocol. With the network I/O protocol, the connected application server is able to access the I/O devices located anywhere in the WAN that are either specified by the users or configured automatically.

References

[1] A. Nye (Ed.). *X Protocol Reference Manual*. O'Reilly & Associates, Sebastopol, CA, 1992.

[2] Boca Research, Inc. *Citrix ICA Technology Brief*. Technical White Paper, Boca Raton, FL, 1999.

[3] Joel P. Kanter. *Understanding Thin-Client/Server Computing*. Microsoft Press, 1998.

[4] Microsoft Corporation. Generating Local Addresses and Communication Sets for Data Parallel Programs. *Comparing MS Windows NT Server 4.0, Terminal Server Edition, and Unix Application Deployment Solutions*, Technical White Paper, Redmond, WA, 1999..

[5] Brian K. Schmidt, Monica S. Lam, J. Duane Northcutt. The Interactive Performance of SLIM: a Stateless, Thin-Client Architecture. *The 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, 1999.

[6] George Coulouris, Jean Dollimore, Tim Kindberg. *Distributed Systems Concepts and Design*. 3rd Edition, Addison-Wesley, Reading, MA, 2001.

[7] Eliezer Levy, Abraham Silberschatz. Distributed File Systems: Concepts and Examples. *ACM Computing Surveys*, Vol. 22, No. 4, December 1990.

[8] Y. S. Chen. A Practical Approach for File Prefetching in Distributed File System. National Taiwan Normal University Graduate Institute of Computer Science & Information Engineering, Master Thesis, Advisor: Gwan-Hwan Hwang, 2003.

[9] The Design and Implementation of Appointed File Prefetching for Distributed File Systems. Gwan-Hwan Hwang, Hsin-Fu Lu, Chun-Chin Sy, and Chiu-Yang Chang. *Journal of Research and Practice in Information Technology*, Vol. 40, No. 2, May 2008..

[10] Mi Rjana Spasojevic, M. Satyanarayanan. An Empirical Study of a Wide-Area Distributed File System, *ACM Transactions on Computer Systems*, Vol. 14, No. 2, May 1996, Pages 200–222.

[11] K. Edwards. *Core JINI.* Prentice Hall, Upper Saddle River, NJ, 1999.