A Case for Replicated Client-Server Model for Optimal Application Response Time in the Presence of Unpredictable Network, Server Loads and Usage Patterns in Mobile Computing Environment

Chuang-Wen You and Hao-hua Chu
Department of Computer Science and Information Engineering,
National Taiwan University
r91023,hchu@csie.ntu.edu.tw

***Abstract*— this paper examines the fundamental limitation of adaptation based methods in the presence of unpredictability and instability in wireless network bandwidth, server loads, and usage patterns. We argue that existing adaptation based methods will fail to produce good application response time given such unpredictability and instability, because they require somehow accurate prediction on resource conditions and usage patterns in order to perform effectively. We propose a new, simple yet powerful, replicated client-server model which overcomes this fundamental problem of unpredictability and instability; at the same time, it can also achieve nearly as good application response time as any adaptation methods. The basic idea behind replicated client-server model is simple – application execution is replicated on both client and server, and the faster result is returned to the user. It provides the benefit of faster response time at the cost of computational overhead in replicating executions on both client and server. In other words, it is about trading off more computing resource with better application response time and user experience. As computing resource continues to be improved exponentially (according to Moore's laws) in speed, capacity, price and abundance, we believe that improvement in user experience becomes appealing tradeoff for the computational overhead in replicating execution.**

I. INTRODUCTION

Application response time is one of the key parameters that affect interactive user experience in Internet-based applications. In this paper, we would like to make a case for a new replicated client-server model that could achieve good application response time in the presence of unpredictable wireless network bandwidth, server loads, and usage patterns in mobile computing environment. The basic idea behind the replicated client-server model is simple – replicate, whenever possible, computation on both the mobile client device and remote server, and bring the faster result to the user. Although this model is simple, it works well to achieve good application response time in mobile and distributed computing environments, where unpredictability and instability in wireless network bandwidth, server loads and usage patterns can occur frequently.

Unpredictability and instability in resource conditions and usage patterns are known to be fundamental limitations for known adaptation based methods. In general, adaptation methods make use of feedback-prediction-adjustment control loop. If future resource conditions or usage patterns cannot be accurately predicted based on feedbacks, adaptation will fail – consider the example of optimizing application response time, inaccurate prediction on network bandwidth or server loads would produce misplacements of application components on mobile client device and server.

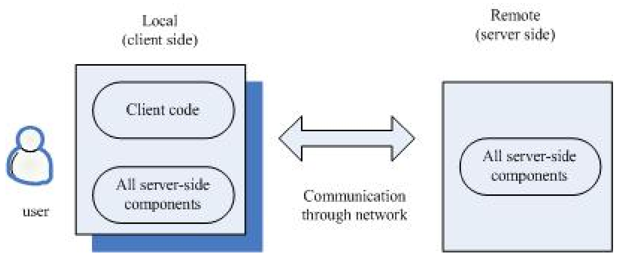
Local
(client side)
Remote
(server side)
Client code
All server-side components
user
Communication through network
All server-side components

Fig. 1. Replicated Client-Server Model

We illustrate how replicated client-server model works. Consider the case when the network bandwidth suddenly decreases by a large margin, so it takes much longer network time to transmit result from server to client. This means that client-side execution is likely to produce faster result than server-side execution. Therefore, the result from the faster client-side execution is likely to be delivered to the web browser first. Consider the 2nd case when the network bandwidth suddenly increases by a large margin, so it takes much shorter network time to transmit result from server to client. This means that client-side execution on a slower processor is likely to produce slower result than server-side execution on a faster processor. Therefore, the result from the faster server-side execution is likely to be delivered to the web browser first.

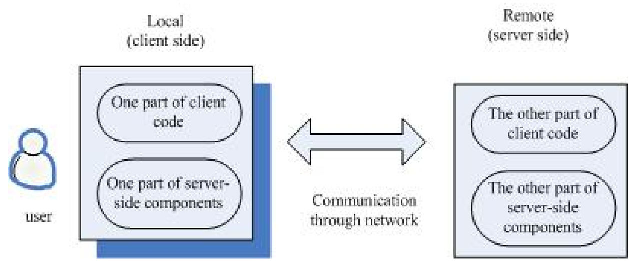
Local
(client side)
One part of client code
One part of server-side components
user
Communication through network
Remote
(server side)
The other part of client code
The other part of server-side components

Fig. 2. Extended Client-Server Model

The replicated client-server model is fundamentally different from both traditional client-server model in distributed system and the extended client-server model in mobile computing [10]. The difference is in replicated vs. non-replicated computation. In traditional client-server model, each piece of computation in a distributed application is assigned to run either on the client or the server, but never both. In the extended client-server model, some traditionally server functions are moved to the mobile client devices, and some traditionally client functions are moved to the servers. However, this extended client-server model never replicates computation. That is, if a server function is moved to the client device, it is executed on the client device, but never on the server device at the same time.

Our replicated client-server model is also different from Coda file system [11], which supports disconnected operations for mobile client devices by caching and hoarding (a form of replication) files from file servers. That is, a mobile client device can continue to perform file operations on its caches while it is physically disconnected from the network file servers. The main difference is that Coda replicates data, whereas our approach replicates computation. In addition, our approach does not distinguish between first-class replica on the server and 2nd class replica on the mobile client device. The results from both the client-side execution and server-side execution are mirrored and consistent.

The main disadvantage for replicating computation is its computational overhead. Since replicated execution would produce two identical results on both client and server, it is considered a waste of computing resources. However, its upside is that users will always get the faster response time of either the client-side execution (i.e., called the thick client approach) or the server-side execution (i.e., called the thin client approach), resulting in improved user experience. In other words, the replicated client-server model can get the better application response time of either the thick client approach or the thin client approach. If replicated execution is applied at the granularity of application component (rather than the whole application), the replicated client-server model can get the best application response time of any application partitions between client and server (i.e., called the smart client approach).

This "always getting the faster response time" is valuable in mobile and ubiquitous computing environments. The replicated client-server model is enabling a new *tradeoff* between higher computing cost and better application response time (user experience) under unpredictability and instability in resource conditions and usage patterns. Given the Moore's law on exponential improvement in processor speed, memory size, and storage capability, computing resources (especially servers running on the fixed infrastructure) are becoming ever more abundant over time. On the other hand, human attention (which includes waiting time for application response) remains relatively static over time [12]. We believe that improvement in user experience is a good tradeoff for the additional computing overhead in computation replication.

The rest of the paper is organized as follows. Section II defines and models the Internet-based applications. Section Ⅲ defines and models application response time function(s) in terms of dynamic factors such as wireless bandwidth, server loads and usage patterns. Section Ⅳ describes related work in application partition and adaptation methods. Section Ⅴ shows the impact of these dynamic factors on application response time using the Java Pet Store application. It also describes limitations in existing adaptation methods in the presence of unpredictability and instability in dynamic factors. Section Ⅵ explains the architecture of the replicated client-server model, and how it can overcome unpredictability and achieve good application response time. Section Ⅶ draws our conclusion and future work.

II. Model of Internet-based Applications

We assume that Internet-based applications are built using components that can be executed either on a client mobile device or server machines. This assumption is inline with the current trend in adapting the service-oriented architecture (SoA) for Internet applications (e.g., XML web services) [15]. That is, applications can be composed from independent components that communicate through well-defined interfaces and can be distributed to run on any machines on the Internet.

We show an application example called Java Pet Store that has been slightly modified to adapt SoA and 3(n)-tier architecture shown in Figure 3. The client-tier is the web browser that takes user inputs and sends them to the web tier. It also renders web pages on a mobile client device. The web-tier contains front-end components that generate webpage presentations. It also has control components that map a HTTP request to the corresponding action or event to EJB tier. The application logic tier contains Enterprise Java

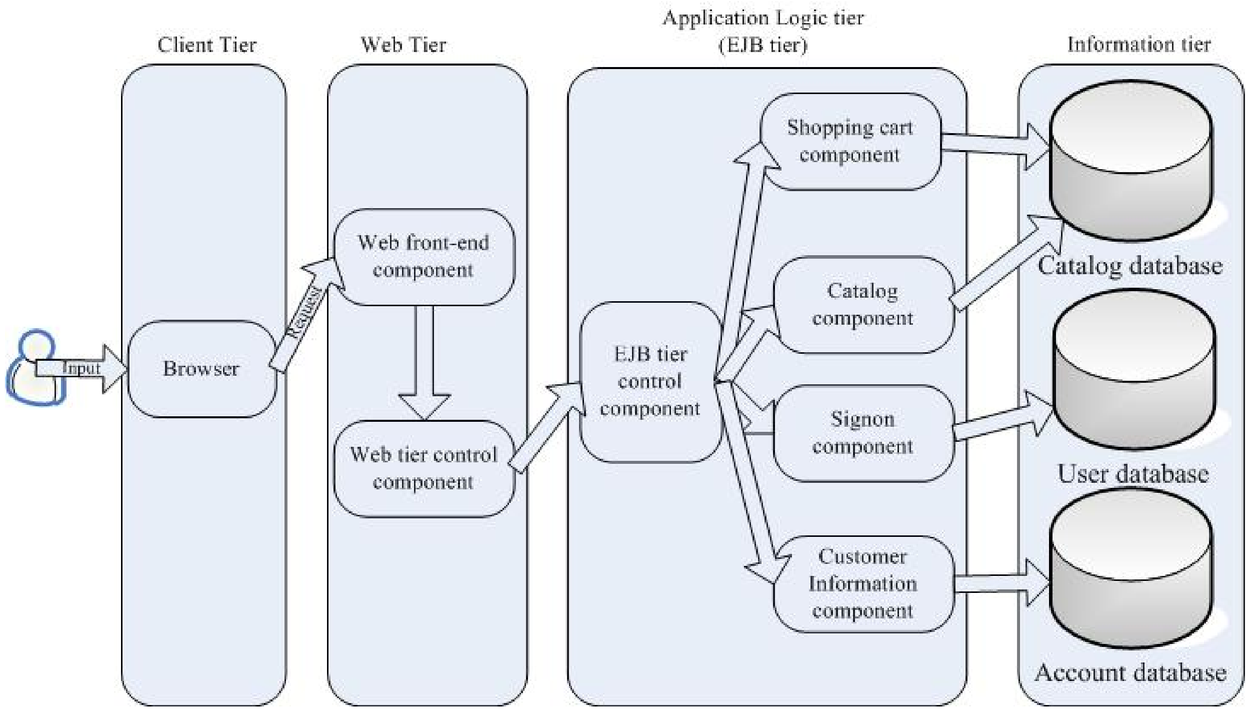

Fig. 3. Component-level View of Java Pet Store Application

Bean (EJB) control component that maps events generated from web tier to corresponding EJB actions. It also invokes EJB beans to process these actions. The EJB beans may need to query the database on the information tier.

Given widespread adoption of SoA in Internet-based applications, we assume that most of application components in the web and application logic tiers can be freely distributed (or replicated) to mobile client devices and/or server machines. In addition, we assume the increased HW/SW capabilities in mobile client devices will enable them to have some limited server capabilities to run application components on the web and application logic tiers. Some early examples of server capabilities on mobile devices are Joeysoft J2EE server [13] and Intel Personal Server [14].

III. Model of Response Time Function

We define application response time of an Internet application (e.g., Java Pet Store) to be the amount of time from the browser receiving input from a user, to a new resulting web page being completely rendered in the browser. In the traditional, extended and non-replicated client-server model, application response time can be decomposed into the following five parts:

- *browser_cli_time* measures client-side processing time of client-tier components. For example, this time corresponds to a user clicking on the form to a web-tier components running on the client or/and server. After the resulting web page is returned, the browser also renders it.
- *webapp_cli_time* measures processing time for application components on the web/application logic tiers that are partitioned to run on a mobile client device. For example, this time corresponds to web-tier or application logic tier components executing some JSP or EJB code when they receive HTTP requests.
- *webapp_svc_time* measures processing time for application components on the web/application logic components that are partitioned to run on the server machine. This is similar to *webapp_cli_time*, except that these components have been assigned to run on the server.
- *db_svc_time* measures database access time in the information tier. For example, this time corresponds to processing a query in a relational database.
- *net_time* measures the network latency to exchange messages between the client-side and server-side components. They can include HTTP request and response messages or RMI calls.

We define the following formula for the application response time function in the traditional extended and

Response Time Component	**Server Loads**	**Wireless Bandwidth**	**Usage Pattern s**
browser_cli_time			
webapp_cli_time			x
webapp_svc_time	x		x
db_svc_time	x		x
net_time		x	x

Table Ⅰ: Relation between Dynamic Factors and Response Time Components

non-replicated client-server model:

$$R_{ext}(application\ partition) = browser\_cli\_time + webapp\_cli\_ptime + webapp\_svc\_time + db\_svc\_time + net\_time \qquad \text{Eq. 1}$$

Note that the above response time function makes a simplifying assumption that all application components are executed sequentially rather than in parallel. The purpose of this assumption is to make it easy to analyze and describe the impact of dynamic factors (e.g., wireless network bandwidth, server loads and usage patterns) on the application response time. In other words, our replicated client-server model can work regardless of whether the execution flow of application components is sequential, parallel or mixed.

These dynamic factors influence different parts of the application response time function. We define them and show how it affects different parts of the response time function. Server load measures the server utilization. If the server is overloaded, requests from clients are likely to queue up on the server, resulting in long waiting time in *webapp_svc_time* and *db_svc_time*. Wireless network bandwidth measures the available data rate of the wireless channel connecting the mobile client device to the network. It affects network latency (*net_time*) of exchanging messages between a mobile client device and server machines. Service usage patterns have a more complex definition. It can be defined as user invoking a sequence of actions, or accessing a set of features (among many) provided by an Internet application.

At the application component level, different service usage patterns result in different execution paths and execution of different sets of application components. Usage patterns can change all aspects or parts of the application response time function, including processing time of selected components on the execution path both on the client or server side, the types of messages exchanged between the client and server, and database query result size. We summarize relationships between dynamic factors and response time function in Table Ⅰ.

Given that the replicated client-server model performs parallel execution on both client and server and takes the faster result, its response time function is different from the traditional extended client-server model:

$$R_{replicated} = min\{ R_{ext}\ (client\text{-}side\ execution), R_{ext}\ (server\text{-}side\ execution)\} \qquad \text{Eq. 2}$$

Adaptation methods translate the problem of finding the optimal application partition (between client and server) into an optimization problem to minimize the application response time function shown in Eq. 1. Since parameters in the function are continuously changing, adaptation methods must try to continuously re-optimize the application partitions with the current values in wireless bandwidth, server loads, and usage patterns. However, if these dynamic factors become unpredictable and instable (i.e., the predicted values are different from the real values in dynamic factors) the computed application partition will not produce good response time. Experimental results in section V will confirm this.

There are many situations where wireless bandwidth, server loads, and usage patterns can become unpredictable. Server loads depend on the number of requests from users. A study done by Padmanabhan et. al. [16] has analyzed server access dynamics in MSNBC web site based on server traces. They have found little stability in the popularity of contents and client access patterns. Wireless bandwidth depends on the radio link condition between the mobile client device to base stations, and the number of active mobile client devices in a given cell. It has been shown that radio link condition is unpredictable and can change drastically over relatively small user movement (e.g., walking in front of a tall building blocking the radio signals). Usage patterns are difficult to predict. It can depend on a lot of complicated factors, such as user group, time of day, type of mobile device, physical location of the user, etc.

In addition to the above-mentioned dynamic factors, there are many static factors that can change the application response time, such as the processor speed of the mobile client device and the server machines. Since static factors do not have the unpredictable or instable problem, they can be easily addressed by existing static partitioning method [7] or runtime adaptation methods [2] [3] [17], so they are not the focus of this paper.

IV. RELATED WORK

We classify the related work into two main categories. The first category is based on compile-time analysis to decide how and where to partition (split) an application

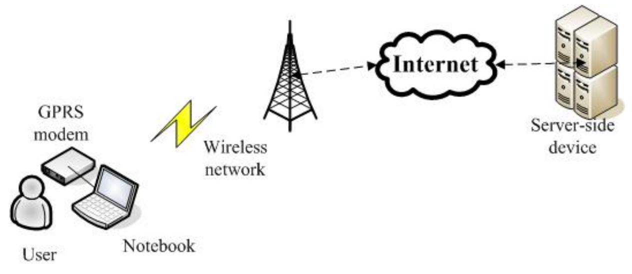

Fig. 4. Experimental Setup

HW/SW	Client-side Device	Server-side Device
CPU clock rate	Pentium Ⅲ 400MHz	PentiumⅣ 2.6GHz
Memory	128MB	512MB
OS	Windows XP	Windows 2000 Server
software building block	1. J2EE v1.4 Developer Release 2. browser	1. J2EE v1.4 Developer Release

TABLE Ⅱ HW/SW Configuration

between client and server, so that it can optimize certain system-level or application-level metrics, such as application response time, network traffic, power consumption, etc. The second category is focused on runtime techniques to reconfigure the application partition according to the changing values in these dynamic factors. Note that these two categories of approaches can complement each other, i.e., it is possible to apply compile-time analysis at application compile time, followed by runtime adaptation at application runtime.

A. Partitioning method and limitations

Method partition [7] in JEcho [8] (a distributed event system) uses compile-time analysis to partition a method into client and server parts. This is how it works – it first represents a method using a control flow graph. Each node in the graph is an instruction and the cost of each edge is determined by a cost model. Then, it tries to find all potential splitting edges in this method. Since each splitting edge incurs communication cost between client and server, it finds an optimal set of splitting edges that minimizes overall communication cost. This compile-time analysis requires time-consuming analysis of the source code. However, there are limitations in the compile-time analysis, such as branch prediction, user input variables and other forms of non-determinisms in the source code. To address non-determinism, it tries to predict the outcomes of these non-determinisms.

Mutable services project [1] partitions application components into edge and remote sets. Frequently used components are cached on edge servers that are closer to the mobile client devices. This helps to reduce application response time. In addition, they group multiple synchronous RMI messages into one large message between edge servers and remote servers. This helps to reduce the network latency of multiple synchronous RMI calls. Its result has shown improvements in application response time for most of the user requests. However, the problem with this approach is that some efforts in program analysis and code modifications are required from the application developers.

B. Adaptation methods and limitations

There are many runtime adaptation systems such as Chroma [3], Coign [2] and Agilos [17]. Chroma [3] is about leveraging remote execution on servers from resource-poor mobile devices. It enables application developers to specify different application partitions that leverage server resources in different mobile environments. These application partitions are called tactics. The Chroma system can support automated tactic selection. The automated selection is based on the amount of available computing resources at runtime, e.g., network bandwidth and server capabilities, as well as history-based predictions on the future application's resource demands. Chroma then chooses the best tactic plan such that it can satisfy the predicted resource demand.

Coign [2] is an automated distributed partitioning system that constructs a graph model of the application's inter-component communications, and co-locating strongly interacting components on the same machine to reduce communication overhead. At runtime, Coign creates a profile of inter-component communication. Then, it uses graph-cutting algorithm to choose an application partition that minimizes execution delay due to network communication. Since network delay is a dynamic factor, Coign needs to periodically monitor the network delay and performs application repartition accordingly.

Agilos [17] is about creating agile, adaptive middleware architecture to support application-aware QoS adaptation. One of its key components is called adaptor which monitors both system-level state, such as CPU load or network bandwidth, as well as application-level QoS. The adaptor also promotes a global awareness of system-level resource availability. Other components called tuner and configurator use the information provided by adaptors to adjust application parameters or activate functional reconfiguration actions based on a fuzzy control model. The rule base of this fuzzy control model is defined by a combination of human expertise and trial-and-error experiments.

In comparison to our replicated client-server model, these related runtime adaptation systems have the following three

Server Loads	Thin Client Partition	Thick Client Partition
0%	0.614 s	2.025 s
10%	1.118 s	2.183 s
20%	2.004 s	2.604 s
30%	3.260 s	2.013 s

TABLE Ⅲ: Impact of Unpredictable Server Loads on Application Response Time in WLAN Network

Usage Patterns	Thick Client Partition	Thin Client Partition
No sign-on	2.489 s	9.016 s
After sign-on	45.065 s	11.241 s

TABLE Ⅳ: Impact of Unpredictable Usage Patterns on Application Response Time

limitations given unpredictability and instability in resource conditions and usage patterns. The first limitation is that the application partition may be instable and does not converge. The second limitation is that there are computing overheads associated with each application reconfiguration. If the application partition does not converge to a stable partition, it will result in significant computing overheads. The third limitation is that some of these systems require application developers to manually configure the application, such as defining tactics or rule base of the fuzzy control model. In comparison, our replicated client-server model does not have the above-mentioned limitations.

V. QUANTITATIVE IMPACTS OF UNPREDICABILITY ON APPLICATION RESPONSE TIME

We describe the quantitative impacts of these dynamic factors (wireless bandwidth, server loads, and usage patterns) on the application response time.

The experimental setup is shown in Figure 4. The mobile client device is a slow Notebook PC running a 400 MHz Pentium III processor, which has roughly equivalent to processing power of a current Pocket PC device. The server machine is a desktop PC running 2.6 GHz Pentium IV processor. They are connected through two possible wireless networks – GPRS or 802.11b WLAN.

We measures the application response time of an Internet application called Java Pet Store [4]. It is released by Sun Microsystems in its J2EE BluePrints program [5]. The Java Pet Store application adapts 3(n)-tier architecture shown in Figure 3. Both the mobile client device and the server machine host J2EE application servers [6]. The HW/SW configuration is summarized in the table Ⅱ.

We assume all replicable application components are already present on both mobile client device and server machines, so we explicitly exclude any one-time mobile code downloading time from the application response time.

The Java Pet Store application has its component-level structure shown in Figure 3. All user interactions with the Pet Store application generate HTTP request messages from the browser to the web-tier components. Different usage patterns may traverse different execution paths on different sets of application components.

In the experiment, we would like to demonstrate that even a simple feature such as "viewing the shopping cart", has different usage patterns with different execution paths and different application response time. Below are two possible usage patterns associated with viewing a shopping cart.

- *No sign-on*: a user views his/her shopping cart prior to sign-on.
- *After sign-on*: a user views his/her shopping cart after sign-on.

In the latter case of after signon, Java Pet Store will generate *mylist* of individual's favorite pets based on the sign-on user id and the user preferences. In the case of no sign-on, Java Pet Store will not generate mylist. The generation of mylist is handled by a web-tier component called mylist.jsp.

Based on our measurements, the optimal application partition for the "no sign-on" usage pattern is the thick client partition (web/application logic components are assigned to run on the mobile client device) on GPRS. On WLAN, the optimal application partition is the thin client partition (web/application logic components are assigned to run on the server). This makes sense given that GPRS has small network bandwidth and high network latency, so it favors the thick client partition which reduces as much communications as possible. On the other hand, WLAN case is reverse. WLAN has high network bandwidth and low network latency; therefore, it favors the thin client partition that leverages the faster processing speed on the server.

We measure the application response time by varying those dynamic factors (server loads, network bandwidth and usage patterns). Because delay and bandwidth variation in GPRS network is common [9], the variance of the response time is larger than wired network. In order to show these impacts clearly, we use the average response time taken over multiple measurements.

Network Bandwidth	Thin Client Partition	Thick Client Partition
GPRS	9.016 s	2.489 s
WLAN	0.614 s	2.987 s

TABLE Ⅴ: Impact of Unpredictable Network Bandwidth on Response Time

A. Impact of unpredictable server loads on response time

Our measurements are based on the following setting: network is WLAN, and a user views his/her shopping cart in the no sign-on pattern. When there is little or no server loads (utilization), our measurements shows that the thin client partition gives the optimal application response time. As the server load increases, the response time of the thin client partition is gradually approaching the response time of the thick client partition as shown in Table III. When the server load reaches or exceeds 30%, the thin client partition (components are placed on the server) becomes worse partition, and the thick client partition (components are placed on the mobile client device) becomes the optimal partition. This shows that *varying the server loads may drastically change the optimal application partition.* In other words, it is difficult to apply adaptive application partition that can give good response time under unpredictable and instable server loads.

B. Impact of usage patterns on application response time

When a user wants to view his/her shopping cart, there are two possible usage patterns described in the previous section. Table Ⅳ shows the impacts of these two usage patterns on application response time. GPRS is used to connect mobile client device and server. The server load is set to be low.

Our measurements in Table V show that the thick client partition gives the optimal application response time in the "no sign-on" pattern. The reason is that this usage pattern does not require access to the database to retrieve user preferences, so the thick client partition can completely avoid sending any messages over the slow GPRS. After we change the usage pattern in which the user views his/her shopping cart after sign-on, the thick client partition becomes the worst possible partition, whereas the thin client partition becomes the most optimal. The reason for such drastic change in response time in that, after user sign-on, the application logic component in the thick client partition would need to issue several query-embedded RMI calls to the database to retrieve user preferences. Each RMI call generates messages over the slow GPRS link in a synchronous, blocking fashion. On the other hand, the thin client partition results only in one HTTP request and response message over the GPRS network.

This shows that *varying the usage patterns may drastically change the optimal application partition.* In other words, it is difficult to apply adaptation methods that give good response time under unpredictable usage patterns.

C. Impact of unpredictable network bandwidth on response time

Our measurements are based on the following setting: the no sign-on pattern is used, and the serve load is set to be low. When the slow GPRS network is used, our measurements show that thick client partition produces the optimal application response time because it avoids using the network. As the user switches to a faster WLAN network (e.g., a user detects the presence of WLAN and performs a vertical handoff from GPRS to WLAN), the thick client partition becomes the worst possible partition, whereas the thin client partition becomes the optimal partition. This is shown in Table Ⅴ. The reason for this drastic change is that high bandwidth and low latency in WLAN favor the thin client partition.

This shows that *varying the network bandwidth may drastically change the optimal application partition.* In other words, it is difficult to apply adaptive application partition that can give good response time under unpredictable wireless bandwidth.

D. Summary of Experimental Results

Based on the above three observations, we have found that changes in dynamic factors can lead to drastic change in the optimal application partition. When these changes are monitored by adaptation methods, application partitions are reconfigured accordingly. Given unpredictability and instability in these dynamic factors, these adaptation methods are likely to reconfigure the application partition frequently and continuously without ever converging on one application partition. Given that each reconfiguration has an associated cost, e.g., in moving components between mobile client device and server, this will use up lots of processing and network resources, resulting in perhaps even worse application response time and user experience.

On the other hand, our replicated client-server model deals gracefully with unpredictability and instability. It does not have the non-converging problem in adaptive methods. The reasons are as follows:

- Any abrupt change affecting the server-side execution (e.g., server loads or wireless bandwidth), only degrades the response time from the server-side execution, but not the client-side execution.
- There is no need to perform any reconfiguration of application partition in the replicated client-server model.
- There is no need to perform any prediction on the resource condition or usage patterns in the replicated client-server model.

VI. Design of Replicated Client-server model

The design of the replicated client-server model needs to address the following three issues brought by the replicated client-server model:

- For a given user request, initiate and manage the flow of replicated execution on client and server.
- Synchronize selective state between client and server at the end of each execution, so that they would produce the same results in subsequent requests.
- Control access and update to external I/O state, such as access to databases and file systems.
- Reduce the network and processing overhead in replicated execution.

The architecture of our replicated client-server model is shown in Figure 5. We describe these system components in the order of execution flow of running an application. At the start of application execution, we assume that the application specifies *replicable application components* (e.g., web/application tier components) that can be executed both on a mobile client and server, and *non-replicable application components* (e.g., client/database tier components) can run either on a mobile client or server. We also assume that those replicable application components are downloaded and started on both the mobile *client-side micro-container* and on the server-side full-feature *container*.

When a web browser generates a HTTP request message (e.g., as a result of user invoking a service on a web page), the *request dispatcher* on the mobile client forwards this message to the client-side micro-server to initiate the client-side execution; and at the same time, it forwards this message to the server-side container to initiate server-side execution.

When the client-side micro-container or the server-side container completes its execution, the result (e.g., a web page generated from a JSP, a RMI call return from EJB) is delivered to *computation coordinator* on the mobile client. Upon receiving a result, the computation coordinator can take two different actions based on whether the result comes from client-side execution or the server-side execution.

In the case that the result first comes from client-side execution (meaning that client-side execution is faster in delivering the result than the server-side execution), the computation coordinator will send the "client-is-faster" notification to the server, asking the server not to transmit (or to halt its current transmission of) its result to the mobile client. This saves network bandwidth. Note that the server container would need to complete its execution, even though its result is no longer needed. This ensures that the server state is consistent with the state of mobile client that completes the execution.

In the case that the result comes from server-side execution (meaning that server-side execution is faster in delivering the result than the client-side execution), the computation coordinator will immediately deliver the server-side result to the browser to give users the good response time. At the same time, the computation coordinator also halts the client-side execution, and sends a "server-is-faster" notification message to the server. Upon receiving this message, the server takes a snapshot of its current session state and transmits it to the mobile client.

The *client-side state synchronizer* receives the server session state snapshot, and then updates its micro-server to be in a consistent state with the server container. This synchronization step is used so that the micro-server does not need to complete its execution in the case that server-is-faster. That is, the mobile device can jump ahead of its execution using the server state. Note that state synchronization is processed after the application response has been shown to the user – this ensures that synchronization does not delay the application response time.

Most of the application need to access databases or file systems, which are considered as an external state to the applications. In the case of database access, these applications issue queries to the databases. Since both the client-side and server-side execution will generate redundant queries (e.g., placing an order), the system must ensure that only one query is sent to database. That is replicated execution does not produce replicated (incorrect) effects on external state. These queries are intercepted by *database access interceptor*, which keeps a log record of all database queries requested by either the mobile client or the server. After the interceptor get database queries from either the mobile client or server, it must first check the log to see if this query has been issued earlier. If yes, it will return the cached database result back to query issuer. Otherwise, the interceptor issues the query to the database. This means that some intermediate query results must be cached by the interceptor.

VII. Conclusion and Future work

In this paper, we have shown that unpredictability and instability in wireless network bandwidth, server loads, and usage patterns can lead to failure and poor application response time in traditional adaptation methods. The reason is that adaptation methods require somehow accuracy predictions on resource conditions and usage patterns, in order to make correct adjustment on application partition. To overcome unpredictability and instability, we propose a new, simple yet powerful, replicated client-server model. In the replicated client-server model, application execution is replicated on both the client and server, and the faster result

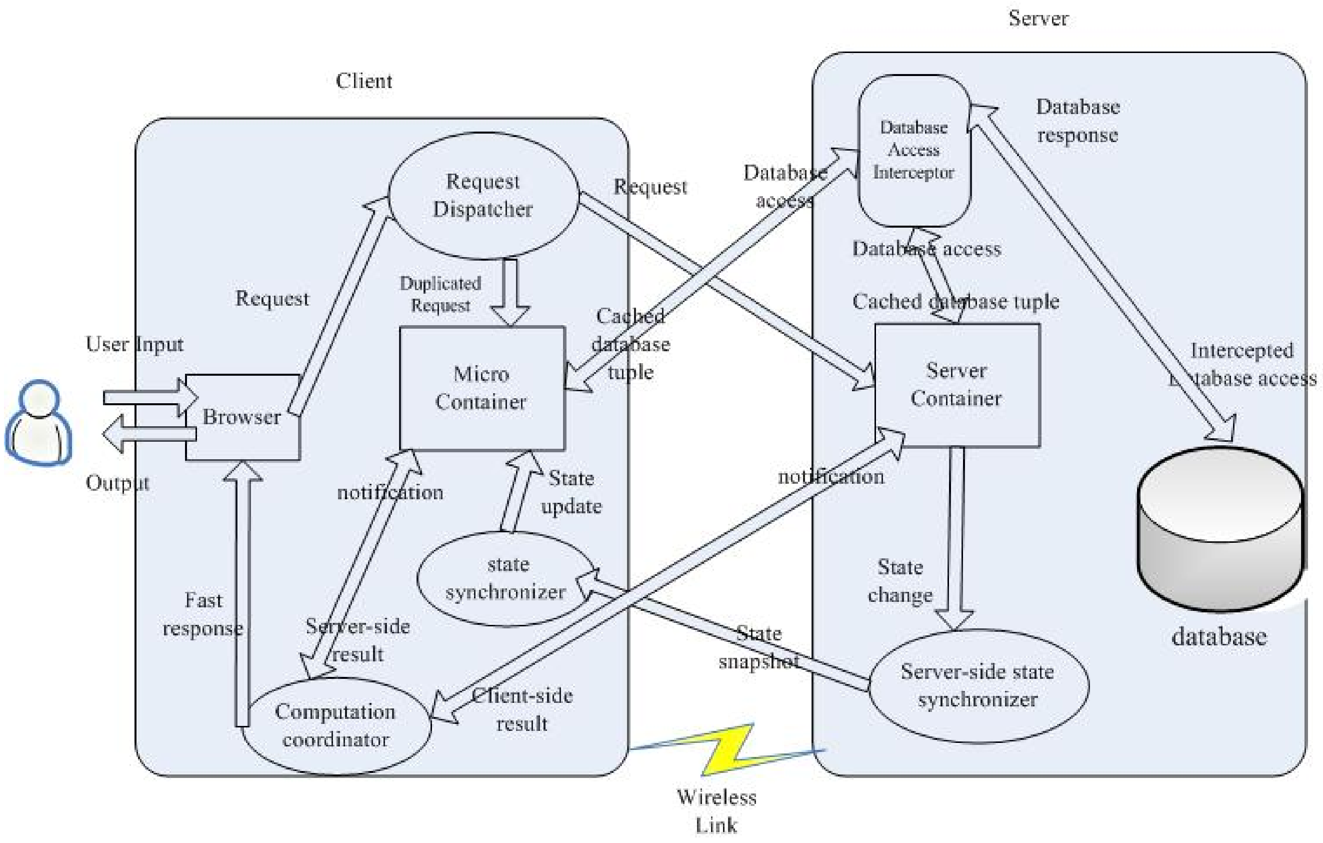

Fig. 5. System Architecture for Replicated Client-Server Model

is returned to the user. This model provides good application response time regardless of unpredictability and instability in resource conditions and usage patterns. We believe that given computing resources are becoming ever more abundant, improvement in application response time and user experience is a good tradeoff for overhead in replicated execution.

We have shown the design of the replicated client-server model. We are in the process of implementing the replicated client-server model. We would also like to make performance comparison with existing adaptation methods to show that the replicated client-server model can indeed work better than adaptation based methods for most of the Internet-based applications.

"Adaptation is the key to mobility" said Satya in his seminal paper on the fundamental challenges of mobile computing [10]. We believe that replication, if design correctly, could do better. We are looking forward to answering this question "Is replication a better key to mobility?"

VIII. References

[1] D. Llambiri, A. Totok, and V. Karamcheti, "Efficiently Distributing Component-Based Applications Across Wide-Area Environments", Proceedings of the International Conference on Distributed Computing Systems (ICDCS), Providence, RI, May 2003.
[2] G. C. Hunt, M. L. Scott, "The Coign Automatic Distributed Partitioning System, Proceedings of the Third Symposium on Operating Systems Design and Implementation", February 1999.
[3] R. K. Balan, M. Satyanarayanan, S. Park, T. Okoshi, "Tactics-Based Remote Execution for Mobile Computing", Proceedings of the 1st USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys), San Francisco, California, USA, May 2003.
[4] Sun Microsystems, Java Pet Store Sample Application. http://java.sun.com/blueprints/code/index.html#java_pet_store_demo .
[5] Sun Microsystem, Java BluePrints, http://java.sun.com/blueprints/.
[6] Sun Microsystems, Java 2 Platform, Enterprise Edition 1.4, http://java.sun.com/j2ee/1.4/download-dr.html.
[7] D. Zhou, S. Pande, K. Schwan, "Method Partitioning - Runtime Customization of Pervasive Programs without Design-time Application Knowledge", 23rd International Conference on Distributed Computing Systems, June 2003.
[8] D. Zhou and K. Schwan, "Eager Handlers – Communication Optimization in Java-based Distributed Applications with Reconfigurable Fine-grained Code Migration", Proceedings of the third International Workshop on Java for Parallel and Distributed Computing, April 2001.
[9] A. Gurtov, "Effect of Delays on TCP Performance", Proceedings of IFIP Personal Wireless Communications 2001.
[10] M. Satyanarayanan. "Fundamental Challenges in Mobile Computing", Fifteenth ACM Symposium on Principles of Distributed Computing, May 1996.

[11] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E. H. Siegel, D.C. Steere, “Coda: A Highly Available File System for a Distributed Workstation Environment”, IEEE Transactions on Computer, 39(4), April, 1990.
[12] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste, “Project Aura: Toward Distraction-Free Pervasive Computing”, IEEE Pervasive Computing, April-June, 2002.
[13] Joey J2EE server, http://www.joeysoft.com.
[14] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, J. Light, “The Personal Server: Changing the Way We Think about Ubiquitous Computing”, Ubicomp’02, June, 2002.
[15] B. Raman, S. Agarwal, Y. Chen, M. Caesar, W. Cui, P. Johansson, K. Lai, T. Lavian, S. Machiraju, Z. M. Mao, G. Porter, T. Roscoe, M. Seshadri, J. Shih, K. Sklower, L. Subramanian, T. Suzuki, S. Zhuang, A. D. Joseph, R. H. Katz, I. Stoica, "The SAHARA Model for Service Composition Across Multiple Providers", Pervasive Computing, August, 2002.
[16] V. N. Padmanabhan, L. Qiu, “The Content and Access Dynamics of a Busy Web Site: Finding and Implications”, ACM SIGCOMM 2000, August, 2000.
[17] B. Li, W. Jeon, W. Kalter, K. Nahrstedt, J. Seo. "Adaptive Middleware Architecture for a Distributed Omni-Directional Visual Tracking System," in Proceedings of SPIE Multimedia Computing and Networking 2000 (MMCN 2000), pp. 101-112, January 25-27, 2000.