International Journal of Research in Advent Technology, Vol.6, No.5, May 2018
E-ISSN: 2321-9637
Available online at www.ijrat.org

CouchDB: Reliable Repository for Big Data generated by IOT

S. Raghavendra Kumar[1], Dr.B. Lalitha[2]
Student[1], Assistant Professor [2]
CSE Dept[1,2], JNTUACEA[1,2], Anantapur, A.P, India[1,2]
Email: rksakali.it29@gmail.com[1], lalitha_balla@yahoo.co.in[2]

Abstract: Internet of Things (IoT) has a capability to develop dynamic systems and real-time applications by the growth of RFID and wireless, mobile, and sensor devices. A spacious of industrial IoT applications have been builded and utilized in present days. Storage is an important research direction of the Internet of Things. Enormous and heterogeneous data of the IoT brings the storage as huge challenges and more complicate in terms of terabyte to petabyte (volume), speed in growth (velocity),hybrid data and un-structured data and structured data (variety) in nature. This is known as 'Big Data'. When data and number of requests increases, structure database cannot handle huge data and requests efficiently. One of the best consequence to overcome these obstacles is to transfer datacenters on NoSQL document oriented databases. In document oriented databases, the CouchDB , provides a REST API, and offers a high set of features targeted to IoT applications. Furthermore, we develop optimized schemes for uploading documents which are specifically customized to resource-constrained IoT devices. We estimate our proposed schemes both analytically and with experiments.

Keywords —Big Data; CouchDB; IoT; JSON; MongoDB ; NOSQL.

1. INTRODUCTION

IoT is the system which connects things with help of the Internet [1] through varieties of information perception devices, in order that all the standard physical objects which can be independently addressed are capable to exchange information with each other, and eventually achieve the goal of intelligent identification, locating, routing, supervising . Data is one of the important characteristic of the IoT. In Internet of Things, data is from different types of sensors and characterize billions of objects. IoT data have attributes such as Multi-source and Heterogeneity, Temporal-spatial, Interoperability and Multi-dimensional.

IoT applications need to access a database, from which they can conveniently obtain the data of interest. To this end, application support leverages a standard application programming interface (API) for web services, such as the REpresentational State Transfer (REST). The storage accredits complex data analysis – e.g., for knowledge mining and semantic analysis – that ease from data being persevered at some location. However, the storage of IoT data also raises major challenges. First, flexibility is needed in supporting diverse data. This requires the definition of a data model which can efficiently describe not only scalar values, but also heterogeneous and multimedia content [2]. Second, the storage infrastructure has to be scalable, as it needs to support a huge number of both IoT devices and end-users. Hence, the corresponding framework can benefit from being distributed, inorder to support load-balancing and clustering for multi-tier query processing.

In this paper, we proposed a NoSQL based document-oriented repository model to overcome the problems in IoT. Here, we evolve a document-oriented approach and illustrate how it supports heterogeneous data. In document-oriented approach, the CouchDB [3] supports a RESTful API and IoT applications which consists of heterogenous data to store in its repository. They include replication for load balancing, distributed query processing, and notifications. Moreover, we plan optimized schemes for uploading documents which are specifically customized to resource-constrained IoT devices. We evaluate such systems with experiment analysis. The achieved results are shown efficiently in proposed work. The paper is organized as follows. Section II-the related work. Section III- the CouchDB implementation . Section IV -proposed work based on both features and performance. Section V-conclusion along with furthermore remarks.

2. RELATED WORK

2.1. IoT and Big Data: Large number of sensors and IoT devices connects with each other from all over the world and generates huge amount of heterogenous data. This is known as Big Data [4]. Only Big Data technologies and frameworks can handle such enormous data volumes that are streaming varied types of information. The more the IoT grows quantitatively, the more Big Data techniques will be required. Within this space, organisations need to shift focus to the rich data, which is easily accessible in real-time. Data from sensors should be processed to find patterns and insights in real-time to advance business goals. Existing Big Data technologies can effectively harness the incoming sensor data, store it and later analyse it efficiently using artificial intelligence.

2.2. IOT and NoSQL: IoT devices have an important role in generating heterogenous data. Just by using these devices and connected sensors, it is possible to create complex systems for data acquisition, with relatively low budget. But, very small amount of these works illustrate the to exchange and make use of information between NoSQL with IoT and the impact of providing platforms that combine these two technologies.Data gathered from sensors and IoT devices should be stored properly and analyzed

efficiently. Hence, NoSQL and Big Data are the better ideas to this kind of challanges. With utilization of these solutions based on the implementation of Big Data and analysis of data flows generated by sensor networks, modern business organizations can find new information that had not been obvious earlier. Those information can be used for improving the business. Their are four types of NoSQL DBs [5], which are different from each other in data storage. They are:

- ***Key-value DB:*** Data is stored with help of key-value pairs. Value is retrieved with of the keys. Examples are Redis, Dynomite, and Voldemort.
- ***Column-oriented DB***: DB stores the data same as the RDBMS tables.But only difference is in storing data items. The data items are stored in columns instead of the rows. Examples are Hbase, Cassandra, Hypertable.
- ***Document-oriented DB***: Data are stored and organized as a collection of documents. Documents are flexible; each document can have a number of fields. Examples are Apache CouchDB and MongoDB.
- ***Graph based DB***: DB stores and retrieves the data based on graph theory .This mainly focus on the inner connectivity among dissimilar parts of data.Data segments are shown with help of nodes and their relationships and are defined by the edge of the connection nodes. Example is Neo4j.

2.3. CouchDB: Storage services in IOT application implemented using the NSQL database system called CouchDB. Document-store databases like CouchDB allow for freedom and flexibility in the structure of documents where the entries can be of any size and structure which are in the form of JSON documents. Queries to this database are written as views using JavaScript and employ the map-reduce algorithm which works in two steps: the map function takes in the data or, in this case, the documents, and filters the input according to a certain condition. The reduce function, then takes the filtered output of the map function and groups the data in accordance to a prescribed criteria. The map-reduce functions improve the scalability and speed of retrieving documents from the database.

Finally, CouchDB also supports master-to-master replication that allows all peer nodes to perform update and insert operations [3]. Here in the system, if a user inserts a new document in the remote database residing on their device, the new entry is replicated to all other databases in the system and becomes accessible by all other users upon syncing.

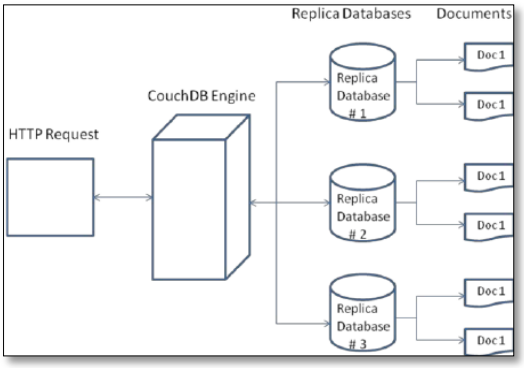

Figure 1: CouchDB Architecture

3. COUCHDB IMPLEMENTATION

3.1. Data model **:** CouchDB stores JSON documents in the form of binary data. The database files of CouchDB is saved as .couch extension. CouchDB stores documents directly inside of its databases. Each document has a unique ID which can be assigned manually when inserting documents, or automatically by CouchDB [3]. There is no maximum number of key-value pairs for documents and no upper limit size; the default max size is 4GB, but this can be changed by editing CouchDB's configuration file.

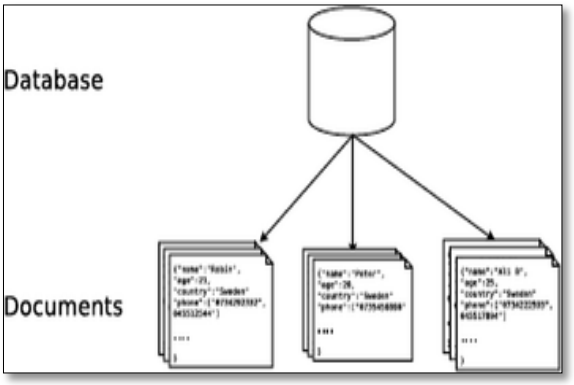

Figure 2: CouchDB's data model showing database and documents

3.2 . RESTful API**:** REST (Representational State Transfer) is an architecture which describes how services can be provided for machine-to-machine communications. It supports developers to use HTTP methods to perform operations such as Create, Read, Update, Delete(CRUD). HTTP methods are mapped to CRUD follows as:

- POST - Create a resource
- GET - Read a resource
- POST - Update a resource
- DELETE - Delete a resource

In a RESTful architecture, resources like databases, documents, attachments etc., get unique identifiers in the form of URIs. Imagine you want to create a database called agriculture on a local CouchDB setup .By the usage of

CouchDB's standard port (5984) and the command-line utility curl doing so would look like this:
curl -X PUT http://localhost:5984/agriculture
With the help of REST API, developer sends information in the form of XML or JSON docments [7]. The above request would be answered by CouchDB with a simple JSON document, to inform the user of success:
{"ok"}:true
Similar requests using curl can be made to execute all of the CRUD operations. CouchDB is a graphical Web interface, and these contains libraries to programming languages, the user always uses the raw HTTP requests.As web browsers use HTTP, they can be able to read JSON documents from CouchDB.

3.3. Scaling and replication: Replicating databases in CouchDB is easy. All it takes to trigger replication is one simple HTTP request that specifies the source database and the target database: POST/replicate HTTP/1.1
{"source":"database","target":http://somewhere.com/db}
It is also possible to replicate from a remote server to the local server by switching the values of the source and target keys. CouchDB [3] supports two-way replication. To make replication even easier, it can be performed from the graphical Web interface Futon. Scaling out databases by splitting them into an array of servers in a cluster is not as much of a trivial matter as replication.

3.4. Querying: Relational database managament systems typically use static data and dynamic queries; schemas are fixed, and SQL queries are dynamic.In CouchDB data is querying by the help of views. There are two kinds of views: permanent views, which are static, and temporary views, which can be provided ad-hoc. Views gives the results of MapReduce functions [6] . Map functions are written by the user, and iterate over all documents in the database to check if the documents match the criteria specified in the function by the user. If everything matches, and a result is hence found, the document (or selected parts of it) are emitted using the emit() function. A simple example follows:

```
function(doc) {
if(doc.age && doc.age > 15 && doc.name)
emit(doc.name,doc.age);
}
```

In the above example, all documents that have the key age, with a corresponding value that is over 15, are emitted. As CouchDB unable to support static schemas, its important to check that certain key-value pair could exists before trying to use it. After a list of emitted documents has been generated by the map function, a reduce function may be used to further operate on the data.

3.5. Indexing: CouchDB views use MapReduce [6] to index user's data. The MapReduce functions generate the results of user's query, which can be obtained via an HTTP request.When data performs added or removed functions in the database, those indexes are updated automatically. In addition, CouchDB [2] stores view results throughout the B-tree data structure it uses for the index. If CouchDB sees that user's query will include all of the children of a given node, it will simply pull the "summary" result from the parent node, preventing it from having to visit each of the child nodes for their individual results. This, and the fact that view results are computed when then view is built, makes querying views very fast.

3.6. Attachments: As mentioned earlier, attachments correspond to arbitrary data associated with a certain content type.As a consequence, attachments are particularly suitable to represent heterogeneous and multimedia data which cannot be otherwise represented as numerical or textual values. Examples of data suitable to be represented as attachments include images, audio, video, and so on. Attachments are main part of a CouchDB document.They are identified by a name (e.g., a *filename*), and are described through two fields: the content type in MIME format [9] and the data itself in Base64 encoding [10] . For instance, the following represents an attachment within a document:

```
"_attachments":{
"hello.txt":{
"content_type": "text/plain",
"data": "SGVsbG8gd29ybGQh"
}
}
```

4. EVALUATION

In this section, we will evaluate our proposed storage infrastructure. Specifically, we will provide first a performance characterization of the document upload process and performance based comparison between couchdb and mongodb.

4.1. Document upload performance: As discussed in Section III-6, attachments can be uploaded together with a document by embedding them in the special attachments field as Base64-encoded strings. As a consequence, this approach has an overhead associated with the encoding process itself, in terms of both processing time and resource utilization. Furthermore, data encoded in Base64 format result in an increased size with respect to the original ones, thus increasing the bandwidth demand for transferring documents. Clearly these aspects are critical in embedded IoT devices.To this end, we consider an alternative process for transferring documents with attachments. Specifically, the document is uploaded first and then the attachments are added to that document through separate HTTP requests. This approach has the advantage of avoiding the Base64 encoding process [10] , since the attachments can be provided as raw data in the corresponding HTTP request.

Transaction analysis: In this section, we will characterize the overhead associated with transferring a document in terms of the size of the corresponding HTTP requests. For simplicity, in the following we limit our analysis to the scenario wherein documents contain a single attachment.In the case where the document also embeds the Base64-

encoded attachments [10] , only a single HTTP request is needed, and the related size is given by:

x_1=h + m + a_{od} + $\hat{n}$= h + m + j_s + f_n+ $\hat{n}$ ———— (1)

From equation (1), where *h* is the length of the HTTP headers; *m* is the size of the (textual) document; a_{od} is the overhead of embedding the attachment in the document; and $\hat{n}$ is the size of the Base64-encoded attachment. The overhead *aod* can be further expressed in terms of the JSON [7] overhead j_s in the document and the filename length f_n.In contrast, the case where the attachments are uploaded separately from the document requires two distinct HTTP requests: one for the document and another one for the attachment in raw format. Specifically, the size of the two HTTP requests combined is given by:

$x_2 = h + m + h + \hat{n} + h_o = 2h + m + n + f_n + a_{or}$ ———— (2)

From equation (2), where h_o is the HTTP header overhead when uploading the attachment in an individual request, consisting in the filename length f_n and in the overhead a_{or} for specifying a revision in the request. The strategy of two separate requests incurs in less overhead than a single request when $x_2 < x_1$. By knowing that the size of a Base64-encoded block of size *n* is $\hat{n} = 4\left\lceil\frac{n}{3}\right\rceil$ and simple calculations, obtains the breakeven point for:

$n \approx 3(h + a_{or} - js) - 4$ ———— (3)

For practical values of the considered parameters, the approach of two separate requests is more convenient than the other one when *n* (from equation (3)) is above a few hundred bytes. The analysis above only focuses on the size of the requests and does not consider the overhead due to additional factors, such as establishing TCP connections for the HTTP requests.

4.2. Performance based comparison between couchdb and mongodb: As IOT data have huge and it can be represented better in document databases, it is needed to further probe and find a better document database among the document databases. MongoDB [8] is popular, but it is needed to analyze the performances of both the databases to fix up with one document database.The various parameters analysed are

- No of Concurrent users vs Latency time
- Data Size vs Latency
- No of Cores vs Latency

1. Experimental Setup:

The Experimental setup was done on windows 7, MongoDB 3.4 version and CouchDB 2.0 was used.

2. Experiments and the Results:

The performance deviations in Latency Time and throughput was checked for the increasing the number of clients, it was found MongoDB performed better when the number of clients was increased. It is inferred from Figure 3 that MongoDB has a lesser Latency time than CouchDB as the number of clients were increased.

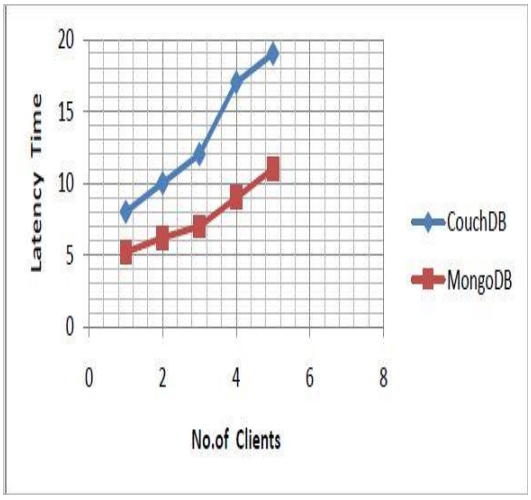

Figure 3: Latency Time Vs No. of clients

The performance of MongoDB and Couch DB was analysed for different Data sizes. It was found that MongoDB performance for Larger files was better compared with CouchDB. It is inferred from Figure 4 that MongoDB has a lesser Latency time than CouchDB as the data size increased.

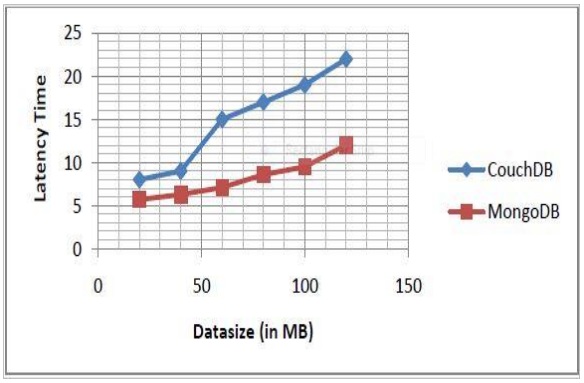

Figure 4: Latency Time Vs Datasize

The analysis was done for varying number of Cores/CPU's for throughput(Number of Images retrieved per minute)a given Query, where Mongo DB's throughput was better compared with CouchDB. It is inferred from Figure 5 the throughput of MongoDB was much better than couchDB.

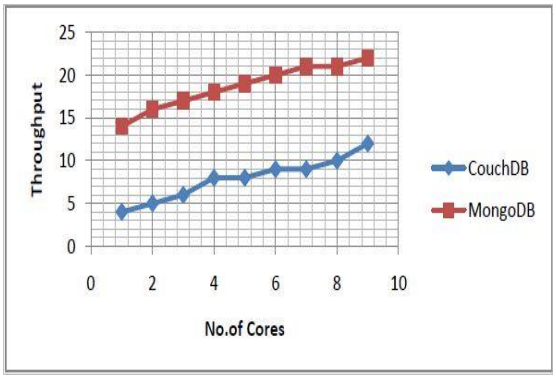

Figure 5: Throughput Vs No. of Cores

5. CONCLUSION AND FUTURE WORK

The qualitative feature of CouchDB is analyzed in this work. CouchDB is one of the NoSQL document oriented database which enables to store IoT data.This supports trivial and non-trivial queries. The CouchDB views with the JavaScript query server are very slow to run, when it contains number of non-trivial documents to process. Here,CouchDB compared with MongoDB(other NoSQL document oriented database) to analyze the performance.

In Section-IV-C proves that MongoDB gives better performance than the CouchDB. One of the aspect,CouchDB supports master-master replication,if a document updated in the remote database then automatically updates in its replications which are present in other databases. Furthermore ,compare the CouchDB with other NoSQL Document oriented databases rather than MongoDB such as Cloudant, OrientDB and ElasticSearch etc., which gives best query performance in storing and retrieving IoT data.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
[2] Apache Software Foundation, "The Apache CouchDB Project," http://couchdb.apache.org, retrieve January 30, 2012.
[3] I. Akyildiz, T. Melodia, and K. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, no. 4, pp. 921–960, 2007.
[4] Yong-Shin Kang, Il-Ha Park, Jongtae Rhee and Yong-Han Lee, Member, IEEE," MongoDB-based Repository Design for IoT-generated RFID/Sensor Big Data", IEEE Sensors Journal-12398-2015.
[5] Vaish, G.: Getting Started with NoSQL, Packt Publishing Ltd., Birmingham, UK, 2013.
[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[7] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 4627, July 1995.
[8] Leavitt, N. (2010). Will NoSQL databases live up to their promise?. *Computer*,*43*(2), 12-14.
[9] N. Freed and N. Borenstein, "Multipurpose InternetMail Extensions (MIME) Part One: Format of Internet Message Bodies," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 2045, November 1996.
[10] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," Internet Requests for Comments,Internet Engineering Task Force (IETF), RFC 4648, October 2006.