

# Distributed and Grid Computing via the Browser

Kevin Berry

Computing Research  
Department of Computing Sciences  
Villanova University, Villanova, PA 19085  
[Kevin.berry@villanova.edu](mailto:Kevin.berry@villanova.edu)

December 8, 2009

## Abstract

Large scale grid computing has only recently become feasible due to the maturation and widespread use of the internet. A browser-based distributed model shows promise, considering the significant amount of “untapped” processing power available. With “Web 2.0” applications quickly becoming the norm, a distributed platform is practical with in-browser scripting and data transport technologies. Currently, 95% of browsers have JavaScript enabled. The new technologies, such as AJAX (Asynchronous JavaScript), Adobe Flash, Microsoft Silverlight, in combination with the tried and true HTTP protocol, make a browser-based distributed model available to anyone simply browsing the internet.

## 1. Introduction

The goal of distributing a problem is to calculate large amounts of data in a short amount time. Distributing a problem over the internet with the web browser establishes a universal, powerful, and flexible platform. Section 2 deals with the feasibility and implications of the browser-based system. Section 3 goes over the types of problems permitted to be accomplished by such a system. Section 4 goes over the specifics of implementation on the browser and the server side, as well as possible ethical and technical considerations. Section 5 discusses the experimentation of the browser-based model with a simple distributed reverse-hashing algorithm using the MapReduce paradigm.

## 2. The Internet and the Web Browser

The breadth of the internet is enormous. Today, an estimated 1.67 billion people use the internet according to the Miniwatts Marketing Group. In the United States, the percentage of population that uses the internet is a lofty 72.4%. [8] Google recently announced that there are over 1 trillion unique URLs, as well as billions of web pages cached. Google employs a distributed architecture to sort of save this much data [9]. Facebook garners over 200 billion monthly page views, and serves over 600,000 pictures per second. [12]

The Web Browser, like the internet, has come a long way. WorldWideWeb, originally developed in 1991, supported plain text, HTTP (Hypertext Transfer Protocol) and FTP (File Transfer

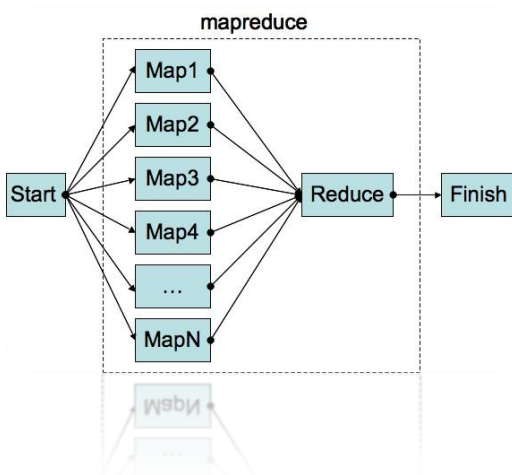
Protocol. [8] Today, browsers support a multitude of protocols and applications, however still revolving around HTTP. Since the development of JavaScript 1995, a browser-side scripting language, client-side applications have become possible. JavaScript is a robust language with support for both functional and procedural paradigms, making it flexible and fitting for a distributed computing project. [3] Today, 95% of browsers support JavaScript, as it has become central to “Web 2.0” applications. [8] There are more yet client-side applications which are fitting for a distributed computing application, such as Adobe Flash, and more recently Microsoft Silverlight. They are usually available through browser plug-ins and extensions. We will focus solely on JavaScript, since it is now considered standard, and thus more universal. Other languages and protocols to be considered are HTTP, HTML, as well as server-side databases and technologies.

### 3. Distributed Algorithms

A browser-based distributed algorithm would likely employ a divide-and-conquer technique on a data set. Data is split into pieces to be computed independently from one another, and finally merged to form a solution. Since pieces have to be computed independently, distributed algorithms only deal with the most easily parallelized of problems to avoid synchronicity issues and dead-lock situations.[7] Synchronicity can be established in a distributed algorithm, but the protocols, programming, and reliability of such a system become extremely complicated on large distributed networks, as demonstrated by a few of the first researched distributed systems, Amoeba and Sprite [1].

#### 3.1 Grid Algorithms

A grid algorithm is a distributed algorithm that takes place over a wide area network (the internet). Due to the relative unreliability and latency of a WAN, the algorithm must be completely asynchronous and independent, or “embarrassingly parallel”. [4] This constraint narrows the problem field further to those of the “data set” model. The MapReduce paradigm, currently employed by Google for a cluster of over 100,000 nodes, is the de-facto system for distributing large problems. The cluster is currently capable of computing over a petabyte of data in a few hours with ordinary machines. [6] Nodes can be added or dropped instantly, which happens to be a crucial requirement for a browser-based model.



MapReduce (left) consists of 3 simple steps. The established problem is set up. The first phase, “map” splits up the data into smaller pieces. Each one of those pieces is sent to a node to be computed. The node can then further “map” the data and send it elsewhere, resulting in a tree-like hierarchy. During the “reduce” phase, the computations are sent back to the parent node, and eventually make their way back to the master node. The data is re-assembled by the master node and

the problem is complete. [6] For a browser-based distributed problem, the map phase considered will only be 1 level: server-client.

#### **4. The Internet and Web Browser/JavaScript as a Distributed Platform**

Due the widespread availability of the internet, as well as the functionality of the browser as an application platform, a possible distributed platform with enormous computational potential arises. The master node would be a standard HTTP web server with CGI capabilities. Each distributed node would simply be a JavaScript-enabled web browser. The computational power of the algorithm is proportional to the number of users and page clicks.

##### **4.1 Implications of Volunteer Computing**

Many grid applications run on a volunteer model where a user agrees to donate resources towards a problem. In a browser based model, this agreement could be breached by having computations performed “invisibly” on anyone’s computer who clicks on a site. There is an ethical issue involved in using people’s computing power, possibly against their will. However, by clicking on a website, one agrees to run any JavaScript code as long as they have it enabled. By keeping computation units minimal, a very small amount of resources and bandwidth are used. The HTTP POST limit is 1kb, which is a minute amount of data to be computed and sent/received by today’s standards. However, the computations themselves may take a long time.

A possible off-shoot of the volunteer model, and a solution to the data-verification issue, would be to employ a “log-in” system common to many websites today. The algorithm may run in the background by employing TLS/SSL. With this situation, user-specific statistics could become available, much like Folding@Home, and sensitive data would be protected by letting trusted users contribute data. This would most likely alleviate some of the concern for false data being reported, but data verification would still come down to trust. However, a log-in based system would eliminate a substantial user-base.

##### **4.2 Security and Data Verification**

Like most distributed computing projects, there is little to no protection against nodes contributing false data. Data could also be sent from outside of a web browser. The most important point to consider is the fact that JavaScript is easily reverse-engineered. JavaScript source code is sent to the browser in plain text. Thus, the inner workings of the algorithm are available to anyone who runs it. Currently, there are obfuscators that aim to make JavaScript difficult to read so applications can’t be reverse engineered, but this protection is very weak, and the code is still readily available to anyone who wants to read it. [2]

Furthermore, there are TSL/SSL libraries available, but performing expensive encryptions on possibly non-sensitive data is a waste of the client and server resources. However, if needed,

TLS/SSL encryption could be employed at either the HTTP level or application level for an extra layer of protection.

### 4.3 Asynchronous JavaScript for Data Transport

Asynchronous JavaScript, commonly referred to as “AJAX”, is a built-in JavaScript technique using `xhr`. `xhr` allows a web page to make an HTTP request without the page being refreshed. Due to JavaScript’s functionality to manipulate the DOM (Document Object Model), or the “look” of an otherwise static web page, pages can be changed and updated without being completely reloaded and re-rendered. Popular AJAX applications include Google’s Gmail, and Facebook. AJAX saves bandwidth because it only needs to load only a part of the page. [3][5]

Using AJAX, an abstraction of the client-server architecture can be established while someone is viewing a web page.

### 4.4 CPU Scavenging Techniques

Many grid computing projects employ a CPU scavenging technique in order to use “spare CPU cycles”. This is a must for a browser-based application. Most users do not want their browser to be used as a computational node, and are often using it to do multiple things at once, such as CPU-heavy video decoding. Although JavaScript is usually run in a single-threaded environment, data can be computed in parallel because of its functional capabilities and event-driven architecture, so a page will not completely freeze while an algorithm is running, however it may slow down. [11]

`setTimeout()` is the built in JavaScript timer. [11] This can be used to run the algorithm in smaller pieces and use a small amount of CPU. It can also be configured independently depending on the user’s machine and browser (Some are much faster than others. For example, Google Chromium is 61% faster than Mozilla Firefox)

### 4.5 Client and Server-Side Algorithm Implementation

Since JavaScript and HTTP are universal, regardless of implementation, a distributed algorithm is extremely portable and can be embedded on any web site. JavaScript libraries have the ability to be cross-site “hot linked” with the HTML script tag. For example, loading 2 separate external libraries is as simple as including the URLs:

```
<script type="text/javascript" src="http://www.google.com/lib.js"></script>  
<script type="text/javascript" src="http://www.yahoo.com/yui.js"></script>
```

The scope of a distributed algorithm may be increased by including the JavaScript library containing the algorithm in a web page and running it. Any browser that visits the website will thus become a node.

Furthermore, AJAX HTTP requests can be made to an external server. Therefore, the master server stays central, but the algorithm may be implemented elsewhere. [11]

## **5. Experimental Results**

A proof of concept reversed hashing distributed algorithm was created. The experiment consisted of 101 random hidden md5 hashes from a dictionary of 80,363 words. The algorithm tests each hash and compared it to a hash from the dictionary in order to find the hidden hashes. Reverse hashing was chosen because it is one of the most straightforward distributed algorithms.

The software used for the program was all open-source. The Lighttpd web server was chosen for the master node, due to its small footprint and touted ability to handle many simultaneous TCP connections. With a lot of I/O, a server must be able to handle many incoming requests. PHP was chosen as the CGI application. MySQL was chosen for the database. On the client side, an md5 JavaScript library was used for the hash computations, and the JQuery JavaScript framework was chosen due to simplicity and robust AJAX library.

The MapReduce paradigm was implemented in the experiment. The map phase is as follows: A user first visits the page and requests it. On the client side, the JavaScript libraries are loaded. The client then sends an HTTP request via AJAX to the “map.php” function from the server. The map.php function then queries the database for 60 random words that haven’t been checked yet. Once the client has the list of words, it hashes each one through the md5sum function and HTTP POSTS the block of computed words/hashes back to the server via AJAX. During the “reduce” phase, the server checks each found hash/word against the database of hidden hashes and updates it if any were found. The process continues every 2 seconds so the algorithm is running continuously on the client until the user stops the script or leaves the page.

Calculating 60 string hashes takes <.1 second with Google’s V8 JavaScript engine on a 2.4GHz CPU. Thus, data units could be computed quickly and easily on a per-click basis, if the problem permits.

The server takes the biggest hit. Like most servers, it must employ a great amount of bandwidth and have efficient database queries. Mapping large data sets can take a long time, such as joining 80,000 rows in the small dictionary. A true reverse string hashing algorithm would most likely calculate permutations of characters instead of querying a dictionary. It would thus deal with extremely large data (exponentially proportioned) sets, however eliminating the need for a database-driven map phase.

## **6. Related Work**

Distributed computing is important for many subsets of computing, including applications geared towards business and science. While reverse hashing is rarely considered useful outside

of the “hacking” realm, there are a number of feasible projects that could be accomplished with the widespread power of the internet, such as Monte Carlo trials (with each node representing a trial), prime number searching, cryptography, and other scientific models and business applications.

With the cloud touted as “the future”, a JavaScript based distributed application seems like the ideal solution. Take the Google Chrome OS, a completely browser-based operating system. Every application is a JavaScript or web application, with all of the data being stored on the cloud. With more and more services being moved from the desktop to the web, it is possible for distributed applications to make the same jump. [3]

## 7. Conclusion

A browser-based distributed computing model is a very feasible implementation, depending on the type and data-sensitivity of the problem at hand. A working implementation has been tested and works as expected, even with regards to data loss and the relative unreliability of the browser. There are other examples available on the internet, however none are in widespread use and can only be considered proof-of-concepts. The model is very loose, and any implementation must be very compromising because of its inherent insecurity and high latency, ad hoc situation of the browser and the internet. The most important issue, however, is that the browser has become so robust that it has presented itself as the basis for the most modern applications, and thus the possibility for an extreme “peta-scale” distributed application is immediately available.

## 7. Reference

- [1] Fred Douglass, M. Frans Kaashoek, John K. Ousterhout, Andrew S. [A Comparison of Two Distributed Systems: Amoeba and Sprite](#). ACM Transactions on Computer Systems, Vol 4, Fall 1991
- [2] Bin Al-Tameem, A.; Chittikala, P.; Pichappan, P., "A study of AJAX vulnerability in Web 2.0 applications," Applications of Digital Information and Web Technologies, 2008. ICADIWT 2008. First International Conference on the , vol., no., pp.63-67, 4-6 Aug. 2008
- [3] Zepeda, J.S.; Chapa, S.V., "From Desktop Applications Towards AJAX Web Applications," Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on , vol., no., pp.193-196, 5-7 Sept. 2007
- [4] Richard Fujimoto, Michael Hunter, Jason Sirichoke, Mahesh Palekar, Hoe Kim, Wonhu Sun. [Ad Hoc Distributed Simluations](#). 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS'07), pp. 15-24, 2007.
- [5] Paulson, L.D., "Building rich web applications with AJAX," Computer , vol.38, no.10, pp. 14-17, Oct. 2005

- [6] Jefferey Dean, Sanjay Ghemawat. [MapReduce: Simplified Data Processing on Large Clusters](#). Google, Inc. Nov 11, 2009
- [7] [CPE 458-Parallel Programming](#). California Polytechnic State University. Oct 2009.
- [8] [World Internet Usage Statistics and World Population Stats](#). Miniwatts Marketing Group. Dec 7, 2009.
- [9] [Official Google Blog: We Knew the Web was Big](#). Google, Inc. Dec 7, 2009.
- [10] [Browser statistics](#). Refnes Data 1999-2009. Dec 7, 2009.
- [11] [ECMA JavaScript Language Specification](#). ECMA International. Nov 1996-Sept 2009. Dec 7, 2009.
- [12] [High Performance at Massive Scale: Lessons Learned at Facebook](#). Nov 24, 2009.