Tiny Client System

Masayoshi Sekiguchi, Haruo Oishi, Takashi Nakashima
NTT Access Network Service systems Laboratories
Chiba-shi, Japan
{masa, oishi, nakasima}@ansl.ntt.co.jp

Abstract - Use of the Internet has increased rapidly, and data communications have changed from being hobby-based to becoming essential for daily living. It is important for everybody to easily obtain necessary information from anywhere and at any time to confidently attain a true information society infrastructure in the future. The terminals are one of the most important elements. To obtain reliable information in the same environment anywhere and at any time, the introduction of a Server-Based Computing (SBC) system is promising, because a SBC system accumulates and processes information in the one place. Cellular phones are also useful as terminals to easily achieve information access anywhere and at any time. To achieve an SBC system for cellular phones that can operate comfortably, we propose a light-weight protocol called the “UI component oriented protocol” and a function of speculative execution based on the prediction of user events. Moreover, we implemented a prototype system, and confirmed that our system worked quickly within the communications environment of cellular phones.

Keywords-component; Thin Client; Server-Based Computing; Speculative Execution; Light-weight Protocol; Prediction of User Events

I. INTRODUCTION

Recently, the number of Internet users has been increasing rapidly. Also, with the remarkable progress in hardware development, CPU capabilities have improved, HDD capacities have increased, and prices have fallen. As a result, almost everyone can own one or more very high performance PC (Personal Computer).

Although we generally use PCs to access the Internet, not everyone can use these because advanced skills are necessary. And, they cannot be used immediately when required because it takes time for startup and shutdown and they cannot always be carried around. To own PC is not a purpose of users. Their true purpose is to ensure necessary information that is provided regardless of the means.

In Japan, the “i-mode” [1] service, which NTT DoCoMo provides is widespread. It enables the Internet to be accessed from cellular phones. It is currently being used by more than 40 million people of all ages and both sexes, and this is much bigger than the number of users using PCs on the Internet.

Table I. lists the “i-mode” services, and most of the content is written in text. Contents providers write this in CHTML (Compact-HTML), which is a subset specification of HTML. Although the functions the user can use are limited by a simple screen based on HTML, there are sufficient functions for what the user really wants. The most important thing is relevant and immediate use by anyone when required.

Against this background of high performance, hardware price-cutting, and the spread of cellular phones, we propose an information society infrastructure achieved through Server-Based Computing (SBC) where the users’ PCs and HDDs are deployed in the network and cellular phones are information I/O terminals.

By deploying users’ PC and HDD as the server, consistency of information can be guaranteed, and the same use environment can be provided anywhere, any time, and any devices. User can continue a current application between different terminals (“Application Roaming”). Moreover, because the carrier maintains this server, there are advantages where users are relieved of the worry of information loss due to security threats such as viruses or failure, and they can use information services with ease.

By making his portable cellular phone a client, the user can actually acquire information immediately, anywhere and at any time when required. Although progress with cellular phones has been rapidly increasing, their processing performance is still far from that of PCs. In addition, there is the additional problem that cellular phones operate in a narrow-band and high latency communications environment.

In this paper, we propose a lighter protocol and a method of concealing communications latency to achieve convenient information access in a narrow-band and high latency communications environment in the SBC system. And, we confirmed their effectiveness through the prototype system implementation. Naturally, this system can work on several devices such as PCs, PDAs, and cellular phones. We called it

<table>
<thead>
<tr>
<th>Category</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Mail</td>
</tr>
<tr>
<td></td>
<td>Chat</td>
</tr>
<tr>
<td>Information</td>
<td>News, Weather Report</td>
</tr>
<tr>
<td></td>
<td>Traffic Information</td>
</tr>
<tr>
<td></td>
<td>Sports</td>
</tr>
<tr>
<td>Network Game</td>
<td>Peer-to-Peer games</td>
</tr>
<tr>
<td></td>
<td>Server-Client games</td>
</tr>
<tr>
<td>Shopping</td>
<td>Network Shopping</td>
</tr>
<tr>
<td></td>
<td>Auctions</td>
</tr>
<tr>
<td>Voice, Image</td>
<td>Ringing Tone</td>
</tr>
<tr>
<td></td>
<td>Standby Screen</td>
</tr>
</tbody>
</table>

0-7803-8785-6/05/$20.00 (C) 2005 IEEE
the “Tiny Client System” because it is a lighter version of the thin client system.

II. SBC SYSTEMS

MetaFrame [2], Go-Global [3], and Bullant [4] are three typical SBC products. The features of the protocol used in these products are different.

A. Existing SBC systems

MetaFrame was developed by Citrix and its protocol is based on pixel data called Independent Computing Architecture (ICA). MetaFrame has the advantage that general applications can be performed. However, it has the disadvantages in that the client for cellular phones is not provided and the data size for communication is excessive for cellular phones because information exchanges are based on pixels.

Go-Global was developed by GraphOn and its protocol is based on a graphic Application Program Interface (API) called RapidX. Go-Global can execute general applications just like MetaFrame. However, it has disadvantages in that the client for cellular phones is not provided and the data size for communication is excessive although this is smaller than types using the protocols based on pixel data.

Bullant was developed by Gravana and its protocol is called the Remote Application Protocol (RAP) that transmits meta-data about UI. Although Bullant has the client for cellular phones, its response is too slow when it operates on a cellular phone. Further, it cannot suspend current tasks and cannot resume them. It also requires exclusive applications and provides a different programming model (API) for each device, which are obvious disadvantages.

Thus, there are no existing SBC products for cellular phones that can achieve quick responses that keep users satisfied.

B. Requirement for the Tiny Client System

Fig. 1 is an overview of our Tiny Client System. The “Personal Application Server” which performs personal applications is accessed from cellular phones, PDAs, or PCs (Thin Client). Information on UI Events that the user performed is transmitted to the server from the thin client, and information on the UI is transmitted to the thin client by the server. When this occurs, the communications between cellular phones and server in a narrow-band and high latency environment is a key issue when constructing an SBC system. This is because a rich UI cannot be obtained since the amount of data which is communicated is restricted in a narrow-band environment, and quick responses cannot be achieved since the Round Trip Time (RTT) is large in a high latency environment.

C. Network characteristics of cellular phones

In Japan, NTT DoCoMo provides two methods of communications of cellular phones. These are called “MOVA” and “FOMA”. “MOVA” is used the method of Personal Digital Cellular (PDC), and its upstream bandwidth is 9.6kbps, and its downstream bandwidth is from 9.6kbps to 28.8kbps. This is too small to achieve an SBC system with existing SBC protocols. “FOMA” is used the method of W-CDMA, and its upstream bandwidth is 64kbps, and its downstream bandwidth is 384kbps. However, according to the result which we actually obtained under the condition of frequent small data communication like an SBC system, its actual bandwidth is as small as PDC. And, there is very high communications latency in both methods. We measured its latency on real cellular phones, and RTT is about 1,000 milliseconds in both methods.

Therefore, to achieve an SBC system with cellular phones, a light-weight protocol that a rich UI can be obtained in such a narrow-band environment and also a technology that offers network-transparent operability by concealing such high latency are required.

The following section describes our SBC features in more detail.

III. LIGHT-WEIGHT PROTOCOL

The following requirements are satisfied in our light-weight protocol. First, the data size needs to be sufficient to display a rich UI. Next, operation with most devices from cellular phones to PCs should be possible. Finally, the same application should be able to be performed between different devices. In addition, to reduce the load of the programmer, the same programming model should be provided regardless of the device types.

The following section describes our light-weight protocol in detail.
A. Problems of existing protocol

A traditional thin client protocol that uses pixel data has problems with not only the large amount of data but also with changes in the number of colors and resolution.

At protocols with interfaces based on graphic API, the amount of data does not change with the number of colors and resolution. When the UI is too rich, an excessive amount of data is required because the number of APIs called from applications increases.

B. UI component oriented protocol

To solve these problems, we propose a UI component oriented protocol.

When a programmer writes a UI program to create a GUI, he generally uses UI components such as “TextField”, “Label”, and “Button”, which are provided as an API. How a UI component is drawn on a screen is determined by the API provider. A programmer can customize the look and feel of UI components such as the background color by changing several property values, which can be set. Since drawing information is in the server, with existing SBC system, it needs to be transmitted as pixel data or graphic API information. This causes increase of the amount of communications data.

The amount of communications data can be reduced by having the drawing information at the thin client beforehand. By managing the default property values of UI components at the thin client side, the amount of data is reduced even more because most property values have not changed. Here, the screen for the application can be reproduced on the thin client side by transmitting only the kinds of UI components and the property values with changes from the server to the thin client (see Fig. 2.).

The advantages of this method are that the amount of data does not change with the number of colors or resolution, or the complexity of the display for GUI. Moreover, the amount of data is smaller than with methods using pixel data or API information because it is not necessary to send screen information on UI components but only to transmit property values with changes.

At the thin client side, parameters of user events that occurred to a UI component are transmitted to the server. The kinds of events, objects of events (UI components), and parameters of events are transmitted. To reduce the amount of data, not all events that occurred are transmitted but only necessary ones at this time. Generally, when a programmer wants to do something against user events, it is necessary for him to describe the content of the process beforehand for each trapped event (event handler). Therefore, an event which does not have a description of event processing is not notified to the server side even if the event occurred. Events on cursor motion are not transmitted, too. As a result, the frequency of communication can be decreased because useless data is not generated. To notify whether there has been a description of event processing or not, the presence of this is defined as the property value of a UI component, and it is transmitted to the thin client side as well as other property values (see Fig. 3.).

C. Unified programming interface

To reduce the load on programmers who create applications, it is important that a single program can work on different devices. Therefore, the thin client protocol needs to be shared by any devices, and an optimal screen for every device should be drawn by thin client middleware that interprets the protocol. As a result, a single program can work on different devices.

Also, an existing program should work without changes in our SBC system so that existing resources can be used more effectively. Therefore, we provide a new library that has exactly the same interface as the existing UI library such as Java AWT (Abstract Window Toolkit). Since the new library has a function to convert the property value into the protocol, existing programs can work without changes.
IV. SPECULATIVE EXECUTION BASED ON EVENT PREDICTION

To achieve high usability, it is necessary to offer network-transparent operability in the SBC system for cellular phones, although this greatly depends on the network. Not only adapting the SBC system to a narrow-band environment with a light-weight protocol but also the technology to conceal the high latency of cellular phones is required.

When the content of the user event is transmitted after the user causes the event and application is performed on the server, even if the processing time on the server is assumed to be 0, the user will be made to wait during the RTT. Since RTT is very large in a high latency environment, the user will be made to wait for a long time. However, it is impossible to reduce the communication latency itself. Therefore, to minimize the “effect” of the communication latency, we propose a function of speculative execution based on predicting user events.

If the content of the user event can be predicted, application can be performed on the server according to the prediction (Speculative Execution), and the execution results can be transmitted to the thin client beforehand. The execution results can be stored in the thin client terminal. When the user actually causes the event, an updated GUI screen can be displayed by loading the prediction which has been stored. Because communication between the server and thin client has not happened in this case, there is no influence by communication latency regardless of its size.

If the prediction can be transmitted by the time the user actually operates the GUI, the time until the updated GUI screen is displayed is only the processing time in the thin client, and this is almost 0. Even if the prediction cannot be transmitted by the time the user operates the GUI, the updated GUI screen can be displayed earlier than the parameters of the user event is transmitted after the event occurs because processing has begun beforehand.

The content on the user events can be predicted for the following reasons. First, the number of events that the user can cause on the GUI is limited. Particularly, at a terminal that has small screen resolution like that on a cellular phone, the possible number of user events occurring decreases because there are few UI components positioned on the screen. Therefore, this is easy to predict because there are few choices. Second, there is generally at least one second until the user recognizes, evaluates, and operates the GUI when the GUI is displayed. This time can be spent predicting on the server, and this is enough. Moreover, when the GUI screen that has been displayed before is displayed again, the possibility to do the same operation as the operation done before is expected. Therefore, the prediction accuracy can be improved by referring the past results. Furthermore, when the programmer does not describe the content of process for the event, the event can be excepted from the object of the prediction. Finally, the longer user operation takes, the more patterns for user events can be predicted, and as a result the prediction hit ratio can be improved. In addition, to reduce the communication frequency, two or more prediction results are transmitted at a time.

When a “prediction engine” starts the prediction at the server side, it lists all possible events in the order of its possibility from high to low. The event is notified to the replicated application in order of the lists, and the event processing can be done. The results are transmitted to the thin client.

At the thin client side, the received results are stored in a terminal. At this time, if the user has not operated the GUI screen yet, another prediction can be executed at the server side, and additional execution results can be transmitted to the thin client. When the user actually causes the event, “event manager” searches for the corresponding result in the “prediction result store”. If it found, the updated GUI screen can be displayed immediately by loading the stored prediction result. When the corresponding result does not exist, the content of the user event can be notified to the server. The usual event processing can be done (see Fig. 4.).

V. PROTOTYPE

We implemented a prototype system that had our light-weight protocol and a function of speculative execution based on predicting user events. We succeeded in operating on a real NTT DoCoMo cellular phone.

We implemented a middleware in “DoJa” [5] on the thin client side, which is a Java specification for NTT DoCoMo cellular phones. We implemented middleware in Java Servlet on the server side, and applications need to be written in Java language using UI library we provided. Applications are deployed on the server.

Since the prototype system is implemented to verify our proposals, we implemented only basic UI components such as “TextField”. We prepared the interface for the UI component of our UI library that has the same interface as the “DoJa” library.

We implemented sample application which enables the end user to control the access network QoS anywhere and at any time (see Fig. 5.).
VI. EVALUATION

We evaluated our Tiny Client System using the sample application.

Traffic was measured to evaluate the performance of our light-weight protocol. And, we measured the response time to evaluate a function of speculative execution based on the predicting user events.

A. Traffic

We established the average number of bytes necessary to compose one screen by measuring the amount of data that was exchanged between the server and the Tiny Client (Table II.). Three kinds, when a user event was predicted on our Tiny Client System, when it was not on our system, and when Bullant system which is other thin client system for cellular phones was used, were measured.

From the results in Table II, we can see that our system generated fewer data than Bullant system, despite the fact prediction data was included.

<table>
<thead>
<tr>
<th>Prediction data</th>
<th>Amount of data (bytes/screen)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our system (prediction data not included)</td>
<td>425.1</td>
</tr>
<tr>
<td>Our system (prediction data included)</td>
<td>879.6</td>
</tr>
<tr>
<td>Bullant</td>
<td>1127.7</td>
</tr>
</tbody>
</table>

TABLE II. EVALUATION OF TRAFFIC SIZE

B. Response time

The response time when the sample application was performed once was measured. While performing this application once, screen changes occur six times. The response time means the interval between the time user actually started operation and the time the GUI screen was displayed. The results are listed in Table III.

From these results, our system can greatly reduce response times with the function for speculative execution based on the predicting user events. 206 milliseconds of response time are short enough to avoid user frustration [6]. In addition, it was only one time that prediction did not hit while six screen changes.

VII. CONCLUSION

We proposed the “Tiny Client System” featuring the newly development of a light-weight protocol and a function of speculative execution based on predictions of user events. “Tiny Client System” can make a rich UI and can operate quickly in the poor communications environment of cellular phones.

We implemented a prototype system using NTT DoCoMo cellular phones. Our proposals work quite well within a real environment.

VIII. REFERENCES