AN ANALYSIS OF USING WEB APPLICATION AS DISTRIBUTED COMPUTING PLATFORM

Parin Chiamananthapong, Krerk Piromsopa
Chulalongkorn University
Bangkok, Thailand
Parin.Ch@Student.chula.ac.th, Krerk.P@chula.ac.th

ABSTRACT
With the emerging in HTML5 and JavaScript, gap between web application and legacy desktop applications have been reduced; making this platform a suitable choice for distributed computing. In this paper, we present a survey of latest web technologies with the focus on performance improvement for distributed computing platform and use web browser as frontend client agent. We present an analysis of tools and compare to existing platform. Finally, performance evaluation has been performed. Several JavaScript features are gathered to benchmark the performance against that of the native compiled applications. The results show that performance achieved in web application is comparable to that of native applications. This platform eliminates the need to install additional software. It lowers the barrier of entrance. Clients can join by just simply opening a specific web page. In addition, it also gets the web application advantages including secure sandbox environment, instant deployment with platform independence and large user base.

KEY WORDS
Distributed Computing Systems, Web Technologies, Desktop Grid, Parallel Computing Systems

1. Introduction
With the emerging in HTML5 and JavaScript, we want to prove that web technologies have been developed to the point that making it suitable for high performance computing.

Desktop grid (aka. volunteer computing) is another powerful architecture. It is similar to traditional grid systems. However, the participants are internet-connected home desktops. The most successful project is SETI@home[1]. The SETI@home project uses BOINC[2] framework. BOINC is designed to grab the large amount of spare processing power from massive number of internet connected computers around the world. However, it requires that each client must install dedicated software, which can raise some compatibility and security issues. Although all problems mentioned above can be solved by using virtual machine based desktop grid like Entropia[3]. It still needs a client to install software. This can discourage some participant. In order to lower the barrier of entrance for participants, we use web browser, the most users’ familiar platform that every computer or even a mobile device has it. With this approach, we can get the profit of the web application advantages, which are instant deployment, maintenance, and sandboxed environment. The participant does not need to install software or to perform additional work to participate. The only thing that a client needs to do is simply browsing a specific web page. In addition, participants can terminate the fail process or leave the computation anytime by closing the web browser without worrying about the damage to the system.

With the portability of web applications and performance that comparable to native application, web applications have been used in other fields. For example, PhoneGap[25] use HTML5 for building mobile applications, mobile cloud computing application using HTML5[26], computing system with cloud-based browser[27] and web search using browser-based volunteer computing[28].

This paper makes the following contributions:
• We present a survey of tools in a web application that enables implementing a distributed computing platform. Those tools are mainly designed for improving performance. We analyze and compare their characteristics.
• We present an analysis of using a web application as distributed computing platform by comparing to existing platforms. Using web for computing has several advantages. However, there are some limitations and trade-off. We compare and show issues between web system and legacy system.
• We perform two benchmarks and discuss their performance. The first benchmark is designed to compare the JavaScript performance to other computer programming languages. The other benchmark is for comparing the performance of JavaScript tools. We provide the analysis of each tool.

The rest of paper is organized as follow. Section 2 is the related works. Section 3 is an introduction to JavaScript. Section 4 discusses tools for distributed computing in JavaScript and compares of web platform characteristics against traditional desktop grid. The performance evaluation and discussion are in section 5.
and 6 respectively. Finally, our conclusions are in section 7.

2. Related Works

2.1 Desktop Grid / Volunteer Computing

Desktop grid (aka. volunteer computing) is a distributed computing platform which aggregates a large number of volunteered internet-connected personal computers. This paradigm mostly used in academic or scientific research applications. There are many desktop grid framework including BOINC and Entropia. Most of them share similar concepts. They employ client-server architecture, which the client is the participant downloading and installing the agent program in their computer. The client will fetch tasks from the server, execute it, and report the completed task. Clients are usually behind NAT or firewall. To communicate, the “pull” model is applied. In this model, clients periodically request for a task from the server. The “push” model, which the master assign task to the specific worker, may not be practical.

Unlike traditional Grid systems, there are two issues related to clients in volunteer computing. First, the client is anonymous, not linked to any real world identity. Malicious clients can reverse engineer the program and send the faulty results in an attempt to sabotage or challenge the server. Second, volunteers are unreliable, because clients may join or leave anytime. Most projects solve this problem by having redundant results per task and validate the results by applying majority voting method to find a consensus result.

2.2 Browser Computing

The term “Browser Computing” (aka. Browser-based distributed computing) as defined in this paper is the distributed computing system that most computational task is performed on the client-side web browser. In particular, browser computing is based on Bag-of-Tasks just like a desktop grid or volunteer computing. The difference between browser computing and desktop grid is that the agent programs used by browser computing is the web browser, which is installed in every computer.

There exist research and projects that use this idea. Luis F.G. Sarmenta and Satoshi Hirano propose Bayanihan[4], a Java applet embedded web-based volunteer computing systems. This system allows developer to build web-based volunteer computing, and client can participate by using a browser that supports a Java applet. This system features eager scheduling, scaling and result validation.

Later, when AJAX is introduced, the web becomes more interactive. AJAX is a technique for a client side that allows an asynchronous request to a server in the background. With AJAX, an application can run continuously without the need for user to press reload button to refresh the entire page. This provides a new way to develop browser-based distributed computing by using JavaScript as the main programming language. This method has been adopted in many applications, such as genetic algorithm[5], genetic programming[6], and evolutionary computing[7]. In [8], Fumikazu Konishi et al. have present RABC, the system that adopt the CMS(content management system) techniques for managing application scripts making flexible for general purpose applications. However, the performance in all works mentioned above is not well. This is because JavaScript performs poorly, and web applications have several limitations comparing to traditional desktop application in the past. However, the environment is difference nowadays. With the increasing performance of the JavaScript and the upcoming new web technologies like HTML5, JavaScript2.0, and WebSocket, the gap between web application and legacy application is reduced. Nowadays, we can develop a web application, which functions and performs similar to that of conventional desktop application.

There are some previous projects that explored about using HTML5 for browser computing. For example, project GridBee[29], which is JavaScript client for BOINC. Another project is using browser-based volunteer computing for web search[28]. These projects encourage the use of web application as distributed computing platform.

3. JavaScript

Embedded in a browser, JavaScript is the main language in web application. JavaScript is a dynamic scripting language with weakly typed. From [9], main characteristic of JavaScript can be describes as portable, rapid development and safe.

Portability is an ability that same software can execute correctly in different environment. JavaScript can write once and run in many web browsers running on different platforms, operation systems and devices. This eliminates the need to recompile the script to a specific platform or to install additional runtime environment.

JavaScript is considered to be a multi-paradigm, which can support many programming style such as object-oriented, imperative and functional. With this paradigm combine with the benefit from dynamic type language, JavaScript is suitable for rapid prototyping. In several cases, JavaScript was often used to enhance user experience and to validate form input. However, the recently improvement of JavaScript engine such as V8 and SpiderMonkey, makes the performance of JavaScript increase exponentially. Nowadays, JavaScript can be used for computation in web application like image processing, graphic rendering and sound processing. Moreover, it has been used in many applications outside of web browser field. For example, Node.js uses V8 engine for light-weight network application framework. Riak and CouchDB use JavaScript for query language in NoSQL database, and Unity game engine use it for game scripting.
Safety is the most important requirement for web application. Unlike traditional desktop application, which users must secure it manually, web application can run instantly when a page is opened. To ensure the security of the user computer, JavaScript uses the sandbox environment, which prevents damage from occurring to physical system.

4. Distributed Computing in JavaScript

In this section, we will discuss tools for distributed computing in JavaScript and compare of web platform characteristics against traditional desktop grid.

4.1 Web Browser Tools for Distributed Computing

The tools for distributed computing in JavaScript can be classified into two groups. First group is the standard features in modern web browser. The items in this group are typed array and web worker. The latter are features that depend on some browser, the items in this group are Google Native Client, WebCL and River Trail. This group trades performance with portability and security.

Typed array[10] is a recent improved array that have structure like normal array in C. Typically, normal JavaScript arrays are not typed. An array may hold anything such as objects, function or even other arrays. This flexible structure adds the overhead when dealing with the arrays of typed data (e.g. array of integers). With the use of typed array, data can be calculated and directly accessed in without the need to lookup and conversion.

Web worker[11] is the threading model specification for JavaScript. In particular, JavaScript is single-threaded with support for asynchronous request (AJAX) to perform background HTTP request. However the thread handling the computation is also the same thread that handles the user interface. It can make the browser unresponsive, when heavy computation occurs blocks the script. Web worker can spawn background worker threads in a web browser. These workers can perform task, I/O or AJAX request without interrupting the main user interface thread. Workers can communicate with the main thread via message passing API.

Native Client[12] (abbr. NaCl) is a sandbox technology developed by Google to execute native code with full privileges of the user. Moreover ActiveX and Silverlight can be used only in Microsoft platform with Internet Explorer web browser. NaCl is similar to JAVA applet. The difference between them is that JAVA applet uses JAVA while NaCl uses C++ as programming language.

WebCL[13] is a JavaScript wrapper API for OpenCL[14], which is a framework for writing parallel programming in heterogeneous platforms. It enables web application to utilize multi-core CPU or GPU to accelerate the performance. WebCL use JavaScript to express the API and C++ for kernel. Currently, WebCL is still in early stage and no natively support in any web browser. However, there are some WebCL implementation like Nokia[15] for Mozilla Firefox plug-in and Samsung[16] for webkit-based browser.

River Trail[9] is an open source JavaScript parallel programming engine. It is developed by Intel and based on Intel OpenCL driver. River Trail provides a high-level data parallel abstraction with ParallelArray and functionality like map, reduce, scatter and so on. Additional, the kernel function can be written in JavaScript entirely, which make it easier and more secure to implement parallel programming without the need to manage the resource manually.

From all of the features mentioned above, we can summarize within the table 1.

4.2 Comparison between Web and Legacy Desktop Grid

In this section, we will briefly compare the issues between browser computing and traditional desktop grid. We choose BOINC framework for representing legacy desktop grid. In fact, the concept and assumption of BOINC is fundamentally different from browser computing. For example, BOINC harvests untapped CPU cycle for computing when computer is idle. This idea, however, cannot be applied in web application. This is because of the sandbox environment prevents the web from accessing and monitoring CPU activity. In contrast, it let the client kindly open the specific web page to join the computing when they use Internet. However, BOINC is the well-known framework for desktop grid. It is open source and well documented, making it easier to study. The details are summarized in table 2.

We will analyze the following aspects: performance, security, available resources, portability, protocol and use case.

In term of performance, BOINC can achieve more performance than JavaScript, because it is a native application. It can utilize the local resource like GPU to
Table 1: Summary of web application technologies

<table>
<thead>
<tr>
<th>features/tools</th>
<th>typed array</th>
<th>web worker</th>
<th>Google NaCl</th>
<th>WebCL</th>
<th>river trail</th>
</tr>
</thead>
<tbody>
<tr>
<td>detail summary</td>
<td>an improve array for store typed variable.</td>
<td>threading model for web application</td>
<td>a sandbox technology that can execute native code within the web browser</td>
<td>a JavaScript wrapper of OpenCL for heterogeneous hardware parallel programming</td>
<td>a JavaScript parallel engine which provide high-level abstraction and data-parallelism for multi-core CPU</td>
</tr>
<tr>
<td>type and dependency</td>
<td>browser native</td>
<td>browser native</td>
<td>specific to Google Chrome web browser</td>
<td>- plug-in - OpenCL driver - WebCL plug-in - Mozilla Firefox (Nokia implementation) - Webkit-based browser (Samsung implementation)</td>
<td>- plug-in - Intel OpenCL driver - River Trail plug-in - Mozilla Firefox</td>
</tr>
<tr>
<td>safe execution</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>develop programming language</td>
<td>JavaScript</td>
<td>JavaScript</td>
<td>C/C++</td>
<td>C++ (kernel), JavaScript (WebCL API)</td>
<td>JavaScript</td>
</tr>
<tr>
<td>hardware utilization</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>CPU, GPU</td>
<td>CPU</td>
</tr>
<tr>
<td>thread model</td>
<td>no</td>
<td>yes</td>
<td>yes (pthread)</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>vendor/standard</td>
<td>Khronos</td>
<td>W3C, WHATWG</td>
<td>Google</td>
<td>Khronos</td>
<td>Intel</td>
</tr>
</tbody>
</table>

accelerate the performance (multi-core CPU is dealt by the BOINC Manager, a client program for BOINC platform, which support multi-core CPU by assigning work units to each core).

For security and available resources, unlike native application execute with full privilege of user, the web has a sandbox model ensure that the damage would not spread out when program crash. Nevertheless, web cannot access GPU or multi-core CPU without any plug-in. Natively, web also cannot store file in client directly. Until recent advance in HTML5 specification, localStorage[17] is introduced. However, each browser has different storage size limitation varying from 2 MB up to 5MB. The different in performance between web application and native application will be evaluated further in the next section.

With high portability, a volunteer project can easily extend the user base. In BOINC, developer must prepare specific compiled program for each platform, while web get instantly deployment. Developer creates application in JavaScript and it can run in any modern web browser without the need to recompile. To port from existing applications, BOINC can use existing program via BOINC wrapper[18]. Alternatively, Emscripten[19] allows existing application (e.g. C++ program) to be compiled from source code to LLVM. The LLVM is then converted to JavaScript. This way, exist application can be ported to run as JavaScript. Another approach is to use Google NaCl. However, it limited only to Google Chrome web browser.

Regarding communication protocol, BOINC use almost the same mechanism to communicate between the client and server. It uses HTTP protocol. However, p2p can be applied only in BOINC. There are some suggestions that using p2p to download input and program file to reduce the server traffic[20]. The only p2p implementation for the web is webRTC[21], which is still in an early stage.

The final issue is the use case. BOINC is mostly used in scientific and mathematical research. These applications require high computing power. So, the performance is an important issue. Most projects have skilled staffs, which make the platform specific optimization a minor problem. Benefit from web application, such as rapid development is insignificant in BOINC use case. We think that our approach does not replace the existed systems, but be an alternative way for implementing distributed computing platform for more
<table>
<thead>
<tr>
<th>key attributes</th>
<th>BOINC</th>
<th>web application</th>
</tr>
</thead>
<tbody>
<tr>
<td>barrier of entrance</td>
<td>Client manually install program, choose project and configuration resource.</td>
<td>Clients open the specific web page.</td>
</tr>
<tr>
<td>execution environment</td>
<td>native application compiled to specific platform</td>
<td>web browser</td>
</tr>
<tr>
<td>storage</td>
<td>local file system with user defined in size</td>
<td>- localStorage (2 – 5 MB)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- file API (exclusive to Google Chrome, Safari)</td>
</tr>
<tr>
<td>developing language</td>
<td>C/C++ (default)</td>
<td>- JavaScript</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- C/C++ with Google NaCl</td>
</tr>
<tr>
<td>deployment</td>
<td>Client manual install BOINC Manager</td>
<td>instantly deployed with platform independence</td>
</tr>
<tr>
<td>protocol</td>
<td>- HTTP with client-server architecture</td>
<td>- HTTP with client-server architecture</td>
</tr>
<tr>
<td></td>
<td>- can extend to use p2p or other protocol</td>
<td>- can extend to use p2p with webRTC (early stage)</td>
</tr>
<tr>
<td>hardware utilization</td>
<td>fully supported</td>
<td>- support multi-core CPU and GPU via WebCL (plug-in)</td>
</tr>
<tr>
<td></td>
<td>- multi-core CPU managed by BOINC Manager</td>
<td>- multi-core CPU with River Trail (Intel OpenCL driver)</td>
</tr>
<tr>
<td></td>
<td>- GPU: developer manual create program with GPU supported.</td>
<td></td>
</tr>
<tr>
<td>safe execution</td>
<td>no</td>
<td>yes with sandbox environment</td>
</tr>
<tr>
<td>thread</td>
<td>support</td>
<td>support with web workers</td>
</tr>
<tr>
<td>portable from existed program</td>
<td>BOINC wrapper, API for other programming language(ex. Python, JAVA)</td>
<td>- Emscripten (compile to LLVM then convert to JavaScript)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- C/C++ (Google NaCl)</td>
</tr>
<tr>
<td>user base</td>
<td>computer user who want to contribute their computing power to science project and do not mind to install additional software</td>
<td>normal computer user who want to contribute their computing power to science project and do not want to install additional software</td>
</tr>
<tr>
<td>resource management</td>
<td>can control CPU cycle</td>
<td>cannot control CPU cycle directly. The only way to control manage the resource usage is limited the time to execute code with setTimeout() function[23]</td>
</tr>
<tr>
<td>use case</td>
<td>scientific and mathematical domain</td>
<td>adhoc projects with more relaxed constraints.</td>
</tr>
</tbody>
</table>

Table 3 Execution time of the benchmark program compare between JavaScript and legacy programming languages

<table>
<thead>
<tr>
<th>programming language</th>
<th>execution time (ms)</th>
<th>GFLOPS</th>
<th>ratio (comparing to JAVA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAVA</td>
<td>1061</td>
<td>0.9429</td>
<td>1.000</td>
</tr>
<tr>
<td>C++</td>
<td>1064</td>
<td>0.9398</td>
<td>1.003</td>
</tr>
<tr>
<td>Python</td>
<td>58847</td>
<td>0.0170</td>
<td>55.49</td>
</tr>
<tr>
<td>JavaScript Chrome</td>
<td>1106</td>
<td>0.9042</td>
<td>1.043</td>
</tr>
<tr>
<td>JavaScript Firefox</td>
<td>1351</td>
<td>0.7404</td>
<td>1.273</td>
</tr>
<tr>
<td>JavaScript IE</td>
<td>4654</td>
<td>0.2149</td>
<td>4.388</td>
</tr>
</tbody>
</table>

438
relaxed and less constrained jobs, such as Genetic Algorithm or Monte Carlo Simulation. This platform will benefit from rapid-development and other web features.

5. Performance Evaluation

In this section, we performed two experiments: gigaflops count and matrices multiplication. The first experiment is designed to measure the performance of JavaScript in various web browsers against the legacy computer programming languages. The second experiment is to compare the performance of various web application tools to native application.

5.1 Gigaflops Count

In this experiment, we benchmark the total time used to complete 1 billion floating point instructions. We implement this program in Java, Python, C++ and JavaScript. The test environment is a desktop computer equipped with 3rd generation Intel Core i7 CPU running at 3.40 GHz with 16.00 GB of memory and Windows 7 service pack 1 (32 bit version). For C++, the program is compiled with Microsoft Visual Studio 2010 express with O2 optimization parameter. For Java and Python, we use Java version 1.7.0 and Python 3.3.0 respectively. For JavaScript, we test them in various web browsers. The candidates are Google Chrome 23, Mozilla Firefox 17, and Microsoft Internet Explorer 9.0. Each result is an average time calculated from five executions. The result is shown in table 3. The performance of JavaScript running in Google Chrome and Mozilla Firefox are similar to those of C++ and JAVA. Notice that JAVA is faster than C++ in this benchmark. This happened because JAVA has JIT (Just-In-Time compilation)[22]. It can perform runtime optimization. In particular, it can query the environment it executed and choose appropriate optimization for it. Unlike, C++ uses static optimization, which compiled to native code optimized decently for running on most machines. However, it cannot fully optimized with just one configuration and not knowing the execution environment beforehand. The performance of JavaScript on Microsoft Internet Explorer is roughly 4.0x slower than the performance of JAVA.

Python showed 55x slower than that of JAVA. This is not surprising because Python is a dynamic language and performance is being traded for productivity (ease of use). The result suggested that JavaScript has a potential for distributed computing platform.

5.2 Matrices Multiplication

Matrices multiplication is used in many applications such as image processing, linear algebra and computer animations. We implement this benchmark based on standard algorithm with O(n^3) and use flat array (1D) to represent the matrices. The performances in this benchmark were measure by the total execution time to complete the multiplication of dense-square matrices with size 128, 256, 512, 1024 up to 2048 respectively.

The details of testing environment are stated as follow. The computer we used is the same from previous experiment. The web browsers used in experiment are Mozilla Firefox 17 and Google Chrome 23. To benchmark the speedup of standard technologies in web application, we implement benchmark using normal JavaScript, typed array with Int32Array and web workers with 2 workers. Same goes for the non standard, we choose Google NaCl and River Trail. For comparison the performance with native application, a C++ implementation is used.

The details of testing method are stated as follow. We generate two matrices in JavaScript. Each element is an integer value ranging between -10 and 10. These prevent the value of result of matrices multiplication exceed the maximum value of 32 bit integer. Since the input is randomly generated, there is no network communication factor in this experiment. The time measurement starts when we begin calling matrix-multiplication function. In case of web workers and Google NaCl, it includes the time to pass the matrices data from JavaScript to worker/module. We run the executions five times for each configuration. Omitting the largest and smallest value, we show the performance speedup. This speedup is

<table>
<thead>
<tr>
<th>Tools / Matrices size</th>
<th>128 * 128</th>
<th>256 * 256</th>
<th>512 * 512</th>
<th>1024 * 1024</th>
<th>2048 * 2048</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox JavaScript</td>
<td>54</td>
<td>529</td>
<td>4513</td>
<td>49242</td>
<td>659306</td>
</tr>
<tr>
<td>Chrome JavaScript</td>
<td>40</td>
<td>228</td>
<td>2311</td>
<td>31071</td>
<td>497782</td>
</tr>
<tr>
<td>Firefox typed array</td>
<td>37</td>
<td>235</td>
<td>2638</td>
<td>26465</td>
<td>518578</td>
</tr>
<tr>
<td>Chrome typed array</td>
<td>18</td>
<td>75</td>
<td>590</td>
<td>7069</td>
<td>172573</td>
</tr>
<tr>
<td>Firefox web worker</td>
<td>44</td>
<td>131</td>
<td>1392</td>
<td>16131</td>
<td>274786</td>
</tr>
<tr>
<td>Chrome web worker</td>
<td>18</td>
<td>58</td>
<td>322</td>
<td>4367</td>
<td>90466</td>
</tr>
<tr>
<td>Chrome NaCl</td>
<td>4</td>
<td>36</td>
<td>225</td>
<td>6493</td>
<td>70569</td>
</tr>
<tr>
<td>Firefox river trail</td>
<td>40</td>
<td>49</td>
<td>172</td>
<td>1836</td>
<td>28873</td>
</tr>
<tr>
<td>C++</td>
<td>0</td>
<td>31</td>
<td>172</td>
<td>7005</td>
<td>66691</td>
</tr>
</tbody>
</table>

Table 4 shows the average time for completing matrices multiplication in each configuration. Figure 1 shows the performance speedup. This speedup is
calculated from the ratio of each configuration to the naïve Mozilla Firefox JavaScript (baseline).

Figure 1. Matrices multiplication performance comparison between web application tools

Among the native web application features, web workers tend to be faster than typed array and naïve JavaScript, respectively. It also shows that Google Chrome web browser is faster than Mozilla Firefox in this benchmark. However, C++ outperformed all of them, except the 1024x1024 matrices that it performs poorer than Google Chrome web workers. We assume that the matrices data size is larger than CPU cache, increasing latency of memory access and making the overall speedup ratio down. The speedup of C++ in 128x128 is not shown, because it is too fast to measure in millisecond. Without additional optimization, River Trail performs best. It is not surprising, because it can utilize the multi-core CPU. The speedup is a factor 26.8 and 22.8 in 1024x1024 and 2048x2048 matrices. The performance of Google NaCl is comparable to C++. Except in small matrices, Google NaCl performs poorly due to the overhead of passing the data from JavaScript to native client module. Both were performed poorly at 1024x1024 matrices compare to web workers and River Trail. We assumed that the reason is the cache size of the CPU. However, further investigation is beyond the scope of this paper.

From this experiment, we can conclude that the performance of web application is less than that of native application. However, it is a tradeoff for portability, security and rapid development. In order to increase the performance, additional feature, which tradeoff some portability can be applied. For example, Google NaCl performance is comparable to that of native C++.

6. Discussion

6.1 Performance

These are several factors that affect the performance. In this paper, we will exclude the communication factors (like network latency), and focus only factors related to the JavaScript virtual machine. Although the performance of JavaScript is increased exponentially recently, and it outperforms other dynamic programming languages like Python, Ruby and PHP. JavaScript is still 3x slower comparing to C++ in our experiment. Nonetheless, it is being traded with the portability and security. In addition, there are some browser specific features that can make the performance comparable to native application.

Unfortunately, WebCL is not included in our experiment, because the implementation is crash in our testing environment. WebCL can provide more flexible parallelism than River Trail, which use data parallelism and let you configure the size of workers. It can also use hardware other than CPU (e.g. GPU). However, River Trail is more secure than WebCL, because developer does not need to deal with memory and device manually, so it is free from data races and deadlocks. The other advantage of River Trail is functionality. From our experiment, it took around 100 lines of code to implement the matrices multiplication in WebCL, while it took around 10 lines of code in River Trail. Finally, according to the Intel River Trail paper[9], we can further optimize the performance by the use of nested array (2D) to represent matrices.

Finally, here are some guidelines for optimizations.

- Always use web worker. Although, an application may not be multithreaded. It can prevent the web from being unresponsive.
- Use typed array when it is possible. Typed array can be used to deal with array of the same data type.
- Try native client accelerator. To achieve the performance like compiled code, Google NaCl can be used. Considering Google Chrome has large distribution[24], Though WebCL and River Trail sound promising, they are not practical. This is because they require users to install OpenCL driver and their plug-in.

6.2 Security

The technique used is no difference from normal web application, so there are no specific concerns about security in using web browser as a client program. Client is protected from malicious script by the sandbox model, same-origin policy and restricted environment that prevent JavaScript to access local resources. The worst case scenario is there are some greedy developers or unoptimized code, make the application consume more resource than it should. However, this is not critical issue because client can easily solve by closing a tab in a browser with no other harm to their computer. The only major security issue is from the use of the plug-in such as Adobe Flash, Microsoft Silverlight and WebCL. Since they increase the attack surface from the plug-in itself, and most plug-in is updated separately via it own vendor update manager, this makes them not up-to-date with the browsers.
7. Conclusion and Future Work

This paper attempts to exploit web application technologies for distributed computing platform. Unlike traditional desktop grid, this approach eliminates the need to install additional software, lowering the barrier to entry. We discuss their key attributes comparing with traditional desktop grid in several aspects. For performance evaluation, several JavaScript features have been introduced and benchmarked against native compiled application.

The most difference between web and native application is, native application can utilize the hardware and storage as much as user defined. The web, although it cannot use hardware acceleration and have less performance, it gains portability advantage. Client can use web browser, which has been already installed in most devices, to join the computing by simply opening the specific web page. In addition, the secure sandbox environment, instant deployment with platform independence, large user base, and the emerging of HTML5 make this platform an interesting choice for distributed computing.

From the details above, choosing the platform is depends on your project’s need. It is being tradeoff between performance and portability.

Finally, the further research is to develop a framework for browser-based distributed computing system with adapt programming model for rapid development and conduct performance evaluation against with legacy systems.

References