Generating dynamic content at database-backed web servers: cgi-bin vs mod_perl

Alexandros Labrinidis
labrinid@cs.umd.edu

Nick Roussopoulos[†]
nick@cs.umd.edu

Department of Computer Science and Institute for Systems Research,
University of Maryland, College Park, MD 20742

Abstract

Web servers are increasingly being used to deliver dynamic content rather than static HTML pages. In order to generate web pages dynamically, servers need to execute a script, which typically connects to a DBMS. Although CGI was the first approach at server side scripting, it has significant performance shortcomings. Currently, there are many alternative server side scripting architectures which offer better performance than CGI. In this paper, we report our experiences using *mod_perl*, an Apache Server module, which can improve the performance of CGI scripts by at least an order of magnitude. Except for presenting results from our experiments, we also briefly describe the implementation of an industrial strength database-backed web site that we recently built and give a quick overview of the various server-side scripting mechanisms.

1 Introduction

In less than 10 years from its debut in the early 90s, the Web has changed our lives dramatically. From comparing prices and shopping online, to viewing realtime stock quotes and managing our bank accounts, the Web is increasingly being used as the means to do everyday tasks. One common denominator for all these activities is the need to generate *dynamic content* [B$^+$98]. Personalization, frequent updates, and searching/querying capabilities are the most common reasons behind dynamically generated web pages.

In order to generate dynamic content, web servers need to execute a program, through some *server-side scripting mechanism*. This script typically connects to a DBMS, performs a query, retrieves the results, and formats them in HTML in order to be returned to the user. Although cgi-bin was the first approach at running programs at the web server, it was not designed to handle the demand for dynamic content creation that web servers face today.

A plethora of server-side scripting mechanisms have been proposed to replace cgi-bin [Gre99, Mal98, FLM98]). One popular mechanism is *mod_perl*, which is a module that can be used with the Apache Server to support efficient CGI-like server-side scripting.

Instead of having a program that generates HTML, the various forms of *annotated HTML* embed scripting commands within an HTML document. The web server parses and executes these commands, before the final HTML page is sent to the user. The most popular annotated HTML approaches are three. 1) PHP [PHP] is free, open source software with many features and support for practically all DBMS platforms. PHP also supports persistent database connections. 2) Active Server Pages [ASP] is Microsoft's solution to server-side scripting and is supported by the Internet Information Server, Microsoft's Web server. ASP scripts can connect to SQL Server, Access, Oracle, Informix or any ODBC-compliant database. 3) JavaServer Pages [JSP] is SUN's ap-

[†]Also with the Institute for Advanced Computer Studies, University of Maryland.

proach to server-side scripting, that uses XML-like tags and scriptlets written in Java to encapsulate the logic that generates content for the page. JavaServer Pages can be used together with the Java Servlets architecture.

Java Servlets [SRV] are another approach at dynamic content generation. They are protocol and platform independent server-side components written in Java. Servlets dynamically extend Java enabled servers.

There is no simple answer when deciding which server-side scripting mechanism is best for a particular setting. However, if moving from or upgrading an existing CGI-based application, then the solution which dramatically improves performance and has the smallest *migration cost* is clearly mod_perl.

In the next section we present the various CGI-based server-side scripting mechanisms in more detail. In Section 3, we describe the implementation of a real-life application, and in Section 4 we present the results of performance experiments comparing cgi-bin to mod_perl. Finally, we discuss our conclusions in the last section.

2 Server-Side CGI Scripting

2.1 The CGI protocol

CGI[1] is a simple **protocol** that specifies the way in which user-defined scripts that run at the web server can communicate with users' browsers. Scripts that follow the CGI protocol are called *CGI scripts*. A CGI script operates in a rather straightforward way. First, it gets invoked by the web server, reads the user's input, which was typically submitted through an HTML form, and parses it. Then it does whatever processing is required and, finally, the script generates an HTML page that is returned to the user (Figure 1).

The CGI protocol dictates how form data should be passed from the web server to the script. Specifically, the data must be transformed into one long string of name-value pairs like: "name1=value1&name2=value2&....", after the values have been URL-encoded by converting spaces to + signs and special characters to %xx sequences, where xx is the ASCII value for the given character. For example, the = sign should be converted to %3d (see [Gun96] for more details). When a CGI script is activated, it must parse this long string into name-value pairs, doing the inverse transformations.

[1] CGI is short for Common Gateway Interface.

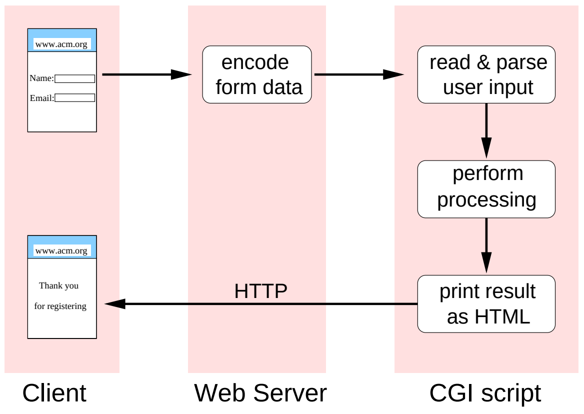

Figure 1: Execution of a CGI script

The CGI protocol also mandates that the standard output of the script is forwarded to the user. This means that the CGI script is expected to print the necessary HTTP headers first, followed by the results page in HTML.

Finally, it should be pointed out that the CGI protocol does not specify how the scripts are to be invoked by the server. Also, the CGI protocol does not restrict the choice of programming language for writing the scripts. The ability to use one's favorite programming language is one of the main reasons behind the popularity of CGI scripts.

2.2 The cgi-bin invocation mechanism

The cgi-bin invocation mechanism was adopted in the implementation of CGI scripts by the first web servers (CERN & NCSA). Each time the server needs to run a CGI script, it has to spawn a new process, setup the CGI environment, run the script, and send the script's standard output to the user's browser (Figure 2).

This approach to server side-scripting has some advantages. First of all, since it follows the CGI protocol, the choice of programming language is not restricted. The second big advantage to forking a new processes for every CGI script is that it

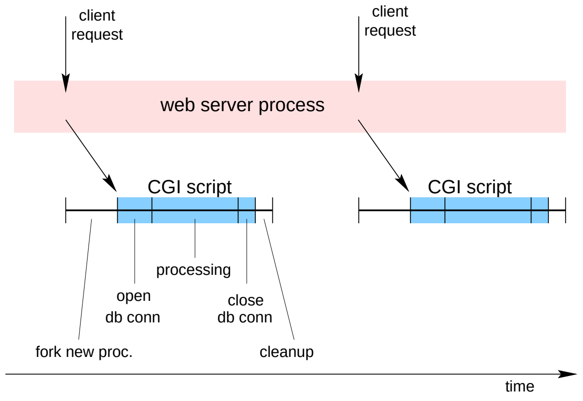

Figure 2: cgi-bin script invocation

provides an easy separation between the web server and application scripts; "bad" programs will not crash or slowdown the server.

There are however many disadvantages to using the cgi-bin invocation mechanism, the most important of which is the poor performance of the web server, especially under high loads. Forking a new process for every CGI script has a big operating system overhead. Moreover, each script is expected to have an initial setup phase, in order for example to establish a connection to the DBMS, initialize variables, etc. Since the lifespan of CGI scripts is typically very small, all this setup overhead is "wasted" and has to be repeated with every script. To overcome this performance problem, a number of solutions have been proposed like FastCGI [FCG], which attempts to eliminate the process spawning overhead and mod_perl, which incorporates the CGI scripts inside the Apache Server, as we see in more detail in the next section.

Finally, another disadvantage of the cgi-bin approach are the long and ugly URLs like *http://www.cs.umd.edu/cgi-bin/search.pl* that must be used.

2.3 The mod_perl invocation mechanism

Mod_perl [MP] is a server scripting module for the widely popular[2] Apache web server [AS]. CGI scripts on a mod_perl-enabled Apache Server run **within** the server processes (Figure 3), thus eliminating the overhead of spawning a new process with every client request. Furthermore, the scripts can

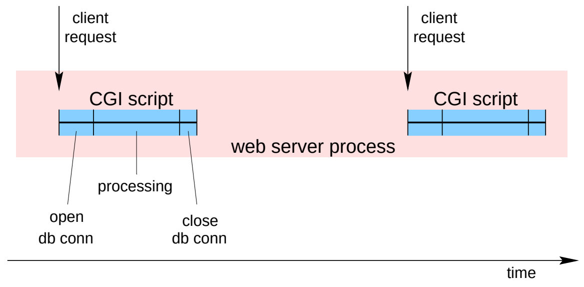

Figure 3: mod_perl script invocation

store state information across multiple invocations and can, for example, maintain open connections to the DBMS, thus avoiding the cost of establishing a connection with every request (Figure 4). Overall, the mod_perl approach minimizes the setup overhead for serving CGI scripts.

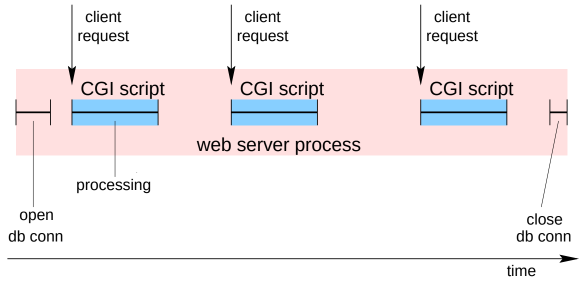

Figure 4: mod_perl script invocation (with persistent database connections)

The Apache Server is well designed and very robust, so that, although mod_perl scripts run within the server processes, "bad" scripts cannot crash the web server. For example, each script can service at most *MaxRequestsPerChild* requests, after which it must be reloaded, thus cleaning up any possible memory leaks it may have had.

Another advantage of the mod_perl approach is that CGI scripts have access to the Apache Server internals via an API, which increases their functionality. A direct consequence of this is being able to get rid of the ugly URLs that were required by the cgi-bin approach. For example we can easily have *http://www.cs.umd.edu/search* instead of *http://www.cs.umd.edu/cgi-bin/search.pl*.

[2]According to the December 1999 Netcraft Server Survey, an estimated 54.5% of web sites over the world use the Apache Server [NSS99].

One last advantage of mod_perl is the ease of migration from existing CGI scripts which were written to work with cgi-bin. Since most of these scripts were written in Perl or C, the transition is straightforward [CMP].

The mod_perl approach has only one disadvantage: the choice of programming language is somewhat restricted, since it has to be Perl or C. However, we do not believe this to be a real problem as these two languages are the most popular ones for writing CGI scripts.

3 A real-life application

In this section, we briefly present our experiences from implementing the *dbgrads system* [DBG], a real-life application on a database-backed web server. The dbgrads system is a searchable database of students that have a background in databases and will be graduating soon. It is intended to be used by prospective employers from academia or industry around the world in order to locate candidates for their job openings.

3.1 Software checklist

Before building the dbgrads system, we had to install a few software packages first: the web server, the DBMS and a few Perl modules. All of the software was free and publicly available.

Web server We used the Apache Server [AS], version 1.3.9, which, except for being free, is fast, efficient, portable, well supported, stable, reliable, extensible, easy to administer and has many features [SM99]. After having installed the Apache Server, we installed the mod_perl module [MPG] version 1.21.

DBMS We chose MySQL [MSQ, YRK99] version 3.22.27 as the database server, primarily because it is free, light-weight and can run as a normal user process (i.e. without super-user privileges) thus minimizing any possible security risks. In order for our scripts to communicate with the MySQL database server, we used the Perl DBI module, a generic Perl interface to relational DBMSs, combined with the DBD (Database Driver) module for MySQL (both available from CPAN [CPN]).

Perl Perl version 5.004 [PRL] was already installed in our system, as well as the CGI.pm module [CGP, Ste98], which is a very useful library for writing CGI scripts in Perl.

3.2 Implementation

Writing code for a database-backed web server is similar to any other code development project, with one notable difference being the need for rapid prototyping and short turn-around times. For that reason, although it is important to come up with an initial design document outlining the functionality of the entire system, most probably there will be a lot of revisions to it by the end of the project. Also, since the web application is to be released to the entire world, special care should be taken to guarantee security, thus preventing attacks to the system. Finally, care must also be taken so that the web application can handle all possible errors internally, without the user ever having to see the generic Error Message page that most web servers have.

One big advantage to using the Apache Server with the mod_perl module is that we can have *virtual URLs*. We can, for example, specify that the URL *http://www.acm.org/mysearch* will correspond to a dynamically generated document (as opposed to a static HTML page), that was computed by a special, user-defined, *content handler* Perl function [SM99].

One big advantage to using the Perl DBI/DBD modules, is that we are not tied to a particular DBMS vendor and can, almost effortlessly, switch to a different database server. For example, the `quote()` function, part of the DBI library, serves to this purpose by correctly quoting SQL strings.

Due to space limitation, we cannot present more information on the dbgrads system. However, for more details on the implementation of a database-backed web site, the reader is referred to [Gre99], [SM99], and [CT98].

4 Experiments

For our experiments, we used the Apache Server version 1.3.6 and, instead of MySQL, the Informix Dynamic Server with Universal Data Option ver. 9.14. Both the web server and the DBMS run on the same machine, a SUN UltraSparc-5 with 320 MB of main memory and a 3.6 GB Seagate Medalist disk, running Solaris 2.6. Multiple clients were generated through multi-threading on another SUN machine, which was on the same local area network to minimize network variations. Each client performed about 60 requests, one after the other.

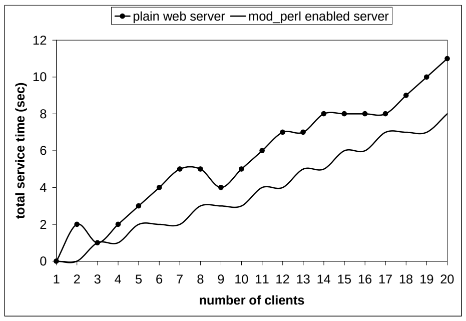

Figure 5: Delivery of static HTML pages

First of all we tried to see if there is any difference between the plain Apache web server and the mod_perl-enabled Apache Server, as far as delivery of static HTML pages is concerned. The results from our experiments are in Figure 5. The x-axis is the number of concurrent clients, and the y-axis is the time it takes for all clients to complete their request workload. We can see that the mod_perl-enabled Apache Server performs equally well with the plain server. In other words, the added functionality and the increased footprint do not affect the performance of static content delivery.

Figure 6 has the results from the second experiment, where we compared cgi-bin with mod_perl on dynamically generated web pages (that correspond to results from SQL queries). We studied two variations of mod_perl: plain mod_perl and *mod_perl+*, where the database connections are kept persistent. As expected, the mod_perl approach outperforms cgi-bin by at least an order of magnitude. Specifically, the performance of plain mod_perl is about 10 times better than that of cgi-bin, whereas the performance of mod_perl+ is about 20 times better than that of cgi-bin. This means that just by keeping the database connections persistent we cut the response times in half.

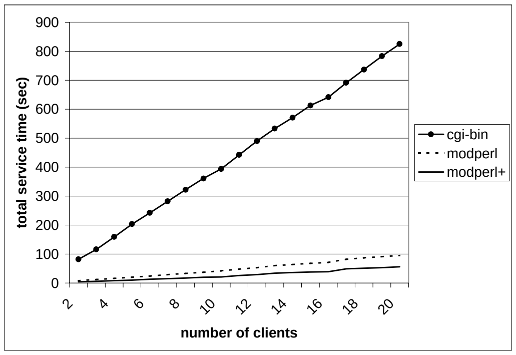

Figure 6: Comparison of cgi-bin to mod_perl

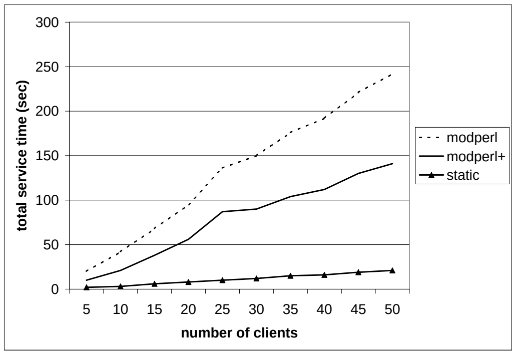

Figure 7: Scaling experiment

In our last experiment, we scaled up the number of concurrent clients and measured the performance of mod_perl, mod_perl+ (mod_perl with persistent db connections) and static (static HTML pages, served by a mod_perl-enabled Apache Server). Although the performance of static pages is, as expected, the most scalable solution, both mod_perl variants scale well, with the mod_perl+ approach giving consistently the best performance for dynamically generated web pages.

5 Conclusions

In this paper we presented a brief overview of the various server-side scripting mechanisms that are used to generate dynamic web content. We also discussed some details from the implementation of a real-life application on a database-backed web server. Finally, we presented experiments to support that the cgi-bin script invocation mechanism is at least an order of magnitude slower than mod_perl, and should thus be abandoned.

References

[AS] Apache Server home page. *http://www.apache.org/httpd.html*.

[ASP] Active Server Pages help page. *http://www.4guysfromrolla.com/*.

[B+98] Phil Bernstein et al. "The Asilomar Report on Database Research". *SIGMOD Record*, 27(4), December 1998.

[CGP] CGI.pm home page. *http://stein.cshl.org/WWW/software/CGI/cgi_docs.html*.

[CMP] Quick guide for moving from CGI to mod_perl. *http://perl.apache.org/dist/cgi_to_mod_perl.html*.

[CPN] Comprehensive Perl Archive Network. *http://www.cpan.org*.

[CT98] Tom Christiansen and Nathan Torkington. *"Perl Cookbook"*. O'Reilly & Associates, August 1998.

[DBG] dbgrads home page. *http://www.acm.org/sigmod/dbgrads*.

[FCG] FastCGI home page. *http://www.fastcgi.com*.

[FLM98] Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. "Database Techniques for the World-Wide Web: A Survey". *SIGMOD Record*, 27(3):59–74, September 1998.

[Gre99] Philip Greenspun. *"Philip and Alex's Guide to Web Publishing"*. Morgan Kaufmann, June 1999. Available at *http://http://photo.net/wtr/thebook/*.

[Gun96] Shishir Gundavaram. *"CGI Programming on the World Wide Web"*. O'Reilly & Associates, First edition, March 1996. Out of print, available at *http://www.oreilly.com/openbook/cgi/*.

[JSP] JavaServer Pages. *http://java.sun.com/products/jsp*.

[Mal98] Susan Malaika. "Resistance is Futile: The Web Will Assimilate Your Database". *Data Engineering Bulletin*, 21(2):4–13, June 1998.

[MP] mod_perl home page. *http://perl.apache.org*.

[MPG] mod_perl development guide. *http://perl.apache.org/guide*.

[MSQ] MySQL home page. *http://www.mysql.com*.

[NSS99] Netcraft Server Survey, Dec. 1999. *http://www.netcraft.com/survey*.

[PHP] php home page. *http://www.php.net*.

[PRL] Perl home page. *http://www.perl.com*.

[SM99] Lincoln Stein and Doug MacEachern. *"Writing Apache Modules with Perl and C"*. O'Reilly & Associates, April 1999.

[SRV] Java Servlets API. *http://java.sun.com/products/servlet*.

[Ste98] Lincoln Stein. *"The Official Guide to CGI.pm"*. John Willey & Sons, April 1998.

[YRK99] Randy Jay Yarger, George Reese, and Tim King. *"MySQL and mSQL"*. O'Reilly & Associates, August 1999.