UNIVERSITY of CALIFORNIA

SANTA CRUZ

Prestan: The Design and Implementation of a WebDAV Server Performance Test Suite

A project report submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Teng Xu

March 2004

The project report of Teng Xu is approved:

Professor Jim Whitehead

Professor J.J. Garcia-Luna

Abstract

Prestan: The Design and Implementation of a WebDAV Server Performance Test Suite

By

Teng Xu

With the rapid growth of WebDAV applications and the increasing scale of data management in WebDAV repositories, there is increasing interest in the performance of WebDAV servers. However, no existing tools provide detailed WebDAV server performance information. In this paper we present an automated client-side testing tool that can accurately and comprehensively measure WebDAV server performance, thereby providing developers with significant visibility into server performance behavior. The contribution of this work is fourfold: first, we initiated research on WebDAV server performance measurement, which had not previously received public attention in the WebDAV community; second, we improved the client-side measuring approach and made it accurately reflect the server's state; third, we designed and implemented a new testing tool called Prestan; fourth, we discovered and resolved a performance bottleneck in the Neon WebDAV client library, thereby improving the performance of WebDAV clients that use this library.

CONTENTS

1 Introduction

Despite the significant effort spent on research and development related to the WebDAV protocol in both academia and industry, to date there has been little research on WebDAV server performance. With the rapid growth of WebDAV applications and the increasing scale of data management in WebDAV repositories, there is increasing interest in the performance of WebDAV servers. Developers have intuitive sense that some operations are faster than others (such as a PROPPATCH of single property should be faster than a PROPPATCH for multiple properties). However, there is no concrete knowledge concerning how much faster and where performance bottlenecks arise. To improve understanding of WebDAV performance, it is critical that quantitative measurements are broadly available to the WebDAV development and user community.

Despite the importance of measuring and understanding the behavior of WebDAV servers, no existing tools provide detailed WebDAV server performance information. In this paper we present an automated client-side testing tool that can accurately and comprehensively measure WebDAV server performance, thereby providing developers with significant visibility into server performance behavior. Furthermore client developers also have benefit from our work by better understanding server performance characteristics (such as throughput and locking delay), thereby supporting rational design choices in client design.

There are several contributions in our work. First, we initiated research on WebDAV server performance measurement, which had not previously received public attention in the WebDAV community.

Second, we improved the client-side measuring approach and made it accurately reflect the servers' state. Third, we designed and implemented a new testing tool called ***Prestan***, which can comprehensively measure WebDAV server performance and help developers gain insight into the server performance characteristics. Fourth, we discovered and resolved a performance bottleneck in the Neon WebDAV client library [4], thereby improving the performance of WebDAV clients that use this library.

The remainder of this paper is organized as follows. Section 2 introduces the background related to WebDAV server measurement. Section 3 presents some existing WebDAV clients that can be used to test WebDAV servers. In section 4, we present the design details for Prestan, including our client-side measurement approach, the method of response time computation, measurement accuracy issues, etc. Section 5 gives the implementation details for each method. In section 6, we use Prestan to measure the performance of a group of WebDAV servers along with analysis of these results. Finally, section 7 summarizes the contributions of this work.

2 Background

This section briefly describes version 1.1 of the HTTP protocol, and then introduces WebDAV, which extends the functionality of HTTP1.1 for remote collaborative authoring.

2.1 HTTP 1.1

The Hypertext Transfer Protocol (HTTP) [3] is a widely used application-level protocol for distributed, hypermedia information systems. It defines eight methods: GET, PUT, POST, OPTIONS, HEAD, TRACE, DELETE, and CONNECT. In practice, however, only GET, POST and CONNECT are widely used by ordinary browsers. PUT and DELETE were designed to allow some limited authoring functionality, but in practice they are not defined well enough to be useful.

To enhance the performance of HTTP, one interesting enhancement in HTTP1.1 is supporting “persistent connections” [3, 6, 7], whose basic idea is sending multiple requests over a single TCP connection. There are two kinds of features in this technique, connection “keep-alive” and “pipelining”.

“Keep-alive” allows multiple HTTP requests to share a single TCP connection, which can greatly reduce the *slow start* [12, 17] between a sequence of operations (such as 1000 consecutive requests in the same test) by avoiding multiple TCP opens and closes.

“Pipelining” allows consecutive HTTP requests to be sent without waiting for the previous response; accordingly, multiple requests and responses can be contained in a single TCP segment. This technique brings no benefit to the response time of individual requests, but can greatly improve client-side throughput. Some existing testing tools take advantage of this technique to overload the server without adopting the complicated implementations of S-Client [10] technique, which consists of a pair of processes, one process is responsible for generating HTTP requests and the other handles the HTTP response.

2.2 WebDAV

WebDAV [1], Web Distributed Authoring and Versioning, is a suite of protocol extensions to HTTP/1.1. It transforms the read-only web into a writeable medium permitting collaborative authoring and management of resources and properties. Unlike SOAP [14], which is layered on top of HTTP, WebDAV is an extension to HTTP. Without changing existing HTTP functionality, WebDAV adds new authoring functionalities, including file storage, directory management and collaborative authoring.

The WebDAV protocol [2] supports all HTTP methods, including OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, and CONNECT. There are added new features such as locking (concurrency control), properties, and namespace manipulations. Table 1 lists all seven new methods supported by WebDAV, as defined in RFC 2518 [17].

PROPPATCH	Set and/or remove properties defined on a resource
PROPFIND	Retrieve property values from a resource
MKCOL	Create new empty collections
MOVE	Move resources or collections
COPY	Create duplicate resources or collections
LOCK	Lock a resource/collection to avoid overwrite conflicts
UNLOCK	Unlock an locked resource/collection to make it available for writing

Table 1: Seven WebDAV Methods

Before WebDAV, it was difficult for people to collaborate on Web-based documents using tools from multiple readers due to the lack of a standard way to synchronize the activities of different authors. WebDAV solves this problem with the introduction of locks. Similar to file system, there are two kinds of locks in WebDAV: shared locks and exclusive locks. Depending on whether or not the LOCK/UNLOCK methods are supported, WebDAV servers can be categorized into two classes. Class 1 WebDAV servers must support all WebDAV methods except for LOCK and UNLOCK, while Class 2 WebDAV server must support all WebDAV methods in RFC 2518 [17].

Another new feature of WebDAV is supporting property management. Properties in WebDAV are essentially the metadata of documents, and are organized as triples of namespace, name and value. The namespace in the triple is an XML namespace. The property name is represented as an XML element name. The property value is represented as a sequence of well-formed XML. With the combination of namespace and property name, the type of property is determined. Properties are represented during network transmission as XML, but can be stored in a wide range of repositories. Some servers implement a WebDAV repository on top of file system, such as Apache [5], Sambar [16], SunONE WS6.1, and Microsoft IIS. To better support DASL, some servers implement the WebDAV repository on top of a database system, examples including Catacomb [4], SoftwareAG Tamino [18], Xythos, and Oracle's WebDAV support.

There are two kinds of properties in WebDAV, live and dead. Live properties are server-defined properties, whose semantics and syntax are enforced by the server; dead properties are user-defined, whose semantics and syntax are not enforced by the server. For example, 'getcontentlength', 'creationdate', 'getlastmodified' are live properties for all WebDAV resources. Properties like 'author' and 'keywords' are dead properties, whose values must be updated by the client.

3 Related work

There are a wide variety of WebDAV clients that can be used to test WebDAV servers in both industry and academia.

Cadaver [4] is a widely used WebDAV client that supports file upload, download, namespace operations and lock operations. Although it supports all the methods specified in RFC2518, these operations need to be performed manually, thereby greatly reducing the utility of Cadaver for automated testing.

Davtool [4] is another WebDAV client with command line style that can perform all WebDAV methods. Unlike Cadaver, Cadaver can work within shell scripts, although it solved the inefficiency problem in Cadaver by supporting batch work, it still does not support performance measurement.

Litmus [4] is a popular WebDAV server compliance test suite, which can comprehensively test all WebDAV methods, ranging from property manipulation to locking management. However, these tests are just WebDAV compliance tests. Similar to the previous three tools, it does not support performance measurement.

Apache benchmark [13] is a widely used web server performance test suite. However, it is restricted to the existing HTTP methods, and does not support WebDAV methods like PROPFIND and MKCOL.

There are undoubtedly some performance measuring tools developed by tester working for specific WebDAV vendors, however, none of these performance testing tools has been publicly released for use by the WebDAV community.

4 Design

A good test tool should correctly and accurately reflect the status of the target system. In this section, we will present our measurement approach, and describe how to bound the response time, how to avoid delays incurred by TCP algorithms, and how to alleviate network latency.

Generally, server performance can be measured at two places, the server side or the client side. Server side measurements provide us with detailed information about the server, but incur some measurement overhead and may impair the accuracy of measurements. In contrast, the client-side approach introduces no overhead to the server, thereby more precisely reflecting the performance behaviors of servers. The problem with this approach is the interference caused by network latency. However, after examining the interaction between client and server and carefully setting up the experimental environment, the accuracy of this approach can be greatly improved.

4.1 Response time

As there are two places for performing measurements (server or client), response time can be defined in two ways.

From the aspect of servers, system response time is the interval between the receipt of the end of transmission of an request message and the beginning of the transmission of a response message to the station originating the inquiry.

From the aspect of clients, the response time is the interval between the last byte sent and the first byte received. Since we take the client-side approach, our measurement should conform to this definition.

In Neon library [4], used by the Prestan tool, WebDAV method is performed using the following sequence: Build request → Open connection (if exists, reuse it) → Send headers → Send request body → Read response headers → Read response blocks → End request. Based on the client side definition, the response time should be the time between the end of "send request body" step until just before the "read response headers" step.

4.2 Nagle algorithm

Nagle algorithm [8] is implemented at TCP layer to reduce the number of small segments by delaying their transmission. The algorithm states that if a given connection has outstanding data, then no small packets will be sent until the existing data is acknowledged. Thus, TCP will always send a full-sized packet if possible.

Nagle algorithm often interacts with TCP delayed ACK algorithm [17], which causes TCP to not send an ACK immediately when it receives data. Instead, an ACK will only be sent after delay, the hope being that during this delay there will be additional data to send back, allowing the ACK to piggyback on this data, thereby saving one TCP transmission segment.

Though the Nagle algorithm improves network efficiency, it can increase client side response time, especially for small requests that cannot fill a full packet, thereby impairing the accuracy of the measurement. In our testing tool, the HTTP request headers and request body were sent out using separate socket WRITEs. This caused the second WRITE will to not be sent until receiving the ACK of the first WRITE. However, the server cannot process the entire HTTP request without receiving the request body, and therefore it delays sending ACK for the first WRITE. Figure 1 shows the interaction of Nagle algorithm with delayed-ACK and Figure 2 shows the effect after disabling the Nagle algorithm.

Our initial experimental results showed that PROPPATCH for a single property was unreasonably 30ms slower than PROPPATCH for multiple properties. After ruling out the possible factors such as the inefficiency of XML parsers, we found out that the Nagle algorithm was reason. After disabling this algorithm, the response time for single property PROPPATCH was almost 30ms faster than before, and the improved methodology now more precisely reflects the state of the WebDAV server being measured.

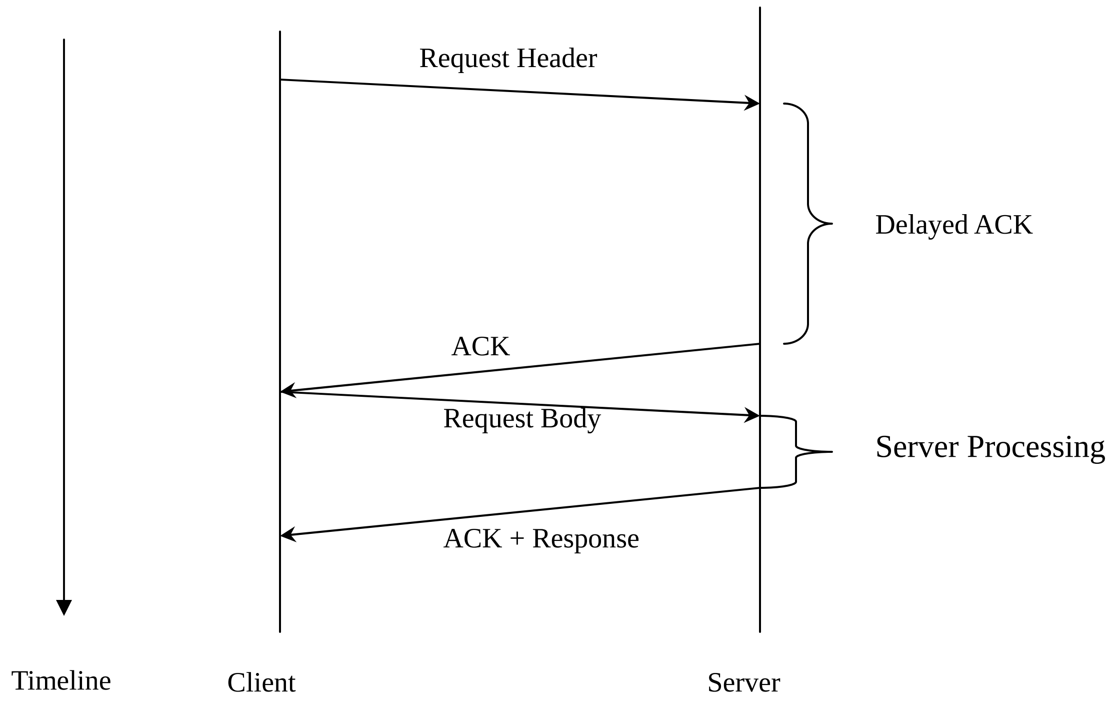

Figure 1: The interaction of Nagle Algorithm with Delayed-ACK

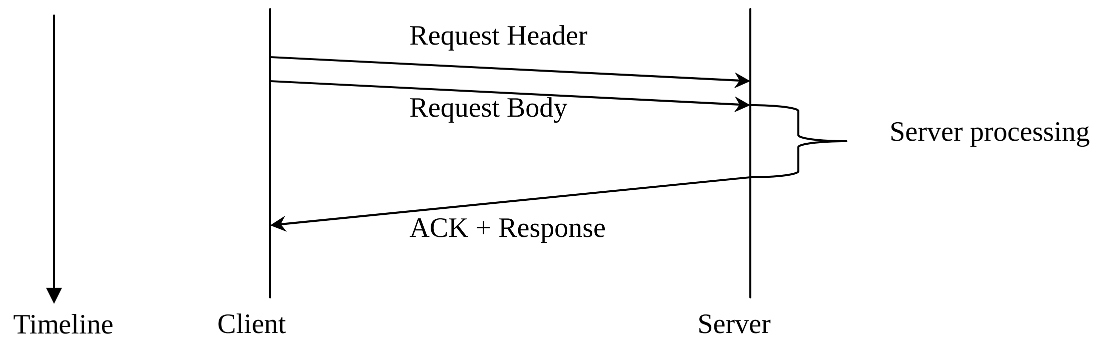

Figure 2: Avoidance of Nagle Algorithm

4.3 Network latency

As mentioned in section 4.1, one difficulty of the client-side approach is the interference of network latency. Generally, network latency contains two components, the transmission delay and the propagation delay.

The transmission delay is the time required to transmit the packet into the link. In our environment, this delay is mostly caused by client-side queuing. Since we disabled the Nagle algorithm, the transmission delay is negligible.

Propagation delay is the time required to propagate data from one end to the other end of the link that connects two computers. This delay in our environment is also negligible, since the test client and servers under test are directly connected by a fullplex100M-Based Ethernet link.

4.4 Concurrent model

To better characterize WebDAV server scalability, we introduce the notion of "concurrency level" into our measurements. Concurrency level captures the degree to which many requests are being made simultaneously. Our concurrency model is based on the WebStone [9, 11, 15] benchmark, which consists of one webmaster and multiple web clients. A webmaster is a process in charge of a group of client processes, and its responsibilities include spawning web-client processes, starting them simultaneously, and collecting testing results back. Web clients are a group of processes issuing requests and directly performing measurements on the WebDAV server.

One difficulty of the concurrency model is guaranteeing that requests are sent simultaneously. As shown in figure 3, a group of clients are started by the webmaster almost at the same time and each of them sends out a continuous sequence of requests. Certainly, those processes cannot be started exactly at the "same time" in a shared time operating system. To attack this problem, we only collect the data from time space in which there are overlapped requests. Due to the overhead of process context switching, the maximum concurrency level supported by Prestan is fifteen.

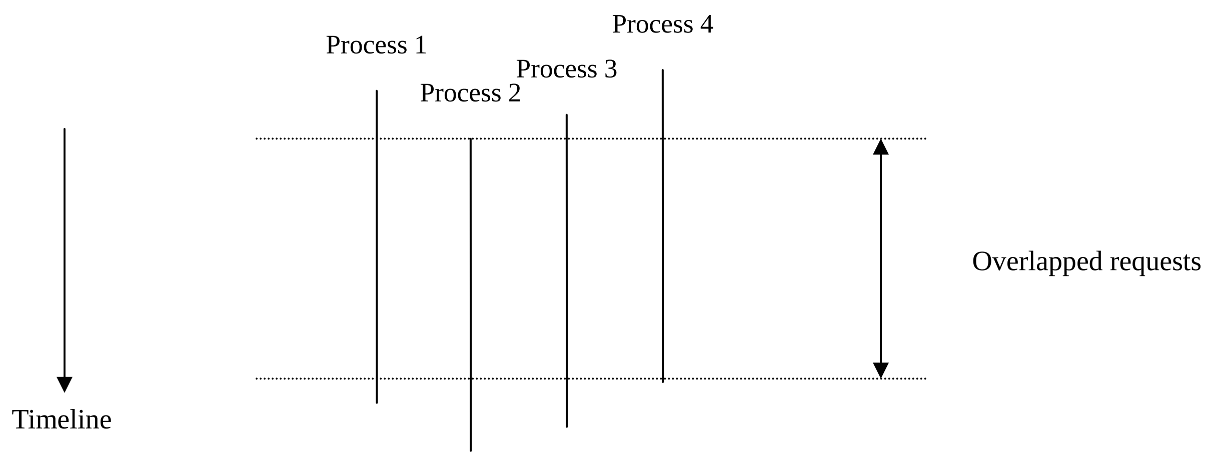

Figure 3: Approach to Obtain Simultaneous Requests

5 Implementation

This section describes the implementation of the Prestan test suite in detail. For each test, we first introduce an objective, and then provide the WebDAV method request and XML request message bodies to illustrate the implementation in detail. In particular, we use "X-Prestan" header to identify each test case of Prestan in the log file, each X-Prestan header includes test case number and name.

According to the functionalities of these methods, we divide them into four categories: properties manipulation, resource management, namespace management and locking management.

As mentioned in Secion 4, each test case was repeatedly sent a certain number of times (the minimum is 100 times in our configuration) and the measurements were gathered only from successful responses, the failed ones were filtered out.

5.1 Properties manipulation

There are two kinds of properties in WebDAV, live properties and dead properties.

Live properties are a fixed number of pre-defined metadata items that are automatically updated by the server. In contrast, the dead properties are unlimited, and can be arbitrarily added and deleted.

5.1.1 PROPPATCH

- Objective: Measure the performance of writing properties. As we mentioned before, a live property is pre-defined and cannot be explicitly changed. This test will focus on writing dead properties. Also, since PROPPATCH collection with "Depth Infinity" is not specified in RFC 2518, we only focus on PROPATCH for a single resource. According to the number of dead properties being set, we divided this method into two classes: PROPPATCH of a single property and PROPPATCH of multiple properties.
- Implementation:

Upload a single resource to the server, and then repeatedly perform Proppatch operations on this resource.

The test can write a:

- Single dead property
- Multiple dead properties (the number of dead properties is configurable, with 10 as the default).

In the figure below, the top box show the HTTP request line and headers while the bottom box gives the HTTP request body.

Method: PROPPATCH [Dead, multiple properties on single resource]

```
PROPPATCH /basic/davtest/prop HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Content-Length: 1065
Content-Type: application/xml
X-Prestan: (null): 4 (propfinddead)
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:propertyupdate xmlns:D="DAV:"><D:set><D:prop><prop0
xmlns="http://webdav.org/neon/DavTester/">value0</prop0></D:prop></D:set>
<D:set><D:prop><prop1
xmlns="http://webdav.org/neon/DavTester/">value1</prop1></D:prop></D:set>
<D:set><D:prop><prop2
xmlns="http://webdav.org/neon/DavTester/">value2</prop2></D:prop></D:set>
<D:set><D:prop><prop3
xmlns="http://webdav.org/neon/DavTester/">value3</prop3></D:prop></D:set>
<D:set><D:prop><prop4
xmlns="http://webdav.org/neon/DavTester/">value4</prop4></D:prop></D:set>
<D:set><D:prop><prop5
xmlns="http://webdav.org/neon/DavTester/">value5</prop5></D:prop></D:set>
<D:set><D:prop><prop6
xmlns="http://webdav.org/neon/DavTester/">value6</prop6></D:prop></D:set>
<D:set><D:prop><prop7
xmlns="http://webdav.org/neon/DavTester/">value7</prop7></D:prop></D:set>
<D:set><D:prop><prop8
xmlns="http://webdav.org/neon/DavTester/">value8</prop8></D:prop></D:set>
<D:set><D:prop><prop9
xmlns="http://webdav.org/neon/DavTester/">value9</prop9></D:prop></D:set>
</D:propertyupdate>
```

Method: PROPPATCH [Dead, single property on single resource]

```
PROPPATCH /basic/davtest/prop HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Content-Length: 192
Content-Type: application/xml
X-Prestan: (null): 3 (proppatch)
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:propertyupdate xmlns:D="DAV:"><D:set><D:prop><prop0
xmlns="http://webdav.org/neon/DavTester/">value0</prop0></D:prop></D:set>
</D:propertyupdate>
```

5.1.2 PROPFIND

- ♦ Objective: Measure the performance of reading properties. Performance measurement of the PROPFIND method is more complicated than for PROPPATCH since it can be applied to live properties. Since some servers turn off the “Depth Infinity” option for collection operations, we only focus on PROPFIND for a single resource. Similar to PROPPATCH, we classify tests by the number of properties: PROPFIND single property and PROPFIND multiple properties. We additionally divide tests into those for live or dead properties.
- ♦ Implementation.

Upload a single resource to the server, PROPFIND corresponding properties to the object, and then repeatedly perform a PROPFIND operation on this resource.

The test can read a:

- o Single dead property
- o Multiple dead properties (the number of dead properties is configurable, 10 is the default).
- o Single live property
- o Multiple live property (11 common used live properties)

Method: PROPFIND [Dead, multiple properties on single resource]

```
PROPFIND /basic/davtest/prop HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Depth: 0
Content-Length: 611
Content-Type: application/xml
```

```
X-Prestan: (null): 4 (propfinddead)
```

```
<?xml version="1.0" encoding="utf-8"?>
<propfind xmlns="DAV:"><prop>
<prop0 xmlns="http://webdav.org/neon/DavTester/"/>
<prop1 xmlns="http://webdav.org/neon/DavTester/"/>
<prop2 xmlns="http://webdav.org/neon/DavTester/"/>
<prop3 xmlns="http://webdav.org/neon/DavTester/"/>
<prop4 xmlns="http://webdav.org/neon/DavTester/"/>
<prop5 xmlns="http://webdav.org/neon/DavTester/"/>
<prop6 xmlns="http://webdav.org/neon/DavTester/"/>
<prop7 xmlns="http://webdav.org/neon/DavTester/"/>
<prop8 xmlns="http://webdav.org/neon/DavTester/"/>
<prop9 xmlns="http://webdav.org/neon/DavTester/"/>
</prop></propfind>
```

Method: PROPFIND [Dead, single property on single resource]

```
PROPFIND /basic/davtest/prop HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Depth: 0
Content-Length: 143
Content-Type: application/xml
X-Prestan: (null): 4 (propfinddead)
```

```
<?xml version="1.0" encoding="utf-8"?>
<propfind xmlns="DAV:"><prop>
<prop0 xmlns="http://webdav.org/neon/DavTester/"/>
</prop></propfind>
```

Method: PROPFIND [Live, multiple properties on single resource]

```
PROPFIND /basic/davtest/prop HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Depth: 0
Content-Length: 404
Content-Type: application/xml
X-Prestan: (null): 5 (propfindlive)
```

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<propfind xmlns="DAV:"><prop>
<creationdate xmlns="DAV:"/>
<getlastmodified xmlns="DAV:"/>
<resourcetype xmlns="DAV:"/>
<supportedlock xmlns="DAV:"/>
<lockdiscovery xmlns="DAV:"/>
<getcontentlength xmlns="DAV:"/>
<getetag xmlns="DAV:"/>
<getcontentlanguage xmlns="DAV:"/>
<getcontenttype xmlns="DAV:"/>
<supportedlock xmlns="DAV:"/>
</prop></propfind>
```

Method: PROPFIND [Live, single property on single resource]

```
PROPFIND /basic/davtest/prop HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Depth: 0
Content-Length: 121
Content-Type: application/xml
X-Prestan: (null): 5 (propfindlive)
```

```
<?xml version="1.0" encoding="utf-8"?>
<propfind xmlns="DAV:"><prop>
<creationdate xmlns="DAV:"/>
</prop></propfind>
```

5.2 Resource management

5.2.1 PUT

- Objective: This test measures the performance of writing resource data to a web server. PUT is a commonly used method. For example, authors often need to periodically refresh their updates to the server and this causes the PUT method to be invoked frequently. To characterize the *write* capability of a WebDAV server under different conditions, we perform this method on three kinds of files: small, medium and large, corresponding to file sizes of

1K, 64K and 1M respectively.

- Implementation:
  - DELETE the resource if it exists, and then PUT single resource with different size of 1K, 64K and 1M respectively, finally clean up the resource by deleting it at the end of all tests.

Method: PUT [1K Bytes single resource]

```
PUT /basic/davtest/res HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Content-Length: 1024
X-Prestan: (null): 6 (put1K)
```

```
This is Prestan test file.
This is Prestan test file.
…
```

Method: PUT [64K Bytes single resource]

```
PUT /basic/davtest/res HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Content-Length: 65536
X-Prestan: (null): 1 (put64K)
```

```
This is Prestan test file.
This is Prestan test file.
This is Prestan test file.
This is Prestan test file.
…
```

Method: PUT [1024K Bytes single resource]

```
PUT /basic/davtest/res HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
```

```
Connection: TE
TE: trailers
Content-Length: 1048576
X-Prestan: (null): 8 (put1024K)
```

```
This is Prestan test file.
This is Prestan test file.
This is Prestan test file.
This is Prestan test file.
This is Prestan test file.
This is Prestan test file.
…
```

5.2.2 GET

- Objective: This test measures the performance of reading resource data from a WebDAV server. GET is by far the frequently used method. Users often need to retrieve resources from the server, either by browsing or editing. To characterize the *read* capability of a WebDAV server under different conditions, we perform this method on three kinds of files: small, medium and large, corresponding to file sizes of 1K, 64K and 1M respectively.
- Implementation:
  - PUT a resource with size of 1K, 64K and 1M respectively, and then repeatedly perform GET on this resource. Note that there is no request body for the following examples, as is typical for GET.

Method: GET [1K Bytes single resource]

```
GET /basic/davtest/res HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
X-Prestan: (null): 6 (get1K)
```

Method: GET [64K Bytes single resource]

```
GET /basic/davtest/res HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
X-Prestan: (null): 7 (get64K)
```

Method: GET [1024K Bytes single resource]

```
GET /basic/davtest/res HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
X-Prestan: (null): 8 (get1024K)
```

5.3 Namespace Management

Namespace management operations are provided in WebDAV to support the needs of the authoring clients and servers that expect a hierarchical namespace. The MKCOL, COPY and MOVE methods are designed to manage such namespaces, and operate much as their operating system counterparts.

5.3.1 MKCOL

- Objective: This experiment tests the performance of creating an empty collection.
- Implementation:
    - A collection hierarchy is created that is 10 levels deep and 100 wide at the bottom level. That is, there are 100 sub-collections at the bottom level.

Note that the MKCOL method does not take a request body.

Method: MKCOL

```
MKCOL /basic/davtest/coll/ HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
X-Prestan: (null): 10 (my_collection)
```

5.3.2 COPY

- Objective: This test measures the performance of the COPY operation for a single resource copy, a deep collection COPY.
- Implementation:

Create either a single resource or a deep, wide collection to the server, and repeatedly copy it to a different location with the “overwrite” header set to true (overwrite the destination); only the copy operation is measured.

The resources being copied are:

- Single resource (1K Bytes)
- Collection 10 levels deep with 100 resources at the bottom level

Note that the COPY method does not take a request body.

Method: COPY [Single resource]

```
COPY /basic/davtest/copy HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Depth: infinity
Destination: http://dav.cse.ucsc.edu:8282/basic/davtest/copy_dest
Overwrite: T
X-Prestan: (null): 1 (begin)
```

Method: COPY [Collection with depth 10 and width 100]

```
COPY /basic/davtest/copy_col/ HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Depth: infinity
Destination: http://dav.cse.ucsc.edu:8282/basic/davtest/copy_col_dest/
Overwrite: T
X-Prestan: (null): 10 (my_collection)
```

5.3.3 MOVE

- Objective: This test measures the performance of the MOVE operation for a single resource copy, a deep collection MOVE.
- Implementation:

  Create either a single resource or a deep, wide collection on the server, and repeat the following two steps: (1) Copy A to B, (2) Move B to C with "overwrite" header set to true. Note that only step (2) is measured.

  The object could be:

  - Single resource (1K Bytes)
  - Collection 10 levels deep with 100 resources at the bottom level

  Note that the MOVE method does not take a request body.

Method: MOVE [Single resource]

```
MOVE /basic/davtest/move HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Destination: http://dav.cse.ucsc.edu:8282/basic/davtest/move_dest
Overwrite: T
X-Prestan: (null): 1 (begin)
```

Method: MOVE [Collection with depth 10 and width 100]

```
MOVE /basic/davtest/move_col/ HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Destination: http://dav.cse.ucsc.edu:8282/basic/davtest/move_col_dest/
Overwrite: T
X-Prestan: (null): 10 (my_collection)
```

5.3.4 DELETE

- Objective: This test measures the performance of the DELETE operation for a single resource copy, a deep collection DELETE.
- Implementation:

Create either a single resource or a deep, wide collection on the server, and repeat the following two steps: (1) Copy A to B, (2) Delete B. Note that only step (2) is measured.

The object could be:

- Single resource (1K Bytes)
- Collection 10 levels deep with 100 resources at the bottom level

Note that the DELETE method does not take a request body.

Method: DELETE [Single resource]

```
DELETE /basic/davtest/delete HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
X-Prestan: (null): 1 (begin)
```

Method: DELETE [Collection with depth 10 and width 100]

```
DELETE /basic/davtest/delete_col/ HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
X-Prestan: (null): 10 (my_collection)
```

5.4 Lock management

Locking is a WebDAV feature that is used to prevent overwrite conflicts among concurrently active authors. Two kinds of locks are defined in RFC 2518, shared locks and exclusive locks. Each collaborating author should request an exclusive lock before writing the object, and unlock it after accomplishing the task in order to let other authors access it. Most WebDAV servers only implement exclusive locks. Due to this, we only test the performance of exclusive locking functionality. Furthermore, since you need to lock a resource to unlock it, LOCK and UNLOCK are tested together.

5.4.1 LOCK/UNLOCK

- Objective: This test measures the performance of the LOCK/UNLOCK pair for a single resource copy, and for a deep collection. All locks have a 3600 seconds duration, and we assume the lock duration does not have a substantial impact on lock performance.
- Implementation:

  We perform an exclusive LOCK/UNLOCK on the following objects:

  - Single resource (1K Bytes)
  - Collection 10 levels deep with 100 non-collection resources at the bottom level

  Note that the LOCK method does not take a request body.

Method: LOCK [Single resource with Exclusive mode]

```
LOCK /basic/davtest/lockme HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Content-Length: 179
Content-Type: application/xml
Depth: 0
Timeout: Second-3600
X-Prestan: (null): 11 (locks)
```

```
<?xml version="1.0" encoding="utf-8"?>
<lockinfo xmlns='DAV:'>
<lockscope><exclusive/></lockscope>
<locktype><write/></locktype><owner>Prestan test suite</owner>
</lockinfo>
```

Method: LOCK [Collection with Exclusive mode]

```
LOCK /basic/davtest/lockme2/ HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Content-Length: 179
Content-Type: application/xml
Depth: infinity
Timeout: Second-3600
X-Prestan: (null): 11 (locks)
```

```
<?xml version="1.0" encoding="utf-8"?>
<lockinfo xmlns='DAV:'>
<lockscope><exclusive/></lockscope>
<locktype><write/></locktype><owner>Prestan test suite</owner>
</lockinfo>
```

Method: UNLOCK [Single resource]

```
UNLOCK /basic/davtest/lockme HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Lock-Token: <opaquelocktoken:d2ac996a-28c7-0310-a170-8d162c9b6438>
X-Prestan: (null): 11 (locks)
```

Method: UNLOCK [Collection]

```
UNLOCK /basic/davtest/lockme2/ HTTP/1.1
Host: dav.cse.ucsc.edu:8282
User-Agent: davtest/0.9.2 neon/0.24.0-dev
Connection: TE
TE: trailers
Lock-Token: <opaquelocktoken:1a309d6a-28c7-0310-95e1-92bacfb4327d>
X-Prestan: (null): 11 (locks)
```

5.5 The user interface of Prestan

Prestan is a command line tool, with the following input parameters and options:

Usage: Prestan [http://]hostname[:port]/path [username password] [options]

Option:

-r, --requests	Number of repeat runs (Default: 10)
-c, --concurrency	Number of concurrent connections (Default: 1)
-p, --properties	Number of dead properties (Default: 10)
-d, --depth	Depth of collection (Default: 10)
-w, --width	Width of collection at the bottom level (Default: 100)
-o, --output	Output file

Example: Prestan http://dav.cse.ucsc.edu:81/basic test1 test1 -r 100 -p 20

Requests: If "-requests" is specified, Prestan will repeat the methods comprising each test corresponding number of times. For example, to measure the average response time for the PROPFIND live single property test for a single resource, we issue a PROPFIND method "-requests" number of times, measuring each request response time. Afterwards, Prestan computes and reports the average response time.

Concurrency: If “-concurrency” is specified, Prestan will indicate “webmaster” building up corresponding number of concurrent client connections.

Properties: If “-properties” is specified, Prestan will apply corresponding number of properties on “Proppatch/Propfind” method.

Depth: If “-depth” is specified, Prestan will apply corresponding depth of collection operations on “Copy/Move/Delete/Lock/Unlock” method.

Width: If “-width” is specified, Prestan will apply corresponding width of collection operations on “Copy/Move/Delete/Lock/Unlock” method.

Output: If “-output” is specified, Prestan will redirect the standard output to the corresponding output file.

Before performing the measurements, Prestan needs to set up the experiential environment: (1) Create a temporary directory called PrestanTest on the server, (2) Pre-send a set of WebDAV methods to warm up any server-side repository cache (i.e., database cache used by the user to store resources), (3) Perform the tests, (4) Delete “PrestanTest” after clean up its underneath resources.

5.6 Prestan Output Format

Output of Prestan contains three parts: copyright information, test configuration and test results. The left column of the results represents the methods tested, the right column is the average response time in terms of microsecond. Below is an example.

```
>    Prestan test.webdav.org/dav

Prestan, Version 2.0.3
Copyright(c) 2003 Teng Xu, GRASE Research Group at UCSC
http://www.soe.ucsc.edu/research/labs/grase

Server Warming Up.......................Done
```

```
Start Testing test.webdav.org/dav:

          *********************************
          * Number of Requests           100
          * Number of Dead Properties    10
          * Depth of Collection          10
          * Width of Collection          100
          * Type of Methods              WebDAV
          *********************************

ProppatchMult             Rsp = 14248 [us]
ProppatchSingle           Rsp = 14240 [us]
PropfindDeadMult          Rsp = 13894 [us]
PropfindDeadSingle        Rsp = 13730 [us]
PropfindLiveMult          Rsp = 15025 [us]
PropfindLiveSingle        Rsp = 14104 [us]
Put1K                     Rsp = 14215 [us]
Get1K                     Rsp = 14229 [us]
Put64K                    Rsp = 88967 [us]
Get64K                    Rsp = 31171 [us]
Put1024K                  Rsp = 487773 [us]
Get1024K                  Rsp = 363265 [us]
Copy                      Rsp = 14125 [us]
Move                      Rsp = 14044 [us]
Delete                    Rsp = 13758 [us]
```

MkCol	Rsp = 14266 [us]
CopyCol	Rsp = 43314 [us]
MoveCol	Rsp = 45393 [us]
DeleteCol	Rsp = 37217 [us]
Lock	Rsp = 14613 [us]
UnLock	Rsp = 13745 [us]
LockCol	Rsp = 30966 [us]
UnLockCol	Rsp = 23005 [us]

6 Experiments

In this section, we use Prestan to measure the average response time (ms) of various WebDAV servers, along with analyzing the test results.

6.1 Practice One

To better understand the bottleneck of the WebDAV server, we performed comparison of the performance of Apache and Catacomb. Apache WebDAV repository is implemented on top of file system while Catacomb is implemented on top of MySQL database. Besides, Catacomb supports DASL protocol. Both servers are installed on the same machine, and we run all tests on the same client machine, which guarantees their measurements are comparable.

The hardware/software configuration of the server machine is as below:

CPU	Intel(R) Pentium(R) 4 CPU 1.70GHz
RAM	512MB
Disk	MAXTOR 6L040L2, ATA DISK drive 40GB IDE
NIC	3Com PCI 3c905C Tornado
OS	Linux RedHat 7.0

The hardware/software configuration of the client machine is as below:

CPU	Intel(R) Pentium(R) 4 CPU 1.70GHz
RAM	512MB
Disk	MAXTOR 6L040L2, ATA DISK drive 40GB IDE
NIC	3Com PCI 3c905C Tornado
OS	Linux RedHat 7.0

The client and server are connected through 100M-based LAN, and the PING time is 100 ~ 150 us.

As shown in Table 2, generally Catacomb will be slower than Apache. Particularly, the collection operations (Copycol, Movecol, Deletecol) are much slower than those on ServerB. These operations seem to be the bottleneck, which arouse our curiosity to find out the reason.

	Catacomb (ms)	Apache (ms)	Catacomb/Apache
ProppatchMult	30	3	10.00
ProppatchSingle	8	3	2.67
PropfindDeadMult	8	3	2.67
PropfindDeadSingle	7	3	2.33
PropfindLiveMult	8	5	1.60
PropfindDeadSingle	7	2	3.50
Put1K	8	2	4.00
Get1K	4	3	1.33
Put64K	13	7	1.86
Get64K	11	8	1.38
Put1024K	128	91	1.41
Get1024K	112	99	1.13
Copy	16	4	4.00
Move	16	4	4.00
Delete	9	2	4.50
MkCol	11	3	3.67
CopyCol	*280*	*20*	*14.00*
MoveCol	*477*	*16*	*29.81*
DeleteCol	*328*	*12*	*27.33*

Lock	7	3	2.33
UnLock	8	2	4.00
LockCol	*344*	*10*	*34.40*
UnLockCol	*251*	*7*	*35.86*

Table 2: Performance Comparison of Catacomb and Apache

Since those methods are performed on a very deep (10 levels) and very wide (100 resources at the bottom level) collection, two possible factors could affect the performance of collection operations. One is the collection depth, the other is the collection width. Therefore, we made the following experiments to find which factor is the determinant.

	Catacomb(ms)	Apache(ms)	Catacomb/Apache
CopyCol	280	20	14.00
MoveCol	477	16	29.81
DeleteCol	328	12	27.33

Table 3: Catacomb vs. Apache with Depth 10 and Width 100

	Catacomb(ms)	Apache(ms)	Catacomb/Apache
CopyCol	261	17	15.35
MoveCol	436	13	33.54
DeleteCol	301	9	33.44

Table 4: Catacomb vs. Apache with Depth 2 and Width 100

	Catacomb(ms)	Apache(ms)	Catacomb/Apache
CopyCol	79	7	11.29
MoveCol	116	7	16.57
DeleteCol	77	5	15.40

Table 5: Catacomb vs. Apache with Depth 10 and Width 10

The original comparison between Catacomb and Apache is shown in Table 3. Table 4 shows the effect after reducing the collection depth from 10 to 2. We found that the gap (the ratio Catacomb/Apache) between Catacomb and Apache remains roughly the same. This rules out the possibility that collection depth is the determinant.

Next, as shown in Table 5, we performed a second experiment in which we shrunk the width of the collection from 100 to 10. The gap between the two servers decreased dramatically (50% ~ 100%), which proves that the performance bottleneck is associated with collection width.

To give more details about the effect of collection width, we compare the scalability of two servers on methods CopyCol, MoveCol and DeleteCol respectively.

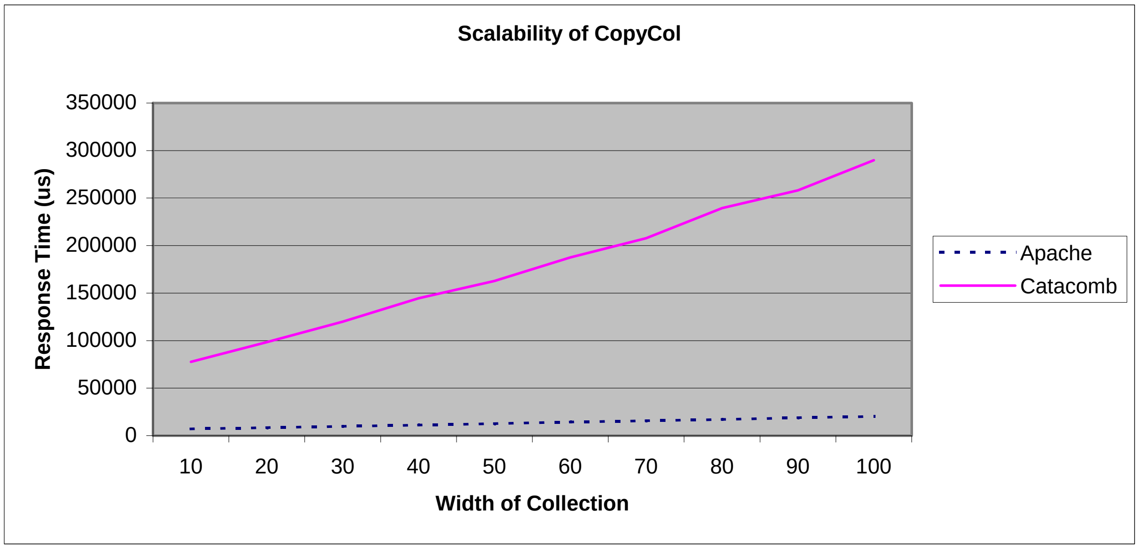

Figure 4: Scalability of CopyCol

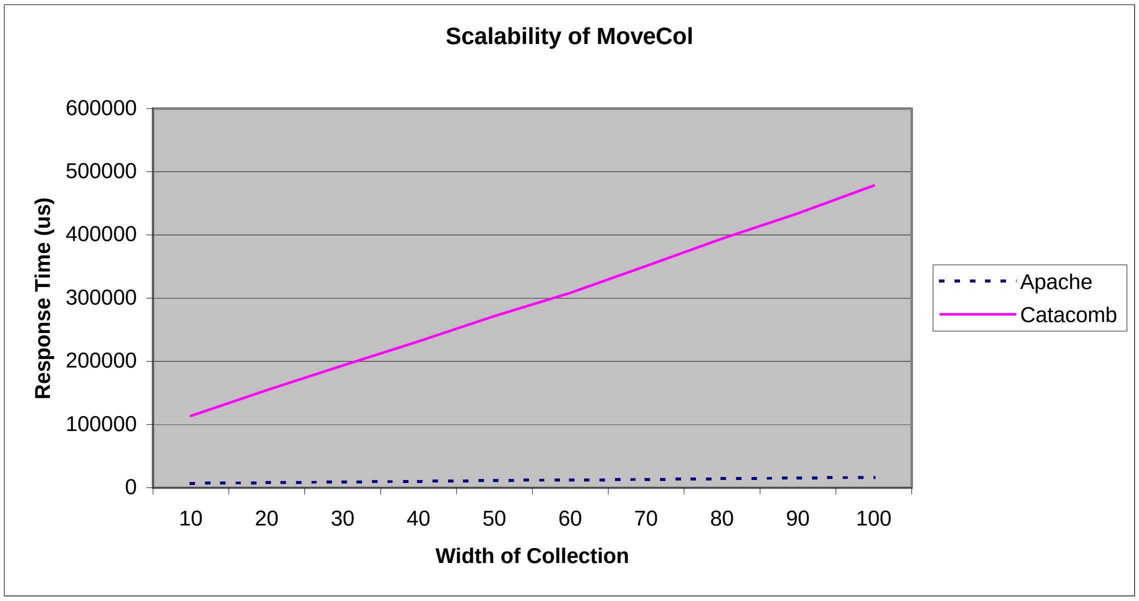

Figure 5: Scalability of MoveCol

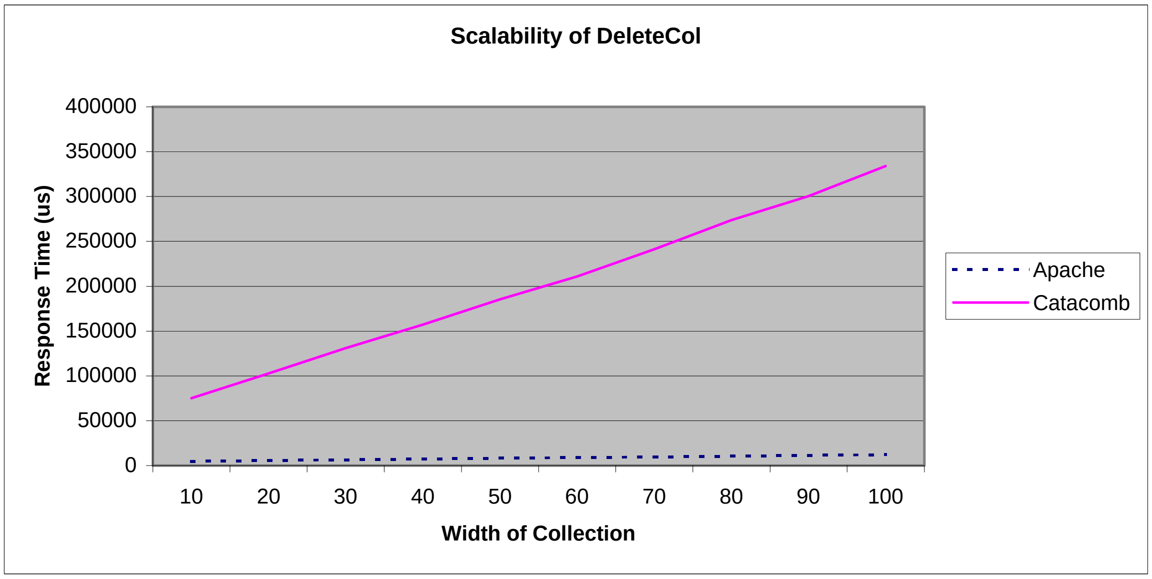

Figure 6: Scalability of DeleteCol

As shown in above three figures, Apache's response time almost remains the same when collection width shrinking from 10 to 100; while Catacomb does not scale well, with a response time that linearly increases much faster when the collection width grows.

6.2 Practice Two

We tested against various WebDAV servers from 12 companies at the annual WebDAV Interoperability Testing Event at UCSC, in September 2004. The goal of the event is to gather together, in one physical location, developers and testers of WebDAV, DASL, DeltaV, and ACL clients and servers so they can exercise as many client/server pairs as possible. Ideally, all functionality of each client will be tested against every server.

As we mentioned in section 4, all these servers are in the same room, most of which them are run on laptops. All of these tests are performed locally through high-speed dedicated network (test client and WebDAV server are directly connected through a 100M-based fullplex Ethernet link), which makes the network delay negligible.

The performance comparison of these servers is shown in Figure 7 and 8. Figure 7 gave a comparison on all WebDAV methods. But, Copycol, Movecol and Deletecol are so outstanding that decrease the granularity of Y-axis. Therefore, we presented the comparison in Figure 8 by filtering out the three methods.

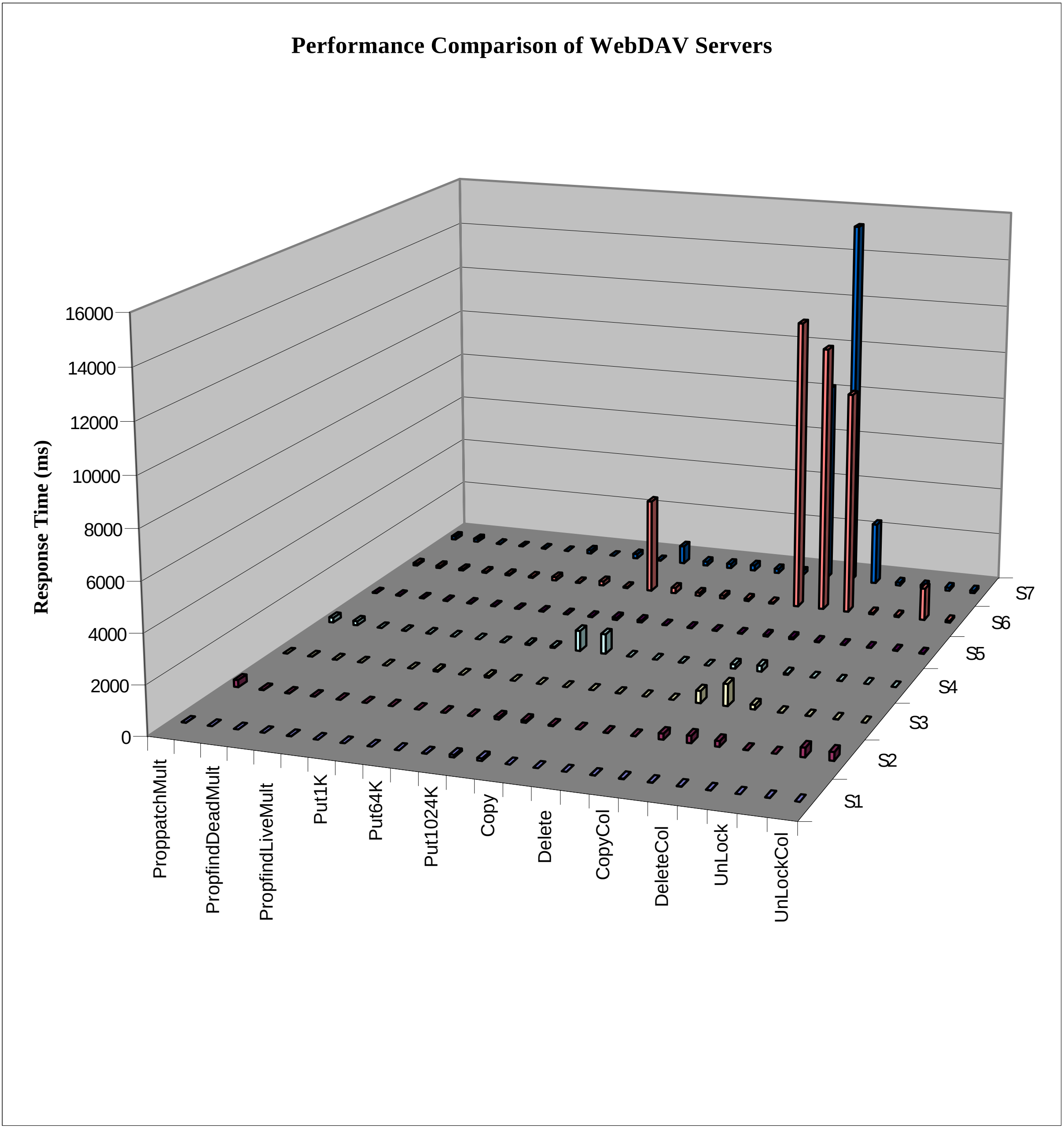

Figure 7: Performance Comparison of WebDAV Servers (All Methods)

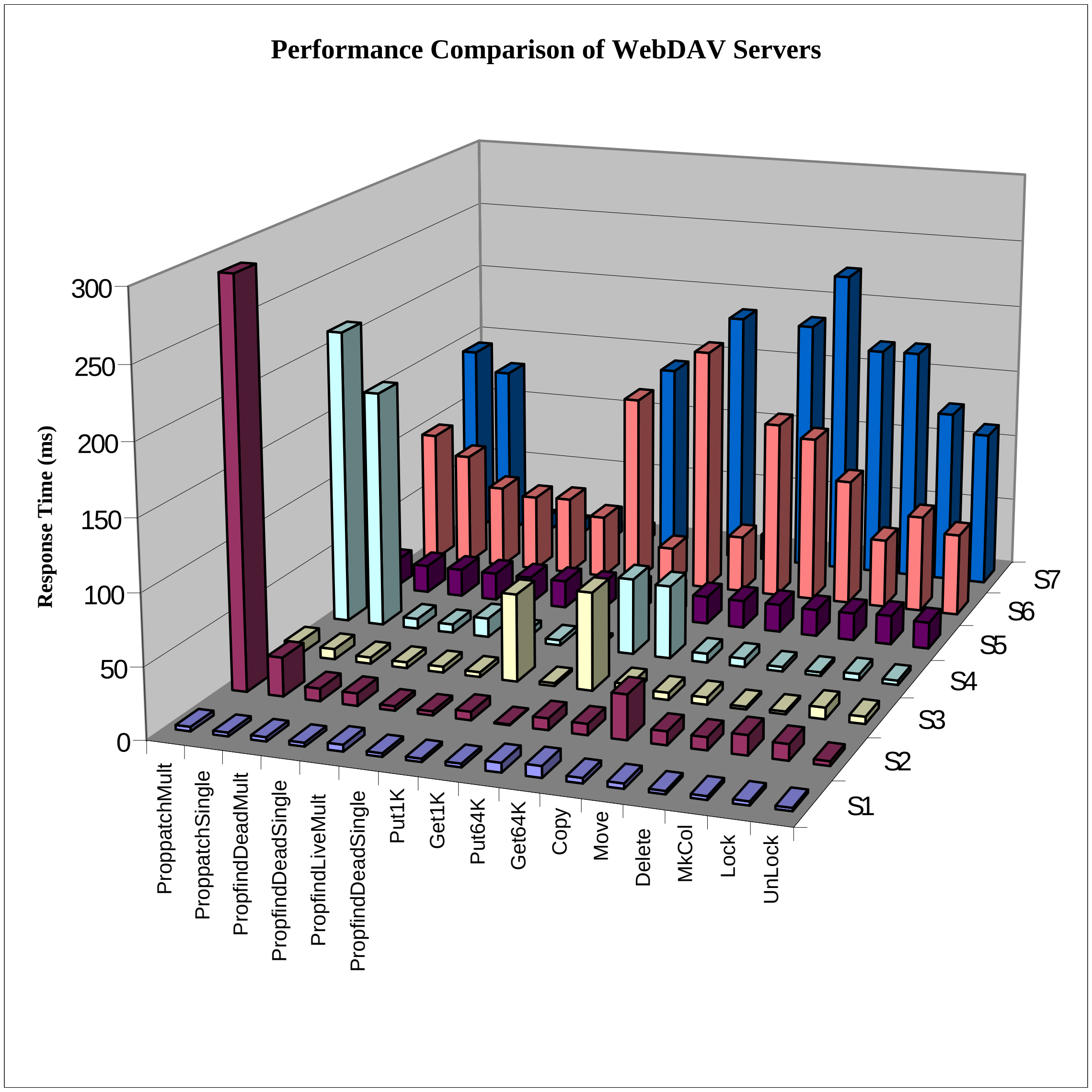

Figure 8: Performance Comparison of WebDAV Servers (Partial Methods)

The servers under testing include Microsoft IIS6.0, a NASA variant of the Catacomb server, SunONE WS6.1, Apache mod_dav, SoftwareAG Tamino, Xythos WFS 4.0 and Sambar 5.0. However, we cannot fairly compare them due to the different machines used for each server, so we refer to them anonymously in the rest of results. The following Table 6 lists the technique details for each server and some performance characteristics.

Server1	It is based on file system. Both resource and property are stored on Ext3 file system. It shows extraordinary performance on every method.
Server2	Database based repository. General performance is pretty good except for collection methods. (COPYCOL, MOVECOL and DELETECOL). However, PROPPATCH for multiple properties is very slow.
Server3	File system based repository. Resource, property and locks are all represented as files. Generally, its performance is pretty good. However, PUT is 10~30 times slower than GET.
Server4	PROPPATCH is almost 30 times slower than PROPFIND. Also, we noticed that COPY/MOVE collection (which is 10 level deep and 100 resources at the bottom level) is 30~40 times slower than COPY/MOVE single resource.
Server5	What we know about this server is both property and resource are stored in file system, while lock database uses Berkeley libdbm. Generally, its performance is very good. In particular, the performance of PUT/GET 1M resource is outstanding. Besides, the performance gap of COPY/MOVE between collection and single resource is the smallest (only 3 times slower) among all the tested servers.
Server6	COPY/MOVE collection is 60 times slower than COPY/MOVE single resource.
Server7	This server is built on database. We notice that PUT is much slower than GET. Also, the COPY/MOVE collection is 40~50 times slower than COPY/MOVE a single resource.

Table 6: WebDAV Server Tests Results and Brief Analysis

As shown in Table 6, the file system based repositories outperform database based repositories in most cases. Particularly, the scalability of the COPY/MOVE is much better for file system than for database repositories. We think this is due to the different storage characteristics of file systems and databases. As we know, most modern file systems (such as Ext3) can take advantage of logical locality of the files in the same directory by storing them in the same cylinder group, and therefore the retrieval of files in the same directory can be achieved by a very small number of disk accesses. The database model has no awareness of such locality, treating individual files in the same directory as unrelated records. Accordingly, it has to retrieve files in the same directory separately, hence response time is degraded when the directory size grows.

In server 2, we notice that the performance of PROPPATCH for multiple properties is degraded as compared to PROPPATCH for a single property. We think this is related to storing each dead property in a different row of database table.

Generally, file system based repositories outperform the database approach in most cases. Also, we should note that the obvious drawback of the file system based approach is the lack of SEARCH capability, which is an important functionality of the DASL protocol for searching WebDAV repositories. In addition, the database approach also has the advantage of supporting full content search in the future.

7 Conclusion

In this paper, we present an automated WebDAV server test tool that can correctly, accurately and comprehensively measure server performance, and help people gain insight into server performance behavior.

There are a number of contributions in this paper. First, we developed a tool for performing client-side performance measurement of WebDAV servers. This has significantly improved visibility into the performance of WebDAV servers, and the relative performance benefits of different implementation strategies. Lastly, we discovered and resolved one performance bottleneck in Neon library, which will benefit other WebDAV client developers in the future.

In the experiments, we comprehensively analyzed the performance degradation of the COPY/MOVE collection operation, which was related to the storage characteristics of different repository schemes. Beyond the performance aspect, we should take other WebDAV functionality issues into the design considerations. Note database generally is slower than file system result.

Although this paper initiates the research work on WebDAV server performance measurements, and provides some understandings of the server's behaviors, we think the work just started. Future research should focus on the WebDAV server measurements associated with different workloads, especially those that mimic the behavior of WebDAV authoring clients.

8 Acknowledgements

First, I would like to thank Professor Jim Whitehead, my advisor, for his insight and guidance in conducting this project. His rich knowledge and experience helped me to tackle many difficulties, and his detailed feedback on this report was very helpful. Second, I would like to thank my teammate Mark Slater, who provided useful information and good ideas in the second half of the project. Third, I would like to thank Kai Pan, Sung Kim and Guozheng Ge, for their unselfish guidance, readiness to help and feedback on the project write-up. Professor Garcia-Luna sacrificed his time and effort to review this report, and I extend my sincere gratitude to him as well.

Reference

[1] E. Whitehead and Y. Goland, “WebDAV – A Network Protocol for Remote Collaborative Authoring on the Web”, Proc. of the Sixth European Conf. on Computer Supported Cooperative Work (ECSCW'99), Copenhagen, Denmark, September 12-16, 1999.

[2] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen, “HTTP Extensions for Distributed Authoring – WEBDAV”. Internet Proposed Standard Request for Comments (RFC) 2518, 1999.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, Internet Draft Standard RFC 2616, June 1999.

[4] G. Stein, “WebDAV resources”, 2003. http://www.webdav.org/

[5] G. Stein, “mod_dav: a DAV module for Apache”, 2003. http://www.webdav.org/mod_dav.

[6] Henrik Frystyk Nielsen, James Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Hakon Wium Lie, and Chris Lilley, “Network performance effects of HTTP/1.1, CSS1, and PNG.” In Proceedings of ACM SIGCOMM'97 Conference, September 1997.

[7] V. N. Padmanabhan and J. Mogul, “Improving HTTP Latency,” In Second World Wide Web Conference '94: Mosaic and the Web, pp. 995-1005, October 1994.

[8] Nagle, J., “Congestion Control in TCP/IP Internetworks.” ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Jan. 1984. RFC-896.

[9] J. Almeida, V. Almeida, and D. Yates, “Measuring the behavior of a world-wide web server.” Technical Report 1996-025, Boston University, Oct. 1996.

[10] G. Banga and P. Druschel, “Measuring the capacity of a web server under realistic loads.” World Wide Web Journal (Special Issue on World Wide Web Characterization and Performance Evaluation), 2(1):69-83, May 1999.

[11] J. C. Hu, S. Mungee, and D. C. Schmidt, “Techniques for developing and measuring high-performance Web servers over ATM networks.” In Proceedings of the Conference on Computer Communications (IEEE Infocom), San Francisco, CA, Mar 1998.

[12] W. Stevens, “RFC 2001 - TCP Slow Start, Congestion Avoidance, Fast

Retransmit, and Fast Recovery Algorithms", Jan 1997

[13] Apache, "Manual Page: ab - Apache HTTP Server benchmarking tool", http://httpd.apache.org/docs/programs/ab.html

[14] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer, "Simple Object Access Protocol (SOAP) 1.1", May 2000. (www.w3.org/TR/SOAP).

[15] G. Trent and M. Sake, "WebSTONE: The First Generation in HTTP Server Benchmarking," http://www.mindcraft.com/webstone /paper.html (February 1995).

[16] Sambar WebDAV server, http://www.sambar.com

[17] M. Allman, V. Paxson, W. Stevens, "RFC 2581 - TCP congestion control", April 1999

[18] Software AG Tamino Server, http://www.softwareag.com/tamino/