Efficiency of Web Based SAX XML Distributed Processing

R. Eggen
Computer and Information Sciences Department
University of North Florida
Jacksonville, FL, USA

A. Basic
Computer and Information Sciences Department
University of North Florida
Jacksonville, FL, USA

Abstract: Given the interest and need for Web processing and programming of distributed systems, researchers are exploring methods and techniques for facilitating programming of such systems. In this paper we explore the performance characteristics of XML processing using SAX for scripting languages PERL and PHP as compared to C (CGI) and Java (Servlets). We use SAX to process XML files via a Web based interface in each of the four environments.

Keywords: PERL, PHP, Java Servlets, C CGI, Distributed Parallel Processing

1 Introduction

Distributed parallel processing belongs to one of the most important areas of research in our modern computing world. From the Web to large scale numerical algorithms, distributed and parallel methods dominate distributed processing applications.[1] It is important for researchers and software engineers to have a wealth of tools available to them for programming such distributed systems. In this paper, we explore the efficiency of extensible markup language (XML) based parallel communications and processing of common Web languages. We consider practical extraction report language (PERL), PHP, common gateway interface (CGI) with C and Java Servlets. We compare the speed and efficiency of these vehicles to demonstrate the efficiency of each implementation of Simple API for XML (SAX) to distribute data and determine how quickly each processes data after the distribution. This study is highly relevant for a typical Web based communication and corresponding processing.

2 Fundamentals

A distributed system is a collection of independent computers interconnected by a network and capable of cooperating to achieve the solution of a problem. Cooperation is required by any Web access. The computers are considered loosely coupled if they do not share memory and have individual processors. Important programming strides have been made in recent years to provide applications programming environments that make distributed computing more common and easier to implement than in the past. The message passing interface (MPI), parallel virtual machine (PVM), Common Object Request Broker Architecture (CORBA), Java’s Remote Method Invocation (RMI), PERL’s SOAP interface, and others have extended ordinary languages to provide methods, functions, or procedures facilitating distributed computing. Web based computing is now common. PERL, PHP, CGI and Servlets are typical tools used in this environment. Communication and processing occur when Web sites are
accessed in a variety of ways, some of which are distributed processing applications.

In this paper, we study the basis of Web distributed communication over a network by processing XML files that distribute a set of integers to a variety of nodes in a network. The nodes process their data, ultimately returning the processed data to the client. We use PERL, PHP, CGI, and Servlets with SAX to process the XML files.

3 PERL

PERL is an open source, cross-platform programming language capable of performing a wide variety of applications. PERL is used in both the public and private sectors and is supported by UNIX, Macintosh, Windows, Linux and many more operating systems. PERL comprises the best features of many languages, including C, awk, sed, sh and BASIC. PERL works with SAX, HTML, and other markup languages to support Unicode. PERL supports both procedural and object oriented paradigms. PERL is extensible. Because of its excellent text processing capabilities, PERL is perhaps the most widely used Web programming language. [2][3]

4 PHP

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. [4]

PHP is an HTML-embedded scripting language. The syntax is very similar to that of C, Java and Perl with a couple of unique PHP-specific features thrown in.

The goal of the language is to allow Web developers to write dynamically generated pages quickly.[5] PHP is a complete programming language unlike the hypertext markup language.

5 CGI

CGI is a standard way for a Web server to pass a Web user's request to an application program and to receive data which is forwarded to the user. When the user requests a Web page (for example, by clicking on a highlighted word or entering a Web site address), the server sends back the requested page. However, when a user fills out a form on a Web page and sends it in, it usually needs to be processed by an application program. The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. CGI defines this method of sending data back and forth. It is part of the Web's Hypertext Transfer Protocol (HTTP). [6]

6 SERVLETS

JavaServer Pages (JSP) is used to dynamically generate HTML information presented on the World Wide Web via XML.

The JSP syntax adds additional XML tags, called JSP actions, to be used to invoke built-in functionality. Additionally, the technology allows for the creation of JSP tag libraries that act as extensions to the standard HTML or XML tags. Tag libraries provide a platform independent way of extending the capabilities of a Web server. The term Web server can mean one of two things:
1. a computer responsible for serving Web pages, mostly HTML documents, via the HTTP protocol to clients, mostly Web browsers;
2. a software program that is working as a daemon serving Web documents.

Every Web server (sense 1) is running a Web server program (sense 2).

The most common Web or HTTP server programs are:

- Apache HTTP Server from the Apache Software Foundation
- Internet Information Server (IIS) from Microsoft
- Zeus Web Server from Zeus Technology
- Sun ONE - Sun Microsystems

JSPs are compiled into Servlets. The Java Servlet API allows a software developer to add dynamic content to a Web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets are the Java counterpart to dynamic Web content technologies such as CGI. However, unlike CGI, (but like PHP), it has the ability to maintain state after many server transactions. This is done using HTTP Cookies, session variables, or URL rewriting.

A JSP compiler is a program that parses JavaServer Pages (JSPs), and transforms them into executable Java Servlets. A program of this type is usually embedded into an application server and run automatically the first time a JSP is accessed, but pages may also be precompiled for better performance, or compiled as a part of the build process to test for errors. [7]

7 SAX

Each of the technologies were implemented using the Simple Application Programmer Interface for XML (SAX). SAX was the first widely adopted API for XML in Java, and is a “de facto” standard. The current version is SAX 2.0.1, and there are versions for several programming language environments other than Java including PERL, C and PHP (as used for this paper). [8]

We considered each of these technologies, implemented them on a Beowulf cluster and measured their relative performance both in terms of communication alone as well as data processing. The project is Web enabled creating a very friendly interface.

8 Hardware and Operating System

The hardware consists of a Beowulf cluster of computers all running RedHat linux v9. The machines are 0.5 GHz machines with 512 megabytes of main memory connected by gigabit fast ethernet.

9 Programming a Distributed System

Programming a distributed system, regardless of the programming environment used, involves several fundamentals. Typically, one of three scenarios are used: boss/worker where the boss distributes a portion of the work uniquely, worker crew where each processor does essentially the same
processing on different portions of the data, often referred to as single instruction multiple data (SIMD), and a pipeline where each worker processes a portion of the data, passing that partial solution to the next worker. In this research we use the boss/worker environment as shown in Figure 1. A boss process defines the task to be accomplished and decides the method of distribution. Several worker processes do the actual work. The worker starts by listening to the socket for the arrival of work from the boss. The boss is the controlling process that has the responsibility of dividing the task appropriately. A portion of the task is sent to each of the workers. The boss waits to receive the completed tasks. Each of the workers execute in parallel. They each receive their task from the boss without interaction with other workers.

![Figure 1](image)

**Communication Dependency**

### 10 Evaluation

The purpose of this research is to determine the relative efficiencies of each of the technologies for a Web based distributed application. We chose a simple sorting algorithm. Sorting is well understood and ubiquitous in computing and Web queries. We programmed an n^-sort so reasonable time would be taken on each of the server machines, creating a measure of both communication and computation times. Boss and workers are programmed using PERL, PHP, CGI and Java, as previously explained. Each application is implemented with exactly the same functionality. The boss machine creates sets of integers whose cardinality vary between 2 and 128,000. Each set was then evenly divided and distributed among the worker machines. The 2 measures just communication time since exactly one integer is sent to each worker node. Each worker sorts their portion of the data, yielding a highly parallel solution. The communication from boss to worker to boss was handled by creating an XML file, processing the file, communicating with the workers, performing the work, and returning the results.

As is often the case, the boss did not participate in the primary task, but rather distributed the work and waited for results. Timings included dividing and distributing the data, sorting each data set, receiving the sorted results from each worker, and merging to arrive at a totally sorted set of data.

### 11 Results

Datasets of sizes 2–4-8, 500, 1000, 2000, 4000, 8000, 16000, 32000, 64000, and 128000, were distributed, sorted, and timed. Since the machines used in this experiment were not dedicated to the researchers, times were taken at 3:00AM when network traffic and users were at a minimum. These machines are lightly used, so unplanned impact on communication is virtually nonexistent. The authors have carefully checked the algorithms, each is correct and implemented in a consistent manner. The
The results for the complete processing are in Table 1. The results for processing just the XML file using SAX are shown in Table 2.

<table>
<thead>
<tr>
<th>DATA SIZE</th>
<th>PHP Total</th>
<th>PERL Total</th>
<th>CGI Total</th>
<th>JSP Total</th>
<th>PHP Total</th>
<th>PERL Total</th>
<th>CGI Total</th>
<th>JSP Total</th>
<th>PHP Total</th>
<th>PERL Total</th>
<th>CGI Total</th>
<th>JSP Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,4,8</td>
<td>0.02</td>
<td>0.06</td>
<td>0.02</td>
<td>0.02</td>
<td>0.05</td>
<td>0.11</td>
<td>0.03</td>
<td>0.04</td>
<td>0.17</td>
<td>0.16</td>
<td>0.07</td>
<td>0.07</td>
</tr>
<tr>
<td>500</td>
<td>0.06</td>
<td>0.11</td>
<td>0.20</td>
<td>0.05</td>
<td>0.08</td>
<td>0.13</td>
<td>0.05</td>
<td>0.07</td>
<td>0.23</td>
<td>0.17</td>
<td>0.19</td>
<td>0.10</td>
</tr>
<tr>
<td>1000</td>
<td>0.12</td>
<td>0.18</td>
<td>0.06</td>
<td>0.09</td>
<td>0.13</td>
<td>0.19</td>
<td>0.07</td>
<td>0.11</td>
<td>0.15</td>
<td>0.23</td>
<td>0.10</td>
<td>0.13</td>
</tr>
<tr>
<td>2000</td>
<td>0.24</td>
<td>0.33</td>
<td>0.14</td>
<td>0.20</td>
<td>0.23</td>
<td>0.32</td>
<td>0.19</td>
<td>0.18</td>
<td>0.26</td>
<td>0.34</td>
<td>0.19</td>
<td>0.22</td>
</tr>
<tr>
<td>4000</td>
<td>0.65</td>
<td>0.78</td>
<td>0.16</td>
<td>0.55</td>
<td>0.48</td>
<td>0.62</td>
<td>0.27</td>
<td>0.39</td>
<td>0.46</td>
<td>0.61</td>
<td>0.25</td>
<td>0.36</td>
</tr>
<tr>
<td>8000</td>
<td>2.11</td>
<td>2.35</td>
<td>1.13</td>
<td>1.92</td>
<td>1.30</td>
<td>1.53</td>
<td>0.11</td>
<td>1.11</td>
<td>0.96</td>
<td>1.20</td>
<td>0.15</td>
<td>0.77</td>
</tr>
<tr>
<td>16000</td>
<td>7.83</td>
<td>8.31</td>
<td>7.01</td>
<td>7.50</td>
<td>4.26</td>
<td>4.67</td>
<td>3.16</td>
<td>3.85</td>
<td>2.61</td>
<td>4.00</td>
<td>1.12</td>
<td>2.20</td>
</tr>
<tr>
<td>32000</td>
<td>31.69</td>
<td>33.02</td>
<td>29.10</td>
<td>30.96</td>
<td>15.78</td>
<td>16.59</td>
<td>14.04</td>
<td>14.94</td>
<td>8.48</td>
<td>9.51</td>
<td>8.54</td>
<td>7.70</td>
</tr>
<tr>
<td>64000</td>
<td>154.90</td>
<td>157.45</td>
<td>153.16</td>
<td>152.95</td>
<td>63.70</td>
<td>65.61</td>
<td>60.14</td>
<td>62.21</td>
<td>31.47</td>
<td>32.96</td>
<td>28.01</td>
<td>29.87</td>
</tr>
<tr>
<td>128000</td>
<td>805.82</td>
<td>795.39</td>
<td>796.42</td>
<td>794.56</td>
<td>307.82</td>
<td>314.07</td>
<td>303.50</td>
<td>305.38</td>
<td>128.03</td>
<td>130.30</td>
<td>121.17</td>
<td>123.80</td>
</tr>
</tbody>
</table>

Table 2: Time for SAX to Process the XML file

<table>
<thead>
<tr>
<th>DATA SIZE</th>
<th>TIME IN SECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>500</td>
<td>0.03</td>
</tr>
<tr>
<td>1000</td>
<td>0.06</td>
</tr>
<tr>
<td>2000</td>
<td>0.11</td>
</tr>
<tr>
<td>4000</td>
<td>0.22</td>
</tr>
<tr>
<td>8000</td>
<td>0.45</td>
</tr>
<tr>
<td>16000</td>
<td>0.89</td>
</tr>
<tr>
<td>32000</td>
<td>1.78</td>
</tr>
<tr>
<td>64000</td>
<td>3.73</td>
</tr>
<tr>
<td>128000</td>
<td>7.17</td>
</tr>
</tbody>
</table>

12 Summary and Conclusions

Surprisingly, all four technologies processed the data in similar times. Interestingly, each scaled very nicely. When the number of processors doubled the time decreased by nearly half. This implies that processing time dominated, as we would expect. The first row of Table 1 essentially sends one integer to each node, thereby timing only the communication. When two workers were present, two integers were sent, one to each worker. When four workers were present, four integers were sent, again one to each node and similarly for eight workers. When more worker nodes are present, additional time is required since communication is between more computers. PERL and PHP ran slower in this area, which is somewhat surprising. PERL and PHP do not process data quickly as determined by the author’s experiments in other research, but normally established communication quickly.
Interestingly, each technology processed large amounts of data at nearly the same rate. This, too, is a surprising result.

Table 2 shows the time to process the XML files. Here we see a major discrepancy in timing among the technologies. CGI ran approximately 30 times faster as did JSP until large amounts of data were present. This indicates that for general Web processing where little data manipulation is involved, CGI and JSP are preferable over PERL or PHP. I find these results very interesting and not what I would have guessed.

13 Future Research

There are emerging technologies for Web processing including C# and the .net framework. Since Java’s JSP and the older technology of CGI performed well, an interesting study would be to compare them to the .net environment.

14 References


