Himanshu Kumar
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 of several different programming languages, Perl, ASP.Net, and PHP, and database engines are analyzed.

Introduction

We first 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 these four tests. The response time is analyzed and recorded using a benchmark tool.

Server Configuration

Server Configuration

Two HP Proliant Servers with Dual Intel Pentium 4 CPU (3.06 GHz), 2.17 GB Ram, 3 GB Virtual Memory and dual Gigabit Ethernet Server Adapters were used as the test servers. Microsoft Windows XP Pro and ubuntu were installed on the servers.

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 for 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 could also be installed to work on the Apache Tomcat server.

Software Installation

We installed Microsoft IIS 5.1 as the web server, along with Perl 5.2.5, Active Perl 5.8.8 and .Net Framework 2.0 for the language support. Microsoft SQL Server 2005 and mySQL Server 5.0 were installed as the database servers. To support these database servers, most languages needed separate drivers to be installed, this includes MySQL connector 5.0.8.1, MySQL ODBC 3.51 driver and Microsoft SQL connection driver. We also used Microsoft SQL Server Manager and MySQL GUI Tools 5.0 for SQL server management, and various software development IDE including Microsoft Visual Studio 2005, Open Perl IDE 1.0.11 and Eclipse. TortoiseSVN/Subversion 1.4.5 was extensively used for version control.

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 ” 10,000 times	10,000
2	Student, professorInfo, personalInfo tables read from database and loaded into memory before rendering the web page	40,000
3	Inner Join on Student and PersonalInfo tables and load the results into memory before rendering the database	10,000
4	Table from #3, Quicksort in code on "Name" column	10,000

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. One such instance is of 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 and 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 lengths 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.

Performance for cached and uncached pages was measured and analyzed separately. Cold tests were performed by restarting the web server and the database server before measuring the performance in the test cases. The Hot tests were performed immediately after the Cold tests, thus caching by the web server and the database server would be seen in these results.

Caching

One problem faced was some type of caching taking place, when a web page is requested multiple times in sequence. The caching was being done by the IIS server when the same web page was requested by the same host. 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 displaying the data received from the database in test #2. #3 and #4 to the client. However, 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.

To find out if caching played a role in the performance of the languages, Cold and Hot scenarios were added to the test plan. Cold tests were performed by restarting the web server and the database server before measuring the performance. The Hot tests were performed immediately after the Cold tests, thus caching by the web server and the database server would result in improved performance of the languages.

Test Results

Raw Data - For all Test cases

	ASP MSSQL	ASP mySQL	Perl MSSQL	Perl mySQL	PHP MSSQL	PHP mySQL
Test 1		1.33		0.13		0.19
		1.22		0.19		0.17
		1.28		0.11		0.20
Test 2	1.68	2.35	1.62	1.62	1.62	0.90
	1.40	2.32	1.47	1.63	1.66	0.89
	1.43	2.26	1.46	1.69	1.70	0.89
Test 3	1.21	1.93	0.97	0.90	0.79	0.58
	1.30	2.02	0.92	0.78	0.80	0.50
	1.28	1.88	0.88	0.83	0.80	0.40
Test 4	1.30	2.05	4.07	3.74	1.23	0.97
	1.30	2.03	3.85	3.76	1.28	0.99
	1.34	1.94	3.83	3.75	1.20	1.01

Cold Scenario: Performance in Seconds

	ASP MSSQL	ASP mySQL	Perl MSSQL	Perl mySQL	PHP MSSQL	PHP mySQL
Printing Test		0.38		0.09		0.14
		0.40		0.07		0.13
		0.29		0.06		0.13
		0.28		0.10		0.14
		0.43		0.16		0.12
		0.43		0.10		0.12
		0.24		0.08		0.16
		0.38		0.06		0.12
		0.41		0.07		0.14

	ASP MSSQL	ASP mySQL	Perl MSSQL	Perl mySQL	PHP MSSQL	PHP mySQL
Select Command Test	0.35	0.52	1.35	1.36	1.44	0.61
	0.35	0.54	1.35	1.37	1.54	0.57
	0.37	0.53	1.35	1.38	1.53	0.58
	0.36	0.53	1.35	1.36	1.57	0.60
	0.38	0.52	1.36	1.36	1.48	0.56
	0.38	0.53	1.33	1.35	1.49	0.58
	0.36	0.52	1.36	1.38	1.44	0.58
	0.36	0.50	1.37	1.40	1.44	0.56
	0.37	0.52	1.36	1.39	1.46	0.58
Join Table Test	0.14	0.23	0.63	0.61	0.73	0.27
	0.14	0.26	0.63	0.57	0.69	0.35
	0.16	0.20	0.62	0.58	0.64	0.27
	0.15	0.19	0.64	0.57	0.63	0.27
	0.15	0.21	0.62	0.57	0.63	0.26
	0.15	0.21	0.69	0.59	0.63	0.26
	0.15	0.19	0.63	0.57	0.64	0.28
	0.15	0.20	0.63	0.57	0.63	0.27
	0.15	0.21	0.62	0.56	0.63	0.26
QuickSort Test	0.17	0.22	3.69	3.47	1.07	0.76
	0.16	0.23	3.71	3.50	1.11	0.75
	0.17	0.27	3.70	3.54	1.12	0.75
	0.17	0.22	3.73	3.53	1.08	0.75
	0.17	0.27	3.71	3.50	1.06	0.75
	0.17	0.23	3.69	3.49	1.06	0.75
	0.18	0.23	3.67	4.31	1.05	0.75
	0.16	0.22	3.70	3.56	1.07	0.74
	0.18	0.27	3.67	3.53	1.05	0.76

Hot Scenario: Performance in Seconds

Averages - For all Test Cases

	ASP MSSQL	ASP mySQL	Perl MSSQL	Perl mySQL	PHP MSSQL	PHP mySQL
Printing Test		1.28		0.14		0.19
Select Command Test	1.51	2.31	1.52	1.65	1.66	0.89
Join Table Test	1.26	1.95	0.92	0.84	0.80	0.49
QuickSort Test	1.31	2.01	3.91	3.75	1.24	0.99

Cold Scenario: Performance in Seconds

HOT AVG	ASP MSSQL	ASP mySQL	Perl MSSQL	Perl mySQL	PHP MSSQL	PHP mySQL
Printing Test		0.36		0.09		0.13
Select Command Test	0.36	0.52	1.35	1.37	1.49	0.58
Join Table Test	0.15	0.21	0.63	0.58	0.65	0.28
QuickSort Test	0.17	0.24	3.69	3.60	1.08	0.75

Hot Scenario: Performance in Seconds

Raw Data - Linearity Test

	100%	50%	25%
ASP - MSSQL			
Select Command Test	0.44	0.20	0.11
Join Table Test	0.18	0.10	0.05
QuickSort Test	0.19	0.11	0.05
ASP - MySQL			
Select Command Test	0.55	0.20	0.12
Join Table Test	0.22	0.08	0.05
QuickSort Test	0.27	0.09	0.06
Perl - MSSQL			
Select Command Test	1.42	0.90	0.63
Join Table Test	0.71	0.57	0.48
QuickSort Test	3.81	1.91	1.08
Perl - MySQL			
Select Command Test	1.40	0.82	0.54
Join Table Test	0.61	0.41	0.31
QuickSort Test	3.69	1.80	0.93
PHP - MSSQL			
Select Command Test	1.55	0.91	0.60
Join Table Test	0.64	0.45	0.41
QuickSort Test	1.18	0.67	0.53
PHP - MySQL			
Select Command Test	0.59	0.33	0.18
Join Table Test	0.29	0.14	0.12
QuickSort Test	0.79	0.42	0.20

Performance in Seconds

Test Summary

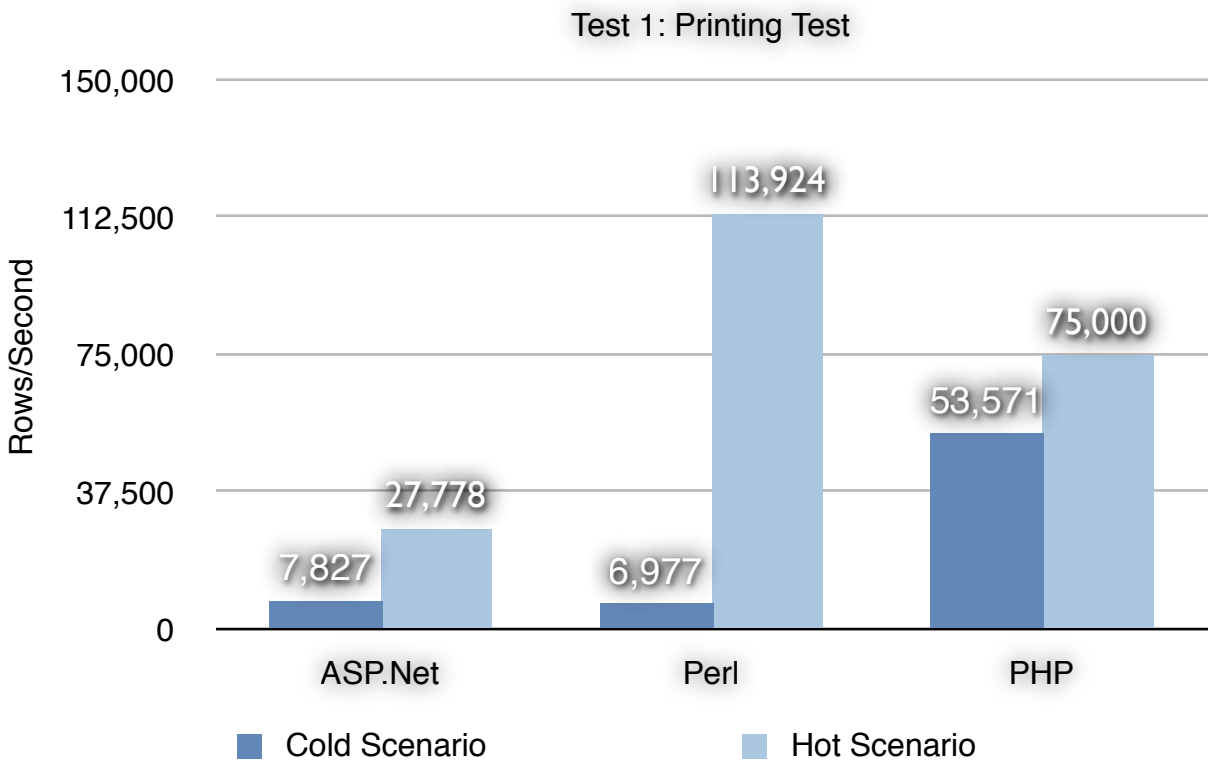
	ASP.NET	Perl	PHP
Printing Test	7,827	6,977	53,571
	MSSQL		
Select Command Test	26,566	26,339	24,067
Join Table Test	7,909	10,819	12,537
QuickSort Test	7,608	2,556	8,080
	MySQL		
Select Command Test	17,294	24,262	44,726
Join Table Test	5,139	11,938	20,270
QuickSort Test	4,978	2,670	10,091

Cold Scenario: Performance in Records/Second

	ASP.NET	Perl	PHP
Printing Test	27,778	113,924	75,000
	MSSQL		
Select Command Test	109,756	29,535	26,850
Join Table Test	67,164	15,762	15,382
QuickSort Test	58,824	2,709	9,298
	MySQL		
Select Command Test	76,433	29,124	68,966
Join Table Test	47,368	17,341	36,145
QuickSort Test	41,667	2,778	13,296

Hot Scenario: Performance in Records/Seconds

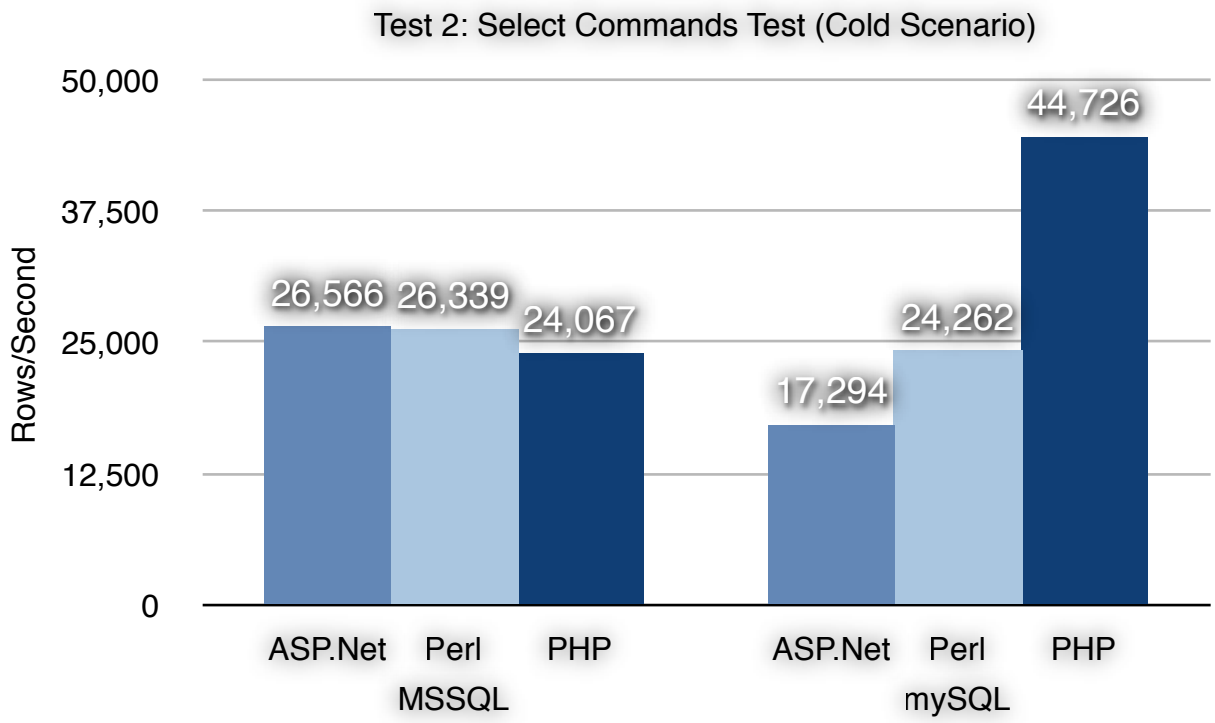
Test 1: Printing Test



For the cold web server scenario, PHP was the fastest of the three languages, faster than ASP by 85% and faster than PHP by 87%.

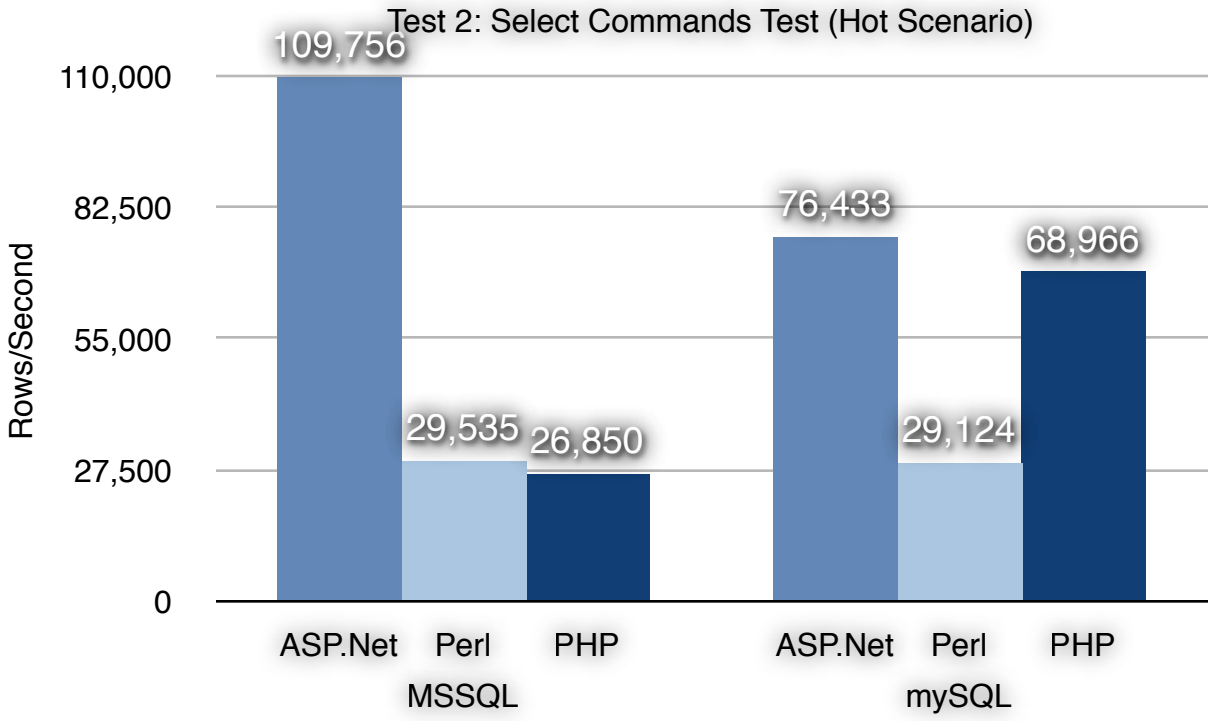
For the hot web server scenario, Perl is the fastest of the three languages in the printing test where the line "Hello World
" is printed onto the webpage. It had a 75% performance advantage over ASP.Net and 34% over PHP.

Test 2: Select Commands Test



For the cold web server scenario, ASP.Net is the fastest of the three languages for MSSQL, but the performance difference was very small. PHP was fastest when accessing mySQL, it was faster than ASP by 61% and Perl by 46%.

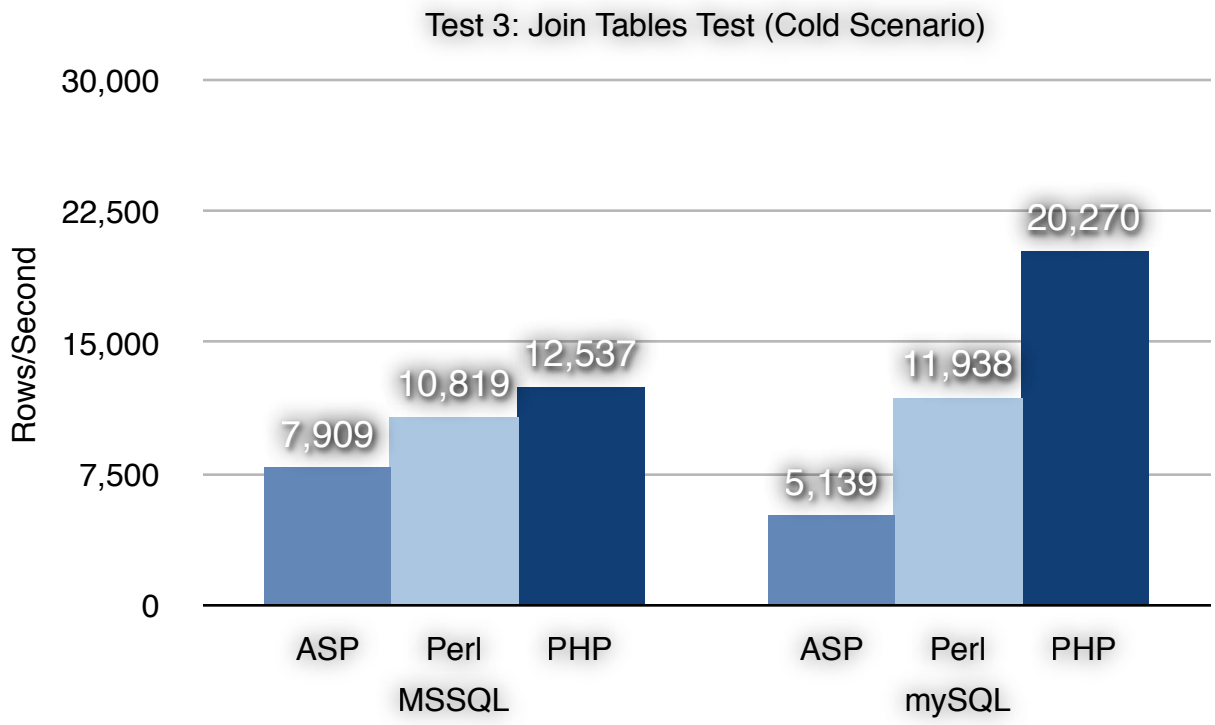
PHP was faster by 46% when accessing mySQL as compared to accessing MSSQL. ASP.Net was in turn faster by 35% as compared to the mySQL database when accessing the Microsoft SQL database.



For the hot web server scenario, ASP.Net is the fastest of the three languages in the test where three SELECT commands are sent to the SQL database and 40,000 records in total are returned to the script. ASP.Net was faster than Perl by 73% and PHP by 77% when accessing MS SQL and by 62% and 10% respectively when accessing mySQL.

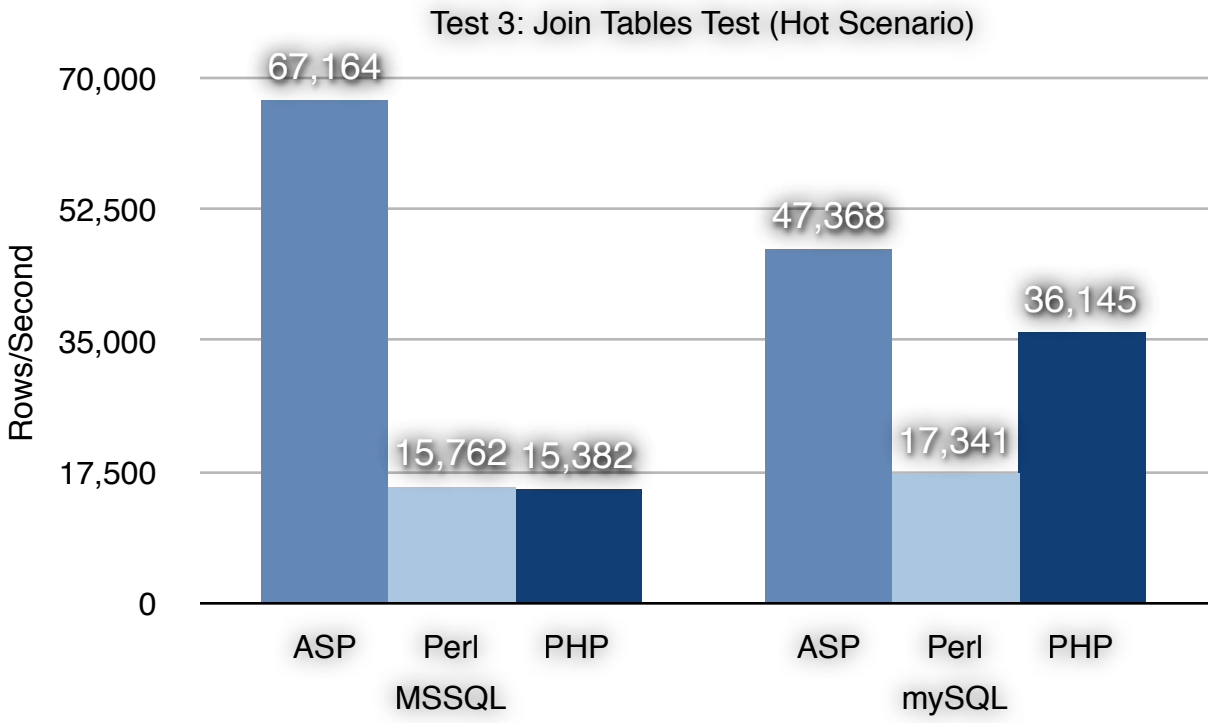
PHP was faster by 61% when accessing mySQL as compared to accessing MSSQL. ASP.Net was in turn faster by 30% when accessing the Microsoft SQL database as compared to the mySQL database.

Test 3: Join Tables Test



For the cold web server scenario, PHP is the fastest of the three languages in this test where two tables are merged using a JOIN command in the SQL database and 10,000 records are returned to the script. PHP was faster than ASP by 37% and Perl by 14% when accessing MS SQL and by 75% and 41% respectively when accessing mySQL.

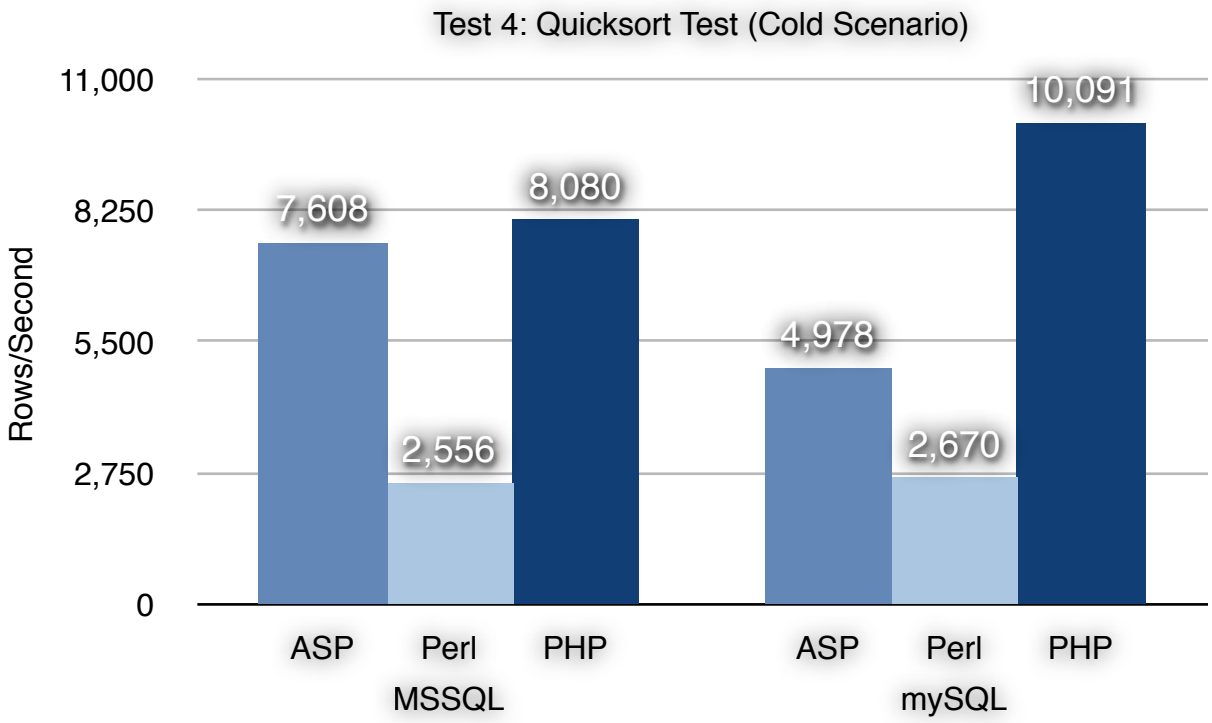
PHP was faster by 38% when accessing mySQL as compared to accessing MSSQL. ASP.Net was in turn faster by 35% when accessing the Microsoft SQL database as compared to the mySQL database.



For the hot web server scenario, ASP.Net is the fastest of the three languages in this test. Two tables are merged using a JOIN command in the SQL database and 10,000 records are returned to the script. ASP.Net was faster than Perl by 77% and PHP by 78% when accessing MS SQL and by 64% and 24% respectively when accessing mySQL.

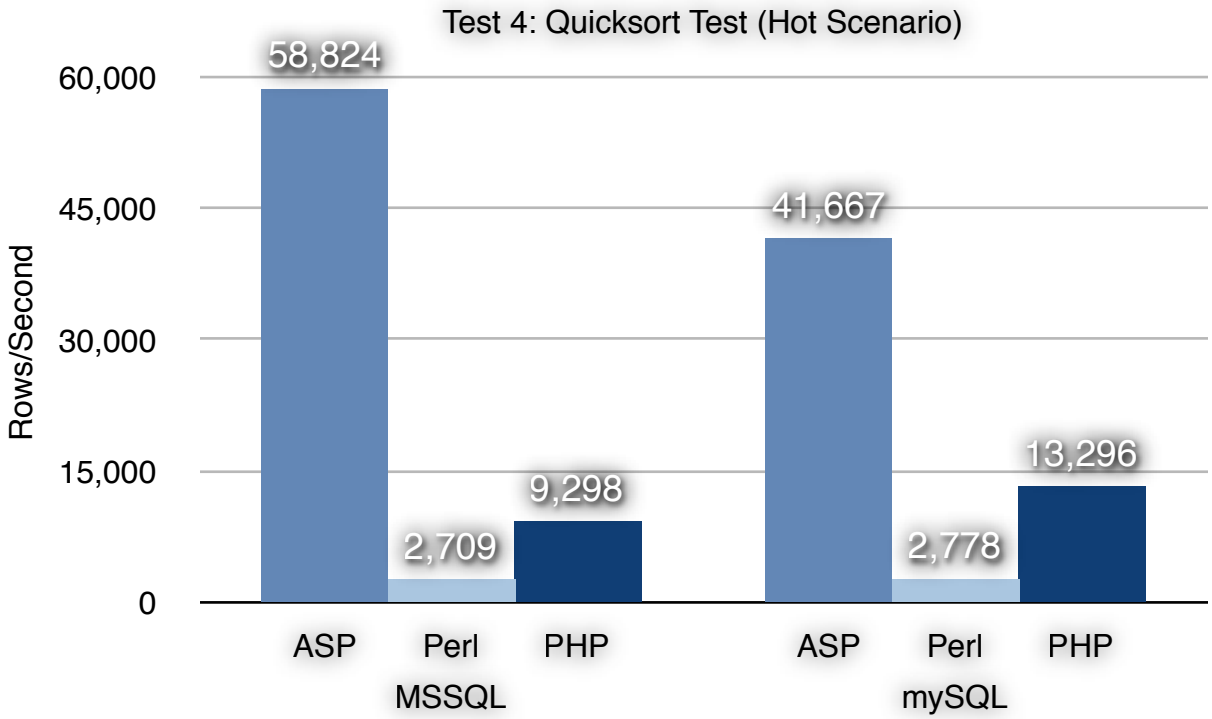
PHP was faster by 57% when accessing mySQL as compared to accessing MSSQL. ASP.Net was in turn faster by 29% when accessing the Microsoft SQL database as compared to the mySQL database.

Test 4: Quicksort Test



For the cold web server scenario, PHP is the fastest of the three languages in this test. PHP was faster than ASP by just 6% and Perl by 68% when accessing MS SQL and by 51% and 74% respectively when accessing mySQL.

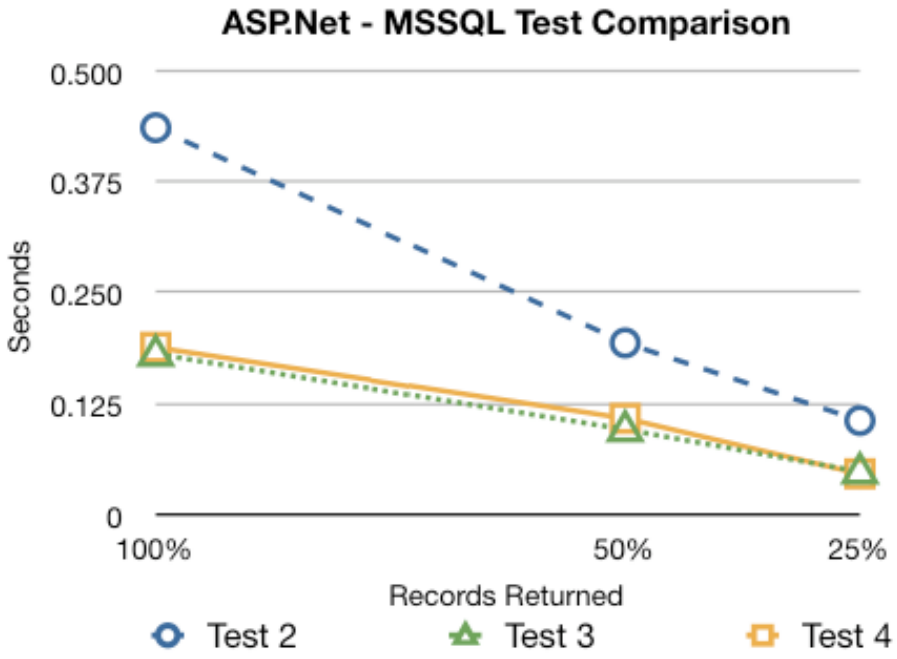
PHP was faster by 20% when accessing mySQL as compared to accessing MSSQL. ASP.Net was in turn faster by 35% when accessing the Microsoft SQL database as compared to the mySQL database, similar to tests 2 and 3 above.

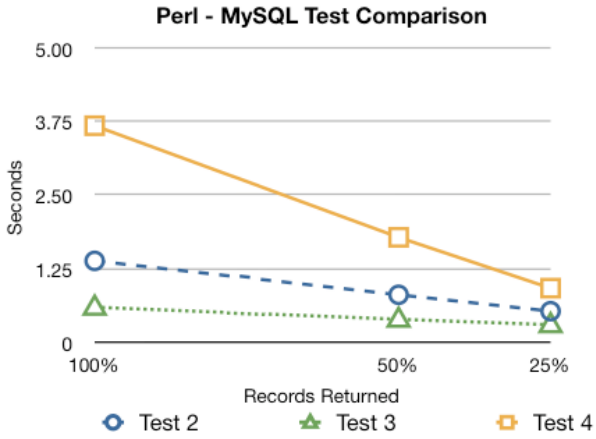


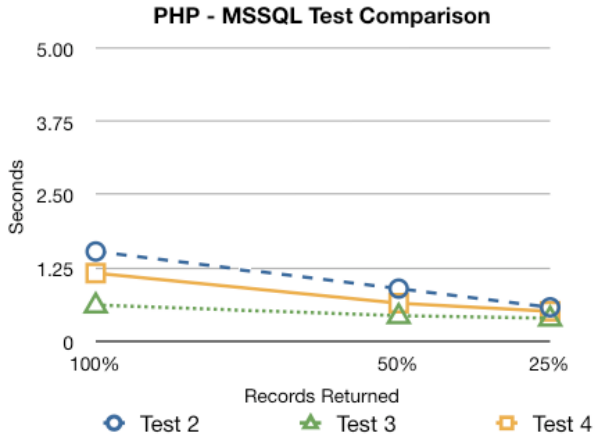
For the hot web server scenario, ASP.Net is the fastest of the three languages in this test. Two tables are merged using a JOIN command in the SQL database and 10,000 records are returned to the script. ASP.Net was faster than Perl by 95% and PHP by 84% when accessing MS SQL and by 93% and 68% respectively when accessing mySQL.

PHP was faster by 30% when accessing mySQL as compared to accessing MSSQL. ASP.Net was in turn faster by 29% when accessing the Microsoft SQL database as compared to the mySQL database.

Verification of test cases







Time taken in seconds to retrieve the data from the databases

Another set of measurements were taken to determine if the results of the tests were linear with varying lengths of the data-sets returned by the two databases. The graphs below show the linear nature of the test results when the tests were run for 50% and 25% of the records retrieved from the database.

Conclusion

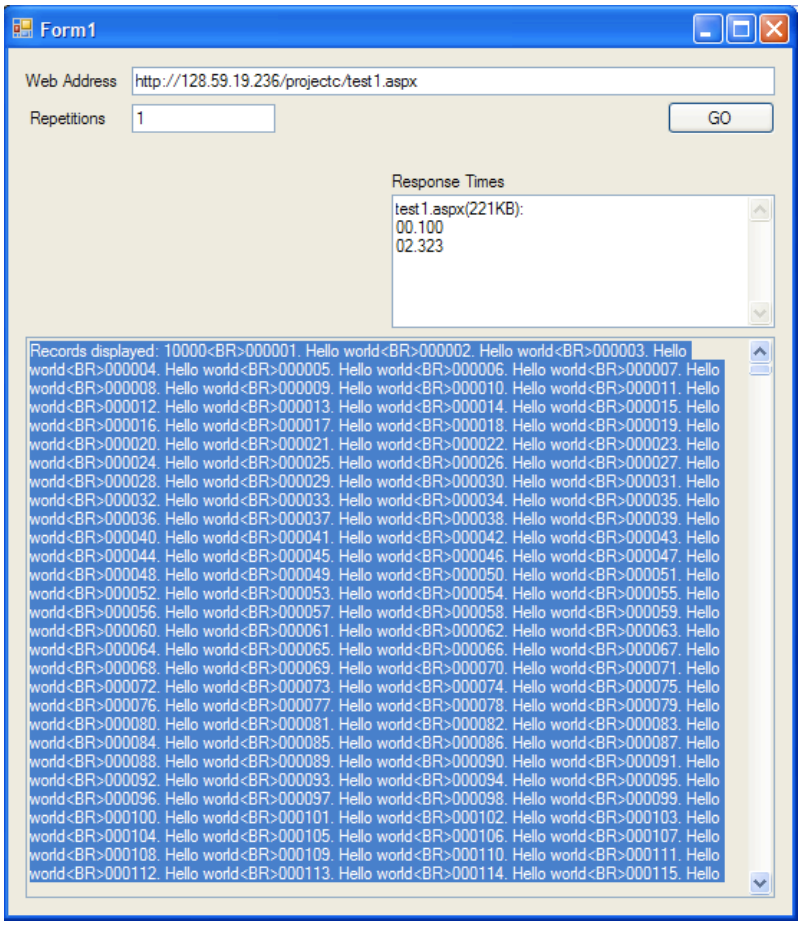
For the cold web server case, PHP was the fastest in displaying large amount of text that was not pulled from a database, while ASP.Net was the second placed language. In the database access and quick-sort tests, PHP was the fastest with ASP.Net and Perl sharing the second places. In the two SQL databases used for testing, MS SQL always performed better when accessed from ASP.Net, while mySQL performed better when accessed from PHP. There was no definitive difference in performance between the two databases when using Perl.

For the hot web server case, 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.

When comparing the hot and cold test scenarios, caching improved the performance of ASP.Net by the highest margin, PHP had a lower performance gain while Perl appeared to be unaffected.

Project Technical Details

Benchmarking Tool



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["StudentDBConnectio
                        nString"].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["mySQLStudentD
                        BConnectionString"].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>";
}
```

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 two 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 to the webpage:

```
for ($i=0; $i < $num; $i++) {
   printf("%06s. Hello world<BR>\n",   $i+1);
}
```

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);
```

The query is then 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 that is 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. This left an exception that was displayed on the web pages rendered by PHPthat 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 is a Windows application which can be used to configure all the aspects of the mySQL installation, along with creation and management of 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')
```

A modified command is used in testing linearity of the test results, “limit N” 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')
```

A modified command is used in testing linearity of the test results, “top N” is inserted into the SQL command, for example:

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

Task List

Installation and configuration of IIS web server, Microsoft SQL Server 2005 and MySQL database server. Installation of database connection drivers for the SQL servers, including ODBC, ADODB, MySQL and MS SQL connection drivers. .Net framework, Active PHP and Perl installation, and configuration of IIS to serve PHP and Perl generated web pages. Setup and configuration of version control through TortoiseSVN/Subversion. Write the ASP.Net script to fill databases with randomized data. Test and analyze data for PHP, Perl, and ASP. Net. Develop the Benchmarking Application.

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/

7. ”MySQL 5.0 Reference Manual”, http://dev.mysql.com/doc/refman/5.0/en/connector-j-reference-configuration-properties.html