

# Distributed Adaptation System for Quality Assurance of Web Service in Mobile Environment\*

SEUNGHWA LEE AND EUNSEOK LEE<sup>†</sup>  
*School of Information and Communication Engineering*  
*Sungkyunkwan University*  
*Suwon 440-746, South Korea*

Context-adaptive services for overcoming various limitations of mobile environment have become important. They have the potential to maintain appropriate service levels according to the changing environment. However, most existing studies concentrate on adaptation modules in a single system: client, proxy, or server. Hence, these systems suffer from a concentrated workload when the number of users increases. This causes response time delays to user requests. This paper proposes a novel adaptation system that performs adaptation over distributed systems. The proposed system monitors the condition of each system using adaptation modules that are deployed on the client, proxy, and server, and selects the appropriate system on which to perform the adaptation. Therefore, the proposed system responds rapidly and stabilizes the system operation, even when overload is increased, such as by the increase in the number of user requests. We implemented a prototype of the system in order to evaluate it, and tested the distributed processing for multimedia content adaptation, simulating the overload on the server. We compared the proposed system with the existing adaptation approaches, in terms of the response time and system stability, and confirmed the effectiveness of the system through these experimental results.

**Keywords:** web content adaptation, context awareness, distributed computing, context adaptive system, ubiquitous computing

## 1. INTRODUCTION

Mobile computing devices of various types are being rapidly deployed due to the growth of information technology. As a result, access to the Web is made possible anytime, anywhere. Many tasks that can be performed at fixed locations using wired network and desktop PC are being changed to free-roaming across time and space. Mobile devices have limitations such as small screen size, battery capacity, low CPU and memory capacity, in compromises to enhance 'mobility'. In contrast to fixed wired networks, a wireless network's characteristics fluctuate as the users move. Therefore, many researchers investigate adaptive systems in order to cope with the limitations of mobile computing environments and maintain suitable service levels [1-20].

The existing adaptation research can be classified into client side, server side, and intermediate side adaptation according to the location of the adaptation module. Most configurations have an issue that system workload burdens when the number of user

---

Received February 19, 2008; revised May 1, 2008; accepted July 10, 2008.

Communicated by Suh-Yin Lee.

\* This work was supported by the Ministry of Knowledge Economy ITRC Program in Korea, and a result of Basic Research Program of the Korea Science and Engineering Foundation. IITA-2008-(C1090-080-0046), grant No. R01-2006-000-10954-0.

<sup>†</sup> Corresponding author.

requests increase, because adaptation modules are concentrated on a single system (client, server, or proxy). In practice, adaptation tasks, such as converting the format of multimedia, consume substantial resource. This increases the response time for user requests.

This paper proposes a novel system architecture in which adaptation modules are distributed into client, proxy, and server. The system distributes the processing through interaction between the modules. Hence, even when the server's workload is overloaded, due to the increase in the number of users, making it hard for adaptation tasks to be focused in a single location, faster adaptation is possible with parallel processing, compared to existing systems, in which the adaptation module is concentrated at a single location. Consequently, it is possible to achieve stable performance for the overall system.

We constructed a prototype to evaluate the proposed system. In the prototype, multimedia content including large size images were converted. The simulation was carried out by gradually increasing the workload on the server, and we compared the response time and system stability with the existing system in which the adaptation modules are concentrated at one location. The result confirms the effectiveness of the proposed system.

The remainder of the paper is organized as follows: In section 2, we discuss related work. This are divided into three types according to the location of the adaptation module. In section 3, we propose a distributed adaptation system that is designed to overcome the limitations inherent in the related work. In section 4, the main methods that perform the distributed process according to each system state are introduced. In section 5, the results of the system implementation and evaluation are described. A conclusion is presented in section 6.

## 2. RELATED WORK

The advent of ubiquitous computing paradigm has stimulated the application of concepts such as context-awareness, calm computing, and autonomic computing [23-25]. The system components developed have common characteristics. They are aware of changes to the system states, and autonomously decide the appropriate plan to respond to the changes. They then automatically implement the plan. In particular, this context-adaptive paradigm is currently being applied to adaptive Web services, in order to offer information of a more suitable form to the user's situation. The adaptive Web returns the most appropriate content based on the user's preference and the characteristics of the user's devices. It performs the adaptation required to maintain stable service levels.

A sure method to customize content based on the user's situation is to prepare in advance the content with all the various types required by users who may access the server. However, this requires much effort by content authors and may require much storage on the server side. Also, predicting and preparing for the advent of new devices is very hard, particularly with rapid changes to the computing environment. Therefore, most adaptation systems dynamically change content. These existing studies can be classified into the three approaches that were adopted depending on the location of the adaptation module: client side, server side, and intermediate side [17].

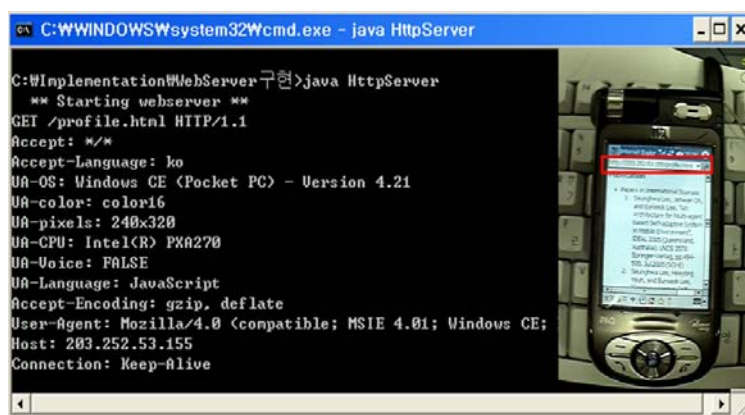
## 2.1 Client Side Adaptation

In client side adaptation, the modules are embedded in the user device, monitoring the changing context and adapting the downloaded content. Representative research is presented in [1-3]. It is not necessary to broadcast the user device's information. Thus, the method can prevent the loss of context information relating to privacy. It can quickly perceive dynamic changes to the user context. It is suited to adjusting the function of the application by reconfiguring the internal components. The method can reformat content, based on user preference or device characteristics, using a style sheet embedded in the device [1].

In this paper, the client is usually considered to be a hand-held device that has relatively poor computing power. Adaptation, such as the format conversion of content, is resource-intensive. The client device may not be able to convert the content transmitted to it if the message's size is not reduced initially. On the client side, only partial adaptation, related to content display or one that does not consume large computation resource, is suitable.

## 2.2 Server Side Adaptation

Server side adaptation is a method whereby adaptation modules are located on the web server. This method binds the appropriate content, prepared in advance, or dynamically creates the adapted content according to the user's situation [4, 5]. To do this, a mechanism is required to inform the user situation to the server. The HTTP protocol standard contains the client's basic information [22]. Hence, content adaptation is possible.



```
C:\WINDOWS\system32\cmd.exe - java HttpServer

C:\Implementation\WebServer 구현>java HttpServer
** Starting webservice **
GET /profile.html HTTP/1.1
Accept: */*
Accept-Language: ko
UA-OS: Windows CE (Pocket PC) - Version 4.21
UA-color: color16
UA-pixels: 240x320
UA-CPU: Intel(R) PXA270
UA-Voice: FALSE
UA-Language: JavaScript
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows CE;
Host: 203.252.53.155
Connection: Keep-Alive
```

Fig. 1. The sample of HTTP header information delivered to the server.

The HTTP request message includes the acceptable file type, language, display size, and CPU information (Fig. 1). However, only limited service personalization is possible using this restricted information, since the information only includes the user's static resource state. For example, it would be hard to adjust the quality of multimedia content

according to the current resource status of the client device; the server cannot recognize the user's current situation in detail. CC/PP (Composite Capabilities/Preferences Profiles) [21] is proposed by W3C to cope with this problem, thereby making it feasible to represent the user's varying situations.

The server side method can offer customized content after receiving context information from the client device. This method is able to create more accurate content because the adaptation rule is created by the content author. However, because the server must perform functions related to adaptation beyond the general response to user requests, extra workload occurs. This causes increases in response time.

### 2.3 Intermediate Side Adaptation

Intermediate side adaptation is a method in which the adaptation modules are located between the client and the server. The intermediate node usually denotes the proxy or gateway. Representative literature includes [6-11, 20]. This method decreases the workload concentrated on the server. It allows adaptation simply by adding the adaptation module to the proxy even if the server does not contain the context adaptation function. However, as in the case of the server side method, the proxy server workload also increases when user requests increase.

These existing systems suffer from the problem of the adaptation module being concentrated in one place: client, server, or proxy. This workload is concentrated at a single location. This results in the increase of the response time to a user's request as the number of users increase. The next section introduces our proposed system designed to cope with such problems.

## 3. PROPOSED SYSTEM

The proposed system recognizes the user's contextual information and performs suitable content adaptation based on it, in order to overcome the various limitations inherent in the mobile environment. The proposed system is designed to efficiently improve the various problems experienced when the adaptation modules are concentrated at one location (*i.e.*, client device, proxy server, web server). The system aims to provide faster response while maintaining the stability of each of the systems.

### 3.1 Overall Structure of the System

As mentioned, existing adaptation systems are located at one location, hence workload will be concentrated. The proposed system performs the distributed processing for adaptation tasks through interaction between systems to perform optimal adaptation.

To perform cooperative work, the proposed system is designed in three parts: *client adaptation module (CAM)*, *proxy adaptation module (PAM)*, and *server adaptation module (SAM)*. The overall structure is presented in Fig. 2.

The component structure of each system is similar. Each module monitors the static/dynamic context information, and then forwards the information to the server when the user requests the web information. *CAM* and *PAM* can attach their executable adaptation

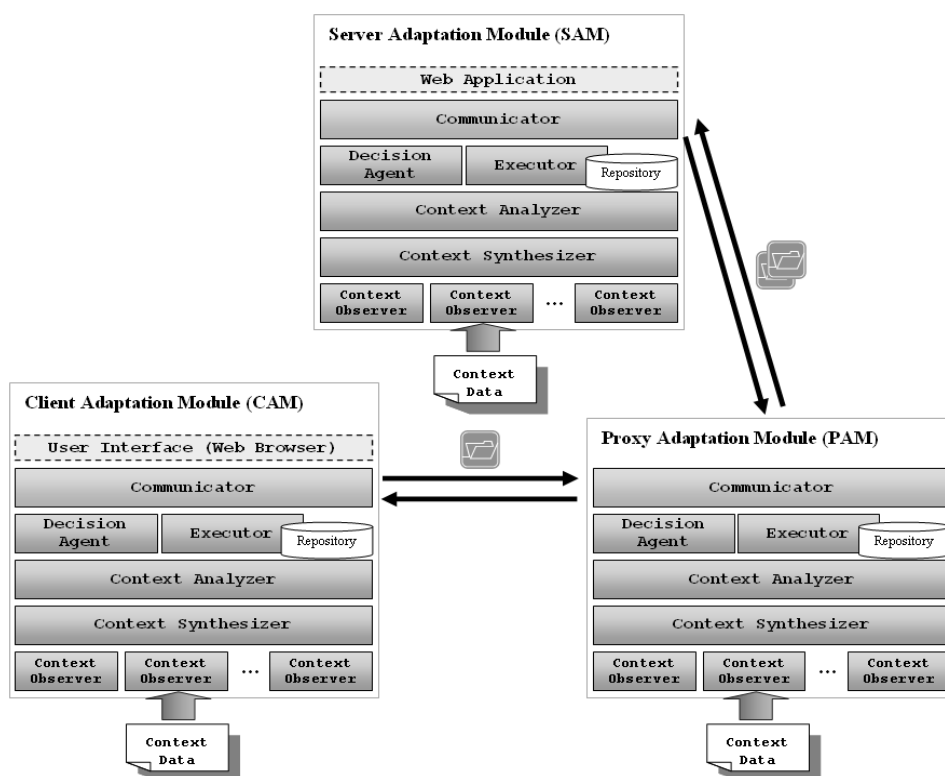


Fig. 2. Overall architecture of the proposed system.

work with the forwarded information, and the final decision for distributed processing the adaptation works is performed by *SAM*. The function of each component is as follows:

- *Context Observer (CO)*: The *CO* is embedded in the adaptation module of the client, proxy, and server. It inspects the dynamically changing context information, such as system resource state and workload. This information is used to decide the adaptation service type and the modules that will perform the work.
- *Context Synthesizer (CS)*: The *CS* gathers the context information monitored by each *CO*.
- *Context Analyzer (CA)*: The *CA* analyzes the current situation based on the context information collected by the *CO*. It converts the current status information to a regular format.
- *Decision Agent (DA)*: The *DA* of the client and of the proxy determines the available adaptation services based on the current situation and pre-defined rules. The server *DA* decides the adaptation services and their intensity. It selects the most appropriate location for performing the adaptation task, based on the status information of each of the systems and the options. Option refers to the list of available adaptation services for each system.
- *Repository*: The various components for content adaptation (*i.e.*, image size converter, image color depth converter, *etc.*) are stored.

- *Executor*: The *Executor* performs the adaptation by calling the service components stored in the repository.
- *Communicator*: The *Communicator* manages the interactions between the modules. It identifies the message type, such as request message, context information, and message to be entrusted.
- *User Interface*: The *UI* is the web browser interacting with the user.
- *Web Application*: The *Web application* is the element that offers the web service (e.g., servlet, cgi, asp)

### 3.2 The Proposed System Behavior

The overall behavior of the proposed system is as follows: First, the *CO* located at each system collects context information. The type of information collected is shown in Table 1. The static resource capacity of the each system can be represented using HTTP 1.1, as mentioned in section 2. However, the proposed system uses the *CO* in order to collect the dynamic context, and to accumulate the new context information according to the addition of the adaptation services.

**Table 1. The type of the context information collected in each system.**

Location	Context Type	Applicable Parts
Client	Display size, Acceptable color depth	Used to decide the type and intensity of adaptation service
	Acceptable media format	
	CPU-RAM capacity, Current usage	Used to entrust the adaptation tasks such as web page reformatting
	Performable adaptation service	
Proxy	CPU-RAM capacity, Current usage, Size of task queue	Used to entrust the adaptation tasks such as media format conversion, consuming much resource
	Performable adaptation service	
Server	CPU-RAM capacity, Current usage, Size of task queue	Used to determine the adaptation tasks that can be performed by the server

The *CS* integrates the context information monitored by several *COs*. Then, a part of the collected information is computed and analyzed by the *AA*. The computation phase of the dynamically changing resource status will be discussed in section 3.3. The *DA* determines the adaptation task list that can be presently performed.

After this, when a user requests any web content, the *Client Adaptation Module (CAM)* sends the context information for the client device and the available adaptation task list to the *Proxy Adaptation Module (PAM)*. Then, the proxy's *Communicator* attaches its context information and the available adaptation task list to the client's message. The *PAM* sends the combined information to the *Server Adaptation Module (SAM)*.

The *DA* of the server determines the adaptation service type and its degree for the requested content based on the client's context information. This decision is based on rules predefined by experts. This paper does not consider rule building. The focus is on the distributed processing of the workload. The proposed system applies an approach similar to [29] in order to build the adaptation rules.

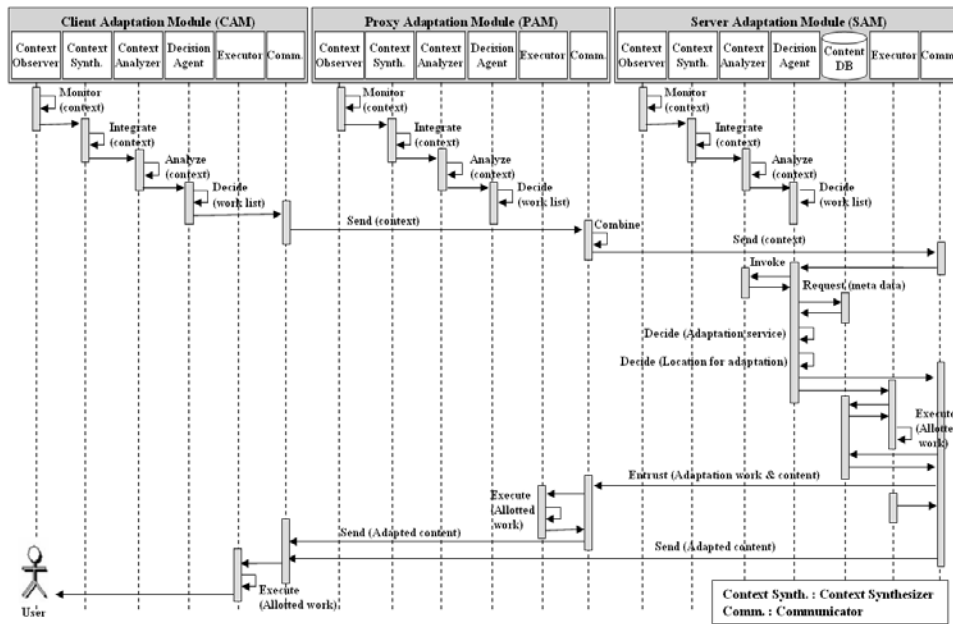


Fig. 3. A sequence diagram that represents the overall behavior of the proposed system.

The server’s *DA* decides the location that will perform the adaptation tasks. The elements of the original content are allocated to the selected systems. At this stage, the statuses of the client and the proxy have priority over the server’s status. That is, even if the server needs to delegate the adaptation tasks to the client or proxy, if the client or proxy does not send the message with the intention to receive the delegated adaptation task, the server cannot delegate the tasks. The overall behavior of the proposed system is represented in Fig. 3.

#### 4. DISTRIBUTED PROCESSING OF ADAPTATION TASKS

The decision to allocate the adaptation tasks is based on the state of each system’s resources and workload. The proposed system reflects formulas used in the grid computing field to analyze the resource status. Grid computing is a field to obtain high performance computing capacity by connecting several devices with low computation power [26]. Hence, the grid algorithm is similar to our proposed system. The objective is to distribute tasks to several computing devices and to integrate the results. In this paper, the client, proxy, and web content server act together as a system for content adaptation.

##### 4.1 Location Selection of the Adaptation Tasks

Each system monitors its own state. It passes the status information to the server that plays the role of the last decision maker in order to efficiently allocate the tasks. The type of context information collected by each system is shown in Table 1.

The server computes the resource state of each system. Resource information, such as the CPU and RAM, is computed using Eqs. (1) and (2).

$$Current_{CPU} = \alpha(1 - CPU_{load}) \frac{CPU_{speed}}{CPU_{min}} \quad (1)$$

$$Current_{RAM} = (1 - \alpha)(1 - RAM_{usage}) \frac{RAM_{size}}{RAM_{min}} \quad (2)$$

where,  $\alpha = \frac{W_{CPU}}{W_{CPU} + W_{RAM}}$ .

Here,  $W$  is the weighted value that is allocated to the CPU and RAM. It is determined by the adaptation service developer. For example, consider the case of the total consumed resource of the *image\_converter*, a selected adaptation task, is given the value of '10'. The CPU and RAM capacity required for this task can be defined as 4 and 6 respectively. The variables are determined by the developer of the *image\_converter*. In this example, the sum of weights is fixed as '10' in order to simplify the explanation, but these values are relative. In this example, the values of ' $\alpha$ ' are 0.4 and 0.6, respectively.

$CPU_{min}$  and  $RAM_{min}$  are the minimum capacity required to execute the task.  $CPU_{speed}$  and  $RAM_{size}$  represent the system's static resource capacity.  $CPU_{load}$  and  $RAM_{usage}$  are the currently used amount of the resource.

After computing the CPU and RAM status respectively, the overall system resource is computed using Eq. (3).

$$Current_{resource} = Current_{CPU} + Current_{RAM} \quad (3)$$

In this paper, we only consider some factors (*e.g.*, CPU, RAM) among the various resource states of each system. In the next investigation, if detailed modeling of the adaptation works and more context factors are considered, more efficient distribution of the adaptation tasks will be feasible.

In addition, we focused on distributing the concentrated adaptation modules of the existing client side, proxy side, and server side adaptation system. However, in practice, web applications are serviced by a server group that consists of a number of web servers. Hence, we applied an additional factor in order to consider a situation where the web server group and proxy server group participate as adaptation workers.

The server can ascertain the current resource status of each system. The system applies the size of the task queue to improve reliability of the result that is computed by Eq. (3). In the case of the proxy and web servers, the sizes of the task queues are indicators that more precisely represent the system workload. Queue size is obtained by using Eq. (4) defined by Little's law [27].

$$Q_{length} = \lambda T \quad (4)$$

$Q_{length}$  is the average number of requested tasks, and  $\lambda$  is the arrival rate of the tasks, the  $T$  is the average time that a task waits in the queue.

The resource usage and the size of the task queue are required to establish the current workload of each of the system. Finally, the proposed system determines the most appropriate system based on these results. The formula used for the final decision is Eq. (5).

$$Situation_{system} = Current_{resource} \times \frac{1}{Q_{length}} \quad (5)$$

#### 4.2 A Case Example

We introduce a case study to explain in detail the computation phase described in section 4.1. We assume the systems have resources shown in Table. 2. The applied adaptation service converts image sizes. The service's weighted value for the CPU and RAM is regarded to equal 5, and the minimum resource requirements of the service is higher than 64MB RAM with a Intel Celeron 333MHz.

**Table 2. The context information collected for each system's resources.**

Location	CPU speed (GHz)	CPU load (%)	RAM size (MB)	RAM usage (%)
Client	0.4	40	256	50
Proxy	3.0	50	1024	30
Server	3.4	70	2048	70

The computation results for each system are as follows:

$$Current_{client} = 0.5(1-0.4) \frac{0.4}{0.3} + (1-0.5)(1-0.5) \frac{256}{64} = 0.4 + 1.0 = 1.4,$$

$$Current_{proxy} = 0.5(1-0.5) \frac{3.0}{0.3} + (1-0.5)(1-0.3) \frac{1024}{64} = 2.5 + 5.6 = 8.1,$$

$$Current_{server} = 0.5(1-0.7) \frac{3.4}{0.3} + (1-0.5)(1-0.7) \frac{2048}{64} = 1.7 + 4.8 = 6.5.$$

In the example, the proxy server is selected as the appropriate system to perform the adaptation task, because the overall resource situation of the proxy is the highest, although the static resource capacity of the web server is higher than that of the proxy server.

The size of the task queue is not applied in the example, but the value may be used to improve the reliability of the result when the proposed system is extended.

## 5. SYSTEM EVALUATION

The performance of the proposed system is measured in terms of response time and each system's overall stability based on our current implementation. The server module was implemented on a HP 4300 workstation equipped with 3.4 GHz CPU, 2048 MB

RAM, and was connected to a wired network. The proxy server is an intermediate server located at or close to the network of the wireless access point to which the mobile device is connected. The proxy server module was implemented on a common desktop PC with 3.0 GHz CPU, 1024 MB RAM, and was connected to a wired network. The client adaptation module was implemented on a HP 5500 PDA equipped with 240 \* 320 resolution, 400 MHz CPU, 256 MB RAM, and was connected to a wireless network.

The operating system used in the server was Microsoft's Windows 2000. The PDA is based on Windows CE. The COs embedded in each system have been encoded using the C++ language that supports the various system APIs in order to facilitate implementation. The remaining modules have been implemented using Java, and interact based on the JADE-LEAP. Hence, the COs create the context information as RDF, the modules implemented with Java read the context information. The image conversion service was mainly used for the adaptation service. It was implemented using JAI (Java Advanced Imaging) [28], and contains the image's size converter, image's color depth converter, and image's format converter.

### 5.1 Evaluation for Response Time

The first test was to evaluate the improved response time. For this, we compared the proposed system with the client, proxy, server side approaches, while overloading the server. The test used web content that included two high quality images of 1680 \* 1050 pixels, 155 Kbytes and 967 Kbytes, respectively.

We first requested the content and checked the point of time at which the display is completed, in order to get the overall response time. We also checked processing time separately at each location to find the factors that affect the response time. The test environment is shown in Fig. 4.

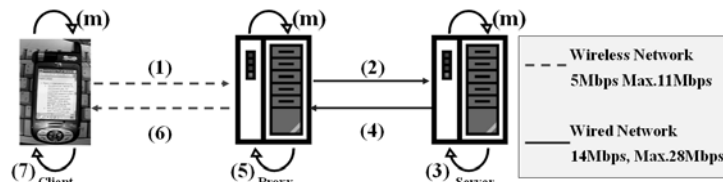


Fig. 4. The communication step for the adapted contents.

The images of the content were downgraded to  $240 \times 150$  pixel and 50% color quality by a predefined adaptation rule based on the client device's context information such as screen size, CPU type. The converted images were reduced to 16 Kbytes, and 9 Kbytes respectively.

In Fig. 4, the phases indicated as (m) are processes that monitor the device's context information, either offline, or in a short time frame not exceeding 20ms. Phases (1) and (2) are request steps during which very little time is consumed. Hence, we disregarded the phases, just focusing on each system's processing time for adaptation and the transmission time of content between each system, in order to check the main causes affecting response time.

The processing time of each system is checked by including the time checking code in each adaptation module. The web browser computes the total response time while the user requests the content and then the content is rendered to the display, and phases (4) and (6) are deduced by using Eq. (6).

$$TT = RT_{total} - \text{MAX}(PT_{server}, PT_{proxy}) - LT_{client} \quad (6)$$

where,  $TT$ : transmission time at phases (4), (6);

$RT_{total}$ : total response time;

$PT_{server}$ : processing time at server;

$PT_{proxy}$ : processing time at proxy;

$LT_{client}$ : loading time at client display;

$\text{MAX}()$ : a function to select the system with highest processing time.

The adaptation tasks of the proposed system are processed in parallel by each system. Hence, the function  $\text{MAX}()$  is used to select the system that consumes the most time of the server and proxy. The test result is shown in Fig. 5.

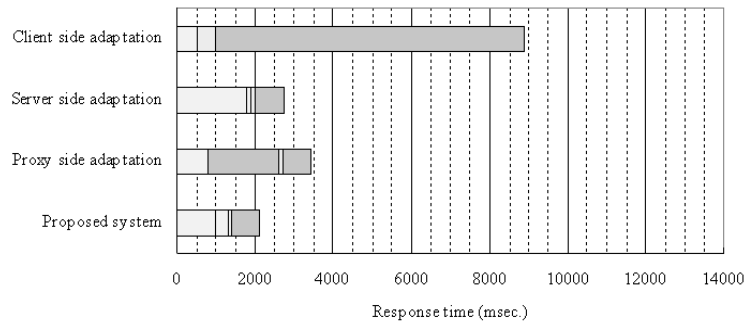


Fig. 5. A comparison of the response time under general conditions.

In the experimental result, the client adaptation approach took on average 8 seconds to convert the image contents and display the result. The client adaptation method downloads original content that does not degrade, and renders the content to fit the display size. Hence, a lot of transmission time is consumed, and total response time is relatively larger than other approaches. The method commonly adopts a web browser that is embedded in the PDA.

In the proxy adaptation approach, the proxy server intercepts the user's context information, and performs the adaptation tasks for original content received from the server. We implemented the method by reference to the basic behavior of IBM's WebSphere Transcoding Publisher [6]. The total response time of the method is slightly higher than for the server side adaptation method, because of transmission time between proxy and server.

In the case of the server adaptation approach, the image processing time at the server took around 0.5 seconds on average, and the image loading time at the client took around 1.5 seconds. Total response time took around 2.7 seconds on average.

Conversely, total response time of the proposed system took around 2.1 seconds on average. We manipulated the adaptation rule to convert the images in parallel on the server and the proxy. Consequently, the image processing time of the server and proxy was reduced to around 0.3 seconds.

Through the experiment, we confirm that the proposed system reduces the response time by up to 20%, in general server situations. The transmission time in phase (4) and (6) dealt with same, although there were some differences because of the wireless and wired network.

After the experiment, we also execute the process generator to observe the change in the outcome when the workload is concentrated at the server. The process generator was developed to simulate increasing of the number of users by using the log data that records the number of users accessing the server and the server's resource utilization.

The response time increased in all approaches, but to a lesser degree in the proposed system. As more load was added on the server, the gap between the response times increased (Fig. 6). We confirm the efficiency of the proposed system through these results.

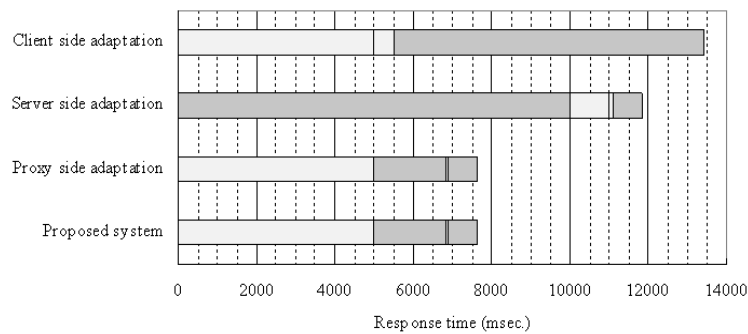


Fig. 6. A comparison of the change in response times simulating overload at the server.

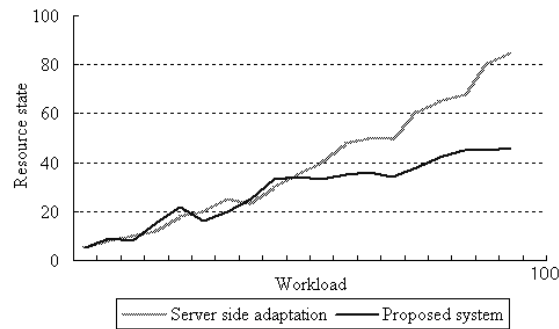


Fig. 7. Evaluation of system stability.

## 5.2 Evaluation for System Stability

The second test was to demonstrate the proposed system's characteristics in maintaining the server's stable condition. The process generator was used to overload the

system, increasing the response times. We checked the change in the server's resource state for each approach (Fig. 7).

The server adaptation approach's resource utilization continually increased as the simulated overload increased. In the case of the proposed system, resource utilization only increased slightly and a stable condition was maintained even when the overload was continually simulated.

Through these experiments, we confirm that the proposed system can maintain a stable condition for resources, as well as improving the response time. We expect to reduce the frequency of faults, such as system shutdown.

## 6. CONCLUSION

This paper presents a novel web adaptation system to cope with the weakness of the existing adaptation systems. The proposed system performs distributed processing of the adaptation tasks according to the condition of each system. We tested the improvement in response time and system stability. This demonstrated that the proposed system is able to deploy web adaptation more efficiently. We expect that these characteristics of the system may ameliorate the various limitations of mobile environments.

In our subsequent work, we will find other practical uses for this system in the various contexts that may occur in the ubiquitous computing environment. A method to collect, analyze, and manage this information will be investigated. We will study a way to predict condition changes through pattern analysis. In addition, the issue of security for the messages communicated among the different modules will be examined.

## REFERENCES

1. M. Butler, F. Giannetti, R. Gimson, and T. Wiley, "Device independence and the web," *IEEE Internet Computing*, Vol. 6, 2002, pp. 81-86.
2. A. Kinno, Y. Yonemoto, M. Morioka, and M. Etoh, "Environment adaptive XML transformation and its application to content delivery," in *Proceedings of Symposium on Applications and the Internet*, 2003, pp. 27-31
3. B. Noble, "System support for mobile, adaptive applications," *IEEE Personal Communication*, Vol. 7, 2000, pp. 44-49.
4. A. Pashtan, S. Kollipara, and M. Pearce, "Adapting content for wireless web services," *IEEE Internet Computing*, Vol. 7, 2003, pp. 79-85.
5. B. Knutsson, H. Lu, J. Mogul, and B. Hopkins, "Architecture and performance of server-directed transcoding," *ACM Transactions on Internet Technology*, Vol. 2, 2003, pp. 392-424.
6. IBM WebSphere Transcoding Publisher, [http://www-306.ibm.com/software/pervasive/transcoding\\_publisher/](http://www-306.ibm.com/software/pervasive/transcoding_publisher/).
7. R. Mohan, J. R. Smith, and C. S. Li, "Adapting multimedia Internet content for universal access," *IEEE Transactions on Multimedia*, Vol. 1, 1999, pp. 104-114.
8. W. Y. Lum and F. C. M. Lau, "A context-aware decision engine for content adaptation," *IEEE Pervasive Computing*, Vol. 1, 2002, pp. 41-49.

9. C. Chang and M. Chen, "On exploring aggregate effect for efficient cache replacement in transcoding proxies," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, 2003, pp. 611-624.
10. W. Y. Lum and F. C. M. Lau, "User-centric content negotiation for effective adaptation service in mobile computing," *IEEE Transactions on Software Engineering*, Vol. 29, 2003, pp. 1100-1111.
11. P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli, "Context-aware middleware for resource management in the wireless Internet," *IEEE Transactions on Software Engineering*, Vol. 29, 2003, pp. 1086-1099.
12. Z. Hua, X. Xie, H. Liu, H. Lu, and W. Ma, "Design and performance studies of an adaptive scheme for serving dynamic web content in a mobile computing environment," *IEEE Transactions on Mobile Computing*, Vol. 5, 2006, pp. 1650-1662.
13. Y. Chen, X. Xie, W. Ma, and H. Zhang, "Adapting web pages for small-screen devices," *IEEE Internet Computing*, 2005, pp. 50-56.
14. D. Billsus, C. A. Brunk, C. Evans, B. Gladish, and M. Pazzani, "Adaptive interfaces for ubiquitous web access," *Communication of the ACM*, Vol. 45, 2002, pp. 34-38.
15. Y. Hwang, J. Kim, and E. Seo, "Structure-aware web transcoding for mobile devices," *IEEE Internet Computing*, Vol. 7, 2003, pp. 14-21.
16. L. Capra, W. Emmerich, and C. Mascolo, "CARISMA: Context-aware reflective middleware system for mobile applications," *IEEE Transactions on Software Engineering*, Vol. 29, 2003, pp. 929-944.
17. M. Margaritidis and G. C. Polyzos, "Adaptation techniques for ubiquitous Internet multimedia," *Wireless Communications and Mobile Computing*, Vol. 1, 2001, pp. 141-163.
18. T. Laakko and T. Hiltunen, "Adapting web content to mobile user agents," *IEEE Internet Computing*, 2005, pp. 46-53.
19. R. Gil, R. Garcia, and J. Delgado, "Delivery context negotiated by mobile agents using CC/PP," in *Proceedings of the 5th International Workshop on Mobile Agents for Telecommunication Applications*, LNCS 2881, 2003, pp. 99-110.
20. M. T. Chebbine, A. Obaid, S. Chebbine, and R. Johnston, "Internet content adaptation system for mobile and heterogeneous environments," in *Proceedings of the 2nd IFIP International Conference on Wireless and Optical Communications Networks*, 2005, pp. 346-350.
21. W3C – Composite Capability/Preference Profiles (CC/PP), <http://www.w3.org/Mobile/>.
22. W3C – RFC 2616 Hypertext Transfer Protocol – (HTTP 1.1), <http://www.w3.org/Protocols/rfc2616/>.
23. M. Weiser, "The computer of 21st century," *Scientific American*, Vol. 265, 1991, pp. 94-104.
24. F. Adelstein, S. K. Gupta, G. Richard, and L. Schwiebert, *Fundamentals of Mobile and Pervasive Computing*, McGrawHill, United States, 2004.
25. P. Horn, "Autonomic computing: IBM's perspective on the state of information technology," IBM White paper, 2001.
26. M. Li and M. A. Baker, *The Grid: Core Technologies*, Chapter 6, 2005, pp. 243-300
27. J. D. Little, "A proof of the queueing formula:  $L = \lambda W$ ," *Operations Research*, Vol. 9, 1961, pp. 383-387.

28. Sun Microsystems – Java Advanced Imaging (JAI) API, <http://java.sun.com/products/java-media/jai/>.
29. S. Lee, J. Lee, and E. Lee, “An inference engine for personalized content adaptation in heterogeneous mobile environment,” in *Proceedings of the International Symposium on Ubiquitous Computing Systems*, LNCS 4239, 2006, pp. 158-170.



**Seunghwa Lee (李昇和)** received his Ph.D. and M.S. degrees in Computer Engineering from Sungkyunkwan University, Korea, in 2008 and 2005, respectively, and his B.S. degree in Information and Communication Engineering from Paichai University, Korea, in 2003. He works at Education Center for Mobile Communications of the Sungkyunkwan University where he is currently a research professor. His research interests include web-based information retrieval, self-adaptation system, and intelligent agent technologies.



**Eunseok Lee (李殷碩)** received his Ph.D. and M.S. degrees in Information Engineering from Tohoku University, Japan, in 1992 and 1988, and B.S. degree in Electronic Engineering from Sungkyunkwan University, Korea, in 1985, respectively. He is a professor of the department of Computer Engineering of the Sungkyunkwan University. From 1994 to 1995, he was an assistant professor of the department of Information Engineering of Tohoku University, Japan. He was a research scientist in Information and Electronics laboratory of Mitsubishi Electric Corporation, Japan, from 1992 to 1994. His research topics include methodologies in software engineering, autonomic computing, and web-based agent technologies.