Third International Conference on Information Science and Technology
March 23-25, 2013; Yangzhou, Jiangsu, China

Application of Document-Oriented NoSQL Database Technology in Web-based Software Project Documents Management System

Shaolong Zhang

Abstract—**In the development of software project system architectures, software programmers, product test engineers and clients who may be distributed over the world need good communication and collaboration. They require easy access to necessary data and sharing of relevant documents of the project which are distributed in a network of resources and users. Hence, web-based applications, decentralized repositories and databases are needed to store and manage project and process development information. However relational databases work poorly in performance, scalability and reliability while processing large amounts of big data spread out across many servers. This paper works for managing software project documents in an Internet-based collaborative environment using NoSQL database technology. The research work focus on the development of a web-based application using Apache CouchDB which is an open source document-oriented database with REST web services interface. The proposed application is described and the main functionalities are illustrated through the examples of use.**

I. INTRODUCTION

IN the development of software project a lot of project documents such as software requirements document, software design documents and project progress reports need to be archived and shared between stakeholders like system architectures, software programmers, product test engineers and clients. The sharing activity can be enhanced through the use of computer and communication technology especially through the Internet which can be considered the best channel for collaboration and knowledge exchange. The distributed storage system plays the most important role which needs good scalability for simple read/write operation with big data files. However relational databases such as MySQL, MS SQLServer and Oracle store data in the form of two-dimensional tables which are good at storing highly structured and interrelated data but works poorly in performance, scalability and reliability while processing large amounts of big data spread out across many servers[1,2].

In this paper, we propose a new approach to manage distributed software project documents which based on Document-oriented Database (DoDB) instead of the usual relational database. Our work implemented a Web-based prototype application which is convenient for users to manage software project documents in the collaborative distributed environment.

Document-oriented Database is one of kind of NoSQL. The definition of NoSQL, which stands for "Not Only SQL" or "Not Relational", is not entirely agreed upon. The key feature of NoSQL systems is "shared nothing" horizontal scaling – replicating and partitioning data over many servers. This allows them to support a large number of simple read/write operations per second.

In the past few years, many scalable NoSQL related distributed storage systems have been proposed, e.g. Google's Bigtable [3], Amazon's Dynamo [4] and Yahoo's PNUTS [5].

Unlike relational (SQL) DBMSs, NoSQL systems have no unique data model. There are four popular data storage types according their data model: Key-value Stores, Document Stores, Extensible Record Stores and Relational Databases [6]. Document stores support more complex data than the others.

The term "document" can be any kind of "pointerless object" including texts, Microsoft Word files, Video files, Microsoft PowerPoint files, etc. Since we aim to deal with all kinds of project documents, document-oriented NoSQL database would be the best choice for our wok. In our research, we implement the storage system based on Apache CouchDB which is an open source document-oriented database supported by Apache project.

The rest of this paper is organized as follows: Section 2 gives a brief description of the basic model of Apache CouchDB [7]. The system architecture and the software design are illustrated in Section 3 and we show the result of our implementation in section 4. The conclusion and analyses the advantages and disadvantages compared with relational database are proposed in Section 5.

II. OVERVIEW OF APACHE COUCHDB

Apache CouchDB is a scalable, fault-tolerant, and schema-free document-oriented database written in Erlang. CouchDB's reliability and scalability is further enhanced by being implemented in the Erlang programming language which used to build massively scalable soft real-time systems with requirements on high availability[8]. In CouchDB data is stored in documents, presented in key-value maps using the data types from JavaScript Oriented Notation (JSON) .

A. Documents And Views In CouchDB

A CouchDB server hosts named databases, which store documents. A CouchDB document is an object that consists of named fields. Field values may be strings, numbers, dates, or even ordered lists and associative maps. A CouchDB document is simply a presented as a JSON object with some associated metadata. Documents can have attachments just like email. All kinds of files can be attached to a document.

CouchDB is designed to store and report on large amounts of semi-structured, document oriented data. With CouchDB,

S Zhang is with the Network Center, Zhejiang Radio & Television University, Hangzhou, Zhejiang, 310012, China (e-mail: zhangsl@zjtvu.edu.cn).

978-1-4673-2764-0/13/$31.00 ©2013 IEEE

no schema is enforced, so new document types with new meaning can be safely added alongside the old. CouchDB greatly simplifies the development of document oriented applications, which make up the bulk of collaborative web applications.

Queries are done with what CouchDB calls "views", which are defined with JavaScript to specify field constraints. Queries can be distributed in parallel over multiple nodes using a map-reduce mechanism.

B. Asynchronous Replication Of CouchDB

CouchDB is a peer based distributed database system. Any number of CouchDB servers can have independent "replica copies" of the same database, where applications have full database interactivity (query, add, edit, delete). When back online or on a schedule, database changes are replicated bi-directionally.

CouchDB has built-in conflict detection and management and the replication process is incremental and fast, copying only documents and individual fields changed since the previous replication. CouchDB will notify an application if someone else has updated the document since it was fetched. The application can then try to combine the updates, or can just retry its update and overwrite.

C. HTTP REST Web Services

CouchDB provides a set of RESTful HTTP APIs for reading and updating (add, edit, delete) database documents. All items in database have a unique URI that gets exposed via HTTP. REST uses the HTTP methods POST, GET, PUT and DELETE for the four basic CRUD (Create, Read, Update, Delete) operations on all resources.

With the powerful features of CouchDB, it is possible to build a distributed document database management application that hold digitized information and replications including text, images, audios or videos.

In the next section the system architecture is presented which aims at supporting our purpose.

III. System Design

A. System Architecture

The goal of our work is to provide a Web-based application which is convenient for users to access software project documents on multiple storage servers. Users from anywhere could log on the local node server in the system and download or upload project documents in the system.

The system is based on distributed CouchDB DoDB. CouchDB works as a database server and a Web Application server which provide RESTful HTTP APIs for client development. CouchDB supports many programming languages including C, C#, Java, JavaScript. In this work, we implement client application in JavaScript which is easy to connect server with an AJAX technology and deal with JSON object returned from CouchDB. A schematic representation of how the system has been implemented is shown in Figure 1.

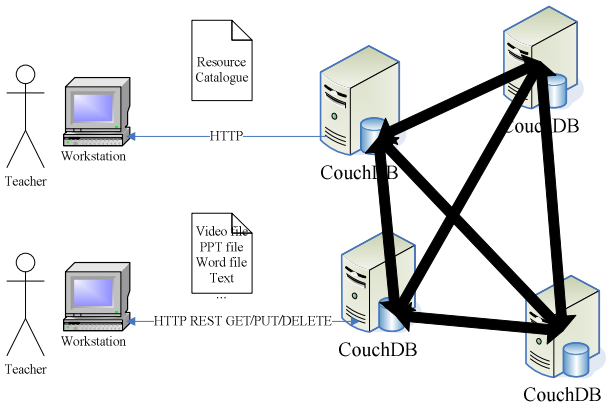

Fig. 1. Distributed system architecture based on CouchDB

We could deploy any number of database servers. Only one of them could be master database server, the others are slave database servers.

Replication between servers is triggered by sending a POST request to the _replicate URL with a JSON object in the body that includes a source and a target member.

POST /_replicate HTTP/1.1

{"source":"example-database","target":"http://example.org/example-database"}

In the next sections, we will describe in detail the design of database and Web client.

B. Entity Relationships Model Of Database

Before we implement the database, we need design the entity relationships model of the system. In our system the core entities are users, projects, project items and project documents. The relationships between them could be described as follows:

1) Every user belongs to a software project and each item belongs to a software project.
2) For each item, only users of the project have rights to add/update documents in this item.
3) Each item could have any number of software project documents and each document only belongs to one item.

The E-R diagram is represented as following figure2.

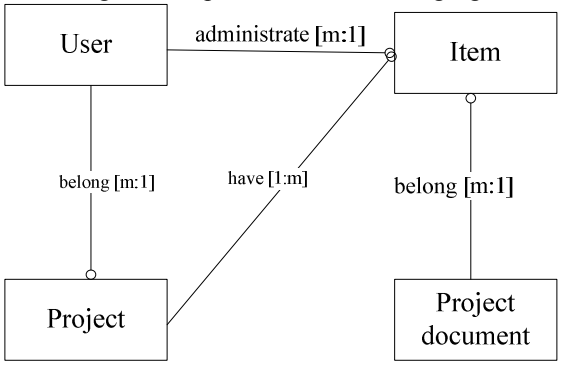

Fig. 2. E-R diagram

The relations between these entities are one-to-many type. In CouchDB, there are two ways to achieve this type:

1) Use separate relationship documents,
2) Use an embedded array in a document.

Since the embedded array is only an option for not too many items to store and bigger documents mean slower handling and slower network transfers. We choose the separate relationship documents way.

Some sample documents in a typical case are:

```
{
  "_id": "a_case_of_project",
  "_rev": "ad2d1c186f853",
  "type": " project",
  "name": "E-Learing",
  " project_id": "E-123480"
}
{
  "_id": "a_case_of_user",
  "_rev": "2f7d520c0d628a",
  "type": "user",
  "name": "Steve Zhang",
  "user _id": "ee001",
  " project_id": "E-123480"
}
{
  "_id": " a_case_of_item",
  "_rev": "0d36967e465",
  "type": "item ",
  "item _name": "Poject Requirements",
  "item _id": "sys_reqs",
"item_description": "any project requirement documents",
" project_id": "E-123480"
}
{
  "_id": " a_case_of_document ",
  "_rev": "f0c0926e7b51b",
  "type": " document",
  " document_type": "DOC",
"document_name": "project hardware requirement",
  " document_id": "dc001",
"item_id": " intros",
  "_attachments": {
    "project hardware requirement.doc": {
      "content_type": "application/vnd.ms-word",
      "revpos": 2,
      "digest": "md5-zFFMEjuhk/5Fe7pndcFNhA==",
      "length": 54800,
      "stub": true
    }
  }
}
```

C. Database View Design

Unlike SQL for relational database, CouchDB use views for querying and reporting on CouchDB documents. Views are the method of aggregating and reporting on the documents.

CouchDB uses Google's MapReduce programming model [9] for views. Views are defined by a JavaScript function that maps view keys to values. To create a view, the functions must first be saved into special design documents. The IDs of design documents must begin with _design/ and have a special views attribute that have a map member and an optional reduce member to hold the view functions.

In our work a design document that defines user_project, all_items and documents_of_item views might look like this:

```
{
  "_id": "_design/dbs",
  "_rev": "10-de6a28a0d90a01c4b42428d658f211c7",
  "views": {
" user_project ": {
      "map": "function(doc){ if (doc.type == 'user') emit(doc.user_id, doc.project_id)}"},
    "all_items": {
      "map": "function(doc){ if (doc.type == 'item') emit(doc. project_id ,{item_id:doc.item_id, item_name:doc.item_name})}"},
    " documents_of_item": {
      "map": "function(doc){ if (doc.type == 'document') emit(doc.item_id ,doc)}"}
  }
}
```

With URL query arguments, we can use views to get data we need. For example, we can retrieve all items of the project from the URL like http://example.com:5984/test/_design/dbs/_view/all_items?key=" E-Learing".

D. The Implementation Of Web Client

Although CouchDB can work as both Web Server and database server where HTML pages and JavaScript files can be stored, it is a weaker Web Server than other professional Web Servers such as Apache HTTP server and MS IIS. In our work we use Apache HTTP server as a reverse Proxy for CouchDB. Browsers will typically enforce the same origin policy and reject requests to fetch data unless the protocol, port and host are identical to the source of the current page. Using a reverse proxy allows browser-hosted applications to access CouchDB while conforming to the same origin policy.

The whole process could be described with a UML sequence diagram.

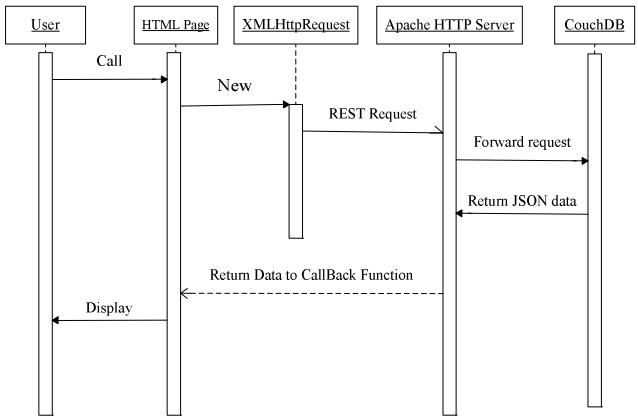

Fig. 3. UML Sequence diagram

When users open the HTML page, events will be triggered automatically or by user. Event function written in JavaScript sends REST request intercepted by the Ajax engine with XMLHttpRequst object. The Web server will accept and forward the request to CouchDB and return back data to client

while in the data transfer users can handle other things.

IV. CURRENT PROTOTYPE

At this stage of the work, we managed to produce a simple prototype of a web application that is able to access software project documents stored in CouchDB under a distributed network environment. The application is designed to allow users log on the system to download or upload any documents for sharing.

The user logs on the system in the local node server where all items are listed, then the user can click one item to see the all the documents information of the item. Attachments are also shown when they are documents like ppt files, word files, images, or video files.

Fig. 4. Page of documents management

The user also could download or add/delete documents of this item.

V. CONCLUSION

In this work we explore the use of recent NoSQL database technologies, namely Apache CouchDB DBMS and REST web services to develop a system for distributed software project documents management. Compared with relational database, application based on CouchDB could take advantages of efficient storage of big data files and data replication over many servers. The peer-to-peer based distributed infrastructure of CouchDB also provides elastic scalability which means that we can add capacity easily by adding more servers.

However some disadvantages of NoSQL also could be found. NoSQL databases have no unique query language like SQL for relation database. Each NoSQL database such as CouchDB provides different query language which would be difficult for complex query programming. NoSQL databases also have weaker authentication and security management compared with relation database like Oracle and MS SQLServer. For the case of CouchDB, it only provides administrator privilege to manipulate database.

Whatever, NoSQL databases like CouchDB shows their potentiality in massive data storage under distributed network environment which would be more appropriate than relation database in Web 2.0 age.

REFERENCES

[1] M. Stonebraker, "SQL Databases v. NoSQL Databases", *Communications of the ACM*, vol. 53, no. 4, pp.10-11, April. 2010.

[2] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?", *Computer*, vol. 43, no. 2, pp.12-14, February. 2010.

[3] F.Chang et al,"BigTable:A Distributed Storage System for Structured Data", in *Seventh Symposium on Operating System Design and Implementation*, Seattle, USA, 2006, pp.205-218.

[4] B.DeCandia et al,"Dynamo:Amazon's Highly Available Key-Value Store", in *Proceedings 21st ACM SIGOPS Symposium on Operating Systems Principles*, Stevenson, USA, 2007, pp.205-220.

[5] B. F. Cooper et al., "PNUTS: Yahoo!'s Hosted Data Serving Platform", in *Proceedings of the VLDB Endowment,* Auckland, New Zealand, 2008, pp.1277-1288.

[6] C. Rick, "Scalable SQL and NoSQL Data Stores." *ACM SIGMOD Record*, Vol. 39, no. 4, pp.12-27, December.2010.

[7] Apache Software Foundation. Apache CouchDB: The Apache CouchDB Project. [Online].Available: http://couchdb.apache.org/.

[8] Erlang.org. Erlang programming language. [Online].Available: http://www.erlang.org/

[9] J. Dean, and S. Ghemawat, "MapReduce: simplified data processing on large clusters", *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, January.2008.