Himanshu Kumar
Yu Song
Columbia University
Fall 2007

PERFORMANCE ANALYSIS OF WEB PROGRAMMING LANGUAGES

Abstract

Using a benchmark script that renders a table from an SQL database, the performance ofseveral different programming languages, such as Perl, ASP.Net, and PHP, is analyzed.

Introduction

As preparation, we configured Internet Information Services (IIS) on a Windows machine, installed SQL Server 2005 and MySQL server on it, and then installed the binaries for the programming languages.

The performance analysis consists of four test cases. Each language was scripted to run those four tests and the response time is analyzed and recorded using a benchmark tool.

Server Configuration

Server Configuration

- HP Proliant Server
- Dual Intel Pentium 4 CPU 3.06 GHz
- 2.17 GB Ram, 3 GB Virtual Memory
- Dual Gigabit Ethernet Server Adapters
- Microsoft Windows XP Pro

We had two similarly configured servers for the project, we initially intended to install Windows and Linux on them so that we could divide the languages and SQL servers on the two servers. During development we ran into problems installing the software on linux, since most of the software needed to be recompiled before installation. For example, to add mySQL connection capabilities to Perl, we needed to recompile perl binaries with extra parameters that did not work so well with other options needed to add MSSQL connection capabilities. Similar problems were also encountered with installation of the Apache web server. We then found that all the software we needed could also be installed on the windows server without the need of recompilation or extensive configurations. All the software needed for the project could be installed to work with the IIS web server without any problems, while some of them were installed to work on the Apache Tomcat server.

Software Installation

- IIS 5.1

- .Net Framework 2.0
- Active Perl 5.8.8
- PHP 5.2.5
- Microsoft SQL Server 2005
- MySQL Server 5.0
- MySQL GUI Tools 5.0
- TortoiseSVN/Subversion 1.4.5
- Microsoft Visual Studio 2005
- Open Perl IDE 1.0.11
- MySQL connector 5.0.8.1
- MySQL ODBC 3.51 driver
- Microsoft SQL connection driver

Test Configuration

Database

MS SQL, mySQL and postgreSQL database servers were used in the testing. The database contained three tables, with 10,000 records each in two of the tables and 20,000 in the third table. An ASP.Net script was written to populate the tables with random data.

- Microsoft SQL Server 2005: The default setup was used without any changes to the security or performance
- mySQL 5.1.11: The default configuration was used, with MySQL GUI Tools 5.0 providing the front end for managing the databases.

Test Case	Description	# of Records
1	Print "Hello World<BR>" 10,000 times	10000
2	Student, professorInfo, personalInfo tables read from database and loaded into memory before rendering the web page	40000
3	Inner Join on Student and PersonalInfo tables and load the results into memory before rendering the database	10000
4	Table from #3, quicksort in code on "Name" column	10000

Database Table Layout								
Student	ID	Name (100 Char)	Track (50 Char)	AdvisorID (int)				
PersonalInfo	ID	Name (100 Char)	Type (20 Char)	Age (int)	Address (200 Char)	Phone (10 Char)	WebAddress (100 Char)	Email (50 Char)
ProfessorInfo	ID	Department (50 Char)	Office (100 Char)	Hours (50 Char)				

Benchmarking tool

We tested many benchmarking tools available on a website [1] that lists numerous web site testing and performance analysis tools. Most of these tools were commercial software geared towards testing performance of websites to help developers optimize their sites. We decided against using these software due to their extreme complexity and somewhat irrelevant results. The tools were not easy to configure and sometimes returned unexpected results when tests were repeated multiple times, for example, returning less than a 10 ms interval for a page of size more than 30 MB that displayed 40,000 records on screen. So we developed a small application using Microsoft Visual Studio 2005 & Visual C# to request a webpage and measure the time taken for the request. The measurement was taken as the time between the HTTP request was made and the last byte of data received. The information gathered by the application is displayed on the application window as well as stored as a log file in the executable's directory.

Testing procedure

The web server was restarted before the tests, and no extra services or applications were allowed to run during the test apart from important system services. Each test case was conducted five times using the order described below, while the test system was connected to university network using an ethernet cable to minimize the network delays.

Another test was later added to determine if the test results were linear in nature when the length of these tests were varied. The new tests added two more tests for each test case where a limit was placed on the number of records returned by the database. These limits were set to 50% and 25% of the records returned by the test scripts.

Caching

One problem faced was some type of caching taking place, probably caching by the IIS server, when a web page is requested multiple times in sequence. To avoid this we used two strategies:

- Add a dummy parameter to the query string for the pages, and set its value to be different for each request. For example: test.aspx?n=1000, test.aspx?n=1001
- Perform the tests in a predetermined sequence to avoid caching on the database side or the web server side. The order chosen was: Test1, Test3, Test2, Test4. This order avoided having tests 3 and 4 next to each other since both tests requested the same query from the database.

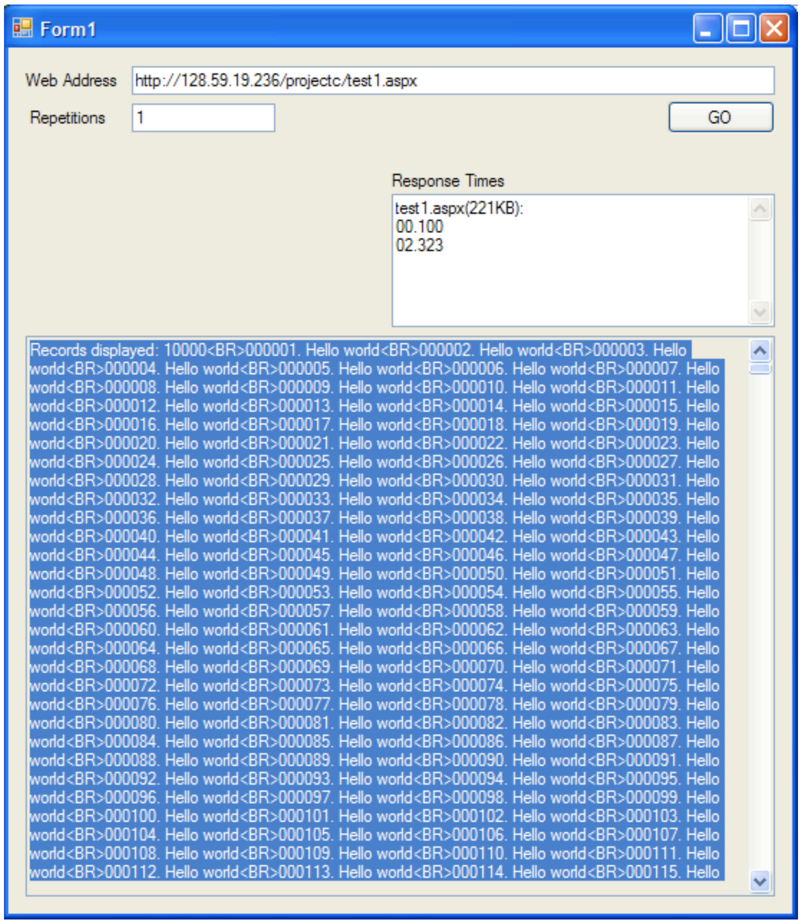
Test Observations

The test cases originally consisted of printing the data received from the database in test #2. #3 and #4. But due to the difference in printing speeds for the languages, these test results were skewed in favor of the language with the fastest printing speeds. So it was decided to remove the print statements to be able to accurately compare the database transaction speeds of the languages.

The test results show that Perl was the fastest displaying large amount of text that was not pulled from a database, while PHP was the second placed language. In the database access and quick-sort tests, ASP.Net was the fastest with PHP coming in second place consistently. In the two SQL databases used for testing, MS SQL performed better when accessed from ASP.Net, while mySQL performed better when accessed from PHP. The performance difference in ASP.Net might be a result of the maturity of the custom connection drivers for the two databases used by ASP.Net. The difference in performance in PHP could lie in the use of an ADODB driver used by PHP to connect to the MSSQL database due to the absence of a custom MSSQL driver for PHP, while it used the mySQL connection driver for accessing the MySQL database. Perl performed equally well accessing both the databases.

Project Technical Details

Benchmarking Tool



Web Address: http://128.59.19.236/projectc/test1.aspx

Repetitions: 1

Response Times

```
test1.aspx(221KB):
00.100
02.323
```

```
Records displayed: 10000<BR>000001. Hello world<BR>000002. Hello world<BR>000003. Hello
world<BR>000004. Hello world<BR>000005. Hello world<BR>000006. Hello world<BR>000007. Hello
world<BR>000008. Hello world<BR>000009. Hello world<BR>000010. Hello world<BR>000011. Hello
world<BR>000012. Hello world<BR>000013. Hello world<BR>000014. Hello world<BR>000015. Hello
world<BR>000016. Hello world<BR>000017. Hello world<BR>000018. Hello world<BR>000019. Hello
world<BR>000020. Hello world<BR>000021. Hello world<BR>000022. Hello world<BR>000023. Hello
world<BR>000024. Hello world<BR>000025. Hello world<BR>000026. Hello world<BR>000027. Hello
world<BR>000028. Hello world<BR>000029. Hello world<BR>000030. Hello world<BR>000031. Hello
world<BR>000032. Hello world<BR>000033. Hello world<BR>000034. Hello world<BR>000035. Hello
world<BR>000036. Hello world<BR>000037. Hello world<BR>000038. Hello world<BR>000039. Hello
world<BR>000040. Hello world<BR>000041. Hello world<BR>000042. Hello world<BR>000043. Hello
world<BR>000044. Hello world<BR>000045. Hello world<BR>000046. Hello world<BR>000047. Hello
world<BR>000048. Hello world<BR>000049. Hello world<BR>000050. Hello world<BR>000051. Hello
world<BR>000052. Hello world<BR>000053. Hello world<BR>000054. Hello world<BR>000055. Hello
world<BR>000056. Hello world<BR>000057. Hello world<BR>000058. Hello world<BR>000059. Hello
world<BR>000060. Hello world<BR>000061. Hello world<BR>000062. Hello world<BR>000063. Hello
world<BR>000064. Hello world<BR>000065. Hello world<BR>000066. Hello world<BR>000067. Hello
world<BR>000068. Hello world<BR>000069. Hello world<BR>000070. Hello world<BR>000071. Hello
world<BR>000072. Hello world<BR>000073. Hello world<BR>000074. Hello world<BR>000075. Hello
world<BR>000076. Hello world<BR>000077. Hello world<BR>000078. Hello world<BR>000079. Hello
world<BR>000080. Hello world<BR>000081. Hello world<BR>000082. Hello world<BR>000083. Hello
world<BR>000084. Hello world<BR>000085. Hello world<BR>000086. Hello world<BR>000087. Hello
world<BR>000088. Hello world<BR>000089. Hello world<BR>000090. Hello world<BR>000091. Hello
world<BR>000092. Hello world<BR>000093. Hello world<BR>000094. Hello world<BR>000095. Hello
world<BR>000096. Hello world<BR>000097. Hello world<BR>000098. Hello world<BR>000099. Hello
world<BR>000100. Hello world<BR>000101. Hello world<BR>000102. Hello world<BR>000103. Hello
world<BR>000104. Hello world<BR>000105. Hello world<BR>000106. Hello world<BR>000107. Hello
world<BR>000108. Hello world<BR>000109. Hello world<BR>000110. Hello world<BR>000111. Hello
world<BR>000112. Hello world<BR>000113. Hello world<BR>000114. Hello world<BR>000115. Hello
```

The benchmarking tool was written in Visual C# as a windows application. It's front end consisted of a text box to enter the web address of the web page to be tested, another text box to enter the number of times to request the web page, a button to start the test, and a couple of output text boxes to display the results. It also saves the results into a text log file as comma separated values so that it can be opened in a spreadsheet application for analysis.

The core code of the benchmark tool is shown below. It records the start time before creating an HTTP web request for the web page in question, and loops through the data that is received from the server in 8KB chunks. The end time of the request is recorded after the last byte of data is received which is displayed by the application and logged in a comma separated text file.

```
byte[] buf = new byte[8192];
start = DateTime.Now;
// prepare the web page we will be asking for
HttpWebRequest request = (HttpWebRequest)
    WebRequest.Create(args);

// execute the request
HttpWebResponse response = (HttpWebResponse)
    request.GetResponse();

// we will read data via the response stream
Stream resStream = response.GetResponseStream();

do
{
    // fill the buffer with data
    count = resStream.Read(buf, 0, buf.Length);
    if(size == 0)
        responseT = DateTime.Now;
    size += count/1024;

    // make sure we read some data
    if (count != 0)     {
        // translate from bytes to ASCII text
        tempString = Encoding.ASCII.GetString(buf, 0, count);
        // continue building the string
        sb.Append(tempString);
    }
}
while (count > 0); // any more data to read?
end = DateTime.Now;
```

ASP.Net

Working with ASP.Net was easy due to the easy installation process to install Visual Studio, which installed IIS by default with ASP.Net. Using the IIS Manager, the folder containing the ASP.Net scripts was setup as a website, and the external IP address of the system was assigned to IIS to allow remote systems to view the webpages rendered by the server. MySQL connector 5.0.8.1 was used to connect to the mySQL database, while the installation contained a default driver to connect to the MSSQL database.

The following code was used in the test #1 to print "Hello World".

```
for (int i = 0; i < count; i++)
        Response.Write((i+1).ToString("000000") + ". Hello
world<BR>");
```

The following code was used to connect to the MSSQL database. It reads the connection string defined in the Web.config file, connects to the database and executes the SQL command and returns a data reader object that is used to read the results of the command. The connection to the database is automatically closed when the data has been read:

```
SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["StudentDBCo
nnectionString"].ConnectionString);
SqlCommand cmd = new SqlCommand("select * from Student", conn);
conn.Open();
dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
while (dr.Read())
{
     String s = "";
     for (int i = 0; i < dr.FieldCount; i++)
         s += dr[i] + ", ";
     x++;
}
```

The mySQL .Net connector can be used in a very similar fashion as shown above. The class names are also similar, with an addition of "My" to the start of the above class names:

```
MySqlDataReader dr;
MySqlConnection myConnection = new
MySqlConnection(ConfigurationManager.ConnectionStrings["mySQLStud
entDBConnectionString"].ConnectionString);
MySqlCommand cmd = .....
```

The following function was written to quick-sort the table retrieved from the database. A new class was written that would take a 2D array as a parameter and return the sorted array:

```
l_hold = left;
r_hold = right;
pivot = a[left];

while (left < right)
{
    while ((a[right][columnIndex].CompareTo(pivot[columnIndex])
>= 0) && (left < right))
        right--;

    if (left != right) {
        a[left] = a[right];
        left++;
    }
    while ((a[left][columnIndex].CompareTo(pivot[columnIndex]) <=
0) && (left < right))
        left++;
    if (left != right)    {
        a[right] = a[left];
        right--;
    }
}
a[left] = pivot;
int piv = left;
left = l_hold;
right = r_hold;
if (left < piv)
    q_sort(left, piv - 1);
if (right > piv)
    q_sort(piv + 1, right);
```

Perl

The following code was used for test #1 to print the static string onto the web page:

```
for $i(1..10000){
      print "Hello World<BR>";      # Print the item
}
```

To connect to the MS SQL database, the following code was used. The connection string is defined first:

```
$connectionInfo="dbi:ODBC:driver={SQL Server};server=
$host;database=StudentDB;";
```

Next the connection is established to the database using an ODBC database connector:

```
$dbh = DBI->connect($connectionInfo,$userid,$passwd)|| die "Got
error $DBI::errstr when connecting to $dsn\n";
```

Next the query is executed and prepared to be accessed by binding variables to the corresponding columns in the database tables:

```
$query = "SELECT * FROM Student";
$sth = $dbh->prepare($query);
$sth->execute();
# assign fields to variables
$sth->bind_columns(\$ID, \$Name, \$Track, \$Advisor);
```

Then a 2 dimensional array is initialized and the data read from the tables is pushed into the array:

```
my @AoA = ([]);
$n=0;
while($sth->fetch()) {
   #print "<tr><td>$ID<td>$Name<td>$Track<td>$Advisor\n";
   $n = $n + 1;
   @a = [$ID, $Name, $Track, $Advisor];
   push @AoA, @a;
}
```

Finally the number of records read is printed to the web page and the connection to the database is closed.

```
print "$n<BR>\n";
$sth->finish();
```

To access the mySQL database, only the connection string is changed to use the mySQL connector that was installed as a module through the Perl setup application:

```
$connectionInfo="dbi:mysql:$db;$host";
```

The following subroutine was written to perform the quicksort operation on the 2D array read from the database in the Test #4:

```
sub qsort {
  @_ or return ();
  if($#_ == 0){
    return (); }
  my @p = shift;
  my @less = ([]);
  my @rest = ([]);
   foreach (@_) { @a=@$_;
      if (@a[0] < @p[0])  {
        push @less, @a;    }
      else                {
        push @rest, @a;    }
   }
   my @t1 = qsort(@less);
   push @t1,@p;
   my @t2 = qsort(@rest);
   foreach (@t2) { @a=@$t;
      push @t1, @a;
   }
  (@t1);
}
```

PHP

The following code was used for printing the "Hello World" statement onto the webpage:

```
while ($i < $num) {
      printf("%06s. Hello world<BR>\n",   $i+1);
      $i++;
}
```

To connect to MS SQL, a COM object is created for ADO DB connector:

```
$conn = new COM ("ADODB.Connection")  or die("Cannot start ADO");
```

Next the connection string is defined and the connection is opened:

```
$connStr = "PROVIDER=SQLOLEDB;SERVER=".$myServer.";UID=".
$myUser.";PWD=".$myPass.";DATABASE=".$myDB;
  $conn->open($connStr);
```

Next the query is executed, and an array is created that holds all the columns returned by the database:

```
$rs = $conn->execute($query);
$num_columns = $rs->Fields->Count();
for ($i=0; $i < $num_columns; $i++) {
    $fld[$i] = $rs->Fields($i);
}
```

Finally looping through the data returned by the database, and the data is compiled together into a row which is added to an array.

```
while (!$rs->EOF) { //carry on looping through while there are
records
    $row=array();
    for ($i=0; $i < $num_columns; $i++)
        $row[]=$fld[$i]->value;
    $array[]=$row;
    $rs->MoveNext(); //move on to the next record
}
```

For connecting to the mySQL database, the default connection driver was used:

```
mysql_select_db("StudentDB") or die(mysql_error());
```

The connection is established and read into a variable. A 2D array is then populated from the result data set and the variable holding the data set is emptied out from memory.

```
$result=mysql_query($query);
mysql_close();
while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
    $array[]=$row;
mysql_free_result($result);
```

The following function was used to quicksort the 2D array read from the database:

```
function quicksort($seq)
{
        if(!count($seq)) return $seq;
        $k = $seq[0][1];
        $x = $y = array();
        for($i=1; $i<count($seq); $i++)        {
                if($seq[$i][1] <= $k)
                        $x[] = $seq[$i];
                else
                        $y[] = $seq[$i];
        }
        return array_merge(quicksort($x), array($seq[0]),
quicksort($y));
}
```

There was a known bug in the mySQL connector that comes with the PHP installation, where the connection threads would not exit cleanly, leaving an exception that was displayed on the web pages rendered by PHP. This resulted in a large delay being added to the end of the page. To fix this problem an older version of the libmysql.dll, we used version 5.2.1 instead of the default version 5.2.5 that was installed.

MySQL

MySQL GUI Tools 5.0 was used as a front end for the mySQL installation. It was a windows application which can be used to configure all the aspects of the mySQL installation, along with create and manage the databases and tables defined on the server. The mySQL installation was easy to setup, and it was set to run as a service and start up with windows.

The SQL commands used in the test cases are:

Test #2:

```
SELECT * FROM Student
SELECT * FROM PersonalInfo
SELECT * FROM ProfessorInfo
```

Test #3 and #4:

```
SELECT     Student_1.ID, Student_1.Name, Student_1.Track,
PersonalInfo_1.Age, PersonalInfo_1.Address, PersonalInfo_1.Phone,
PersonalInfo_1.Email, PersonalInfo_1.WebAddress FROM
Student AS Student_1 INNER JOIN PersonalInfo AS PersonalInfo_1 ON
Student_1.ID = PersonalInfo_1.TypeID WHERE
(PersonalInfo_1.Type = 'student')
```

Modified command for use in testing linearity of the test results, "limit 2500" is added to the SQL command, for example:

```
SELECT * FROM Student limit 2500
```

MSSQL

Microsoft SQL Server 2005 was installed on the server, which installed SQL Server Management Studio as the front end used to manage all the aspects of the server. The tables and user permissions were defined using this tool.

The SQL commands used in the test cases are:

Test #2:

```
SELECT * FROM Student
SELECT * FROM PersonalInfo
SELECT * FROM ProfessorInfo
```

Test #3 and #4:

```
SELECT     Student_1.ID, Student_1.Name, Student_1.Track,
PersonalInfo_1.Age, PersonalInfo_1.Address, PersonalInfo_1.Phone,
PersonalInfo_1.Email, PersonalInfo_1.WebAddress FROM
Student AS Student_1 INNER JOIN PersonalInfo AS PersonalInfo_1 ON
Student_1.ID = PersonalInfo_1.TypeID WHERE
(PersonalInfo_1.Type = 'student')
```

Modified command for use in testing linearity of the test results, "top N" is inserted into the SQL command, for example:

```
SELECT top 2500 * FROM Student
```

Task List

Himanshu Kumar

- IIS installation & configuration
- Microsoft SQL Server 2005 installation
- MySQL database server installation
- Database connection drivers (ODBC, ADODB, MySQL, MS SQL)
- PHP installation
- Perl installation
- Version control installation (TortoiseSVN)
- Script to fill databases with randomized data
- Test and analyze data for PHP, Perl, and ASP. Net.
- Develop the Benchmarking Application

Yu Song

- Tomcat5.5 installation
- PostgreSQL server installation
- Database connection drivers (MySQL, MS SQL, PostgreSQL)
- Python installation
- Ruby installation
- Tcl installation
- JDK installation
- Test and analysis data on Ruby, Python, Tcl, and JSP.

References

1. Rick Hower, "Web Site Test Tools and Site Management Tools", http://www.softwareqatest.com/qatweb1.html
2. Michael Gossland and Associates, "Perl Tutorial Course", http://www.gossland.com/course/index.html
3. Wikibooks, "Algorithm implementation/Sorting/Quicksort", http://en.wikibooks.org/wiki/Transwiki:Quicksort_implementations
4. Nik Silver, "Perl Tutorial: Start", http://www.comp.leeds.ac.uk/Perl/start.html
5. "Perl DBI/DBD::ODBC Tutorial", http://www.easysoft.com/developer/languages/perl/sql_server_unix_tutorial.html
6. "Configuring and Testing a PERL Script with Internet Information Server", http://support.microsoft.com/kb/q150629/