JasDB

Project Report-by

Huy Khac To , Dinesh Aggarwal, Krishna Chaitanya Tatikonda ,

Department of Computer Science,

Georgia State University,

Atlanta,30003.

Introduction:

The word NoSQL is there in existence from 1998 in various different forms. There are several good projects that have made good use of NoSQL concepts like Lotus notes and very popular Facebook. These incarnations of NoSQL is the outcome of the problems experienced by the software developers in various different areas like scaling of data and quick retrieval of data. When these solutions are invented using NoSQL concepts, they look like solutions for the specific problem rather than a enterprise product. These solutions are developed just as the need of the product. These solutions are unable to solve large number of other problems.

Now a days the availability of internet and smart phones are more and they became more affordable, because of this there is a huge increase in data which is generated by the users. All these loads of data has to be stored and need to be accessed fast enough so as to engage users with the commercial products. Relational databases with conventional mechanisms for storing data were not loaded with all these functionalities so as to deal with these activities. This is the reason where NoSQL officially came into existence.

NoSQL stands one step ahead of conventional database design. Relational database stores data into tables containing rows and columns. Columns are used maintain relationships among the data that's been stored. NoSQL is a combination of all good qualities taken from various database design mechanisms and approaches. There are different kinds of NoSQL databases and each one of them are having their own applications and their own design trade off's. Every business has their own set of requirements and their own needs. Based on their requirements business people choose different NoSQL databases.

NoSQL Database:

The major factor which lead to database scalability is multi-table correlation which is the major and very important concept of relational databases. Because of different varieties of business requirements there are wide range of databases have been created. These databases are quite different from traditional relational database systems. So these databases are coined as NoSQL database. It can also be called as "Not only SQL" just to show the advantage of NoSQL.

Features of data model:

These are the main advantages of using NoSQL database as enterprise databases:

- Reading and writing data quickly.
- Support for mass data storage.
- Expansion of database is easy.
- Low cost when compared to regular RDBMS since all these databases are developed using open source software's.

NoSQL has some disadvantages too:

- It does not support SQL which is Industry standard.
- It lacks the capability of transactions.
- It lacks reporting functionality etc..

All these features of NoSQL are common to all NoSQL databases theory but in reality every product have their own strengths, different data models and CAP theorem.

Most commonly used Data Model in NoSQL databases:

NoSQL data modeling often starts from the application-specific queries as opposed to relational modeling. Traditional databases have a data model which is relational, specifically it supports associated class operations and ACID operations. In case of NoSQL database fields very commonly used databases are:

- Key - Value
- Column Oriented
- Document

Key - Value: In simple words key value data model mean that a value corresponds to a key but still structure is simple and query accessing speed is higher than any other relational database. It also support mass storage and high concurrency. In this data model , query and modifying operations are

through the primary key and they are supported well. It is very simplistic and powerful technique/data model. Many of the problems that have been faced by huge data applications are successfully handled by this data model.

Column Oriented: This database uses table as the data model but this will not support table association. It has the following characteristics:

- Data is stored by column and data stored separately for each column
- Each column is the database index
- Only column with data involves in the querying part so less number of I/O of system
- Has the capability of parallel processing and each column is processed by one process
- Contains same type of data(no data types as such) and has good compression ratio

This kind of data model is highly used in where high aggregation of data is required and can be formed as highly available data warehouse.

Document: Both document based and key value based data model have very similar structure. But it varies in one approach the values of document database is semantic. Documents are stored in JSON or XML format. Document databases maintains a secondary index so as to facilitate upper applications. This feature cannot be supported by key value data model.

Document data model advanced the BigTable concept with two major improvements.

- It has values with schemes of obituary complexity
- Database managed indexes

This approach is taken from full text search functionality in the sense that they provide automatic indexing and database-managed indexes. The main difference is that document data model is that it uses group indexes by field names. Oracle is now using these functionalities in their databases via addition of indexes and in-database entry.

Ex: JasDB and MongoDB

JasDB : JasDB is a database which is between relational and non- relational database. JasDB is flexible and fault tolerant database which supports JSON data format. It comes with REST web service that allows retrieval of data from JasDB database.

JasDB has following features:

- JasDB is non-relational database that has richest features of relational database.
- It has very good support for complex data types.
- It has powerful query language and querying with database is simple like querying data from

single table in relational database.

- High speed access to mass data.

JasDB has following definitions:

Instance: An Instance is like a schema in RDBMS. It contains bags and entities and it also contains storage path on the disk associated to this bags and entities. All bags and entities are stored in the storage location.

Bags: Bags are like tables in the relational database. In JasDB it is called bag and it is big bag of data without any data definition associated with that bag. Any bag is related to any instance and the data files(JSON) are stored inside the instance storage location.

Entities: Entity contains data and it is stored inside a bag. Structure of entity is so simple that is can store any data of any data type. JasDB stores data as data files in JSON format.

JasDB can be used in two ways : In process and Running as remote storage.

In process is a mechanism that helps us to use JasDB as java service withour using remote server to access database. This mechanisms helps faster access of data and allows quick storage of data into database. Accessing JasDB using Inprocess mechanism is so simple, we just need to include all jar files to java project.

we can enable jasDB database using following kernel binding in the configuration file:

<jasdb kernel="nl.renarj.jasdb.core.SimpleKernelBinding">

we can control the lifecycle of jasDB using following commands:

//Forcible initialize JasDB, can also be lazy loaded on first session created

SimpleKernel.initializeKernel();

//There is a wait for shutdown method

SimpleKernel.waitForShutdown();

//shutting down JasDB on program end / web app shutdown

SimpleKernel.shutdown()

we can create instance at run time on local process where JasDB is running using API. This process is so handy while testing the database without changing configuration files. Below code will show how to start database and creating instance:

```
try {
        SimpleKernel.initializeKernel();
        try {
           SimpleKernel.getInstanceFactory().addInstance("myId", "/Path/To/DB");
            DBSession session = new LocalDBSession("myId");
           EntityBag bag = session.createOrGetBag("MyBag");
           bag.addEntity(new SimpleEntity().addProperty("Property", "Value"));
        } finally {
           SimpleKernel.shutdown();
        }
     } catch(JasDBStorageException e) {
        LOG.error("Unable to store", e);
     } catch(ConfigurationException e) {
        LOG.error("Unable to create instance", e);
  }
```

REST Web Service:

JasDB comes with REST webservice so as to use to retrieve data from database. By default it runs on 7050 port.

Operations:

- Creation of bags:

```
curl -H "Content-Type: application/json"
-X POST "http://localhost:7050/Bags" -d '{"instanceId":"default","kto2":"inventory"}
```

- Retrieving bags: In order to retrieve bags GET operation can be used:

  GET http://localhost:7050/Bags

-
- Creation of Entity: Use simple HTTP post in the body of valid JSON document:

```
POST http://localhost:7050/Bags(wikidata)/Entities
```

- we can use the same line in JSON document like this:

```
curl -H "Content-Type: application/json" -X POST
http://localhost:7050/Bags\(wikidata\)/Entities -d
'{
"comment": "My Comment when adding inventory data",
"contributorid": "174285",
"contributorip": "null",
"id": "34",
"minor": "true",
"revisionid": "29705046",
"revisiontimestamp": "2012-02-28T03:06:56Z",
"title": "Test2"
}'
```

Querying Entities:

Criteria	Type	Example	Description
myCategory=Soaps	Equals	/Bags(inventory)/Entities(Cname=liril)	This will retrieve all entities having a field called ‘cname’ containing the value ‘liril’
Criteria , Criteria	And	/Bags(inventory)/Entities(name=liril , rate=20)	This allows an AND operation on two criteria, name and rate should have

			specified values
Criteria | Criteria	Or	/Bags(inventory)/Entities(name=kris **|** qty=2)	This is an OR operation on two criteria, name with kris or qty with 2.

Available HTTP method in REST API:

HTTP method	URL Pattern	Data Format	Success Code	Error Code
GET	`/resources/groups`	application/json	**200** Returns the list of user groups defined on theWorkload Deployer appliance. See the example below for a sample of the data returned.	**403** This code is returned if the requester does not have sufficient permission to view the list of user groups. **500** This code is returned if the Workload Deployer appliance encountered an internal error while processing the request.
POST	`/resources/groups`	application/json	**201** The user group has been defined and is included in the response body. The URL of the new user group is included in the Location header of the response.	**400** This code is returned if there are problems parsing the JSON data in the request. **403** This code is returned if the requester does not have sufficient permission to define a new user. **500**

				This code is returned if the Workload Deployer appliance encountered an internal error while processing the request.
GET	`/resources/groups/{id}`	application/json	**200** Returns information about a user group defined to theWorkload Deployer appliance.	**403** This code is returned if the requester does not have permission to view the user group. **404** This code is returned if the requested user is not defined. **500** This code is returned if theWorkload Deployer appliance encountered an internal error while processing the request.
PUT	`/resources/groups/{id}`	application/json	**200** The user group was successfully updated. The response body contains a JSON representation of the current state of the user group.	**400** This code is returned if there are problems parsing the JSON data in the request. **403** This code is returned if the requester does not have permission to update the specified user group. **404** This code is

				returned if the request references a resource that is not defined. **500** This code is returned if the Workload Deployer appliance encountered an internal error while processing the request.
DELETE	`/resources/groups/{id}`		**204** The user group has been deleted.	**403** This code is returned if the requester does not have permission to delete the user group. **404** This code is returned if the requested user group is not defined. **500** This code is returned if theWorkload Deployer appliance encountered an internal error while processing the request.

User Group has following attributes:

Created: It specifies creation date of the user group. It is in the format, number of milliseconds from mid night, May 3, 2013. It is generated by the system.

ID: Auto generated value created by the system for the user group.

Name: Specifies group name and accepts string value of 64 character.

URL: Specifies the URI of the user that owns this user group. The URI is relative and should be resolved against the URI of the user group.

Updated : Specifies the time the user group was last updated, represented as the number of milliseconds since midnight, January 1, 1970 UTC. This value is numeric and is automatically generated by the appliance.

Users : Specifies a list of the URIs for the users that belong to this user group. The URIs are relative should be resolved against the URI of the user group that contains them

Inventory Management Application using JasDB(Final Project):

we have created online shopping inventory website using JasDB as database. Using our application we can add categories and we can add items in the following categories which we have added. We have provided functionality to add / delete/ modify the inventory items and categories. We can also add admin for the application and database so that they can have full right on the database and web application. Please find the screen shots and the corresponding code.

Login screen:

Code:

```
if(action.equals("login")) {

        String userid = request.getParameter("userid");
```

```
String password = request.getParameter("password");
if(userid.equals("") || password.equals("")) {
throw new JasDbException("User or Password is blank");
}

if (!jasdb.findUser(userid, password))
        throw new JasDbException("User not found");
                                        mySession.setAttribute("userid",userid);
Cookie userCookie = new Cookie("userid",userid);
userCookie.setMaxAge(3600);
response.addCookie(userCookie);
Cookie passCookie = new Cookie("password",password);
passCookie.setMaxAge(3600);
response.addCookie(passCookie);
refreshCategories();
if(jasdb.isEmpty("inventory")) {

        if (jasdb.isEmpty("category")) {
        throw new JasDbException("Category is empty. <a
href=\"newCategory.jsp\">Add new category</a>");
        } else {
        throw new JasDbException("Inventory is empty. <a
href=\"newInventory.jsp\">Add new inventory</a>");

        }
} else {
```

```
            refreshUsers();
            refreshInventories();
            RequestDispatcher rd = request.getRequestDispatcher("inventory.jsp");
            rd.forward(request, response);
    }
    }
```

Inventory screen: inventory.jsp

Add new inventory screen:

Code: Add category

```
if(action.equals("addCategory")) {
        String category = request.getParameter("category");
        String description = request.getParameter("description");
        jasdb.addCategory(category, description);
        refreshCategories();
        RequestDispatcher rd = request.getRequestDispatcher("inventory.jsp");
        rd.forward(request, response);
}
```

Update inventory screen:

Code: Update Category

```
if(action.equals("updateCategory")) {
        String id = request.getParameter("id");
        String description = request.getParameter("description");
        String category = request.getParameter("category");
        //String categoryID = jasdb.getCategoryID(category);
        jasdb.updateCategory(id,category, description);
        refreshCategories();
        RequestDispatcher rd = request.getRequestDispatcher("inventory.jsp");
        rd.forward(request, response);
}
```

Delete inventory code:

```
if(action.equals("removeInventory")) {
        String id = request.getParameter("id");
        jasdb.removeInventory(id);
        refreshInventories();
        RequestDispatcher rd = request.getRequestDispatcher("inventory.jsp");
        rd.forward(request, response);
}
```

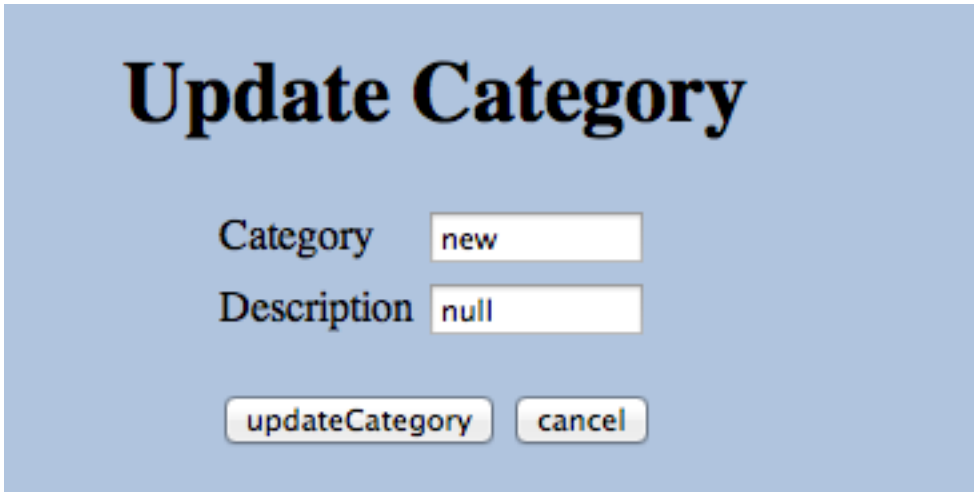
Add new category screen:

Code:

```
if(action.equals("addCategory")) {
        String category = request.getParameter("category");
        String description = request.getParameter("description");
```

```
        jasdb.addCategory(category, description);
        refreshCategories();
        RequestDispatcher rd = request.getRequestDispatcher("inventory.jsp");
        rd.forward(request, response);
}
```

Update category screen:



Code:

```
if(action.equals("updateCategory")) {
        String id = request.getParameter("id");
        String description = request.getParameter("description");
        String category = request.getParameter("category");
        //String categoryID = jasdb.getCategoryID(category);
        jasdb.updateCategory(id,category, description);
        refreshCategories();
        RequestDispatcher rd = request.getRequestDispatcher("inventory.jsp");
```

```
        rd.forward(request, response);
}
```

Delete category:

```
if(action.equals("removeCategory")) {
        String id = request.getParameter("id");
        if(jasdb.inventoryByCategory(id)) {
                jasdb.removeCategory(id);
                refreshCategories();
                RequestDispatcher rd = request.getRequestDispatcher("inventory.jsp");
                rd.forward(request, response);
        } else {
                throw new JasDbException("Unable to delete category. Remove all inventory
                belongs to this category");
        }
}
```

Add new user screen:

Code:

```
if (action.equals("addUser")) {

        String userid = request.getParameter("userid");

        String password = request.getParameter("password");

        String name = request.getParameter("name");

        String role = request.getParameter("role")
jasdb.addUser(userid,password,name,role);

        refreshUsers();

        RequestDispatcher rd = request.getRequestDispatcher("user.jsp");

        rd.forward(request, response);

}
```

Update user:

Code: Update user

```
if (action.equals("updateUser")) {
        String id = request.getParameter("id");
        String userid = request.getParameter("userid");
        String password = request.getParameter("password");
        String name = request.getParameter("name");
        String role = request.getParameter("role");
        jasdb.updateUser(id,userid,name,password,role);
        refreshUsers();
        RequestDispatcher rd = request.getRequestDispatcher("user.jsp");
        rd.forward(request, response);
}
```

Delete user:

```
if (action.equals("updateUser")) {
        String id = request.getParameter("id");
        String userid = request.getParameter("userid");
        String password = request.getParameter("password");
        String name = request.getParameter("name");
        String role = request.getParameter("role");
        jasdb.updateUser(id,userid,name,password,role);
        refreshUsers();
        RequestDispatcher rd = request.getRequestDispatcher("user.jsp");
        rd.forward(request, response)
}
```

Conclusion: In this paper I have described background for NoSQL databases and different data models available for NoSQL databases. We can choose available database according to business requirement. Here in this report we have shown pros and cons of different data models and how to use JasDB document database. JasDB provided a new approach for increased flexibility of document based software application where saved data has a complex structure. In this database adding the data is very easy and there is no need to change structure of the database.

Experiments has been conducted so as to create an extra storage layer so that JasDB can provide very good performance for heterogeneous data. This JasDB is proposed to handle huge amounts of data and number nodes can be added anytime so as to make the system highly available.

References:

1. Official JasDB website

   http://www.oberasoftware.com

2. Software Flexibility Improvement for Document Oriented Applications by "Politehnica" University/" Romania.

3. Data Storage for smart Environment using non-SQL databases by
Haller Piroska and Farkas , University of Petru, Romania

4. Survey on NoSQL Database by
Jing Han, Hai Hong, University of Posts and Telecommunication, Beijing, China.

5. A comparison between several NoSQL databases with comments by
Tudircia, B.G. , Dept of Econ. Math, Ploiesti , Romania.

6. NoSQL data modeling techniques.
http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/