Overhaul
Extending HTTP to Combat Flash Crowds

Jay A. Patel and Indranil Gupta

Department of Computer Science
University of Illinois at Urbana - Champaign
Urbana, IL 61801

Abstract. The increasing use of the web for serving http content, for database transactions, etc. can place heavy stress on servers. Flash crowds can occur at a server when there is a burst of a large number of clients attempting to access the service, and an unprepared server site could be overloaded or become unavailable. This paper discusses an extension to the http protocol that allows graceful performance at web servers under flash crowds. We implement our modifications for the Apache web server, and call the new system as *Overhaul*. In Overhaul mode, a server copes with a stampede by offloading file transfer duties to the clients. Overhaul enables servers to chunk each requested document into small sections and distribute these partial documents to clients. The clients then share the sections amongst themselves to form a complete document. Overhaul enables a web server to remain responsive to further requests from other clients and at the same time helps conserve the amount of bandwidth utilized by a flash crowd. We present detailed experimental results comparing the benefits of using Overhaul under flash crowds and under normal operating situations. Although we restrict our studies to static content, the Overhaul architecture is applicable to improving web services in general.

1 Introduction

There have been numerous reports of flash crowds ruining the performance of web sites. Flash crowds can bring a web server to a screeching halt. For example, an order of magnitude increase in traffic can be expected as a result of linkage from a highly popular news feed. Most web servers are not designed to cope with such a large spike in traffic. There are not many reasonable, economically-sound solutions to control a massive surge in traffic. There are only a few choices a web master can make to combat the sudden influx of traffic: invest in extra resources (be overly insured), temporarily shut down the web site, or change the content of the site. The first solution is not a viable alternative for many enthusiast web masters or not-for-profit organizations as buying surplus resources is an expensive proposition. Additionally, since most flash crowds aren't malicious, the later two solutions could pose a problem to the regular visitors of the site. If the site is shut down, one may alienate the regular visitors and, at the same time, fail to attract new ones.

C.-H. Chi, M. van Steen, and C. Wills (Eds.): WCW 2004, LNCS 3293, pp. 34–43, 2004.

The performance of a web server degrades as the number of concurrent requests increases. Additionally, a pending document request blocks the server from addressing requests from other clients. Flash crowds create both these problems and therefore present a challenging research question. Previous research to address flash crowds, which is discussed in detail in Section 2, can be broadly categorized as solutions that either propose modifications to the server architecture, institute protocol changes, or depend on client collaboration for data sharing.

In this paper, we propose a systematic yet simple change to HTTP that reduces the resources required to serve a flash crowd. The primary goal of this paper is to present a technically feasible solution with minimal changes to the current HTTP standard. It is important that the solution be deployable on today's most-popular web servers and clients. The process presented by this paper is seamless to the end-user and requires no special input from the web master once it is deployed. We strive to minimize the economic costs of a flash crowd from the server administrator's perspective. The main contribution of this paper is the massive reduction of resources (primarily bandwidth) utilized by the server during a flash crowd.

Overhaul is a modification to HTTP that involves both the client and the server. During a flash crowd, the server steps into Overhaul mode and serves only portions of requested documents. Each file is divided into multiple portions and distributed amongst the clients. The clients collaborate together (independent of the server) to share the various portions to form a complete document.

2 Related Work

Previous research can be divided into three different broad categories: architectural changes, protocol changes, and cooperative sharing.

SEDA [12] is a staged event-driven architecture used to support massively concurrent requests for well-conditioned services like HTTP. Capriccio [11] is a highly scalable thread package that can be used with high-concurrency processes. The thread package is implemented as a user-level process, making it easily portable across platforms and far removed from inconsistencies of the underlying operating system. Another paper [10] examines the impact of various connection-accepting strategies used to boost web server performance. Web booster [7] is a dedicated machine that intercepts the TCP connection from clients to the server during peak times and only bothers the server if the document is not present in its cache.

DHTTP [8] is a replacement for the TCP-based HTTP protocol. Client requests are initiated over UDP streams, which frees up the server from the overhead of maintaining expensive and non-cacheable TCP connections.

Backslash [9] is a collaborative web mirroring system. Squirrel [3], based on Pastry, implements a web caching mechanism over a P2P overlay in lieu of a centralized proxy server. Another collaborative web caching mechanism [5] based on Kelips performs well during heavy churn (rapid arrival and departure

of member nodes). Content distribution networks (CDN) [4] like Akamai are utilized by many well established sites. CDNs are used to minimize the number of hops required to fetch a document. FastReplica [1] is an algorithm used to efficiently replicate a large file using subfile propagation. BitTorrent [2] is a protocol for distributing large files over a P2P network. The file provider makes the file available by providing a meta-info (torrent) file and a tracker URL. Clients exchange blocks of the file with other peers connected to the tracker.

3 Design

Overview. Overhaul is set of changes to HTTP that lets overloaded servers distribute content through a P2P network. As described in this paper, Overhaul assumes a fully collaborative environment. A web server under the rampage of a flash crowd enters into Overhaul mode and splits up the requested document into n chunks. Each request results in a response that includes the i^{th} chunk and the IP addresses of m other clients accessing the document. A signature for each of the n chunks is also provided in the header. A client supporting the Overhaul extension connects to other clients to retrieve the remaining chunks.

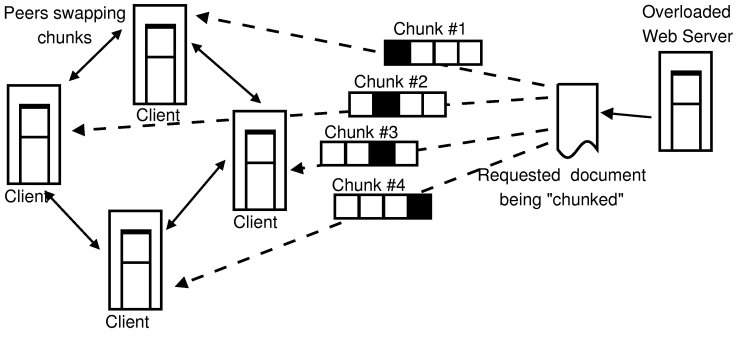

Fig. 1. Overhaul: An overview

A simple Overhaul model is shown in Fig. 1. A document is divided into chunks and distributed amongst four unique clients. The clients collaborate to merge the chunks together and form a coherent document. The Overhaul process enables the server to save approximately three-fourths of the bandwidth utilized by a regular fetch. By transferring only a small portion of the document, the Overhaul process frees up the server to satisfy requests from other clients.

Overhaul provides an HTTP-integrated solution for collaborative fetching and sharing of documents. This differs from the approach taken by BitTorrent, which excels as a specialized tool for distributing large files over a separate P2P

network. Unlike Overhaul, BitTorrent requires a dedicated meta-info file for each requested document. This results in additional traffic to a server. BitTorrent also requires a dedicated tracker (basically a server which is not compatible with HTTP clients). Moreover, BitTorrent is not geared towards short-time downloads, as it depends on peers sharing documents after completing a document fetch.

Details. The Overhaul solution requires modification to regular HTTP client-server interaction. To maintain backwards compatibility, clients inform the server of their Overhaul functionality by sending a `Supports` tag with a regular HTTP request. This allows the server to maintain a list of the clients that will engage in the Overhaul process, and additionally send an error message to legacy clients. A sample tag takes the form of:

```
Supports: Overhaul <Speed> <Port>
```

In most cases the server will ignore this tag and respond normally. However if the server is overloaded, it will respond in Overhaul mode with only one chunk of the document. Three special tags: `Overhaul-chunk`, `Overhaul-hosts`, and `Overhaul-md5sum` are sent in addition to the header. The first tag provides the sequence number of the current chunk being sent to the client. The second tag includes a list of peers (and their respective server ports) that are currently engaged in the Overhaul process. The last tag is a list of md5 signatures of the chunks. The md5 hash is provided to verify the contents of the chunks. The server will need to return HTTP code `206` to indicate partial content and the corresponding `Content-Range` tag with the byte range of the transmitted chunk.

If a client gets an Overhaul mode response back from the server, it opens the aforementioned port to allow requests from other peer-clients. Meanwhile, the client establishes connections to other peer-clients to gather more chunks. After establishing a connection to another peer-client, a client sends a request to fetch a missing chunk in the given form:

```
GET /file.html CHUNK <#> HTTP/1.0
HOST www.host.com
Overhaul-port: <Port>
Overhaul-hosts: <List of known hosts:port>
Overhaul-chunks: <List of available chunks>
```

The requesting peer-client provides the server port, a list of known peer-clients and available chunks with the request. In response to an Overhaul request, the serving peer-client sends a HTTP return code of `206` and the relevant data if it possesses the requested chunk. Otherwise, a HTTP return code of `404` is sent. The response header also includes `Overhaul-hosts` and `Overhaul-chunks` tags to exchange information with the requesting peer-client.

After the initial interaction, both peer-clients learn about new peer-clients and the available chunks on the other peer-client. This enables clients to grow

their list of known peer-clients. The clients repeat the process with other peer-clients until the document is completely assembled.

If the peer-clients list is fully exhausted without successful completion of the document, a client may re-request chunks from known peer-clients. If the client fails to collect the document chunks even after successive tries, the client may retry the server. The server will respond either with a new list of peer-clients or the document in its entirety depending on the current server load.

To further reduce server workload, a client may query known peer-clients for other documents being Overhauled on a particular host with the `INFO` tag. This tag helps reduce the number of active TCP connections and the amount of bandwidth utilized by affected hosts as HTML documents generally include many embedded objects. A query seeking information about documents being served takes the form of:

```
INFO www.host.com HTTP/1.0
```

In response to this tag, the peer-client indicates that there is no associated body content with HTTP response code `204`. Additionally, the following set of header tags is sent for each document currently being fetched in the Overhaul mode:

```
Overhaul-URL: <An Overhaul URL>
Overhaul-md5sum: <List of md5 hashes>
Overhaul-hosts: <List of host:port>
Overhaul-chunks: <List of available chunks>
```

Once the document fetch process is complete, the peer-client may shut down the port and decline further connections.

4 Implementation

The modifications required by Overhaul on a server were implemented as a module to the Apache/2.0 web server. Apache was chosen because of its popularity [6], extensibility, and openness. Apache web server can be extended by loading modules at run time for added functionality. `mod_overhaul` implements the Overhaul extension for the Apache web server. The module can be customized by specifying the minimum chunk size, the maximum number of chunks, and the number of peers (to keep track of) per document.

The client was not implemented directly: a proxy server was developed using the Java programming language. A proxy was deemed to be a superior solution as it can be used with most web browsers without any source modifications to the browser. The proxy can also be run on a vast multitude of platforms that have a Java virtual machine. The proxy normally fetches HTTP documents but acts as an Overhaul client when a server responds in Overhaul mode.

5 Results

Server. The performance of `mod_overhaul` was tested by executing a few controlled experiments. Apache/2.0 web server was compiled with the worker MPM, which uses a single process with multiple child threads to handle requests. The tests were performed on a medium-range web server with a 2.5 GHz AMD Athlon XP+ processor and 1 GB RAM powered by RedHat's Fedora GNU/Linux operating system. The client machine utilized for the experiments was a low-end computer powered by a 650MHz Intel Pentium III processor and 320 MB RAM. Both machines were placed on the same 100 Mbps switch as the web server to minimize network latency and maximize throughput. Testing was performed using the `ApacheBench` utility. All experiments were performed with 25 concurrent accesses from the client machine. The CPU utilization of the client machine was minimum. However, network bandwidth was saturated on both the server and client machines when performing some large experiments.

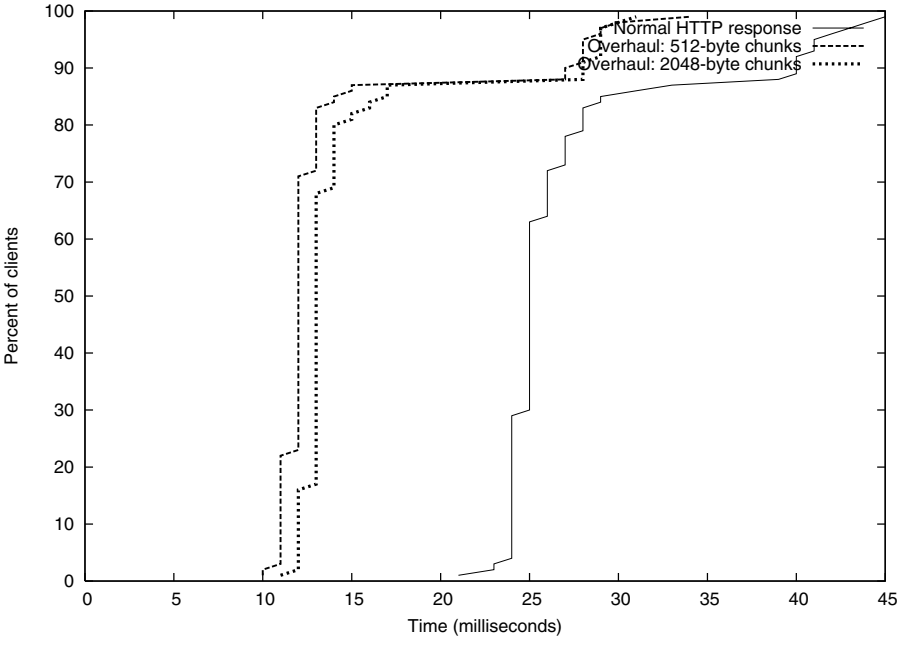

Fig. 2. Time to serve or a medium-sized document (10,000-bytes)

Overhaul's chunked responses lead to a substantial performance gain over regular responses to a medium-sized static document as shown in Fig. 2. Smaller chunks lead to better performance. However, 512-byte chunks only provide a minor performance gain over 2048-byte chunks.

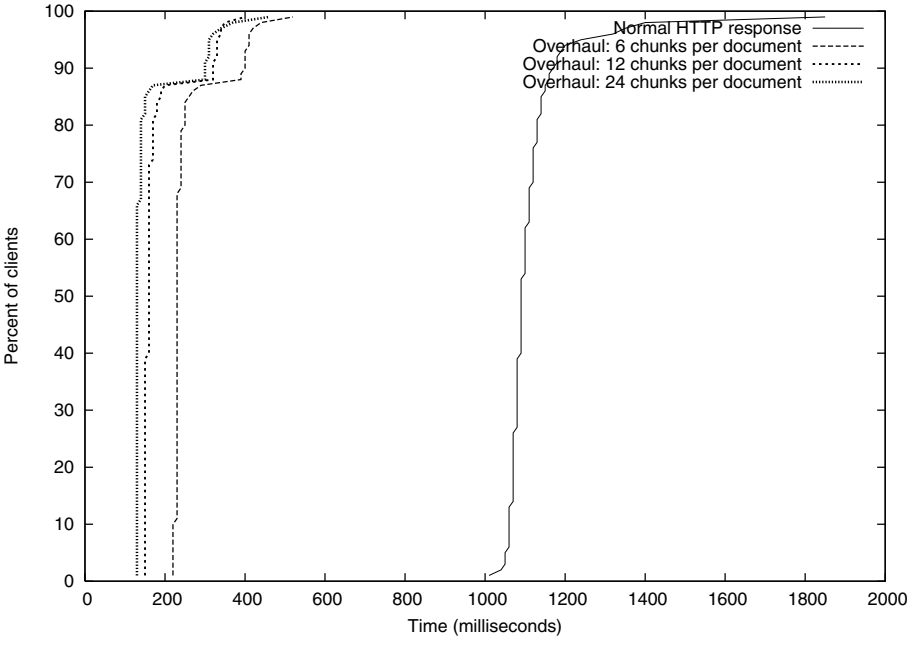

Fig. 3. Time to serve a large document (50,000-bytes)

With larger documents, a constant chunking size creates numerous chunks. Fig. 3 maps the performance of the server when configured to split a document in a fixed number of chunks. This experiment verifies the the diminishing benefits of smaller chunk sizes, possibly due to the constant overhead incurred for establishing a TCP connection. Additionally, the work load increases by having small chunks as clients need to fetch more chunks from other peer-clients. A better solution would be to have a fixed-number of chunks per document, with limitations on the minimum chunk-size.

An experiment to benchmark performance of Apache while serving different size files was setup. The server was configured to have minimum chunk size of 512-bytes with the maximum number of chunks set to twelve. Fig. 4 shows the result of the experiment. In Overhaul mode, the requests are satisfied at a much higher rate than regular responses. Apache is up to eight times more responsive when using as few as twelve chunks for medium sized files. The solution also scales well as file sizes increase.

In other experiments, chunking dynamically generated documents was not beneficial. This behavior is not surprising because the bottleneck in dynamic content generation process is the CPU and not the network. Moreover, dynamic content creates incompatible chunks as many documents (or parts there of) are uniquely created for specific clients.

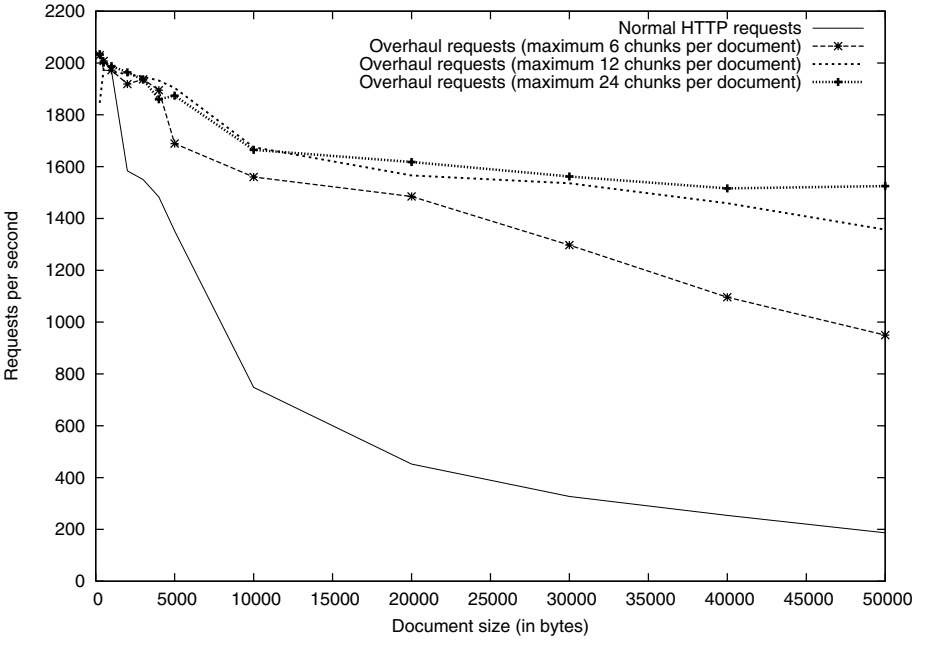

Fig. 4. Request satisfaction rate of static documents

Client. Successful deployment of the Overhaul extension depends on user experience and satisfaction. To gauge the performance of the Overhaul client process, lab experiments were performed on a cluster of 25 high-end workstations. Each machine in the cluster was equipped with a Pentium 4 processor running at 2.8 GHz with 1 GB RAM running the RedHat 9 GNU/Linux operating system. The machines were all part of the same subnetwork. One of the workstations was used as a server and the remaining were utilized as client machines.

For the first client-side experiment, 24 clients simultaneously fetched a 50,000-byte document in Overhaul mode. The server was configured to distribute each document in 12 chunks of 4,167-bytes each. This result was compared against regular HTTP fetches from a server under the flash crowd effect, simulated by requesting the same document concurrently from 150 clients for a brief period. The results are presented in Table 1 show that a fetch in Overhaul mode is more responsive than a regular HTTP request on an overloaded server. Additionally, the bandwidth utilized by the server in the Overhaul mode is approximately one-twelfth of a normal request.

The clients fetched multiple files in the next experiment. Eight files were used: one file of 30,000-bytes, three files of 20,000-bytes each and four files of 5,000-bytes each. The server was configured to send a minimum of 2048-bytes per chunk, with a maximum of 12 chunks per document. The requests were

Table 1. Time required to fetch a large document

	Regular request in flash crowds	Overhaul mode experiment
Fastest	1 `sec`	3 `secs`
Slowest	184 `secs`	8 `secs`
Average	13 `secs`	5 `secs`

staggered in two batches to take advantage of the `INFO` tag: the first twelve clients requested all eight files, whereas the next batch were restricted to requesting only the 30,000-byte file. The regular requests under a flash crowd were simulated as in the previous experiment. Table 2 shows that Overhaul mode fetches are also more responsive than regular HTTP requests when fetching multiple files under a flash crowd.

Table 2. Time required to fetch multiple documents

	Regular request in flash crowds	Overhaul mode experiment
Fastest	1 `sec`	14 `secs`
Slowest	355 `secs`	24 `secs`
Average	26 `secs`	17 `secs`

In the last experiment, the amount of bandwidth utilized by the server in Overhaul mode was approximately only one-eighteenth of the bandwidth consumed by the normal HTTP requests. The `INFO` tag reduces the number of direct TCP connections to the server and therefore the amount of bandwidth required to serve clients decreases as the flash crowd grows.

6 Conclusion

Final Thoughts. A server can benefit tremendously from chunking. In our experiments, the Apache web server is able to serve medium-sized files up to eight times faster in Overhaul mode. Since most of HTTP bandwidth is utilized on transferring medium-sized static objects (usually images), Overhaul can greatly increase the responsiveness of a server. Additionally, the amount of per capita bandwidth utilized by the server decreases as the flash crowd grows. These two properties work in tandem to lengthen the liveliness of a web site being visited by a flash crowd.

Overhaul presents a two-fold benefit to the server administrator: increased server responsiveness and decreased utilization of bandwidth. Additionally, the user is presented with the desired content in a timely manner instead of a scenario involving timed out requests, sluggish server response or complete unavailability.

Future Work. Currently, Overhaul works under the assumption that every user will be cooperative. There are no strong safeguards against freeloaders and malicious users. However, malicious peers can not corrupt chunks because of md5 hash verification. To counter freeloaders, a Overhaul client needs to maintain a trust matrix for each peer-client. The trust score for each peer-client can be a function of chunks shared, reputation amongst other peer-clients, and the bandwidth and latency of network connection. A further modification can be placed in the Overhaul extension to let clients swap trust matrices with other peer-clients.

Besides user behavior issues, many problems are associated with the structure of the Internet. There are many clients that reside behind firewalls and NATs. These clients cannot make their Overhaul port accessible to other peer-clients easily. Another issue is the heterogeneity of network connections. A future implementation can use the speed provided by the client to organize peer-groups.

References

1. L. Cherkasova and J. Lee. FastReplica: Efficient large file distribution within content delivery networks. In *USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, USA, March 2003.
2. Bram Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, May 2003.
3. S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *ACM Symposium on Principles of Distributed Computing*, Monterey, CA, USA, July 2002.
4. J. Kangasharju, J. Roberts, and K. Ross. Object replication strategies in content distribution networks. In *Proceedings of Web Caching and Content Distribution Workshop*, Boston, MA, USA, June 2001.
5. P. Linga, I. Gupta, and K. Birman. A churn-resistant peer-to-peer web caching system. In *ACM Workshop on Survivable and Self-Regenerative Systems*, Fairfax, VA, USA, October 2003.
6. Netcraft. April 2004 web server survey.
7. V. V. Panteleenko and V. W. Freeh. Instantaneous offloading of transient web server load. In *Proceedings of Web Caching and Content Distribution Workshop*, Boston, MA, USA, June 2001.
8. M. Rabinovich and H. Wang. DHTTP: An efficient and cache-friendly transfer protocol for web traffic. In *INFOCOM*, Anchorage, AK, USA, April 2001.
9. T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *International Peer To Peer Systems Workshop (IPTPS 2002)*, Cambridge, MA, USA, March 2002.
10. D. Pariag T. Brecht and L. Gammo. accept()able strategies for improving web server performance. In *USENIX Annual Technical Conference*, Boston, MA, USA, June 2004.
11. R. von Behren, J. Condit, F. Zhou, G. C. Necula, and E. Brewer. Capriccio: Scalable threads for internet services. In *ACM Symposium on Operating Systems Principles*, Lake George, NY, USA, October 2003.
12. M. Welsh, D. E. Culler, and E. A. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *ACM Symposium on Operating Systems Principles*, Banff, Canada, October 2001.