# Parse & Dispatch: Parallelizing the Generation of Dynamic Web Content

**Stavros Papastavrou**
**Department of Computer Science**
**University of Cyprus**
**75 Kallipoleos Str., P.O.Box 20537**
**Nicosia, Cyprus**
**stavrosp@ucy.ac.cy**

**George Samaras**
**Department of Computer Science**
**University of Cyprus**
**75 Kallipoleos Str., P.O.Box 20537**
**Nicosia, Cyprus**
**cssamara@ucy.ac.cy**

**Paraskevas Evripidou**
**Department of Computer Science**
**University of Cyprus**
**75 Kallipoleos Str., P.O.Box 20537**
**Nicosia, Cyprus**
**skevos@ucy.ac.cy**

**Panos K. Chrysanthis**
**Department of Computer Science**
**University of Pittsburgh**
**Pittsburgh, PA 15260, U.S.A.**
**panos@cs.pitt.edu**

## ABSTRACT

The use of dynamically generated Web content is gaining in popularity over traditional static HTML content. Dynamic Web content is generated on the fly according to the instructions embedded in HTML script files. Such instructions generate HTML by carrying out various forms of tasks such as session tracking, database queries and transactions, dynamic image generation, or even fetching information from remote servers. Popular portal and e-commerce Web pages contain a number of tasks that are executed in a serial manner by traditional multithreaded Web servers. In this ongoing work, we propose a parallel approach for dynamic content generation, and elaborate on how it affects the design and performance of Web servers. We have developed a prototype Web server that supports the parallel processing of tasks involved in the dynamic content generation with improved throughput and response time as compared to the classical (serial) approach.

## Keywords

Web Servers, Parallel Processing, Dynamic Content, HTML

## 1. INTRODUCTION

Web servers are the basic component of the World Wide Web [1] in terms of content delivery. Early Web servers, such as the NSCA HTTPd Web server, were used for dissemination of static documents (files) based on a specification called the Hypertext Markup Language. The need, however, for personalized or customized content, also called dynamic content, led to the introduction of the Common Gateway Interface [2]. CGI could generate dynamic content by running external programs/scripts that typically accessed an application specific content database. With the great expansion of the Web, CGI was substituted by more persistent and scalable technologies. Modern Web servers support multiple runtime environments in which script code is embedded in an HTML script file which is then parsed to generate a dynamic Web page.

Such a runtime environment executes either as library code within the Web server process, or as a separate process communicating via a standard application interface. Microsoft's Active Server Pages, Macromedia's Cold Fusion, and PHP are examples of such a runtime technology that support augmented versions of HTML. A typical augmented HTML script file that is parsed in order to generate a dynamic Web page includes static content (in HTML), and multiple script code blocks each one performing a specific task (static content and script code blocks are assumed to be interleaved). Personal menu bar generation, lengthy distributed database transactions, remote XML fetching, image generation, shopping cart validation, credit card verification, catalog generation are only a few examples of such tasks. For instance, a stock-related Web page may include: (a) left menu bar generation with links dynamically generated from the local content database, (b) top menu bar generation with links customized to a particular client's preferences

which are stored in the local content database, (c) one or more main articles/stories retrieval, with articles extracted from a corporate news database, (d) current stock market indices retrieval retrieved from a remote financial provider via XML (presented either graphically or in text), and (e) currency exchange rates retrieval. The execution time of these tasks may vary from tens to hundreds of milliseconds.

Modern multithreaded (pool-based) Web servers process HTML script files by parsing their contents. For each client request, a worker thread is selected out of a poll of pre-activated worker threads in order to parse the file. During parsing, static content is stored in a buffer where a script code block (that performs a task) forces the worker thread to halt parsing in order to execute it. The content generated by the task is appended in the buffer, and parsing resumes. When the end of the file is reached, the worker thread transmits the buffer to the Web client that requested the dynamic page. This blocking multithreading of the tasks introduces a processing bottleneck that reduces Web server throughput.

In this ongoing work, we focus on improving Web server performance in producing dynamic Web content by introducing parallelism in processing the tasks included in an HTML script file. For that reason, we propose the *Parse and Dispatch* approach, and show that it outperforms the traditional serial approach adopted by modern Web servers. The scope of our initial experimentation, that is reported in this paper, was to provide proof of concept of our methodology. For this reason, we did not deal with dependencies between tasks. The handling of dependencies in parallel processing is a well-researched topic [4]. It is straightforward to apply techniques developed by the parallel processing community in handling dependencies in dynamic content generation. This will be part of our future work.

## 2. PARSE AND DISPATCH APPROACH

The key idea of the Parse and Dispatch approach is to decouple the parsing of an HTML script file and the execution of the embedded tasks, enabling uninterrupted/non-blocking parsing of the HTML script file and parallel execution of the script code blocks (tasks). The approach consists of two phases: the *Content Expansion*, and the *Content Serialization*.

During the Content Expansion phase, the worker thread parses the HTML script file, stores temporarily the static content in a buffer, but does not halt in order to execute an encountered script code block (task). Instead, an auxiliary thread is dispatched to perform the task, freeing the worker thread to resume parsing. In order to ensure that both static and dynamic content are delivered at the end to the client in the right order (since the auxiliary threads may not terminate in the order that they were dispatched), the buffer used to store temporarily the content is slotted and indexed.

With the end of the file reached, the worker thread enters the Content Serialization phase where it monitors the auxiliary threads. Once all auxiliary threads terminate and store their generated content in the indexed buffer, the worker thread iterates and flashes the buffer's contents to the client that requested the dynamic page. In order to utilize the time and resources that the worker thread spends while waiting the auxiliary threads to finish, the last task in an HTML script file is assigned for execution to the worker thread.

## 3. PERFORMANCE EVALUATION

We want to compare the performance between a traditional multi-threaded Web server that executes the tasks of a dynamic Web page in serial, and an experimental multi-threaded Web server that executes the tasks of a dynamic Web page in parallel according to the Parse and Dispatch approach. The traditional Web server was developed by adopting code from the Java-based Apache Web server [3]. The experimental Web server is a modification of the traditional one in order to ensure maximum compatibility between results.

For both Web servers, we use a worker thread pool of size 21. This selection is based on the recommendation of the popular Web server vendors which advise a pool size between five and twenty threads. We found out that this range provides a complete scope of results for evaluating the two approaches.

We measure the performance of a Web server in terms of (1) *average throughput*, and (2) *average client response time* (perceived latency), both under different (stable) workloads. Formally, we define throughput as the average number of client HTTP requests that are completely processed by a Web server in a period of one second. Throughput is computed at the Web server site. In our experiments, Web server workload depends on the following two metrics: (a) the number of client requests that are simultaneously admitted by the Web server for processing (we refer to this metric as 'Concurrent Clients'), and (b) the number of tasks embedded in an HTML script file. In addition, client response time is the average perceived latency from the moment the client issues an HTTP request, until the complete HTTP response is sent by the Web server. In our experiments, Concurrent Clients range from 1 to 21, and embedded tasks range from 2 to 8.

Since the blend and nature of tasks embedded in dynamic Web pages differ across various Web applications, we assume a typical task with an approximate execution time of 50 milliseconds that favors no particular type of Web application. The typical task includes script code for accessing a content database plus CPU intensive script code.

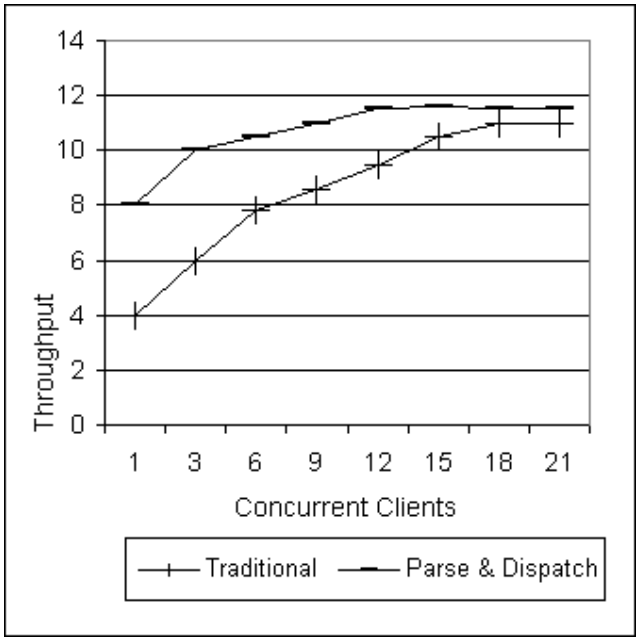


**Figure 1. Average Throughput with 5 Tasks per HTML Script File.**

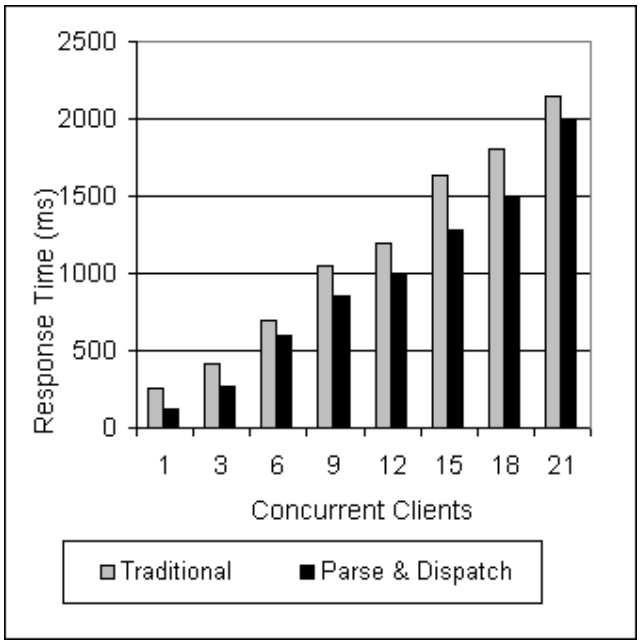


**Figure 2. Average Client Response Time with 5 Tasks per HTML Script File.**

The results shown in Figure 1 (throughput) and Figure 2 (client response time) clearly indicate the performance benefits of the Parse and Dispatch approach with 5 included tasks in an HTML script file. As expected, we found out that the performance gains are less obvious if we decrease the number of tasks. This is because by reducing their number, we reduce the amount of parallelism that the Parse and Dispatch approach can exploit. On the other hand, by increasing this number towards eight in an attempt to increase parallelism, we tend to overload the server and in particular when the tasks involve remote access to content database servers. This overload has an immediate negative impact on the performance of the Parse and Dispatch approach, which for eight tasks and more than three concurrent clients it drops much below the throughput of the traditional approach.

## 4. CONCLUSION AND FUTURE WORK

In this paper, we suggest a parallel approach in executing the dynamic parts (tasks) of a dynamically generated Web page. The approach, namely the Parse and Dispatch, was used in building an experimental Web server and its performance was compared to that of a traditional Web server that executes the dynamic parts in serial. The experimental results yielded significant performance gains in favor of our approach in terms of Web server throughput and client response time under certain conditions. Maximum performance is achieved by identifying and working with a moderate amount of parallelism in processing dynamic web pages. We expect that by further fine-tuning the multithreading environment, we will achieve even better results. As part of our future work, we will focus on applying parallel processing techniques in handling dependencies among tasks and developing more efficient multi-treading Web server architectures.

## 5. ACKNOWLEDGEMENTS

This work has been supported in part by the European IST project DBGlobe, IST-2001-32645 and by the U.S. National Science Foundation under grant IIS-9812532.

## 6. REFERENCES

1. T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, A. Secret: *The World-Wide Web*. Communications of the ACM 37(8): 76-82(1994).
2. G. Ehmayer, G. Kappel, S. Reich: *Connecting Databases to the Web: A Taxonomy of Gateways*. Proc. of DEXA Conference, Springer-Verlag, LNCS 1308, 1-15 (1997).
3. R. T. Fielding, G. E. Kaiser: *The Apache HTTP Server Project*. IEEE Internet Computing 1(4): 88-90 (1997).
4. C. D. Polychronopoulos: *Parallel Programming and Compilers*. Kluwer International Series in Engineering and Computer Science, Volume 59, ISBN 0-89838-288-2 (1998).