mod_dav_dbms:
A Database Backed DASL Module forApache

Sung Kim, Kai Pan, Elias Sinderson
Department of Computer Science
University of California at Santa Cruz
{hunkim, pankai, elias}@soe.ucsc.edu

March 14, 2002

Abstract

This paper describes the implementation of mod_dav_dbms, a database backed repository layer for the Apache web server that fulfills the requirements for a *Class 1* DAV server and the current DASL specification. We describe the WebDAV and DASL protocols, and briefly describe the Apache server architecture and the mod_dav module structure. The schema used in the database is discussed, placing an emphasis on the discussion of design choices and trade offs. The implementation of mod_dav_dbms is covered in detail, followed by a presentation of the testing methodology and our results.

Contents

1 Introduction

The mod_dav_dbms Apache module implements a database backed DAV repository that supports server–side searching of resource contents and metadata. The Apache WebDAV module, mod_dav [1], was used as a foundation for this work, in the hope that it would be relatively straightforward to implement the DASL SEARCH method on top of it. We quickly realized that due to the large number of file operations involved the file system based repository layer, mod_dav_fs, would have poor performance characteristics. Hence, we decided to replace mod_dav_fs with a new repository layer, mod_dav_dbms. Thus our project grew to include the development of mod_dav_dbms as well as the DASL SEARCH method. In practical terms, what this means is that instead of only implementing the SEARCH method, we also had to implement the methods specified by HTTP/1.1 and WebDAV [2].

There are several unique contributions made by this project. The client and server implementations of the DASL protocol specification are necessary steps on the path towards having DASL become an Internet Engineering Task Force (IETF) recommended standard. Furthermore, the DASL SEARCH algorithm serves as a reference implementations for anyone who wishes to implement the protocol. Finally, the development of an open source, database backed repository is a valuable contribution to many communities who will not have to reproduce this effort themselves.

This rest of this document is organized as follows. Section two provides some background on WebDAV, DASL, and the architecture of the Apache 2 server. Section three outlines the design of the WebDAV module structure, and illustrates how our work fits into the existing architecture. The database schema used by mod_dav_dbms is presented, with attention paid to the design choices made and their trade offs. Section four examines the implementation in detail, covering the DBMS layer, mod_dav_dbms, the DASL SEARCH method and a DASL client based on cadaver [3]. In section five we discuss our testing methodology and present the test results. We conclude by indicating several promising directions for future work in this area.

2 Background

2.1 WebDAV

Web Distributed Authoring and Versioning (WebDAV, or DAV for short) is a suite of protocol extensions to HTTP/1.1 [4] which supports collaborative authoring and management of resources and metadata properties on remote web servers. The core DAV protocol defines seven new HTTP/1.1 methods: PROPPATCH for associating metadata properties with resources, PROPFIND for retrieving properties of resources, MKCOL for creating collections, MOVE and COPY methods to manipulate the resource namespace, and LOCK and UNLOCK methods for serializing access to a resource. DAV servers are divided into two classes depending on which methods they support. A *Class 1* DAV server

must support all of the DAV methods except for the LOCK and UNLOCK methods, while a *Class 2* DAV server must support all the DAV methods as defined in RFC 2518, the WebDAV protocol specification [5].

2.2 DASL

The DAV Searching and Locating (DASL) protocol defines a lightweight SEARCH method for executing server–side searches of resource contents and metadata properties. The full DASL protocol consists of the SEARCH method which transports the query from the client to the server, the DASL response header which indicates the server support of the SEARCH method and the query grammars recognized, the DAV:searchrequest XML element which contains the actual query, the DAV:queryschema property which allows the client to discover what properties can be referred to in the query, the DAV:basicsearch query grammar which implementations must support, and the DAV:basicsearchschema which defines how properties may be used in the query (selectable, searchable, or sortable), and which of the special DAV operators are supported [6].

The DASL protocol specification defines the basic life cycle of a SEARCH request as follows. First the client constructs a XML query using the desired search grammar. After constructing the query, the client invokes the SEARCH method on a *search arbiter*, a resource that performs the search on the client's behalf, and includes the query in the request body. The arbiter then executes the query and sends the results back to the client in the body of the response. The text/XML MIME type is used for both the request and response bodies. The response body must conform to the PROPFIND response body, as specified in RFC 2518.

2.3 Apache2

Apache is the most widely used web server software on the Internet, claiming over 65% of the market share [7]. The Apache web server architecture consists of a number of processing modules which handle client requests, acting on them in various ways depending on how the server is configured. As a direct result, Apache has a very flexible and extensible design. Using this modular approach it is possible to implement server–side functionality without having to modify the main Apache source code. This allows developers to leverage the efficiency and robustness of the core Apache architecture so that they don't have to reimplement the existing code base. There are several distinct steps Apache uses to process requests, such as initilize, get method, read request, etc. Depending on the module, it is possible to hook additional processing steps into the server's behavior.

The WebDAV protocol is implemented in the Apache server as two modules, shown in Figure 1. The main module, mod_dav, takes care of the initial processing of method requests, while the second module, mod_dav_fs, handles the interface with the underlying repository structure. In mod_dav_fs, the repository is implemented on the standard file system with metadata stored as key : value pairs in files at the directory level. Although this is a very direct approach, the implementation of any search mechanism over this

architecture would result in many operations to open and close the files. This leads to poorer performance characteristics that won't scale well for large installations. Since scalability and performance are primary factors in server adoption, this behavior is unacceptable in the long run. This fact led us to the decision to replace mod_dav_fs with our own database backed repository that would be able to support searching more efficiently.

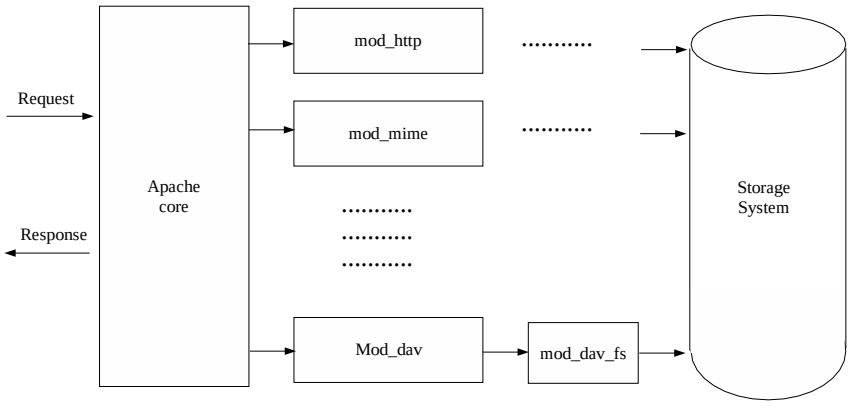

Figure 1: Apache Module Architecture

3 Design

3.1 DB Schema

We use MySQL [8] so that we can make use of the strong conditional capabilities of SQL queries to search metadata and resources. In mod_dav_dbms, all resources and metadata are stored in the database. The text and the binary contents of resources are stored in different fields. Unlike the hierarchical structure in a file system, the resources in mod_dav_dbms are organized in a flat structure in the database table. The hierarchical containment relations of resources and collections can be realized by the URI of each resource. There is a 'resourcetype' field in the table to indicate whether the resource is a regular file or a collection. With a hierarchical structure, it is difficult to efficiently implement conditional searching because the search process must recursively process the subdirectories. In contrast to this, one can easily implement search function in a database due to its flat organization of records in tables and the relations between the tables.

There are two tables in the database. The dasl_resource table contains resources and live properties, while the dasl_property table contains user defined dead properties. We have to use a separate table for the dead properties because, unlike live properties, dead properties are not known ahead of time. Any resource can have its own, unique dead properties and new ones can be created at any time. The relationship between the dasl_resource and dasl_property tables is one-to-many, related by serialno.

The dasl_resource table contains both the resources and their live properties. The fields are shown in Table 1. The dasl_property table contains the dead properties associated with a resource. The fields in are shown in Table 2. To facilitate efficient searching, there is also an extra index placed on (name, value(245)) for the dasl_property table.

3.2 Design Choices

3.2.1 Primary key

Which is a better choice for the primary key for the dasl_resource table, URI or serialno? Though URI is a unique field in the dasl_resource table and can serve as the primary key, we introduce a serialno field as the primary key for efficiency. serialno is a long integer field, while URI is a text field which could be at most 65535 bytes in length. If URI were used as the primary key, it will take much longer when we perform joins on the dasl_resource and dasl_property tables. In addition to the above, serialno has another function of keeping track of the order the resources were inserted into the table. The cost for using serialno as the primary key is that we add one more field to the dasl_resource table and MySQL should maintain its serial property. In practice, this cost is not considered high.

3.2.2 date–time format

In the dasl_resource table, we use 'bigint' data type to represent the date–time fields, such as creationdate and getlastmodified. The values in those fields indicate the seconds since Jan. 1st, 1970, 00:00:00. Besides this decision, there are two other choices to present the date–time.

The first choice is to use a string type. For example, "20020128080601" indicates 08:06:01 on Jan. 28th, 2002. We can only use the "YYYYMMDDHHMMSS" format to present the date–time with a string because we need to perform comparison operations on the field. There are two disadvantages to this approach. First, the format to present the date–time in this approach is not flexible since we have to transform it when we want to display it in different forms. Second, arithmatic operations can not be easily applied. For example, the arithmatic operation that calculates the difference of two date–times would require inefficient conversion routines.

Field	Type	Description
serialno	integer, not null, AUTO_INCREMENT, Primary Key, indexed	The serial number of the resource record in the table.
filename	text, not null, indexed	The filename of the resource.
uri	text, not null, uniquely indexed	The absolute URI of the resource.
creationdate	bigint, indexed	The date time the resource is created.
displayname	text, indexed	A description of the resource that is suitable for presentation to a user.
getcontentlanguage	varchar(20), indexed	Content-Language header on a GET.
getcontentlength	integer, not null, indexed	Content-Length header in response to a GET.
getcontenttype	varchar(20), indexed	Content-Type header in response to a GET.
getetag	text, indexed	The ETag header returned by a GET.
getlastmodified	bigint, indexed	The last-modified date time on a resource.
lockdiscovery	varchar(255), indexed	The active lock information on a resource.
resourcetype	integer, indexed	The nature of the resource, 0: regularfile; 1: collection(directory).
source	integer, indexed	Indicates that the resource is source or not. 0: regular; 1: source.
supportedlock	varchar(255), indexed	A listing of the lock capabilities supported by the resource.
depth	integer, not null, indexed	Depth of containment hierarchy from the root.
istext	integer, indexed	Indicates that the resource is text or binary. 0: none; 1: text; 2: binary.
textcontent	longtext	Field storing the content of a text file.
bincontent	longblob	Field storing the content of a binary file.

Table 1: dasl_resource Table

Field	Type	Description
serialno	integer, not null, indexed	The serial number of the resource which is related to the serialno in the dasl_resource table.
Name	varchar(255), not null	Name of the property.
Value	text	Value of the property.

Table 2: dasl_property Table

The other approach considered was to use the date–time data type supported by MySQL. The advantage of this approach is that we can make use of the date–time functions in MySQL. The drawback is that it is vendor specific since we have to depend our code that deals with the date–time on MySQL. If we decide to use another database in the future, this approach dictates changes to the source code. On the other hand, it is not flexible to process the date–time in our program, if our code depends on the presentation format and process functions of date–time in MySQL.

3.2.3 Text and Binary Fields

We use a `longtext` field to store text files and a `longblob` field to store binary files (e.g. image, pdf, mp3, video, et cetera). There are three reasons that we store the text file and binary file in different fields. First, the content of the text file can join the conditional search. For example, we can easily use this field in a search condition to find resources that have certain content, while it makes no sense to apply this on binary files. Second, we can retrieve the text content from a document with a special format and store it in the 'textcontent' field. For example, .pdf and .doc files have a special document format. We can save these files in the `bincontent` field and, at the same time, extract their text content and save it in the `textcontent` field. It is then possible to not only make use of the search abilities of SQL on their content, but also keep the original documents in the repository. There is an `istext` field maintained that indicates whether the resource is a text file or a binary file. Third, the database can apply full-text searching on text fields. Full-text indexes can be created on varchar and text fields to allow a query to search for resources with specific contents.

There is not much space wasted when we create both `textcontent` and `bincontent` fields in the dasl_resource table. The size of a record in the table depends only on the size of the actual contents of the record. For example, a text file only has content in the `textcontent` field and its `bincontent` field is empty. Thus, the `bincontent` field contributes zero bytes to the overall size of the record. The only extra space required is for the record that has contents in both `textcontent` and `bincontent`.

3.2.4 Live vs Dead Properties

We put live properties together with the resource due to the considerations of the search efficiency. Since the search condition may be over the content and live properties at the same time, we can avoid table joins by putting them in the same table. It was necessary to put live properties and dead properties in different tables because dead properties are different from live properties. Dead properties are not predetermined so users can define their own properties of their resources. Therefore, dead properties have a different representation than live properties. Live properties are organized in columns, while dead properties are organized in rows. We will see later that this representation of dead properties in this way causes some difficulties when parsing the SEARCH request to generate the SQL query.

The organization of the live properties in the dasl_resource table ensures the searching efficiency when using live properties as the searching conditions. But there is an extensibility problem with the design of the dasl_resource table. If someone wants to define a new live property, both the dasl_resource table and the source code of the search method must change. As we can see, this is a trade off between efficiency and extensibility.

4 Implementation

We implemented an Apache 2 server module that interfaced the WebDAV protocol with a DBMS layer over a MySQL DB. All resource contents and metadata properties are stored in MySQL. The decision to use MySQL was made because it is lightweight and supports a simple C API. Within this architecture, we implemented the DASL protocol SEARCH method. For testing out the server, we added search capabilities to cadaver, a command line WebDAV client.

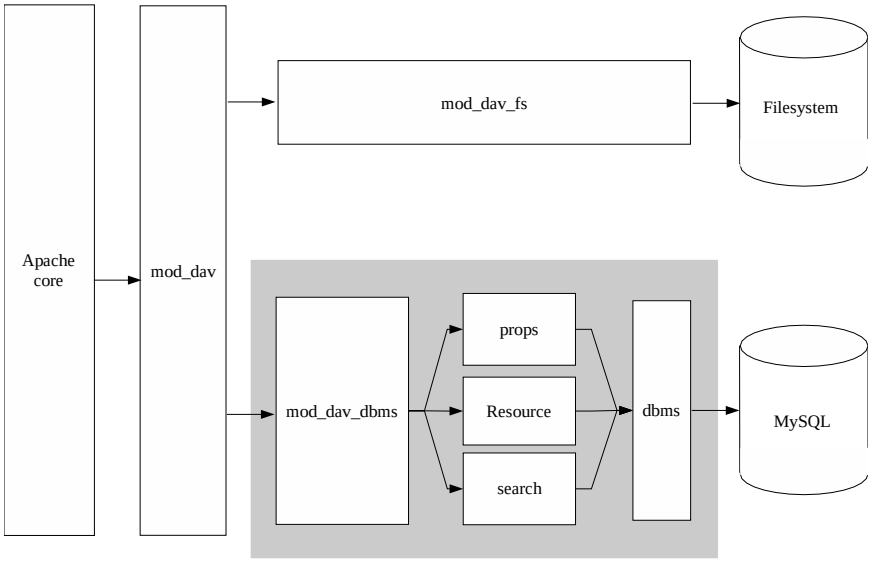

Figure 2: mod_dav_dbms Architecture

4.1 dbms.c

The DBMS module uses the MySQL C API to implement the database operations. This module is at the lowest layer in mod_dav_dbms, and conceptually sits on top of the DB. The functions in this module provide database operations for the other modules. The functions in this module can be divided into the following groups: database connection, property processing functions, file content processing, and searching.

4.1.1 Data Structures

The main data structure used in the DBMS module is dav_dbms_resource. It serves as a container that returns query results to the search module. dav_dbms_resource is a linked list, every node in which is a resource structure containing major information of a resource. A dav_dbms_resource node also contains a list, which holds the list of dead properties belonging to the resource. This structure can also serve as the container for the individual resource record and the container for resource records in a collection. dav_dbms_property is a data structure that stores dead property records. It is also a link element of dav_dbms_resource.

4.1.2 Functions

The major parts in the functions of DBMS module are the SQL statements in them. In this section, we explain a few of the more interesting functions.

The dbms_read_content function reads the contents of a resource from the DB and saves the content to a file. MySQL provides the ability to retrieve the contents of text or blob fields directly into a file. Thus, we don't need to write the code to handle the file operation. Conversely, the dbms_write_content function uses UPDATE statement to write the content of a file into the database. Examples of the SQL statements used are below.

```
SELECT bincontent
INTO DUMPFILE <filename>
FROM dasl\_resource
WHERE serialNO= aSerialno ;
```

or

```
UPDATE dasl\_resource
SET textcontent = LOAD_FILE("aTextFileName")
WHERE serialNO = aSerialNo ;
```

By simply changing the URI field of a resource, the dbms_move_resource function moves the resource. Just like moving files or directories in a file system, we only change the path (URI) of the resource(s) instead of copying and deleting files. We make use of the string operations provided by MySQL to construct the SQL statement that can change the URI(s) of the resource(s) in only one statement. In doing so, we avoid recursing through the DB and ensure the atomicity of the move operation on a collection. For example, if we want to move the resource /project to /project_bak, the following statement will change /project to /project_bak, and change all the resources in /project, /project/*, to /project_bak/* if it is a collection.

```
UPDATE dasl_resource
SET uri = concat( '/project_bak' +
SUBSTRING( uri , 8 ))
WHERE uri like '/project%' ;
```

The dbms_copy_resource function copies a resource (and it's properties) to another URI. New records are created in the DB to contain the copy of the resource. If the resource being copied is a collection, all the files contained therein are copied as well. The procedure to copy resource(s) in the database is described below. First, create a temporary table and save source resource(s) with changed URI(s) into the temporary table. The SQL statements used are:

```
CREATE TEMPORARY TABLE tmp
SELECT ...
FROM dasl_resource
WHERE ...
```

Second, copy the records in the temporary table to dasl_resource table. The SQL statement is:

```
INSERT INTO dasl_resource
SELECT *
FROM tmp
```

4.2 mod_dav_dbms.c

The Apache 2 server configuration directives are used to configure the database options used by mod_dav_dbms, thus each instance of the server can have only one DB host, DB name, DB user id, etc. specified. The advantages of this approach is that we can connect to the DB when the Apache thread starts and disconnect from the DB when apache thread exits. This connection life cycle improves the database performance which is responsible for the overall performance of mod_dav_dbms. The limitation of this approach, of course, is that we can only use one database per server instance.

Since the interface of mod_dav is designed for a file system, there are open, read, seek style interfaces on the back end. MySQL doesn't support these sorts of file handling mechanisms, so we needed to use temporary files as a medium to deliver content between the server and the DB. This process makes mod_dav_dmbs slower than it would be otherwise.

4.3 search.c

The processing of the SEARCH method is handled by the search module. The general approach is as follows. First the XML body of the request is parsed, and the corresponding SQL query constructed. The query is then executed on the underlying database, with the results returned as a linked list of resources and their properties. The results of the search are packaged into an XML body that is sent with the HTTP response. Due to the DB schema we used in , there are a few difficulties that were faced in implementing the SEARCH method.

One of the difficulties faced was that dead properties are stored as rows of the dasl_property table. This necessitates the aliasing of the dasl_property table in the SELECT portion of

the query and then using self join operations to get the desired results. Aliasing is an operation that allows one to refer to a table by two names, allowing the dasl_property table to be effectively used as multiple tables. Self joins refer to the fact that we are really joining result sets from the same underlying table. To see why this is necessary, recall that the dasl_property table has three fields, serialno, name and value. In order to find the resources that are written by 'Patrick' with the keyword 'Germany', we specify a query such as

```
SELECT uri
FROM dasl_resource, dasl\_property
WHERE name='author' and value ='Patrick' and
name='keyword' and value='Germany';
```

This query will return an empty set, as fields cannot have more than one value (as with name='author' and name='keyword' above). This query must instead be written as

```
SELECT uri
FROM dasl\_property t1, dasl\_property t2
WHERE t1.name='author' and t1.value ='Patrick' and
t2.name='keyword' and t2.value='Germany';
```

where the dasl_ property table is aliased twice to t1 and t2. This example also illustrates how the dead properties that occur in the WHERE element of the XML request affect the FROM portion of the SQL query. In reality, the dead properties in the WHERE element will also affect the SELECT portion of the SQL query as well. To make matters worse, dead properties in the SELECT element will also affect both the SELECT and WHERE portions of the SQL query. These dependencies dictate that we must process the XML request out of order, collecting the dead properties first from the different elements of the request, and then building the SQL query from string fragments generated during the parsing. More work is needed to determine if there is a better DB schema that will result in a more direct parsing of the SEARCH request without negatively affecting the performance. If MySQL supported 'EXISTS' operation, sub-queries or stored procedures this self join approach would not be necessary.

4.4 cadaver_dasl

Cadaver is a command-line WebDAV client for UNIX. It supports file upload, download, on-screen display, namespace operation(move/copy), collection creation and deletion, and locking operations.We extended the functionality of cadaver by adding limited support for the DASL protocol. The user's search condition is taken from the command line and transformed into a SEARCH method compliant with the DASL specification. After sending the request to the server, the client receives the search result from the server and displays the URIs on the terminal. The syntax of the cadaver search command is `search <arbiter> <condition>`, where `arbiter` is the URI of the search arbiter executing the query, and `condition` is the set of conditions we are trying to resolve. The current implementation does not support the `orderby` or `limit` portions of the DASL

specification, nor does it support parameters for the displaying fields. Furthermore, the client only displays the URI field of the search result. We plan on supporting the full search functionality in the future.

A challenging part of the cadaver_dasl implementation was to transform the search condition into a compliant XML request body. In order to do this, we examined the BNF formula for the `condition`. The BNF formula is a simplified version of the BNF formula of the condition portion of ANSI SQL. After devising the BNF formula for the condition parameter, we can recursively parse the condition. For each term in the formula, we translate it into corresponding XML text. The cadaver_dasl portion of our project currently only supports conditions without parentheses, 'not', 'in', and 'between'. We will support the entire syntax in the future.

5 Test Results

5.1 Standards Compliance

In order to test the compliance of our implementation with respect to WebDAV, we ran several test suites against the server. The SkunkDAV [9] test suite reported no problems, except for the test related to locking, which we have not implemented. Furthermore, the Litmus test suite [10], based on the neon client library [11] reported that our server implementation was 92.3% compliant with RFC 2518. There are no publicly available test quites for the DASL protocol, so we have been working on developing our own. Since the protocol is currently in a draft state, and likely to evolve, we will continue to develop this test suite and try to keep it up to date with the latest DASL draft.

5.2 Performance

To compare the writing and reading performance, we used the same apache server program and machine. We configured /dasl directory for mod_dav_dbms and /dav directory for mod_dav_fs and then mounted each directory on the local filesystem using Davfs [12] and used Iozone [13] as the benchmark test suite. The Linux kernel's reading and writing interfaces include not only PUT or GET methods, but also PROPFIND and PROPPATCH. So the reading results includes PROPFIND/GET, while writing results includes PROPPATCH/PUT. These results show that mod_dav_dmbs is fast enough to use for a content server. Table 3 shows our results.

Module	Writing	Reading
mod_dav_dbms	94	48
mod_dav_fs	97	50

Table 3: Reading and Writing Performance [KB/sec]

For each method's performance test, we modified the apache benchmark test tool, ApacheBench V2.0.32, to use the same methods several times and give us timing and performance results. For GET and PUT methods, we used 1 KB files. We executed PROPFIND methods in a directory with 101 files. For PROPPATCH we use one file and set three properties per request. To get accurate results, we request each method 1000 times and compute the average. We ran the benchmark tool on both local and remote clients to see the effect of network latency on the performance. The average ping speed of the remote client was 101.363 msec.

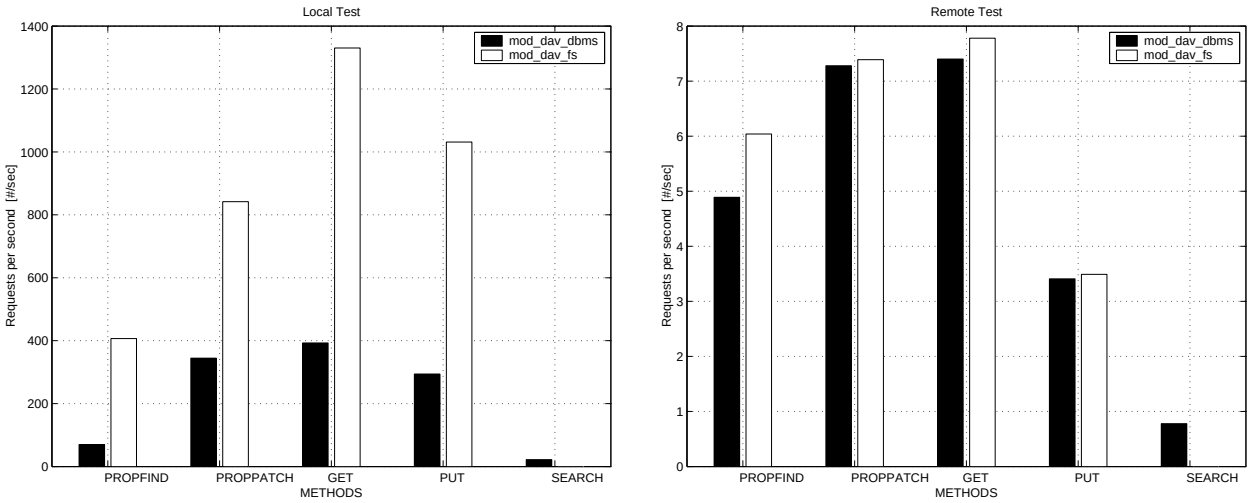

Figure 3: Performance in Local and Remote Clients

There is a big difference in the local test between the GET and PUT methods because mod_dav_dmbs dumps a file from DBMS into the file system and then delivers this file to Apache. So it takes almost double the time as compared to mod_dav_fs. With the PROPFIND methods, the performance is almost same since mod_dav_fs needs to look all files in the directory, mod_dav_dbms only needs several SQL queries. If we connect from remote server the differences between GET and PUT methods is very small. The search method is quite slow compared to other methods. We have 1124 files in a /dbms directory, and do infinity option for search. Unfortunately, there is nothing to compare this behavior against since the DASL protocol is not implemented for mod_dav_fs.

While the above represents a good start, it should be clear that more performance testing is needed before we can draw any firm conclusions. There are a number of factors influencing the performance including the number of resources, the number of properties, and the value of the depth field to name just a few. It will be very interesting to see the performance results from other similar project as they become available.

6 Related Work

There are several areas of work that relate to mod_dav_dbms in different ways. Local search engines, database backed web servers, and metadatabases all have aspects that can

be compared and contrasted with our approach in different ways.

A local search engine (LSE) is a software package that can be installed on a Web server to allow searching the contents of the server [14]. Most LSEs are based on CGI scripts, as opposed to being based on a standard protocol. Also, for faster searching, LSEs typically employ an indexing process which reads the contents of the server and creates a dictionary. This approach can cause synchronization problems between the dictionary and the actual contents of the server. Another drawback is that current LSEs only supports searching of contents, while DASL supports searching metadata properties as well as resource contents. As DAV servers become more popular, it is likely that LSEs will be developed to search metadata as well, and some will support the DASL protocol.

The Oracle web server saves resource contents in the Oracle DB, as we do, and supports the searching of resources via proprietary CGI scripts or server APIs [15]. Again, this proprietary approach falls short of the broad, interoperable goals set forth by the DASL working group. However, with a small amount of effort, one should be able to substitute any DB for the repository used in mod_dav_dbms. Given some of the limitations of MySQL, the effort may be well worth it.

There has been much work in the fields of metadata and metadatabases. Many systems exist to support metadata management for various areas including digital libraries, the Web, geographic information systems, images, video clips, and almost anything else you can imagine [16, 17, 18, 19].Unfortunately, there is precious little information available on generic DB schemas that support the efficient representation of arbitrary metadata. Indeed, the flexibility required in the representation and management of metadata is a difficult problem to overcome despite the huge amount of effort that has been put into standardization [20]. A major problem in developing a sufficiently general metadatabase schema is the fact that metadata is usually very structured and application specific. As a result, it is extremely difficult to implement a generic schema that is efficient with many different types of metadata.

7 Conclusions and Future Work

The development of a Class 1 DAV server that supports the DASL protocol is not easy. Several contributions of this project can be identified, including the reference implementations of the DASL protocol on both the client and server side and the initial work towards developing a generic, yet reasonably efficient database schema for combined resource and metadata management. The value of providing an open source, DAV/DASL aware web server with a database backend cannot be underestimated, and is an important first step towards the development of a personal document management/digital library system, a market that currently has no commercial offerings.

There are a number of ways in which this project could be extended and developed further. Research into developing a high performance, flexible DB schema that can store arbitrary

metadata is high on the list of priorities, as is finishing the work necessary to be fully compliant with RFC 2518 and the DASL protocol. The project could also be functionally improved by adding auditing facilities, a more flexible security model than the one provided by Apache, versioning of resources and collections via the Delta-V protocol [21], or developing a richer containment model that allowed multiple containment. The integration with event notification services by adding the appropriate hooks into mod_dav_dbms is another promising direction to embark upon. This body of work will likely require the long–term attention of several, perhaps many, developers. To address this need, this work will continue as an open source project in the hope that other people will recognize the long–term benefits to the community and join our effort. In the future, we hope to be able to distribute mod_dav_dbms with the Apache server in a standard fashion, thus providing a scalable, database backed, open source web server that supports the entire DAV protocol suite.

8 Acknowledgements

This work would not have been possible without the gracious support of our advisor, Professor E. James Whitehead, who provided invaluable feedback on earlier drafts of this paper. We also wish to thank the individuals affiliated with the DAV working groups and the Apache software foundation, without whose efforts none of this would be possible.

References

[1] G. Stein, "mod_dav: a dav module for apache," 2002. Web site. http://www.webdav.org/mod_dav/.

[2] G. Stein, "Webdav resources," 2002. Web site. http://www.webdav.org/.

[3] J. Orton, "Cadaver," 2002. Web site. http://www.webdav.org/cadaver/.

[4] R. Fielding, "Hypertext transport protocol – http/1.1," 1997. Network Working Group RFC 2068, January 1997. http://ds.internic.net/rfc/rfc2068.txt.

[5] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen, "Http extensions for distributed authoring – webdav," 1999. Internet Proposed Standard Request for Comments (RFC) 2518.

[6] E. Whitehead, "Dav searching and locating (dasl) home page," 2002. Web site. http://http://www.webdav.org/dasl/.

[7] E-Soft, "Security space," 2002. Web site. https://secure1.securityspace.com/s_survey/data/man.20020

[8] A. Unknown, "Mysql," 2002. Web site. http://www.mysql.com/.

[9] J. Smullyan, "Skunkdav dav client," 2002. Web site. http://skunkdav.sourceforge.net/.

[10] J. Orton, "litmus – webdav server protocol compliance test suite," 2002. Web site. http://www.webdav.org/neon/litmus/.

[11] J. Orton, "neon http and webdav client library," 2002. Web site. http://www.webdav.org/neon/.

[12] S. Kim, "Dav file system," 2002. Web site. http://dav.sf.net.

[13] D. Capps, "Iozone filesystem benchmark," 2002. Web site. http://www.iozone.org.

[14] T. ht://Dig Group, "Internet search engine software," 2002. Web site. http://www.htdig.org/.

[15] Oracle, "Oracle 9i application server," 2002. Web site. http://www.oracle.com/appserver/.

[16] P. Atzeni, G. Mecca, P. Merialdo, and G. Sindoni, "A logical model for metadata in web bases."

[17] D. Eichmann, T. McGregor, and D. Danley, "Integrating structured databases into the Web: The MORE system," *Computer Networks and ISDN Systems*, vol. 27, no. 2, pp. 281–288, 1994.

[18] S. Timpf, M. Raubal, and W. Kuhn, "Experiences with metadata," 1996.

[19] E. Department, "Sequoia 2000 metadata schema for satellite images, jean t. anderson and michael stonebraker."

[20] M. Staudt, A. Vaduva, and T. Vetterli, "The role of metadata for data warehousing," Tech. Rep. ifi-99.06, 11, 1999.

[21] E. Whitehead, "Ietf delta–v working group homepage," 2002. WebSite. http://www.webdav.org/deltav/.

A Installation Instructions

```
1. Get mod_dav_dbms.tar.gz from http://dasl.sf.net

2. Checkout the "httpd-2.0" cvs module from apache.org.  Put it
   whereever you wish;  it's an independent project.

   cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic login
      (password 'anoncvs')

   cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co httpd-2.0

3. cd httpd-2.0/srclib/, and checkout the "apr" and "apr-util"
   modules into this directory:

   cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co apr
   cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co apr-util

4. At the top of the httpd-2.0 tree:

   ./buildconf
   ./configure --enable-dav --enable-so --prefix=/usr/local/apache2

   The first arg says to build mod_dav.
   The second arg says to enable (and build everything) as shared libs.
   The third arg is where you will ultimately install apache.

5. make depend; make ; make install

6  Untar mod_dav_dasl and go into the directory. Then run

   ./configure --with-apache=/usr/local/apache2 \
           --with-mysql=/usr/local/mysql

   This argument tells how to build mod_dav_dbms, and where to
   find the required information to do so.

7. make clean;  make; make install

   After the make install, the libmod_dav_dbms.so should be installed
   in /usr/local/apache2/modules/.

8.  Add this to the *bottom* of /usr/local/apache2/conf/httpd.conf:
```

```
    <Location /dbms>
            DAV dbms
    </Location>
```

9. Add this server configuration :

```
        DavDBMSHost localhost
        DavDBMSDbName   props
        DavDBMSId       hunkim
        DavDBMSPass     xxxxx
        DavDBMSTmpDir   /tmp/
```

   - DavDBMSHost is hostname for DBMS server
   - DavDBMSDbName is database name of the DBMS
   - DavDBMSId is the user name who has the read/write permission of DBMS.
   - DavDBMSPass is the password for the user
   - DavDBMSTmpDir is temporary directory for contents

10. Change table.sql for your site. Create table and insert initial field :

```
        mysql <dbname> < mod_dav_dbms/table.sql
```

11. Now fire up apache 2.0:

```
        /usr/local/apache2/bin/apachectl stop
        /usr/local/apache2/bin/apachectl start
```

    Check /usr/local/apache2/logs/error_log to make sure it started
    up okay.

12. Enjoy the DASLed apache server.