Optimized Performance of Web Applications

GLEN OAKLEY, CSC260-01, The College of New Jersey
DR. M. PULIMOOD, Instructor

Web Applications are a breed of programs that are designed to be lightweight and run across a network. Due to recent advances in technology, Web Applications are gaining both popularity and function with developers and consumers alike. As with all platforms, there are many different facets of developing and deploying a Web Application. Prominent among these is an analysis of the processing power required to drive the application. This report will attempt to address some of the issues that arise when developing for a web-based platform, focusing on concerns with overall performance and the distribution and utilization of processing power.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Design Techniques—*Web-Based Applications*

General Terms: Design, Performance

Additional Key Words and Phrases: Web Application, Web Development, Networking

1. INTRODUCTION

The term "Web Application" (also knows as "WebApp") is used to refer to a program or application that is accessed through a network. The workload is shared between the client accessing the application and the server presenting the application. The client usually handles front-end elements such as the user interface and light calculations, while the server handles back-end elements, i.e. database queries and resource-intensive calculations. Web applications are diverse; a WebApp could be as simple as static HTML files that present basic information, or as grandiose as dynamically-changing web pages that update automatically and interact with other users.[Pressman 2010]

2. CHARACTERISTICS OF WEB APPLICATIONS

2.1. Notability

WebApps have become popular due to the persistence of Internet browsers; the web is a platform available to a large percentage of the population, and has a single standard that is maintained across many different architectures and operating systems. The permeability of WebApps benefits both producers and consumers; the former is able to create an application that can be deployed to a single platform, while the latter can easily access that application.

2.2. Technology

The foundation of a WebApp lies with the HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript, which give these applications structure, formatting, and function, respectively.[W3C 2012] Many languages had been created or adapted to enhance the features of WebApps, including PHP for dynamic pages[PHP-Group 2012], Ajax for real-time client-server communication[Ullman and Dykes 2007], and Java servlets for extending the features of a typical client-server communication model[Bodoff 2012]. There are numerous technologies such as these, each of which has many features that overlap with others. It is up to the developer to determine the best language(s) to use in developing his or her application, and there have been many studies done on the benefits of different languages and platforms.[Prechelt 2011]

2.3. Issues

The apparent simplicity of WebApps to an end user is misleading; the nature of web applications lends itself to a host of problems and complications. These issues and more are subjects that must be addressed when creating a WebApp, and may affect a developer's decision to use this platform. While some issues are largely outside of the developer's control, proper design and programming practices can help prevent many problems from arising.

There are a whole host of issues that become relevant when dealing with an application that runs over a network, including security vulnerabilities in client-server communication, transparency as it relates to a user's data (which is usually trapped on a server and inaccessible to a user outside of the application), and server-side errors and complications outside of a user's control that may disable a WebApp. Complications related to the structure of the network on which the application is run are beyond the scope of this paper.

2.4. Application Processing Power

Most Web Applications are run through a user's web browser or custom-designed web interface; the resources for a web browser are much more limited than a typical platform, and thus the client-side computations are much more limited on this platform.[Okamoto and Kohana 2010] Additionally, that massive amount of usage a web application may incur must be handled by the backend server; combined with the low-end processing power of the client, this forces a developer to invest in a stable (and potentially costly) server solution, one that is much more powerful than might be necessary with a application on a different platform.

When designing a Web Application, one must remember these limitations. The most important thing to consider is whether a program can even be designed as a WebApp. A program that relies heavily on local resources would not be suitable as a WebApp (such as a Computer-Aided Drafting program). Web Applications are not designed to access those low-level components that enable powerful computing, such as a client's underlying operating system and hardware, so program such as file trackers and hardware monitors could never be made as WebApps. Once the appropriateness of the application as a WebApp has been established, concerns with implementation must be address.

3. PROCESSING DISTRIBUTION

As Web Applications are all supported by a server[1], it is important to ensure that the server is able to handle the workload incurred by the application. An important factor in this is scalability; as interest and use of an application grows, more instances of that application will be running at one time, which in turn will force the server to process more data. Indeed, it is possible for a server to reach a point at which it cannot process the incoming data as fast as it would on a smaller scale. The results are noticeable to the user in the form of response latency.

Latency can be a killer for applications; users expect 'immediate' responsiveness from an application. If a user feels a program does not consistently respond within a reasonable time limit, they will react negatively, possibly abandoning the application for an alternative solution.[Selvidge 2002] Upgrading the server referenced by a Web Application is a possibility, however the overhead of this can be very large depending on the current size of the server.

A largely effective solution to the problem of scalability is using a technique known as load distribution. Load distribution is often used on the server level to distribute a single problem across multiple servers, letting each server handle a different block of data. Again, this may present a problem in which a single server is forced to process a large or complex

[1]In this paper, the general term "server" will be used to refer to either a single computer or collection of computers that perform data processing on the backend for a client-server application.

chunk of data, causing the entire system to bottleneck. A potentially better distribution solution lies in web workers.

3.1. Web Workers

Web workers are scripts executed by a web page. These scripts run in the background and are independent of the other scripts related to their parent web page, as well as independent of each other. There are a number of marked benefits to this method of load distribution:

— Web workers are native to modern-day web browsers, the platform of Web Applications, meaning that there are no extra packages or services to install or configure in order to enable web workers. This makes them an ideal solution for developers, who need not deal with a setup process before their application is allowed to run.
— Web workers can be instantiated on the client side, allowing the application to take advantage of spare processing power available on the user's computer.
— The threaded nature of web workers allow networked data requests to be executed fully asynchronously (that is, the application will not 'hang' while waiting for a response).

Possibly one of the largest benefits for developers is this: Web workers can distribute the data-crunching needed by an application across its clients, reducing the processing power required by the application's server, thus reducing the monetary and time costs to the developer.

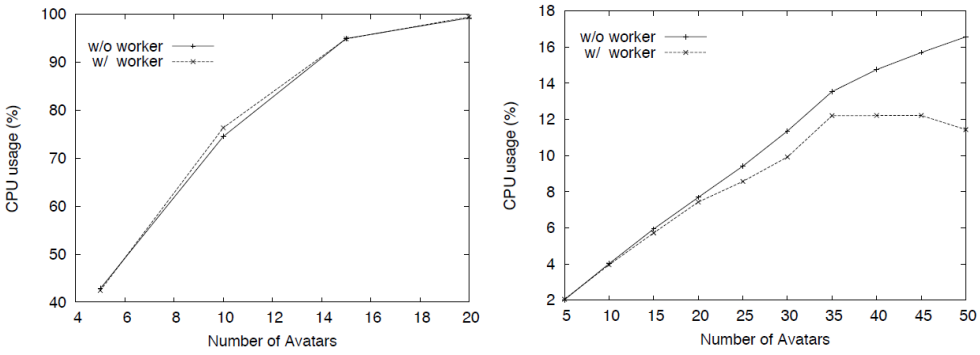

Fig. 1. CPU usage of a testbed web-based video game as a function of the number of users (avatars) on both a low-end (left) and high-end (right) server.[Okamoto and Kohana 2010]

A case study done at SEIKEI University in Japan involved implementing web workers in a web-based video game and measuring the difference in performance (see Figure 3.1). Past the 20 user mark, as the number of users increased, the difference in performance between a version of the application implementing web workers and version not implementing them increased at a fairly constant rate, with the web worker model requiring less CPU usage.[Okamoto and Kohana 2010] There is a marked benefit to using the processing power of a client to handle calculations normally computed by the server.

There are, of course, possible detriments to using web workers. Instantiating too many web workers requires a large amount of client-side processing power, perhaps more than a specific client could handle, causing the application massive slowdown or halting execution completely. An overabundance of web workers also presents a problem with concurrent edits. For instance, if two or more web workers attempt to simultaneously edit a web page's

DOM[2], the results of the edit will be unexpected, and almost certainly different than what the developer/user intended. A developer must be conservative in the use of web workers, and keep in mind the limited systems that their application may be run on.

4. NETWORK COMMUNICATION

Network communication can be a difficult challenge to overcome for developers. Networks depend on a host of different devices and software in order to function properly, and thus can be unpredictable and encounter problems outside of an application's control.

One such issue is with bandwidth. For developers creating an application for use on an internal network or intranet, this is much less of an issue, as the scope of such a program is within a company or organization. However, when scaling up to an application designed to run on the Internet, a developer has to assume that at some point, there will be a slowing or interruption in communication over that network. The application must be designed with this problem in mind.

4.1. Compression

One solution to the problem of network communication speeds is to use compression. With this method, the data being sent across a network is packaged into a condensed form and then unpacked when it reaches the client. The client receives the same information it would without compression, but due to compressed data having a smaller footprint, the data is able to reach the client faster.

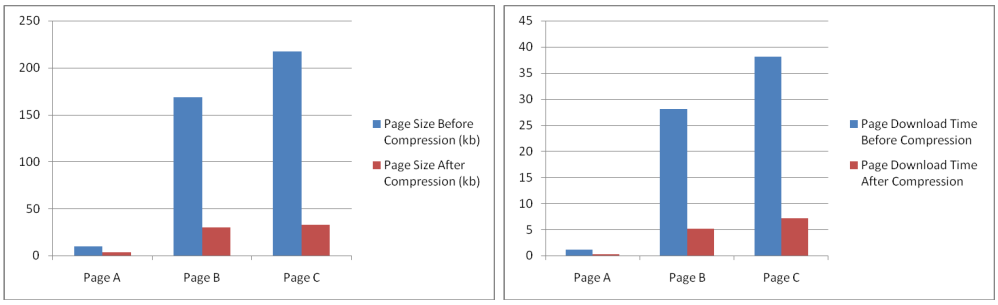

Fig. 2. The size and download time of compress and uncompressed files[Al Fararjeh and Abu Jabal 2010].

Figure 4.1 shows clear benefits in network performance while using compression. The amount and time of processing power required to uncompress on the client side is nearly negligable, making compression an ideal solution for speeding up Web Applications.[Al Fararjeh and Abu Jabal 2010]

4.2. Asynchronous JavaScript

At their disposal, a developer has a few basic tools that a client system is guaranteed to have available. Among these is JavaScript, a language implemented by all web browsers and used for scripting. JavaScript is able to handle networked requests in two ways; synchronously or asynchronously. After making an HTTP request, the former method of requesting will cause the script to wait for all of the requested data to be returned to the client or for a timeout to occur before continuing operation[Al Fararjeh and Abu Jabal 2010]. If a network

[2]The Document Object Model refers to the structure of a web page, consisting of a root document object and its children (who themselves have children, etc.). Changes to the DOM are reflected in the content of a web page and the way it is displayed.

is running slowly, this will cause a massive runtime slowdown, depending on the amount of requests the application must make.

A much better alternative is to use the asynchronous requests. This will allow the application to continue execution while the request is being processed and delivered; the application need not wait for the request to be downloaded. In JavaScript, this method of requesting is known as Ajax[Al Fararjeh and Abu Jabal 2010]. Not only will Ajax prevent an application from halting for networked requests, but invalid requests or missing results can be handled in realtime instead of causing the program to hang.

An Ajax request is created in JavaScript using a special object whose type is XMLHttpRequest. This is very similar to the typical HTTP request, with the major difference that when the request is sent, the web page that is parent to the request need not change or refresh; the request will be completely invisible on the front-end. The format of the requests depends on what handler is expected to process the request on the server side; different languages require different request formats. In addition to a request, Ajax also requires a 'callback method': A function that will be called after a response is obtained. When the server receives an Ajax request, it builds a response. The response type is not specified by the Ajax request, so the application must know what to expect from the server[3]. Once the response has been built, the server returns the generated response to the point in the Web Application where it was called from. Upon receiving a response, Ajax calls the callback method assigned to the original request, which can then process the data.[Albert et al. 2008]

There are a few downfalls of using Ajax. It is very important that the server response is checked *before* the application begins processing the data. It is very possible that, given a 'large' number of users, the number of Ajax requests the server needs to process could cause a race condition[4] that may halt the server or cause it to crash[Albert et al. 2008]. This is not limited to the server side; as with web workers (Section refsec:webwork), it is possible that two callback methods will both operate on the same object, causing unexpected and unwanted results. Ajax should be implemented with caution, and a developer should be aware of the possible race conditions that may form as a result of too many asynchronous requests.

4.3. Peer-to-Peer Connections

A style of networking exists in which the idea of a server has been completely removed from the process; peer-to-peer (p2p) communications are a style of networking in which data is sent from client-to-client with no server to process or redirect the messages. A decentralized model such as this completely eliminates the need for a server, which would need to be consistently maintained by the developer or related parties.[Schollmeier 2001]

There are, of course, drawbacks to this type of distribution. Each client must sacrifice part of its resources, such as extra disk space, processing power, and bandwidth. For some applications, this may be a trivial amount of resources, but this becomes unfeasible for applications with features such as a large persistent database (Where would the main database be stored? See Section 5.1). The performance of a p2p system is entirely dependent on the number of users. That is, a p2p system with a small number of running clients will be much less effective than one with a relatively large number. Removing a central server also disables some of the convenient features of Web Applications, such as always being inherently up-to-date.

Peer-to-peer Web Applications are not feasible for a large number of applications. An application should not be forced into this model, as the results of such actions could cause

[3]Though the Ajax request is officially typed as an XML request, the response from the server could be of any format, such as XML, HTML, or JSON.

[4]A race condition is whcn two processes try to access the same resource, or when two processes executing in parallel are dependent on each other.

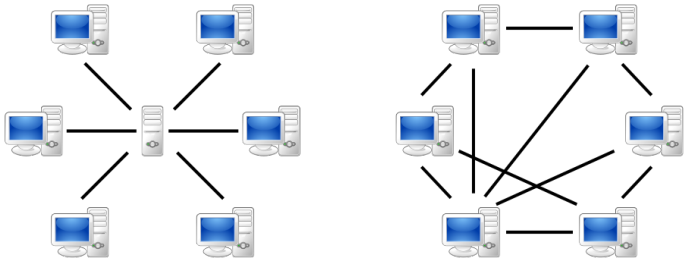

Fig. 3. A server-based application compared to a p2p-based application[Bieg 2007].

large problems for the application, user, and developers. For an appropriate program, however, a p2p system has the potential to remove the need for a large server or high bandwidth, and could be very cost-saving.

5. LONG-TERM STORAGE

For many applications, user-generated data must be stored long-term for access at a future point in time. This allows for data persistence in the event that the application halts operation on either the client or server side. The developer of a Web Application is presented with a few options for storing data indefinitely.

5.1. Databases

Databases are used in applications to store data in an easily accessible manner over long periods of time, for access at a later date. Databases are very valuable to a developer, as they usually store data in a way that allows for quick access using a set of algorithms. Most applications rely heavily on databases, as they allow large amounts of data to be cached in permanent storage instead of in memory (caching all data in memory is unfeasible for large-scale applications).

A typical Web Application will execute the following procedure many times during its lifespan[Li et al. 2003]:

(1) The application sends a request from the client to the server in order to obtain data.
(2) The server looks in its cache (memory) for the requested information.
(3) If the information is not found, the server requests the information from its database(s).
(4) The information is returned to the client, who uses it as needed.

Ideally, the server has a large enough cache that a large amount of its requests do not need to directly query the database; database (disk) reads are much slower than reading from memory. Since cache is located in the memory, reading from the cache instead of the database(s) greatly improves the performance of an application.

The underlying structure of a database is often of little importance to a developer. There are a large number of different database structures to choose from, a number of which perform at very similar, high levels. A developer will almost always implement a database that has already been developed, as the overhead of designing a new database for a specific application is unfeasibly high. This is especially true for Web Applications, which can take advantage of a number of database management systems (DBMS) created specifically for web-based languages and scripts (see Figure 5.1).

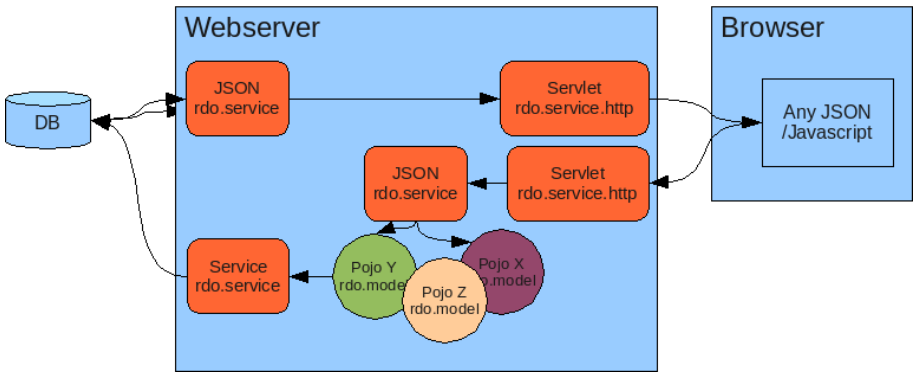

Fig. 4. A sample database interaction, using the JavaScript language[Redora]. The web server parses data from the browser and stores/retrieves information in the database.

Since database reads are relatively slow operations, a developer must be concerned with choosing optimizing their interaction with a database. Data stored inside of a database is accessible through 'queries'. A typical database will handle a multitude of different queries in a short amount of time. Not only do different DBMS have syntactical differences in the way queries are formulated, but they each handle the 'stack' of queries in different ways. The DBMS and database itself are tasked with scheduling and executing the queries in the best way possible[Ahmad et al. 2008]. In order to obtain the best performance, however, the queries must be written in an optimal way, i.e. they should not contain extraneous commands or information (though many DBMS are able to detect and remove these). It is also in the best interest of the developer to group multiple requests into a single query, if possible. This reduces the number of queries the database needs to process, and can cause a measurable difference in database (and thus application) performance.

5.2. Cookies

Cookies are sets of formatted data that are received by a client from a server and stored in the client's persistent storage. It can be quite beneficial to a developer to allow some of the user's data to be stored locally on their own machine, instead of forcing the server to categorize and handle that data.

Unfortunately, this also gives users direct control over the cookies. A large number of Internet users have been programmed to think that cookies are intrusive and even malicious (some are), and popular web browsers have been designed to control the storage of cookies very carefully[Millett et al. 2001]. A user may decide to delete the cookies that the developer is using to store data. Thus, important long-term and sensitive information should still be stored on the server side of the Web Application. It is up to the developer to determine the trade-off between data security and server capacity.

6. CONCLUSION

This paper discussed some of the ideas and methods behind developing optimized web applications. The raw processing power required by a server can be mitigated by using client computers for calculations, and implementing web workers in an application can further increase the speed at which calculations can be carried out. A major factor in the performance of networked applications in general is network communication speed. Compressing data allows for smaller transfer times between network nodes, while asynchronous data requests (such as Ajax) can trickle data back and forth between the server and client, eliminating the need for an application to transfer periodic large bursts of information. Developing an ap-

propriate, well-designed peer-to-peer platform can eliminate the need for complicated tasks such as server maintenance. Using suitable long-term storage methods can further increase the throughput of data for an application. It is important for a developer to utilize these methods wisely, and not abuse them, which leads to problems in itself.

REFERENCES

Ahmad, M., Aboulnaga, A., Babu, S., and Munagala, K. 2008. Modeling and exploiting query interactions in database systems. In *Proceedings of the 17th ACM conference on Information and knowledge management*. CIKM '08. ACM, New York, NY, USA, 183–192.

Al Fararjeh, A. M. and Abu Jabal, A. M. 2010. Recommendations to improve performance of an enterprise web-based application. In *Proceedings of the 1st International Conference on Intelligent Semantic Web-Services and Applications*. ISWSA '10. ACM, New York, NY, USA, 29:1–29:6.

Albert, T. J., Qian, K., and Fu, X. 2008. Race condition in ajax-based web application. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*. ACM-SE 46. ACM, New York, NY, USA, 390–393.

Bieg, M. 2007. Networking diagrams.

Bodoff, S. 2012. Java servlet technology. Website. http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html.

Li, W.-S., Po, O., Hsiung, W.-P., Candan, K. S., and Agrawal, D. 2003. Engineering and hosting adaptive freshness-sensitive web applications on data centers. In *Proceedings of the 12th international conference on World Wide Web*. WWW '03. ACM, New York, NY, USA, 587–598.

Millett, L. I., Friedman, B., and Felten, E. 2001. Cookies and web browser design: toward realizing informed consent online. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. CHI '01. ACM, New York, NY, USA, 46–52.

Okamoto, S. and Kohana, M. 2010. Load distribution by using web workers for a real-time web application. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*. iiWAS '10. ACM, New York, NY, USA, 592–597.

PHP-Group. 2012. Php: Hypertext preprocessor. Website. http://www.php.net/.

Prechelt, L. 2011. Plat_forms: A web development platform comparison by an exploratory experiment searching for emergent platform properties. *IEEE Trans. Softw. Eng. 37*, 95–108.

Pressman, R. S. 2010. *Software Engineering: A Practitioner's Approach* 7th Ed. McGraw-Hill, New York, New York.

Redora. Javascript db interaction.

Schollmeier, R. 2001. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*. 101–102.

Selvidge, P. R. 2002. The world wide wait: Effects of delays on user performance. *International Journal of Industrial Ergonomics 29*.

Ullman, C. and Dykes, L. 2007. *Beginning Ajax* 1st Ed. Wrox.

W3C. 2012. World wide web consortium standards. Website. http://www.w3.org/standards/.

Received March 2, 2012; revised March 23, 2012; accepted