An Analysis on the Web Technologies for Dynamically Generating Web-Based User Interfaces in Ubiquitous Spaces

Ilsun You[1] and Chel Park[2]

[1] Department of Information Science, Korean Bible University,
205 Sanggye-7 Dong, Nowon-ku, Seoul, 139-791, South Korea
isyou@bible.ac.kr
[2] Fasoo.com, Product Planning Team,
KByeoksam bldg. 5th fl., Yeoksamdong, Kangnamgu, Seoul,
135-911, South Korea
pcman@fasoo.com

Abstract. In this paper, we study the web technologies that allow ubiquitous spaces to create dynamic web pages in accordance with user profiles. Especially, we explore the server-side scripting approach, the most popular technology for dynamic web pages. This approach mainly adopts the execute-while-parsing model, which suffers from the interpretation overhead. Recently, the compile-then-execute model was proposed to address the overhead. This paper compares and analyzes the two models, while performing benchmark test in Microsoft ASP and ASP.NET environment. The benchmark results show that, due to the high initialization overhead, the compile-then-execute model cannot substantially improve the execute-while-parsing model. Also, the best performance can be achieved through optimization rather than compiled execution. Based on the results of the benchmark test, we develop a speedup model, which estimates the maximum performance improvement achievable by the compile-then-execute model.

1 Introduction

Since the introduction by Mark Weiser, ubiquitous computing technology has received significant attention in the last few decades [1-5]. In ubiquitous computing environment, users carry mobile access devices such as PDAs, mobile phones and wristwatches, through which they can seamlessly access resources and services within ubiquitous spaces. For example, as Alice enters an intelligent hotel room, which is a ubiquitous space, her mobile access device is automatically detected and joins the space. Then, her device alerts her that she is within the ubiquitous space, thus allowing her to use resources and services provided by the space.

Nowadays, the dynamic web page service is becoming important in ubiquitous computing environment because of the followings:

- It is preferred to use the web and its underlying HTTP protocol for interaction between mobile access devices and ubiquitous spaces because they are standard and mature technology easy to implement. Furthermore, since most mobile access devices include a web browser, it is desirable to use the web browser as an interface to ubiquitous spaces.

- Typically, ubiquitous spaces tend to offer various resources and services, all of which should not be given to users. Therefore, interfaces to the spaces need to be personalized according to user profiles. Such personalization requires a service that dynamically generates web-based user interfaces for controlling resources and services according to user profiles.

In this paper, we study the web technologies that allow ubiquitous spaces to create dynamic web pages in accordance with user profiles. Especially, we explore the server-side scripting approach, the most popular technology, which mainly adopts the execute-while-parsing model. However, the execute-while-parsing model has a critical burden that server-side scripts must be interpreted every time they are requested. Recently, the compile-then-execute model was proposed to address this burden. This model allows a script page to be executed without any compilation, after the page is first compiled. Thus, it is expected that the compile-then-execute model improves the execute-while-parsing model. This paper compares and analyzes the two models, which are expected to be popular in ubiquitous computing environment. For the purpose, we design a benchmark program, implement three different versions of the program and perform benchmark test in Microsoft Active Server Pages (ASP) and ASP.NET environment.

The rest of the paper is organized as follows. Section 2 reviews the web technologies that enable creating dynamic web pages, and section 3 gives a brief overview of ASP and ASP.NET. Section 4 describes test environment and our benchmark program. In section 5, the results of the benchmark test are analyzed, and then a speedup model is provided. Finally, section 6 draws some conclusions.

2 Web Technologies for Generating Dynamic Web Pages

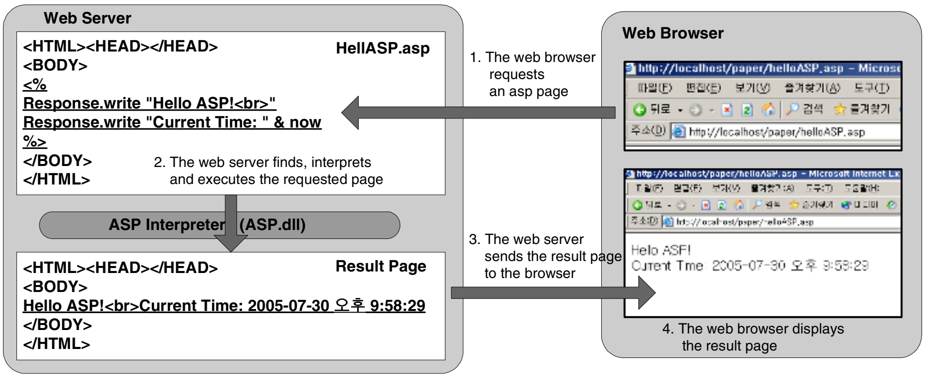
Since the introduction of the web, there has been a tremendous demand for mechanisms that enables creating dynamic web pages in accordance with user requests. The Common Gateway Interface (CGI), a standard for running external programs (CGI programs) on a web server, was the first widely means for generating dynamic web pages. Though CGI has benefits such as ease of understanding, language independence, platform independence and so forth, it has the two significant drawbacks [6-10]: low performance and high programming overhead.

The limitations of CGI have led to various approaches such as web server extensions, Fast-CGI, java servlets and server-side scripting [7,8]. Unlike other approaches whose goal is to address the low performance, server-side scripting focuses on minimizing the programming overhead. Since script languages are easy and convenient to build, debug and modify, this approach achieves the purpose, while becoming a popular technology. However, it has a critical burden that server-side scripts must be interpreted every time they are requested. To address this burden, the web technologies such as Practical Extraction and Report Language (PERL), Microsoft Active Server Pages (ASP) and PHP: Hypertext Preprocessor (PHP) 3.0 implement an interpreter based on web server extensions. In spite of reducing the burden, the technologies still require the web server to interpret the scripts. Unlike the interpreter-based technologies using an execute-while-parsing model, advanced technologies such as PHP 4.x/5.x, Sun Java Server Pages (JSP) and ASP.NET use a compile-then-execute model for removement of the interpretation overhead. This model allows a script page to be executed without any compilation, after the page is first compiled.

3 Overview of ASP and ASP.NET

3.1 Active Server Page

Active Server Pages (ASP) is a server-side scripting technology that supports the creation of dynamic web pages [11]. This technology allows a web developer to combine Hypertext Markup Language (HTML), scripts, Extensible Markup Language (XML), and reusable Component Object Model (COM) including ActiveX controls to build powerful interactive web sites. An ASP page is an HTML page that contains server-side scripts, and is executed as shown in Fig. 1. ASP.dll interprets a requested ASP page and executes any script commands in it, while running as a script language interpreter in the web server process. Also, it provides access to COM objects including ADO and ASP components.


Web Server
HellASP.asp
2. The web server finds, interprets and executes the requested page
ASP Interpreter (ASP.dll)
Result Page
1. The web browser requests an asp page
3. The web server sends the result page to the browser
Web Browser
http://localhost/paper/helloASP.asp
4. The web browser displays the result page

```
<HTML><HEAD></HEAD>
<BODY>
<%
Response.write "Hello ASP!<br>"
Response.write "Current Time: " & now
%>
</BODY>
</HTML>
```

```
<HTML><HEAD></HEAD>
<BODY>
Hello ASP!<br>Current Time: 2005-07-30 오후 9:58:29
</BODY>
</HTML>
```

```
Hello ASP!
Current Time 2005-07-30 오후 9:58:29
```

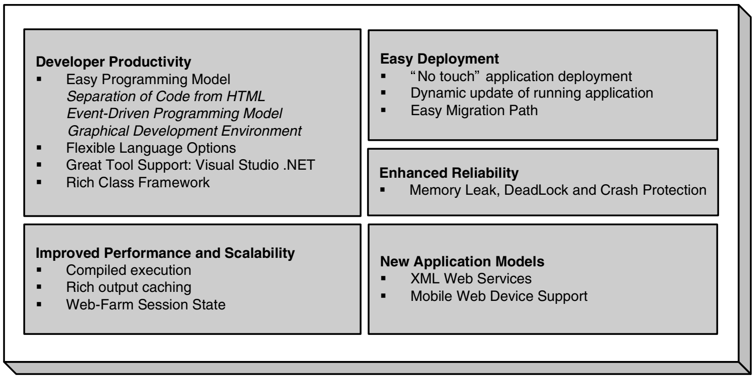
Fig. 1. ASP page execution model

ASP offers the following competitive features:

- Server-side scripting
- Easy and Flexible Database Access
- Extensibility through COM Objects

3.2 ASP.NET

ASP.NET, which is more than the next version of ASP, is a set of technologies in the Microsoft .NET framework for building web applications and XML web services [12-14]. It provides a unified web development model that enables developers to build enterprise-scale web applications. Also, it uses a compiled, event-driven programming model that improves performance and enables the separation of application logic and user interface. Because of being based on the fundamental architecture of .NET framework, it allows web applications to be created in any .NET compatible language, such as Visual Basic .NET, C#, and JScript .NET. Furthermore, developers can easily leverage the benefits of .NET framework, which include the managed common language runtime environment, type safety, inheritance, and so on. Fig. 2 describes five key advantages of ASP.NET [13].



Developer Productivity
- Easy Programming Model
  Separation of Code from HTML
  Event-Driven Programming Model
  Graphical Development Environment
- Flexible Language Options
- Great Tool Support: Visual Studio .NET
- Rich Class Framework

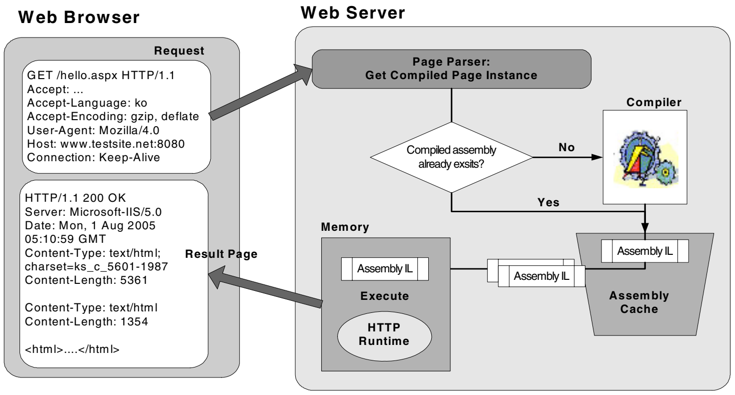
Easy Deployment
- "No touch" application deployment
- Dynamic update of running application
- Easy Migration Path

Enhanced Reliability
- Memory Leak, DeadLock and Crash Protection

Improved Performance and Scalability
- Compiled execution
- Rich output caching
- Web-Farm Session State

New Application Models
- XML Web Services
- Mobile Web Device Support

Fig. 2. Key advantages of ASP.NET



```
GET /hello.aspx HTTP/1.1
Accept: ...
Accept-Language: ko
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: www.testsite.net:8080
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 1 Aug 2005
05:10:59 GMT
Content-Type: text/html;
charset=ks_c_5601-1987
Content-Length: 5361

Content-Type: text/html
Content-Length: 1354

<html>....</html>
```

Fig. 3. ASP.NET page execution model

In contrast to classic ASP pages, ASP.NET pages are compiled and then executed as illustrated in Fig. 3. When an ASP page is first requested, it is compiled into a .NET assembly. Without the requirement of interpretation, subsequent requests are directly processed by the assembly, which is cached in assembly cache until its source page is changed. Such a compile-then-execute model makes ASP.NET overcome the performance penalties caused by interpreting the scripts. In addition to the improved performance, ASP.NET requires no explicit compile step, thus making web application development easier, faster and much more cost-effective.

4 Experiments

A main goal of this paper is to analyze the execute-while-parsing and compile-then-execute models, which are expected to be popular in ubiquitous computing environment. For this goal, we measure web server performance through benchmark test.

4.1 Test Environment

For performance measuring, we use as a benchmark tool WebBench 5.0 developed by VeriTest [15]. Fig. 4 shows our test environment based on WebBench. In this environment, clients execute WebBench tests while sending repeated requests to the web server, and controller provides a means to set up, start, stop, and monitor the WebBench tests.

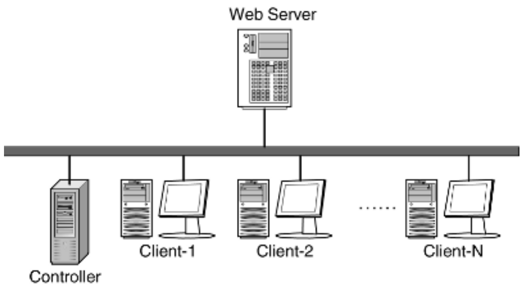


Fig. 4. Test environment architecture

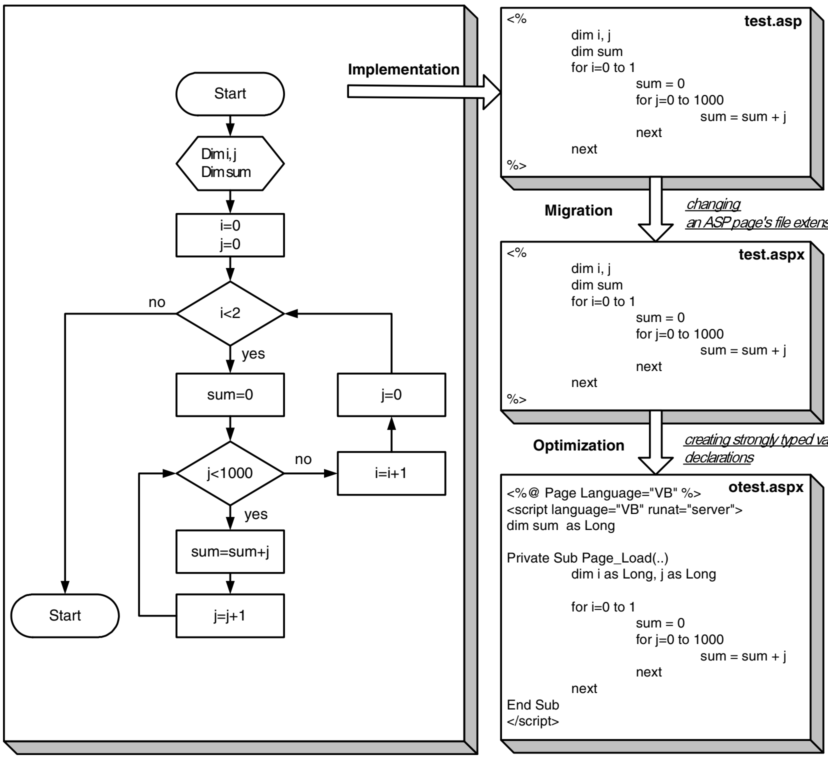
System specification is as follows.

- Web Server
  H/W: Intel Pentium III 1GHz processor, 256MB main memory
  S/W: Microsoft Windows Server 2003, Internet Information Server 6.0
- Controller
  H/W: Pentium III 733MHz processor, 256MB main memory
  S/W: Microsoft Windows XP Home Edition, WebBench Controller
- Client
  H/W: Pentium III 733MHz processor, 256MB main memory
  S/W: Microsoft Windows XP Home Edition, WebBench Client

4.2 Benchmark Program

We design a benchmark program as shown in Fig. 5. The benchmark program is first implemented as an ASP page, which is then migrated into an ASP.NET page according to [16-19]. After migrated, the ASP.NET page is optimized. Especially, for migration from ASP to ASP.NET, we change just an ASP page's file extension from .asp to

.aspx. Such a port allows the impact of compiled execution on performance to be measured. In order to examine performance improvement caused by new features of ASP.NET besides compiled execution, we optimize the migrated ASP.NET page by creating strongly typed variable declarations.



test.asp
```
<%
          dim i, j
          dim sum
          for i=0 to 1
                    sum = 0
                    for j=0 to 1000
                              sum = sum + j
                    next
          next
%>
```

test.aspx
```
<%
          dim i, j
          dim sum
          for i=0 to 1
                    sum = 0
                    for j=0 to 1000
                              sum = sum + j
                    next
          next
%>
```

otest.aspx
```
<%@ Page Language="VB" %>
<script language="VB" runat="server">
dim sum  as Long

Private Sub Page_Load(..)
          dim i as Long, j as Long

          for i=0 to 1
                    sum = 0
                    for j=0 to 1000
                              sum = sum + j
                    next
          next
End Sub
</script>
```

Fig. 5. Benchmark program

5 Benchmark Results and Analysis

In our benchmark test, three different versions of the benchmark program mentioned above are tested to compare their performance. For that, multiple clients (from 16 to 96) concurrently send repeated requests to the web server during 300 seconds. The main metric used for performance measuring is throughput, which is number of requests processed per second.

5.1 Results of Benchmark Test

Fig. 6 compares throughputs produced by the three versions of the benchmark program. Because of compiled execution, performance of an ASP.NET page is expected to be better than that of an ASP page. However, our benchmark test shows that the

ASP page has better performance than the ASP.NET page. This strange result may be caused by high initialization overhead of the ASP.NET page. That is, the ASP.NET page, which is a .NET assembly, should be interpreted to native code at runtime by Just-in-time (JIT) compiler and then loaded in memory before its execution. In addition, since the benchmark program has small code size, the ASP page does not suffer from the interpretation overhead.

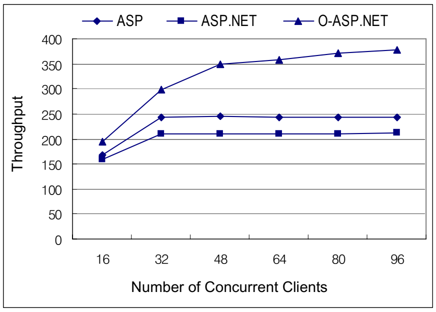
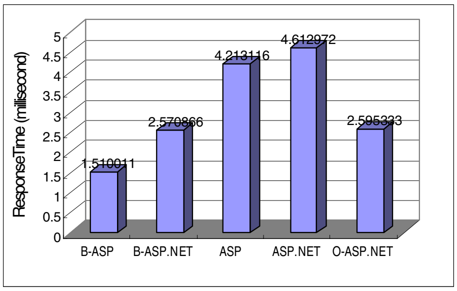


Fig. 6. Comparison of throughputs produced by three different versions



B-xxx: Blank xxx Page
O-xxx: Optimized xxx Page
Tbasp : response time of the blank ASP page
Tbaspx : response time of the blank ASP.NET page
Tasp : response time of the ASP page
Taspx : response time of the ASP.NET page
Toaspx : response time of the optimized ASP.NET page
eTasp : execution time of the ASP page
eTaspx : execution time of the ASP.NET page
eToaspx : execution time of the optimized ASP.NET page
eTasp = Tasp - Tbasp = 2.703105
eTaspx = Taspx - Tbaspx = 2.042106
eToaspx = Toaspx - Tbaspx = 0.024467

Fig. 7. Comparison of response times (64 clients, period of 600 seconds)

To analyze the initialization overhead, we make two blank pages with a file extension such as .asp or .aspx. Since such blank pages execute no commands, their response time reflects only non-execution time including initialization, network transfer time and so forth, thus being able to be used as non-execution time of three different versions. These blank pages and three versions are tested in a way that 64 clients repeatedly send requests to the web server during 600 seconds. Fig. 7 shows that the response time of the blank ASP page, *Tbasp*, is much less than that of the blank ASP.NET page, *Tbaspx*. Such difference between *Tbasp* and *Tbaspx* indicates that the ASP.NET page has higher initialization overhead than the ASP page. But, the actual execution time of the ASP.NET page, *eTbaspx*, is faster than that of the ASP page, *eTbasp*. Thus, in ASP.NET environment, compiled execution cannot substantially improve performance due to the initialization overhead.

To analyze the impact of code size on performance, we modify the benchmark program to perform the same action without for-loop, and then build three different versions from the modified one, which have large code size resulting in high interpretation overhead.

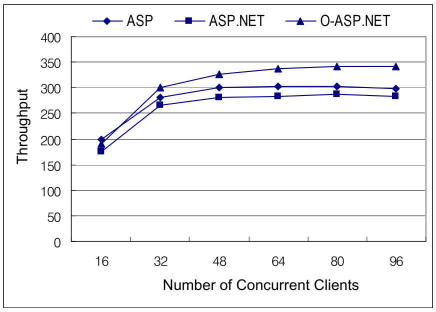


Fig. 8. Comparison of throughputs produced by three different versions without for-loop

Fig. 8 compares throughputs produced by these three versions without for-loop. As depicted in Fig. 8, the ASP page still has better performance than the ASP.NET page, though their performance difference becomes small. On the other hand, the optimized ASP.NET page, unlike the ASP.NET page, achieves the best throughput in both cases. Thus, from the above benchmark tests, we can know that optimization rather than compiled execution may considerably improve performance of the ASP.NET page.

5.2 Analysis

In this section, we develop a speedup model based on the above result. For that, we first analyze the response times of both the ASP page and the ASP.NET one. Since, as illustrated in Fig. 9, not only the optimization degree *d2* but also the initialization overhead degree *d1* may influence performance, the speedup model should consider them together. The speedup model is derived as follows:

$$d1 = neTaspx - neTasp \approx Tbaspx - Tbasp, \quad \text{where } neTaspx \approx Tbaspx,\ neTasp \approx Tbasp \tag{1}$$

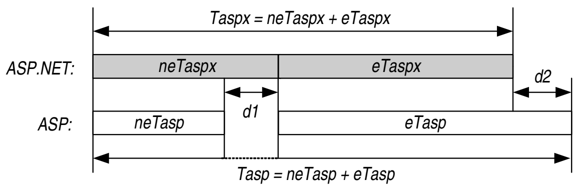
$$\begin{aligned} d2 &= eTasp - eTaspx = (Tasp - neTasp) - (Taspx - neTaspx) \\ &= (Tasp - Taspx) + (neTaspx - neTasp) = (Tasp - Taspx) + d1 \\ &\approx (Tasp - Taspx) + (Tbaspx - Tbasp) \end{aligned} \tag{2}$$

$$\begin{aligned} \text{Speedup } S &= Tasp/Taspx = Tasp/(neTaspx + eTaspx) \\ &= Tasp/((neTasp + d1) + (eTasp - d2)) = Tasp/(Tasp - (d2-d1)) \\ &= Tasp/(Tasp-D), \text{ where } D = d2 - d1,\ d2 \le eTasp \end{aligned} \tag{3}$$

$$\text{The maximum Speedup } Smax = Tasp/(neTasp+d1) \tag{4}$$

$$d2 = Taspx \times (S-1)/S + d1, \text{ where } S > 1 \tag{5}$$

Given *Tasp*, *Tbasp* and *Tbaspx*, we can calculate speedup *S* according to *d2* and the maximum speedup *Smax* through equation (3) and (4). Also, equation (5) enables *d2* to be approximated according to speedup *S*.



Tbasp : response time of the blank ASP page
Tbaspx : response time of the blank ASP.NET page
Tasp : response time of the ASP page
Taspx : response time of the ASP.NET page
eTasp : execution time of the ASP page
eTaspx : execution time of the ASP.NET page
neTasp : non-execution time of the ASP page
neTaspx : non-execution time of the ASP.NET page

Fig. 9. Analysis of response times

Table 1. Parameters given from the benchmark results presented in Fig. 7 and results computed by our speedup model (* indicates the results computed from our speedup model)

Parameter or Result	Values
Tbasp	*1.510010615 ms (millisecond)*
Tbaspx	*2.570865919 ms*
Tasp	*4.213116274 ms*
Taspx	*4.612971676 ms*
Toaspx	*2.595333072 ms*
*d1**	*1.060855304 ms*
*The maximum d2**	*eTasp = Tasp-Tbasp = 2.703105659 ms*
d2 of the ASP.NET page*	*(Tasp - Taspx) + d1 = 0.661 ms*
d2 of the optimized ASP.NET page*	*(Tasp - Toaspx) + d1 = 2.678638506 ms*
S1: speedup of the ASP.NET page*	*Tasp / Taspx = 0.913319346*
S2: speedup of the optimized ASP.NET page*	*Tasp / Toaspx = 1.623343192*
*Maximum speedup Smax**	*Tasp/(neTasp+d1) = 1.63879269*

We apply the above equations to the benchmark results presented in Fig. 7. In Table 1, there are the parameters given from the benchmark results and the results computed by our speedup model. In Fig. 10, our speedup model estimates speedup *S* according to *d2* and *d2* according to speedup *S*.

The above figure and table show that, in the benchmark test of Fig. 7, the ASP.NET page can achieve at most *Smax* (less than 2.0) and *S2*, the speedup of the optimized ASP.NET page, is almost same as *Smax*. Thus, we can see that optimization is the most important factor for substantially improving performance in ASP.NET environment.

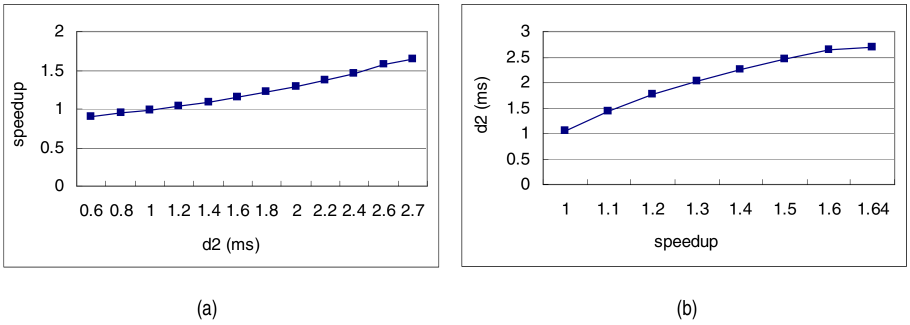


Fig. 10. (a) speedup *S* according to *d2* (b) *d2* according to speedup *S*

6 Conclusions

Due to not only the popularity of the web technology but also the importance of personalized interfaces, the dynamic web page service is becoming important in ubiquitous computing environment. In this paper, we explore the web technologies that allow ubiquitous spaces to create dynamic web pages in accordance with user profiles, especially concentrating on the server-side scripting approach, which is expected to be popular in ubiquitous computing environment. The server-side scripting approach mainly adopts the execute-while-parsing, which suffers from the interpretation overhead. Recently, the compile-then-execute model was proposed to address the overhead. Thus, it is assumed that the compile-then-execute model improves the execute-while-parsing model. We compare and analyze the two models. For this goal, we perform benchmark test in Microsoft ASP and ASP.NET environment by using WebBench 5.0 as a benchmark tool.

The results of the benchmark test may be summarized as follows.

First, the compile-then-execute model cannot substantially improve performance due to the high initialization overhead. Second, the best performance can be achieved through optimization rather than compiled execution. Thus, it is necessary for the compile-then-execute model to be accompanied by optimization to gain the maximum performance improvement. Since optimization needs expensive development costs, it is desirable for developers to estimate the maximum possible speedup in advance. For that, we develop a speedup model based on the benchmark results.

References

1. M. Weiser, "The Computer for the Twenty-First Century," Scientific American, pp. 94-10, Sept. 1991
2. M. Weiser, "Hot Topics: Ubiquitous Computing" IEEE Computer, Oct. 1993.
3. A. Shahi, V. Callaghan, M. Gardner, "Introducing Personal Operating Spaces for Ubiquitous Computing Environments," In Proceedings of Pervasive Mobile Interaction Devices (PERMID 2005), pp. 10-14, May 2005

4. T. Nakajima and I. Satoh, "Personal Home Server: Enabling Personalized And Seamless Ubiquitous Computing Environments," In Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communication (PerCom'2004), pp.341-345, Mar. 2004
5. R. Jimeno, Z. Salvador, A. Lafuente, M. Larrea, A. Uribarren, "An architecture for the personalized control of domotic resources," In Proceedings of the 2nd European Union symposium on Ambient intelligence, pp.51-54, Nov. 2004
6. S. Coolbrandt, "An introduction to CGI," Software Engineering Group 5, Feb. 2004 http://wilma.vub.ac.be/~se5/files/tutorials/CgiIntroduction.htm
7. B. Kothari and M. Claypool, "Performance Analysis of Dynamic Web Page Generation Technologies," Proceedings of International Network Conference (INC), July 2000
8. G. Gousios and D. Spinellis, "A Comparison of Portable Dynamic Web Content Technologies for the Apache Server," Proceedings of the 3rd International System Administration and Networking Conference, pp.103-119, May 2002
9. B. Doyle and C. V. Lopes, "Survey of Technologies for Web Application Development," ACM Journal Narne, Vol. 2, No.3, pp. 1-43, June 2005
10. P. Simons and R. Babel, "FastCGI The Forgotten Treasure," ApacheCon Europe 2001, Oct. 2001
11. Microsoft Corporation, "Active Server Pages Tutorial," Microsoft MSDN, Dec. 2000
12. Microsoft Corporation, "Introduction to ASP.NET, " Microsoft MSDN
13. Microsoft Corporation, "Why ASP.NET?," Microsoft ASP.NET, http://www.asp.net/ whitepaper/ whyaspnet.aspx
14. Sudhirmangla, "Beginners Introduction to ASP.NET, " The Code Project, June 2003 http://www.codeproject.com/
15. VeriTest and PC Magazine, "WebBench 5.0", ZDNet, 2002 http://www.veritest.com/ benchmarks/webbench/
16. B. VedanthaRamanujan, "Migration From ASP to ASP.NET," BadreNarayanan.V's Radio Weblog, Sept. 2003
17. J. Kieley, "Migrating to ASP.NET: Key Considerations," Microsoft MSDN, Nov. 2001
18. S. Mitchell, "Converting ASP to ASP.NET, " Microsoft MSDN, Nov. 2001
19. Microsoft Corporation, "Using the ASP to ASP.NET Migration Assistant," Microsoft MSDN