Architecture and Data Model of a WebDAV-based Collaborative System

Sunghun Kim, Kai Pan, Elias Sinderson, E. James Whitehead, Jr.
Dept. of Computer Science
Baskin Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064 USA
{hunkim, pankai, elias, ejw}@cs.ucsc.edu

ABSTRACT

Web Distributed Authoring and Versioning (WebDAV, or DAV for short) is a suite of protocol extensions to HTTP/1.1 which support collaborative authoring. The development of the WebDAV protocol has explored the hypothesis that an authoring protocol built on HTTP is the best way to deploy collaborative authoring protocol capability. For three and half years, WebDAV and its many implementations have provided a strong indication that this hypothesis is correct. More recently, DeltaV and DASL are exploring a similar hypothesis that an HTTP-based protocol is the best way to deploy searching, versioning, and Software Configuration Management (SCM) capabilities. However, due to the protocol's complexity and lack of reference implementations, there are few DeltaV/DASL implementations and scarce publicly available information on architectures and data models for such implementations. Our key contribution is an architecture for performing DeltaV versioning and DASL searching operations within the context of the Apache web server. We also present the relational database schema used to represent the DeltaV/DASL data model. As a verification of the architecture and database schema, we implemented Catacomb, an open source WebDAV/DeltaV/DASL server. The Catacomb design and implementation are offered as a reference architecture for DASL/DeltaV servers.

Keywords

Web Infrastructure, Collaborative Applications, Design of Collaborative Systems, Interfaces for Collaborative Work.

1. INTRODUCTION

It is common to collaborate on a project with several people in different geographical locations. What is a good collaborative system to support such a project? A network-accessible server is required that provides namespace management to create directories, copy and move files, and a common mechanism for overwrite protection to avoid the "lost update problem." Metadata management is also desirable, to set metadata properties such as author, keyword, or copyright information, as is versioning, to record the various states in the evolution of a resource. Furthermore, the ability to search for a resource among many other resources is very useful.

WebDAV [1, 2], generally, is a suite of extensions to the HTTP/1.1 protocol, which fulfill the above requirements [3, 4]. The core WebDAV protocol explores the hypothesis that an application layer protocol, built on HTTP, is the best way to deploy collaborative authoring capabilities, in an interoperable manner, into today's applications and content management servers. DASL [4] and DeltaV [5], address similar questions for searching and versioning, respectively. For three and half years, WebDAV and its many client and server implementations – including Apache, Microsoft Internet Information Server, Microsoft Office, Adobe Photoshop, Internet Explorer, and Dream Weaver, among others – have largely proven that this hypothesis is correct [5]. One of the key benefits of the WebDAV, DeltaV and DASL protocols is that they provide a standard mechanism for accessing and manipulating resources.

The DASL protocol extends WebDAV with searching and locating capabilities [6]. Development of DASL is ongoing, currently available as an Internet draft. The DeltaV protocol extends WebDAV with versioning and Software Configuration Management (SCM) capabilities. It was approved as an IETF Proposed Standard specification in March 2002 [5, 7]. Due to the complexity of the specifications and lack of reference implementations, there are few DeltaV/DASL implementations and no open architectural or data models for such implementations. This paper presents an overview of the DeltaV/DASL protocol and a reference architecture that supports DeltaV/DASL versioning operations within the context of the Apache server. We also show the relational database schema used to represent the DeltaV/DASL data model. To verify the models and the data schema, we implemented Catacomb, an open source WebDAV/DeltaV/DASL server.

There are several contributions made by the Catacomb project. The relational database implementation of the DeltaV/DASL data model, which differs from traditional versioning data models such as RCS [8], acts as a reference for other DeltaV/DASL implementations. Secondly, it provides an example of how the WebDAV/DeltaV/DASL methods are used for collaborative authoring, searching and basic versioning of resources. Finally, the development of an open source, database-backed distributed authoring, searching, and versioning system is a valuable contribution to many communities who will not have to reproduce this effort themselves.

The rest of this paper is organized as follows: Section 2 provides some background on the WebDAV protocol suite, Apache2, and Catacomb. In section 3 we examine related work, including XQuery [9] Z39.50 [10], RCS [8], WWRC [11], CVS [12] and Subversion [13], drawing comparisons where appropriate. Section 4 describes the design and implementation of

DeltaV basic versioning and DASL basic searching functionality within Catacomb, and section 5 discusses future work this area and concludes.

2. BACKGROUND

Before we dive into the data model, design, and implementation issues, we provide some background on the WebDAV protocol suite and WebDAV-related servers. In this section, we briefly introduce the WebDAV, DeltaV and DASL protocols implemented in Catacomb, and describe the Apache server and the Apache mod_dav module, which provides the server framework for Catacomb.

2.1 WebDAV

WebDAV is a suite of protocol extensions to HTTP/1.1 which support collaborative authoring (depicted in figure 1), namespace management of resources, setting and retrieving of metadata properties, access control, and resource versioning on remote web servers [1, 14]. The suite includes the DeltaV [5], DASL[6], ACL [15], Binding [16], Redirect Reference Resources [17], Ordered Collections [18], and WebDAV core [1] protocols. In this paper, the WebDAV protocol refers to the WebDAV core protocol.

The core WebDAV protocol defines seven new HTTP/1.1 methods, providing functionality in four main areas: resource management, metadata management, concurrency control, and resource namespace manipulation [1]. The PUT and DELETE methods defined in HTTP/1.1 are used for resource management [1]. The PUT method creates a new resource, while the DELETE method deletes a resource in the WebDAV server.

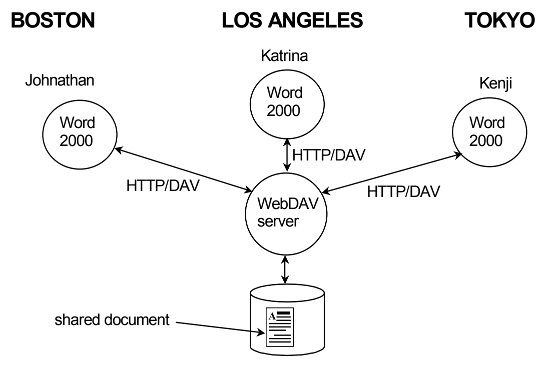

Figure 1. Three collaborators, at different sites, are jointly authoring a document using the WebDAV capabilities of Microsoft Word 2000/XP. Word uses the WebDAV protocol to interact with the shared document, stored on a WebDAV server in the Los Angeles office. Authoring takes place using turn taking, and is not simultaneous.

For the metadata management, the WebDAV protocol provides a notion of properties (metadata). A property is an arbitrary name and value pair, where a name is an XML namespace URL/path pair and a value is a sequence of well-formed XML. Using the PROPPATCH method, we can set one or more properties for a resource. The PROPFIND method lists and discovers the properties of resources. PROPFIND also lists the membership of collections.

Within the context of collaborative authoring, it is possible that two or more people can edit or update the same resource on the same server at the same time and one of the changes may overwrite the other. In order to avoid the resource overwriting (or lost update) problem referred to above, one can use the LOCK method. If a user locks a resource, the WebDAV server provides a lock token and only the owner of this lock token is able to modify the resource. The UNLOCK method releases a lock. The MKCOL method creates a collection, a notion which is similar to a directory in a filesystem. The MOVE and COPY methods are used to move or copy resources from one collection to another. Together, these methods permit manipulation of the resource namespace.

2.2 DeltaV

Using WebDAV, we can create and update resources on WebDAV servers. DeltaV extends WebDAV by providing versioning functionality to the protocol stack. The DeltaV protocol defines eleven methods, and related properties, to provide a variety of versioning features [5]. There are two levels of versioning support, basic and advanced. Basic versioning enables users to check in, check out, create new revisions, discover revisions, and access each revision. Workspaces, parallel development, and configuration management functionality are specified in the advanced versioning portion of the specification, which we will not go into here due to space constraints.

The VERSION-CONTROL method puts a resource under version control. A Version Controlled Resource (VCR) has an associated version history resource, which contains version history information about the resource. The basic versioning model of DeltaV is a linear versioning model (depicted in figure 2), where each version of a resource has its own URL. To ensure that authors have a stable URL to use for editing, the most recent version of a VCR always has the same URL. In the linear versioning model, branching is not permitted and, hence, multiple simultaneous CHECKOUTs of a resource are not allowed. Thus, if another client wants to work on the same resource, the client must wait until the resource is checked back in.

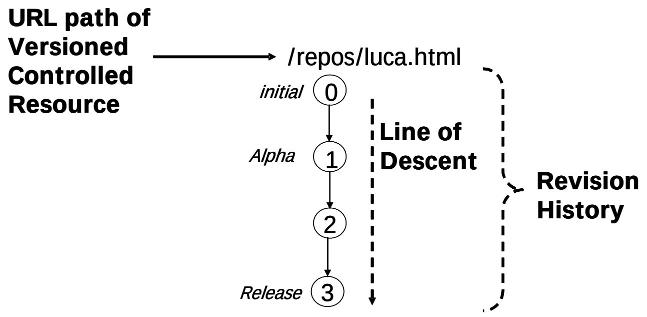

Figure 2. DeltaV basic versioning model. The latest version of the resource will always be accessed via the URL path /repos/luca.html.

With advanced versioning, one can create workspaces on the server and can CHECKOUT the resource into the workspace. This allows for parallel development activities among collaborators. Branching and multiple checkout functions require merge operations. In a DeltaV server, each resource can change separately so we need to be able to capture the state of each resource at a given time. A configuration is used to capture the state of each version-controlled resource in a project. DeltaV introduces a baseline version resource which captures the essentials of a configuration. The baseline version resource concept provides configuration management capability.

The Catacomb server implements the basic versioning functionality of DeltaV. Advanced versioning functionality is not currently implemented due to its complexity. The implementation of basic versioning is the first step towards implementing the full DeltaV feature set and is sufficient to demonstrate the database representation of the DeltaV data model, which will be leveraged and extended to support advanced versioning.

2.3 DASL

The DAV Searching and Locating (DASL) protocol is an extension to the WebDAV protocol, consisting the SEARCH method, the DASL response header, the <DAV:searchrequest> XML element, the <DAV:basicsearch> XML element and query grammar, the <DAV:queryschema> property, and the <DAV:basicsearchschema> element. While WebDAV and HTTP protocols provide a limited resource locating mechanism, the DASL protocol provides support for client specified, server-executed queries to locate resources based on WebDAV properties and text content.

The basic lifecycle of a SEARCH request is as follows. First the client constructs an XML DASL query using the desired search grammar. After constructing the DASL 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 for the WebDAV protocol specification [1].

A DASL query consists of 5 parts, the result record definition, search scope, search criteria, sort specification, and search limits [6]. The result record definition defines which properties are returned in a result record (DAV:select). The search scope part indicates the set of resources to be searched (DAV:from). The search criteria contain an expression against which each resource in the search scope is evaluated (DAV:where). The sort specification defines the sort order of the result set (DAV:orderby). The search limits indicate a bound on the number of result records in result set.

2.4 Mod_dav

Apache is a well-known open source web server [19]. Apache is an exceptionally modular server, so much so that even core HTTPD parts such as request handling and protocol handling are implemented as separate Apache modules. This flexible architecture enables developers to develop various third party modules that add new functionality to the Apache server, and today there are more than three hundred Apache modules [20].

Mod_dav is a built-in Apache module that supports the WebDAV protocol suite [21]. Mod_dav_fs is a sub-module for mod_dav which acts as an interface between the filesystem and mod_dav. Mod_dav_fs stores resources as files in the filesystem with properties stored in a separate file [22]. This filesystem based data storage model is not well suited for implementing the WebDAV protocol suite. It is expensive to open multiple content and property files when handling SEARCH requests and cannot efficiently represent the referential containment relationships among VCRs, version history resources, and version resources as required by the DeltaV data model.

2.5 Catacomb

Catacomb is a collaboration infrastructure technology that supports the WebDAV, DeltaV, and DASL protocols. Catacomb is an Apache module that enables mod_dav to use a relational database for its storage layer [23]. Catacomb stores all resources and properties in the database (currently MySQL). Since a database is more flexible than a filesystem, it is straightforward to implement searching and referential containment operations. Catacomb supports the WebDAV core protocol [1, 5, 6] as well as DeltaV basic versioning and the DASL basic search functionality, described in section 4.

3. RELATED WORK

XQuery [9] is a functional query language for querying XML documents and collections of these documents. XQuery is more complex than SQL primarily because it contains many constructs for dealing with nested structures, absent in SQL. XQuery is also a language with a strong notion of typing [9]. Unlike SQL, which works on relational data, XQuery is designed to perform queries on structured documents and collections of these documents. SQL is sufficient for supporting relational data search in DASL. If a DASL implementation wanted to support searching of structured data, XQuery may be used in the server to process requests that search over XML documents.

Z39.50 is a client/server based standard for information retrieval [10]. Z39.50 specifies procedures and structures for a client to search a database provided by a server, retrieve database records identified by a search, scan a term list, and sort a result set [10]. Z39.50 has goals similar to DASL for enhancing the interoperability of information retrieval, but Z39.50 is different from DASL in many ways. Z39.50 focuses on the database resources, while DASL searches web resources. Furthermore, Z39.50 is a connection-oriented protocol that uses a custom syntax, while DASL is connectionless and leverages the HTTP infrastructure. Finally, Z39.50 repositories typically do not support arbitrary, user-defined (or dead) properties.

RCS is a source code version control system [8]. RCS is focused on source code version control in the local filesystem. There is no remote version control support in RCS and, in order to use it, we need login access to the machine on which RCS is installed. RCS uses an archive file to store version information. For example, if we put a foo.c under version control, RCS creates an archive file, foo.c,v and archives version history and delta information into the archive file. With this data model, it is difficult to control binary files or directory versions.

World Wide Revision Control (WWRC) extends version control to the WWW environment [11]. WWRC is implemented as a CGI program built on top of RCS. Using a generic web browser and a customized application viewer, a user can use the version control system through WWRC. Since the data model is inherited from RCS, WWRC suffers the same limitations as RCS.

As an additional obstacle to widespread adoption, the WWW/CGI style transaction method limits interoperability.

CVS addresses the remote access problem using its own protocol [12]. CVS doesn't use a lock mechanism to prevent the overwrite problem. CVS uses optimistic concurrency control which allows concurrent editing, with merging of changes occurring later. Still, CVS uses an RCS-like data model, so CVS has difficulty when versioning binary files or directories.

James Hunt reported his experience implementing an early pre-standardized version of the DeltaV protocol. He implemented a DeltaV server on top of the Revision Control Engine (RCE) [24], and his work focused on the mapping between DeltaV and RCE. Since this work focused on mapping DeltaV to an existing version control system, which doesn't support all of the functions that DeltaV defines, the system is not scalable enough to support the full DeltaV protocol.

Subversion is an open source version control system similar to CVS [13]. The goal of the Subversion project is to build a better version control system than CVS, but using the same user interaction model. Subversion uses the WebDAV and DeltaV protocols but is not a general-purpose WebDAV/DeltaV server, as it uses several custom options and properties. Subversion sends only deltas for PUT, a behavior not understood by general purpose WebDAV clients. Further, Subversion doesn't support LOCK and UNLOCK, which are basic WebDAV functions. As a result, even though Subversion partially implements the DeltaV protocol, its lessons are not completely generalizable for all DeltaV implementations.

4. DESIGN AND IMPLEMENTATION

In this section, we present the design and implementation of Catacomb. The database schema that represents the DeltaV/DASL data model is described and we examine the Catacomb architecture. Additionally, we explain the data structures and interfaces between mod_dav and Catacomb, and then describe the algorithm employed to parse DeltaV/DASL requests and generate responses. We start by describing the database design.

4.1 Database Design

Catacomb stores all contents and properties of resources into a database since a database is much more flexible than a filesystem. At first blush, it may seem that we could use a filesystem and RCS style archive files instead of a database, however a filesystem has several drawbacks. First of all, it is hard to represent the WebDAV resource data model. WebDAV resources consist of content and its properties. To store the content and properties in a filesystem, each resource must have a content file and a property file. It is hard to maintain the consistency of these related files. PROPFIND with depth infinity is resource intensive work in the filesystem since it requires recursively browsing the filesystem. Implementing SEARCH is very inefficient in a filesystem since it has to open all of the content and property files to execute a query. Filesystems make it complicated to represent the relationships between files within a versioning data model. For example, version resources have a relationship between root revision and child revisions or a relationship between previous revision and next revisions.

In order to overcome the drawbacks of the filesystem, Catacomb needs a flexible data storage system such as a relational database. Search related operations can benefit from the database's search capability, especially SQL, and a database can easily represent complex relationships among resources. Catacomb currently stores all resource content in the database to support content searching. However, storing all contents in the database is limited by the maximum data size for each data field. For example, in MySQL 3, the maximum data size for each field is 16M bytes [25]. We can solve this problem by storing content as an external file or by dividing the content into smaller data chunks and storing each piece separately. Since we were primarily focusing on implementing DeltaV/DASL, we have left this problem for future work.

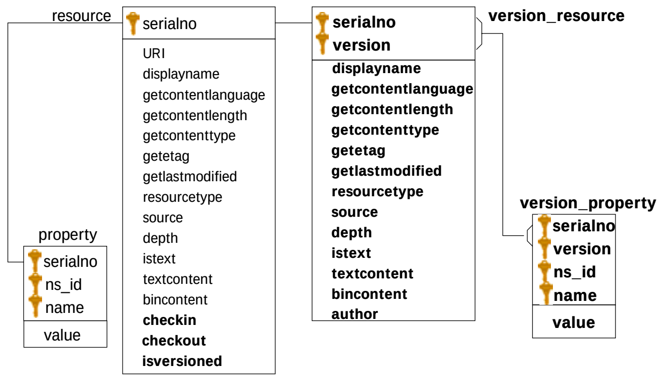

Figure 3. Database schema for Catacomb. Key icons indicate primary key fields. Version related tables and data fields are shown in bold.

Two main database tables store the contents and properties of resources. The resource table contains the resource content and predefined properties specified in RFC 2518 [1]. The property table is used to store user defined dead properties. The relationship between the resource table and the property table is 1 to n. In order to support resource versioning, we use three fields in the resource table. These version specific fields, checkin, checkout, and isversioned, keep track of the state of a resource. We use two tables, version_resource and version_property, to store content of a version resource and its properties.

The version_resource table is a clone of the resource table, and the version_property table is a clone of the property table. There are several reasons to have separated resource and version tables. The data fields in two tables may have different semantics. For example, the textcontent field in the resource table contains the content of a resource, but the textcontent field in the version resource table may contain only a delta. Also note that most of the WebDAV/DeltaV methods such as PUT, COPY, MOVE, and PROPFIND need only the data present in the resource table. In that sense, having two separated tables is helpful for Catacomb's performance. For a version resource, we store the full content in the textcontent field of version resource, although the database schema is flexible to support storage of deltas if storage space is an issue. The resource and property tables keep the latest version of the resource content and its properties, while version_resource and version_property tables keep the resource content and properties of each revision. The relationship between resource and version_resource is 1 to n, since each resource can have one or more revisions.

4.2 DeltaV

Catacomb is a module for Apache2, so its architecture depends on the Apache module architecture. Since Catacomb has to communicate with mod_dav, the Catacomb module conceptually lies under mod_dav. The basic architecture of Apache, mod_dav, and Catacomb is shown below in figure 4. The Apache core module handles HTTP requests and responses. Mod_dav interprets WebDAV requests and passes the information to lower-level modules such as mod_dav_fs or Catacomb. Mod_dav gets some information back from the lower module in order to send responses back to clients.

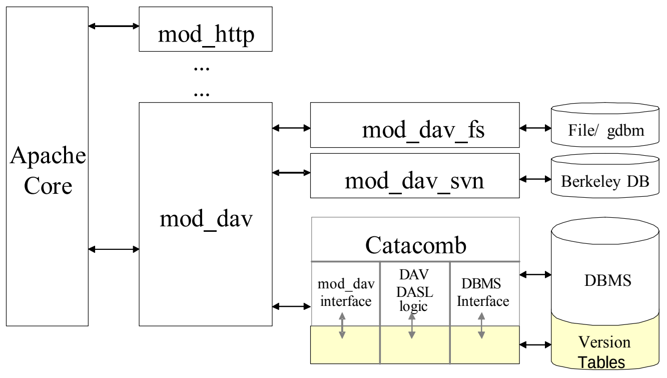

Figure 4. Catacomb architecture. The shaded sections indicate modules used to support versioning.

The Catacomb module receives request information including the request URL, request method, and request message from mod_dav. If PUT, PROPPATCH, LOCK, UNLOCK, CHECKIN, or CHECKOUT methods are being handled, the Catacomb module stores information into the database. To support the DeltaV protocol, we added function hooks for communicating with mod_dav. The DAV core module in Catacomb was extended to understand the DeltaV requests and to generate the DeltaV responses. The database module was also extended to manipulate the new DeltaV database tables.

The Catacomb implementation process can be divided into three parts: implementing the mod_dav versioning hooks, adding database management capability, and adding DeltaV logic modules. To get the DeltaV requests from mod_dav, we implemented function hooks between mod_dav and Catacomb. In between the mod_dav hooks and the database management module, the DeltaV logic module manipulates the information, and stores it into the database. In addition, the DeltaV logic reads information from the database and sends it back to mod_dav. To support the basic versioning functionality, Catacomb has to understand the VERSION-CONTROL, REPORT, CHECKIN, CHECKOUT, and UNCHECKOUT methods. The method requests are parsed in mod_dav, which passes the request information to Catacomb via the registered hooks.

The second part of the implementation is the DeltaV logic module, which maintains the state of the resource, performing actions based on that state and the request being processed. A resource in Catacomb uses a state machine to model the current versioning state and allowable transitions. The resource state cycle is depicted in figure 5.

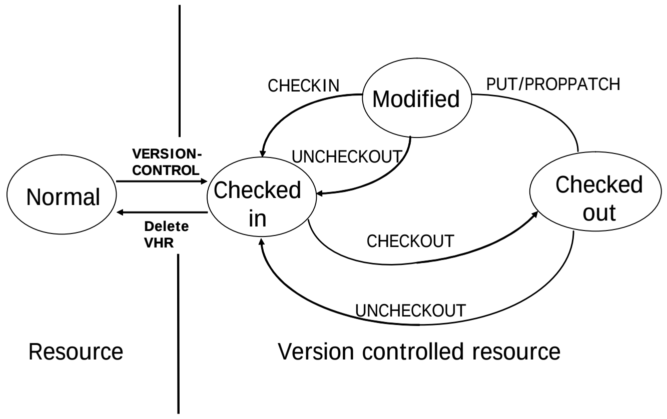

Figure 5. Resource States in Catacomb

Initially a resource is in the normal state. If the client requests the VERSION-CONTROL method, the state of resource is changing to the version controlled state. All version controlled resources are in the checked-in state at first. We checkout resources in the checked-in state by using the CHECKOUT method, which makes it possible to modify the resource. If a resource is in the checked-in state, we cannot update the resource until the resource is in the checked-out state. After a resource has been checked out and updated using PUT or PROPPATCH, the resource can be checked in.

At the moment of the CHECKIN request, the content and properties of previous version resource are copied into the version_resource and version_property tables in the database. This action creates a version resource of the resource. Since the version resource is read-only resource, the new version resource preserves the content and properties of a resource when the resource is checked in. If a resource is in checked-in state, the resource and its latest version resource have the same content and properties.

Each revision resource has a unique URL that is not changeable. Catacomb creates the revision URL by combining the root revision URL, version separator, version number, version separator, and file name. For example, if a resource's root revision URL is /repos/jim.html, the revision URL for revision number 2 is /repos/jim.html/!/2/!/jim.html. Catacomb uses '/!/' as a version separator, since it is rarely used in URI. The reason that Catacomb uses the file name for the revision URL suffix is because many WebDAV/DeltaV clients determine the content type of a resource by looking the extension of the filename in the URL.

Several fields in the resource table are used to keep the status of resource. The isversioned field in the resource table indicates whether the resource is in the version controlled state. The checkin and checkout fields in the resource table are used to keep track of the version controlled resource state, as follows: one of two fields contains the current version number and the other field is set to -1 value. For example, if the checkin value is set to -1 and checkout is set to 5, the resource is in the checked out state and next version number is 5. If there is a CHECKIN request, Catacomb creates a new version resource and its version number is 5. After that, Catacomb increases the version number by one, and flips the checkin and checkout values to indicate the state change. The resource is in checked-in state and version number is 6.

Catacomb supports the auto-versioning function which is specified in DeltaV. The auto-versioning function is an automatic checkin/checkout function for generic WebDAV clients which do not support DeltaV. For example, if the client tries to PUT or PROPPATCH without CHECKOUT request, the server automatically does CHECKOUT for PUT or PROPPATCH and does CHECKIN after the PROPPATCH or PUT requests.

There are two ways to implement the auto-versioning function: change-based and lock-based auto versioning. Change-based auto-versioning function wraps a checkout/checkin pair around every PUT or PROPPATCH method. Lock-based auto-versioning performs a checkout when there is a LOCK request and a checkin when there is a UNLOCK request. Change-based auto-versioning creates a revision whenever the resource is changed, so it has the possibility of creating more revisions than strictly necessary. In order to avoid this, Catacomb supports only lock-based auto-versioning.

4.3 DASL

The DASL logic module in Catacomb implements the specific semantics of DASL operations. In the processing of a SEARCH request, the mod_dav module and the mod_dav interface components parse the WebDAV and DASL protocol requests into operations on the repository. The DASL logic component handles the SEARCH method. Particularly, the DASL logic translates the XML DASL query in the body of a SEARCH request into a SQL query that will be executed against the database. Additionally, when the query results are returned, the DASL logic component is responsible for converting the search results into XML format, which will be put into the response message. The DBMS interface provides DBMS APIs to handle the resources and their properties in the database.

One major task of the DASL logic module in Catacomb is to translate the XML DASL query in the SEARCH request into a SQL query that will work on the 'dasl_resource' and 'dasl_property' tables in the database. The difficulty in translating the DASL query is that dead properties are stored as rows in the dasl_property table. This necessitates aliasing the dasl_property table in the SELECT portion of the query and then using left join operations to self-join the 'dasl_property' table 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. A sample SEARCH request and its corresponding SQL query are shown in Table 1.

In general, the major rules used in translating a DASL query to a SQL query are as follows. Each live property in the result record definition section of the DASL query goes to the SELECT clause of the SQL query directly. For each dead property in the result record definition section of a DASL query, add 't.name, t.value' to the select clause of the SQL query. Also, add 'LEFT JOIN dasl_property t USING (serialno)' to the right of 'dasl_resource' in the FROM clause of the SQL query, and use OR to connect the boolean expression 't.name = <dead property name>'. Each logical expression on a live property in the search criteria of the DASL query becomes part of the WHERE clause of the SQL query. For each logical expression on a dead property in the search criteria of the DASL query, add 'LEFT JOIN dasl_property <dead property name>_t USING (serialno)' to the end of the FROM clause of the SQL query. Further, add 'AND (<dead property name>_t.name = "<dead property name>" AND <dead property name>_t.value <condition on this dead property>)' to the end of the WHERE clause of the SQL query. The live properties in the sort specification are mapped to the 'ORDER BY' clause directly. There is currently no support for sorting the results by a dead property. The search scope is translated into a search condition in the WHERE clause: uri like '<href in scope>%', where '<href in scope>' stands for the value of the 'href' XML element in the search scope.

Table 1. A sample DASL query and its corresponding SQL query.

```
<d:searchrequest xmlns:d="DAV:">
  <d:basicsearch>
    <d:select>
       <d:prop>
         <d:displayname/><d:foo/><d:bar/>
       </d:prop>
    </d:select>
    <d:from>
      <d:scope>
        <d:href>/</d:href>
        <d:depth>infinity</d:depth>
      </d:scope>
    </d:from>
    <d:where>
      <d:gt>
        <d:prop><d:bar/></d:prop>
        <d:literal>2518</d:literal>
      </d:gt>
    </d:where>
  </d:basicsearch>
</d:searchrequest>
```

```
SELECT
          dasl_resource.displayname,
          t.name, t.value
 FROM
         dasl_resource
         LEFT JOIN
         dasl_property t USING (serialno)
         LEFT JOIN
        dasl_property bar_t USING (serialno)
 WHERE
          ( bar_t.name = 'bar' AND
          bar_t.value > 2518 )
 AND
          ( t.name = 'foo' OR t.name = 'bar' )
```

5. SEARCH PERFORMANCE

In order to test the search performance of Catacomb, we ran several tests using Cadaver [26], a WebDAV/DASL client. We used various database loading conditions, shown in table 2, including small numbers of resources and large numbers of resources. The results of the tests helped us determine the conditions that impact the search performance of the DASL server and suggest ways to alter the algorithm to improve the search performance.

Table 2. Database Conditions for Test

	Collection under which the tests were run	Number of resources under this collection	Total number of resources	Number of dead properties
1	/repos/dasltest/cadaver1/	78	1161	1694
2	/repos/dasltest/	780	1161	1694
3	/repos/dasltest/cadaver1/	78	24332	1694
4	/repos/dasltest/	23961	24332	1694

Table 3 shows the test cases and the test results of the search performance of Catacomb under different search conditions and database situations.

We should take notice of the two test results in table 3, where the searching time is exceptionally long. The first case is when the search condition is a dead property condition, 'Author=Joe', and the number of the resources in the search collection is 23961. The second case is when the search condition is 'Author=Joe And comments=XML', and the number of the resources in the search collection is also 23961.

The reason for these exceptional test results is that MySQL works inefficiently on left-joining large sets of data. We have to use left joins in the algorithm to construct the SQL query, because it is necessary for the 'OR' operator in the search condition. But we can alter our algorithm to use natural joins instead of left joins under the circumstances where left joins are not necessary. Thus, the left-join performance issue can be partially overcome with some further work.

6. CONCLUSIONS AND FUTURE WORK

In this report, we have presented an open architecture for performing DeltaV versioning and DASL searching operations in the context of the Apache web server. The WebDAV/DeltaV/DASL data models are represented using a relational database. These tables and schema can be reused for implementing DeltaV advanced versioning functionality by adding several status fields. As validation of this architecture, we have shown the design of Catacomb, the first open source reference implementation of a DeltaV/DASL server.

The Catacomb project is an ongoing open source project that has achieved several milestones, however the project is far from complete. Currently, Catacomb only supports MySQL as its backend database [23, 25]. It is desirable to support the use of other database systems such as PostgreSQL, Oracle, Sybase, Informix, and DB2 with Catacomb. In order to support multiple database systems efficiently, Catacomb will have to provide an abstract database layer as a different module. Mod_dav has a sub-module for its repository access such as mod_dav_fs or Catacomb modules. Like this relationship, Catacomb may have Catacomb_mysql or Catacomb_oracle modules for database access.

Catacomb supports only basic versioning features at this point. In order to be a full software configuration management framework, Catacomb should be able to support advanced versioning functionality such as workspaces, baselining, activities, and configuration recording features specified in the DeltaV specification. In spite of this, however, Catacomb already provides a rich environment for collaborative authoring of resources.

Table 3. Search Performance Test Results

Search Condition	Explanation	Database Situation	Avg. Time Used (Second)	Records Returned
getcontentlength =5770	1 live property condition	1	0.0442	1
		2	0.0574	10
		3	0.0443	1
		4	0.0591	10
Author=Joe	1 dead property condition	1	0.0546	2
		2	0.0933	2
		3	0.0548	2
		4	12.7452	2
Author=Joe AND comments=XML	2 dead property conditions	1	0.0551	1
		2	0.0949	1
		3	0.0554	1
		4	12.9884	1
Getcontentlength =5770 AND author=Joe AND comments=XML	1 live property condition and 2 dead property conditions	1	0.0445	1
		2	0.0459	1
		3	0.0447	1
		4	0.0460	1
Getcontentlength =5770 AND author=Joe AND comments=XML AND school=UCSC AND department=CS AND project=catacomb	1 live property condition and 5 dead property conditions	3	0.0452	1
		4	0.0466	1

7. ACKNOWLEDGEMENTS

This project is supported by the National Science Foundation under Contract Number NSF CAREER CCR-0133991. We also would like to thank the Catacomb developers and user community for their commitment and valuable feedback.

8. REFERENCES

[1] Y. Goland, E. J. Whitehead, Jr., A. Faizi, S. Carter, and D. Jensen, "HTTP Extensions for Distributed Authoring -- WEBDAV," *Internet Proposed Standard Request for Comments (RFC) 2518*, 1999.

[2] E. J. Whitehead, Jr. and Y. Y. Goland, "WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web," *Proceedings of Sixth European Conference on Computer Supported Cooperative Work*, Copenhagen, Denmark, pp. 291-310, 1999.

[3] J. Feise, "Posties: A WebDAV Application for Collaborative Work," *Proceedings of Eleventh ACM Conference on Hypertext and Hypermedia (Hypertext '00)*, San Antonio, Texas, USA, pp. 228-229, 2000.

[4] E. J. Whitehead, Jr., "WebDAV and DeltaV: Collaborative Authoring, Versioning, and Configuration Management for the Web," *Proceedings of Twelfth ACM Conference on Hypertext and Hypermedia (Hypertext '01)*, Aarhus, Denmark, pp. 259-260, 2001.

[5] G. Clemm, J. Amsden, T. Ellison, C. Kaler, and J. Whitehead, "Versioning Extensions to WebDAV," *Internet Proposed Standard Request for Comments (RFC) 3253*, 2002.

[6] A. Babich, J. Davis, R. Henderson, D. Lowry, S. Reddy, and S. Reddy, "DAV Searching and Locating," *Unpublished manuscript, draft-davis-dasl-protocol-00*, 2000.

[7] G. Clemm, "IETF Delta-V Working Group Home Page," http://www.webdav.org/deltav/, 2003.

[8] W. F. Tichy, "Design, implementation, and evaluation of a Revision Control System," *Proceedings of 6th International Conference on Software Engineering*, Tokyo, Japan, pp. 58-67, 1982.

[9] S. Boag, "XQuery 1.0: An XML Query Language - W3C Working Draft," http://www.w3.org/TR/xquery/, 2003.

[10] Z39.50 Maintenance Agency, "Information Retrieval (Z39.50)," *Application Service Definition and Protocol Specification (ANSI/NISO Z39.50-1995).* Bethesda, MD, USA: NISO Press, 1995.

[11] J. Reuter, S. U. Hanssgen, J. J. Hunt, and W. F. Tichy, "Distributed Revision Control via the World Wide Web," *Proceedings of Sixth Software Configuration Management Workshop*, Berlin, Germany, pp. 166-174, 1996.

[12] B. Berliner, "CVS II: Parallelizing Software Development," *Proceedings of Winter 1990 USENIX Conference*, Washington, DC, pp. 341-351, 1990.

[13] B. Behlendorf, C. M. Pilato, G. Stein, K. Hancock, and B. Collins-Sussman, "Subversion Project Homepage," http://subversion.tigris.org/, 2003.

[14] R. Fielding, J. Gettys, J. Mogul, H. F. Nielsen, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," *Internet Draft Standard Request for Comments (RFC) 2616*, 1999.

[15] G. Clemm, A. Hopkins, E. Sedlar, and J. Whitehead, "WebDAV Access Control Protocol," *Internet-Draft, work-in-progress, draft-ietf-webdav-acl-12*, 2003.

[16] G. Clemm, J. Crawford, J. Reschke, J. Slein, and E. J. Whitehead, "Binding Extensions to WebDAV," *Unpublished manuscript, draft-ietf-webdav-bind-01.2*, 2003.

[17] J. Slein, E. J. Whitehead, Jr., J. Davis, G. Clemm, C. Fay, and J. Crawford, "WebDAV Redirect Reference Resources," *Internet-Draft, work-in-progress, draft-ietf-webdav-redirectref-protocol-02*, 1999.

[18] J. Slein, E. J. Whitehead, Jr., J. Davis, G. Clemm, C. Fay, and J. Crawford, "WebDAV Ordered Collections Protocol," *Internet-Draft, work-in-progress, draft-ietf-webdav-ordering-protocol-02*, 1999.

[19] Apache Foundation, "The Apache HTTPD Server Project," http://httpd.apache.org/, 2003.

[20] Apache Foundation, "Apache Module Registry," http://modules.apache.org/, 2003.

[21] G. Stein, "mod_dav: a DAV module for Apache," http://www.webdav.org/mod_dav/, 2003.

[22] GNU, "gdbm - GNU Project - Free Software Foundation (FSF)," http://www.gnu.org/software/gdbm/gdbm.html, 1999.

[23] S. Kim, K. Pan, and E. Sinderson, "Catacomb Project Homepage," http://www.webdav.org/catacomb, 2001.

[24] J. J. Hunt and W. F. Tichy, "RCE API Intro. and Ref. Manual," DuraSoft 2000.

[25] MySQL AB, "MySQL: The World's Most Popular Open Source Database," http://www.mysql.com, 2003.

[26] J. Orton, "Cadaver Resources," http://www.webdav.org/cadaver/, 2003.