

EmeWeb

by Mustafa Nassir

FILE	6367_MUSTAFA_NASSIR_EMEWEB_93287_1690532142.DOCX		
TIME SUBMITTED	27-MAR-2018 02:09PM (UTC+0100)	WORD COUNT	2740
SUBMISSION ID	85653563	CHARACTER COUNT	18050

EMERGING WEB TECH

STUDENT: MUSTAFA NASSIR (1708461)

COURSE: BSC WEB DEVELOPMENT

MODLE: EMERGING WEB TECHNOLOGY (M6X01096)

LECTRER: **MICHAEL DACEY**

Executive summary

There has been the creation of massive amounts of data in the past decade. This data creation will continue with the emergence of the notion of the Internet of Things (IoT). The incorporation of sensors, chips, and other computing devices will see to it that more data is continuously collected and stored. The other source of huge amounts of data in the recent past has been the growth and the use of social media, mobile applications, and the alternative web technologies. To support these incessant creations of the massive amount of data various emerging web technologies and database models are used. The generation and storage of these huge amounts of data have led to the coining of the name “Big Data.” These data is unstructured and dynamic hence it will be costly and unpractical to administer via SQL databases as the data lacks structures, requires high scalability and elasticity. The emerging technologies used in the handling of these data implements NoSQL data storage that targets unstructured data. The NoSQL data utilizes databases system such as the MongoDB and Cassandra in the provision of a reliable platform that is fast and efficient for data queries. This data has increased the significance of the cloud applications, social media, e-commerce material science and bioinformatics. In efforts to boost the querying capabilities of a conventional database, this paper will evaluate the Cassandra and the Hadoop engine. This research will evaluate a wide range of representative cases, compare and then contract the usability of Cassandra and Hadoop in big data. The paper will also effectively tackle the five V’s of Big Data as a challenge to many researchers.

Table of Contents

Executive summary	2
Introduction.....	5
Big Data.....	6
Volume	6
Velocity	6
Variety	7
Veracity	7
Value.....	7
Hadoop Distributed File System	7
Name Nodes	8
Data Node.....	8
Cassandra File System	9
Comparison of the Hadoop File system and Cassandra File system	10
Conclusion	12
References.....	13
Appendix.....	15

List of Figures

Figure 1: The HDFS Architecture.....	8
Figure 2: The Cassandra File system	10

Introduction

The recent evolution of the computing world has made a robust database that can handle and accommodate huge amounts of data from users on demand an essential ingredient in emerging web technologies. The distributed storage databases have become a defacto method of data storage in the new web technologies [1].

The mention of big data has always been associated with the effective handling of colossal amounts of data that are currently being produced every second. The pace at which the data is being generated has made it essential to adopt new ways to manage data, especially in analysis and querying. These adaptations of the right data handling method are significant for the various data-driven decisions. Most organizations have put a lot of efforts to get vital insights from the massive data they collect [2]. This data is vital for the daily operation of these organization providing the organization's competitive advantages in their business activities [2]. These activities require effective data handling systems and architectures for the organizations to remain in profitability

Effective and efficient data curation

Effective and efficient storage of data

Effective data analysis [2]

Effective presentation and evaluation of data

Identification of sagacious pattern in data

This research paper will define the various technologies used in big data, evaluate these technologies in terms of features and properties in relation to big data [3]. Additionally, the paper will evaluate the Hadoop Distributed File System (HDFS) and the Cassandra File System. A tabular comparison of the various features of the two file system will be presented.

Big Data

Big data is that is beyond the conventional storage capacity and processing capabilities of the classical computing structure [2]. Thus big data is data that cannot be effectively handled, analyzed and processed by the traditional systems, tools and database tools. These data are typically unstructured, semi-structured and in raw form that poses a limitation in the conventional systems. The classification of the usable part of the data becomes the substantial part of the challenge. The technique selected in handling the multidimensional data will be a defining factor for mining Big Data.

Data can be classified as structured, unstructured and semi-structured. Globally data is generated via satellite for weather forecasting, large organizations, seismic activities, data from sensors, GPS, data from social media, e.g., Twitter, Facebook, and Instagram, mobile applications such as WhatsApps, GTalk, etc. The data expands in five directions depending on a variety of characteristics. These characteristics give the 5 V's of Big Data.

Volume

This describes the data in terms of amount, giving the size and shape of the data. This characteristic cannot allow data to be stored in the conventional storage units [1]. The volume of the data leads to the adaptation of new tools and techniques for the storage of data.

Velocity

This describes the frequency of data creation and the pace at which the data needs to be handled. It also describes the frequency of variation of the data. The effective and timely processing of the data is significant in the successful handling of the Big Data [2].

Variety

This phenomenon describes the various type and formats of data that is being generated every second. The variability of the data defines the system and the architecture that will be used in the handling of data [4].

Veracity

This characteristic defines the biases, flaws, and anomalies in the data [2]. The veracity of any data illustrate the accuracy of the data and primarily focuses on the quality of the data being created. The isolation of meaningful data from Big Data has continuously become the biggest challenge in Big Data [2].

Value

This defines the meaningful value that can be extracted from the Big Data [2]. The obtain the best value from any data there is the need for effective handling the big data through the key data extraction operations of addition, updating, deletion sorting, mining, searching and storage [4].

Hadoop Distributed File System

This is an open source software framework that exploits data via the divide and conquers concept in the storage of data while applying the client-server architecture. This file system has been applied to the Google file system [5]. The file system entails the division of the Big Data into small chunks of a typical size of 64 MB. These chunks are called blocks, and they are desired as the calculations are done close to where the data is stored. The Hadoop File system allows parallel processing and handling of Big Data set applying easy programming models [2]. The Hadoop Distributed File System is used for the storage of the data as illustrated in figure 1 below. The primary model used in this architecture is the write-once-read-many model. This

model restricts data writing to one user at any particular time [1]. However, the system allows multiple users to read data simultaneously. The HDFS is made up of two primary components, i.e., name nodes and data nodes.

Name Nodes

This component is made up of the metadata information. It is responsible for file opening, directory opening, file closing, directory closing, and renaming. The component is also responsible for the tracking of the block mapping to the Data nodes. It also determines the creation of the replica blocks [2].

Data Node

This component primarily functions to serve the read and write requests from the clients. It also executes requests from the Name nodes to create blocks, replicate blocks and delete blocks [5]. The blocks in the HDFS system are saved in a distinct file in the local file system in the data node. Thus by using heuristics, the data node can be able to determine the ideal number of the files per directory and then create the subdirectories accordingly [2].

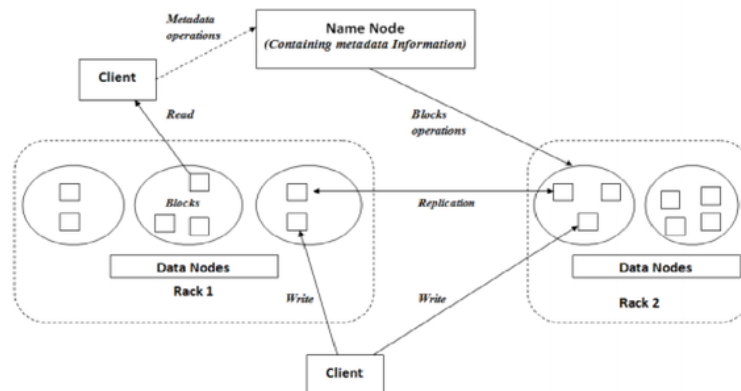


Figure 1: The HDFS Architecture

The HDFS system uses a scale-out architecture and commodity servers that have their local storage unit. The servers are configured as a cluster while the data in Hadoop is configured and partitioned into blocks that are spread throughout the clusters [6].

Cassandra File System

This is an Apache database which is flexible and easily expandable NoSQL Database [2]. This database is easily used for a real-time NoSQL database system that should not have any down time. Contrary to the HDFS system it does not have the master-slave architecture. Its architecture is that of a peer to peer distributed ring that is easy to set up and maintain [7]. The peer to peer does not provide any single point of failure in Cassandra. The Cassandra File System (CFS) utilizes the Cassandra system to save and execute the real-time data analysis [2]. The CFS has an in-built data replication system that ensures even replication of data across in real-time to the search nodes and the analytics [8]. The metadata in this case, especially with regards to the analytics data, is stored in the Cassandra keyspace. The keyspace is made up of two column families that contain the actual data. The families are inode and sblock column family [2]. These family columns are similar to the name node in the HDFS. The families store the metadata information, permissions, location of files, block locations, permissions, list of block ids and type of files. The sblock in the CFS resemble the data node in Hadoop especially in the storage of actual data [3]. The rows in the sblock serve as blocks of data linked with the rows in the inode column family [2]. The Cassandra file system uses the Gossip protocol for communication between the nodes thus each node in CFS plays a similar role. Each node in the clusters can acknowledge the read-write requests with disregard of the location of the data storage. In case of the failure of a single node, the requests to that node can be served by any node in the cluster [4].

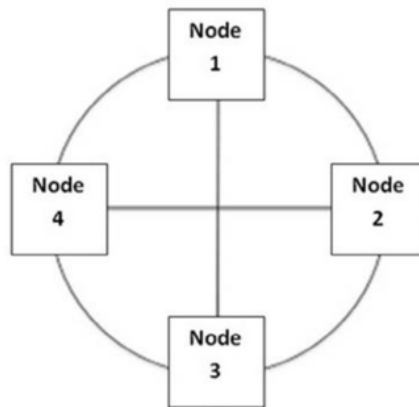


Figure 2: The Cassandra File system

Comparison of the Hadoop File System and Cassandra File System

File system Feature	HDFS	CFS
Support for Big Files	Supports	Supports
Balancing	The Name node effects the load balancing in this system.	Uses automated tools like the OpCenter that ensure that the nodes are balanced in the clusters
Scalability	Highly scalable but dependent on the increase in the hardware.	Highly scalable, where the throughput can be increased with the increase in the nodes in the cluster.
Architecture	Adheres by the master-slave architecture where the name	They use the peer to peer ring type architecture. The nodes

	node are the masters while the data nodes are the slaves.	in a ring are similar and have the same ability to serve the ability to serve read-write requests.
Replication	The Hadoop rack aware replica placement policy is used for replication	The following policies are used for replication i.e. Rack aware, datacenter aware, rack unaware etc.
Failure detection and the failure tolerance	The HDFS client software uses checksum to check the content of the HDFS files system so as to identify the corrupted data and take a corrective measure.	Uses the modified version of Accrual Failure Detector
Data Storage Model	Data is stored in blocks that are 64 MB by default in size. The metadata is stored in the name node while the actual data is stored in the datanodes.	Data is stored in the inode and sblock columns families. The inode column family stores the metadata while the sblock stores the actual data.
Data Integrity and Error	Checksum is used for checking the content of the HDFS data.	Use AID i.e. Atomicity, Isolation and Durability.

Security, Authentication and Validations.	Has no default in-built security. It can utilize the Kerberos protocol for authentication and authorizations.	Utilizes the Kerberos and SSL for authentication and authorization. The system also utilizes the Commit log to ensure there is no data loss.
---	---	--

Conclusion

Big data was coined in response to the huge amounts of data that is being created every second. Most of the data generated are enterprise dependent where the data can be in the orders of 20 TB. This order is not definite and can go to the orders of 50 PB. This research paper has evaluated the various file systems that are used in the storage and the handling of Big Data. It is evident that the Cassandra is a better file system than the HDFS. The Cassandra system has better authentication and security features, data Integrity and Error handling, failure detection, and the failure tolerance and storage model. The use of peer to peer ring distributed system in CFS allows the system to protect itself from failures.

References

- [1] E. Dede, B. Sendir, P. Kuzlu, J. Hartog and M. Govindaraju, "An Evaluation of Cassandra for Hadoop," in 2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, 2013.
- [2] M. A. Ahad and R. Biswas, "Comparing and Analyzing the Characteristics of Hadoop, Cassandra and Quantcast File Systems for Handling Big Data," Indian Journal of Science and Technology, vol. 10, no. 8, pp. 1-6, 2017.
- [3] K. S. Rajith and R. R. Mary, "Comparative Performance Analysis of various NoSQL Databases: MongoDB, Cassandra, and HBase on Yahoo Cloud Server," Imperial Journal of Interdisciplinary Research, vol. 3, no. 4, pp. 265-269, 2017.
- [4] B. R. Prasad and S. Agarwal, "Comparative Study of Big Data Computing and Storage Tools: A Review," International Journal of Database Theory and Application, vol. 9, no. 1, pp. 46-65, 2016.
- [5] G. R. Chandangole and S. A. Joshi, "Processing Cassandra or MongoDB dataset with Hadoop-Streaming Based Approaches," International Journal of Engineering Development and Research, vol. 4, no. 3, pp. 27-31, 2016.
- [6] K. Dwivedi and S. K. Dubey, "A Taxonomy and Comparison of Hadoop Distributed File System with Cassandra File System," ARPN Journal of Engineering and Applied Sciences, vol. 10, no. 6, pp. 6870-6876, 2015.
- [7] A. A. Mutha and V. M. Deshmukh, "Cassandra File System Over Hadoop Distributed File System," International Journal on Recent and Innovation Trends in Computing and Communication, vol. 2, no. 3, p. 634 – 637, 2014.

- [8] M. D. Khade and P. A. Sable, "Apache Cassandra @ Business Field," International Journal of Engineering Trends and Technology, vol. 26, no. 5, pp. 263-269, 2015.

Appendix

The code fore for the application

Application

```
var express = require("express"),  
  
app = express(),  
  
mongoose = require("mongoose"),  
bodyParser = require("body-parser"),  
expressSanitizer = require("express-sanitizer"),  
methodOverride = require('method-override');  
  
mongoose.connect("mongodb://localhost/todo_app");  
app.use(express.static('public'));  
app.use(bodyParser.urlencoded({extended: true}));  
app.use(expressSanitizer());  
app.set("view engine", "ejs");  
app.use(methodOverride('_method'));  
var todoSchema = new mongoose.Schema({  
  text: String,  
});  
var Todo = mongoose.model("Todo", todoSchema);  
  
app.get("/", function(req, res){  
  res.redirect("/todos");  
});
```

```
app.get("/todos", function(req, res){

  Todo.find({}, function(err, todos){

    if(err){

      console.log(err);

    } else {

      res.render("index", {todos: todos});

    }

  })

});

app.get("/todos/new", function(req, res){

  res.render("new");

});

app.post("/todos", function(req, res){

  req.body.todo.text = req.sanitize(req.body.todo.text);

  var formData = req.body.todo;

  Todo.create(formData, function(err, newTodo){

    if(err){

      res.render("new");

    } else {

      res.redirect("/todos");

    }

  });

});

app.get("/todos/:id/edit", function(req, res){
```

```

    Todo.findById(req.params.id, function(err, todo){

        if(err){

            console.log(err);

            res.redirect("/")

        } else {

            res.render("edit", {todo: todo});

        }

    });

});

app.put("/todos/:id", function(req, res){

    Todo.findByIdAndUpdate(req.params.id, req.body.todo, function(err, todo){

        if(err){

            console.log(err);

        } else {

            res.redirect('/');

        }

    });

});

app.delete("/todos/:id", function(req, res){

    Todo.findById(req.params.id, function(err, todo){

        if(err){

            console.log(err);

        } else {

            todo.remove();

        }

    });

});

```



```

        res.redirect("/todos");
    }

    });

});

app.listen(3000, function() {

    console.log('Server running on port 3000');

});

// // Uncomment the three lines of code below and comment out or remove lines 84 - 86 if using cloud9

// app.listen(process.env.PORT, process.env.IP, function(){

//     console.log("The server has started!");

// });

```

The Packing JSON code

```

{
    "name": "ToDo",
    "version": "1.0.0",
    "description": "",
    "main": "app.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "Ian",
    "license": "ISC",
    "dependencies": {

```

```

"body-parser": "^1.14.1",
"ejs": "^2.3.4",
"express": "^4.13.3",
"express-sanitizer": "^1.0.1",
"method-override": "^2.3.5",
"mongoose": "^4.2.3"
}
}

```

Stylesheet Code

```

body {
padding-top: 80px;
}

```

Code for Editing

```
<% include partials/header %>
```

```
<div class="row">
```

```
  <div class="col-md-6 col-md-offset-3">
```

```
    <h1>Edit Item</h1>
```

```
    <form action="/todos/<%=todo._id%>?_method=PUT" method="POST">
```

```
      <div class="form-group">
```

```
        <label>Item Text</label>
```

```
        <input type="text" value="<%= todo.text %>" name="todo[text]"
```

```
        class="form-control">
```

```
      </div>
```

```

        <button class="btn btn-primary">Update Item</button>

    </form>

</div>

</div>

```

```

<% include partials/footer %>

```

Code for Indexing

```

<% include partials/header %>

```

```

<div class="row">

    <div class="col-md-6 col-md-offset-3">

        <h1>To Do Items</h1>

        <ul class="list-group">

            <% todos.forEach(function(todo){ %>

                <li class="list-group-item">

                    <span class="lead">

                        <%= todo.text %>

                    </span>

                    <div class="pull-right">

                        <a href="/todos/<%=todo._id%>/edit" class="btn btn-

sm btn-warning">Edit</a>

                        <form style="display: inline" method="POST"

action="/todos/<%=todo._id%>?_method=DELETE">

```

```

<button type="submit" class="btn btn-sm btn-
danger">Delete</button>

</form>

</div>

<div class="clearfix"></div>

</li>

<% }}; %>

</ul>

<a href="/todos/new" class="btn btn-primary">New Item</a>

</div>

</div>

```

```

<% include partials/footer %>

```

Code for Inserting New

```

<% include partials/header %>

```

```

<div class="row">

  <div class="col-md-6 col-md-offset-3">

    <h1>New Item</h1>

    <form action="/todos" method="POST">

      <div class="form-group">

        <label>Item Text</label>

        <input type="text" name="todo[text]" class="form-control">

      </div>

```

```
<button class="btn btn-primary">Create Item</button>
```

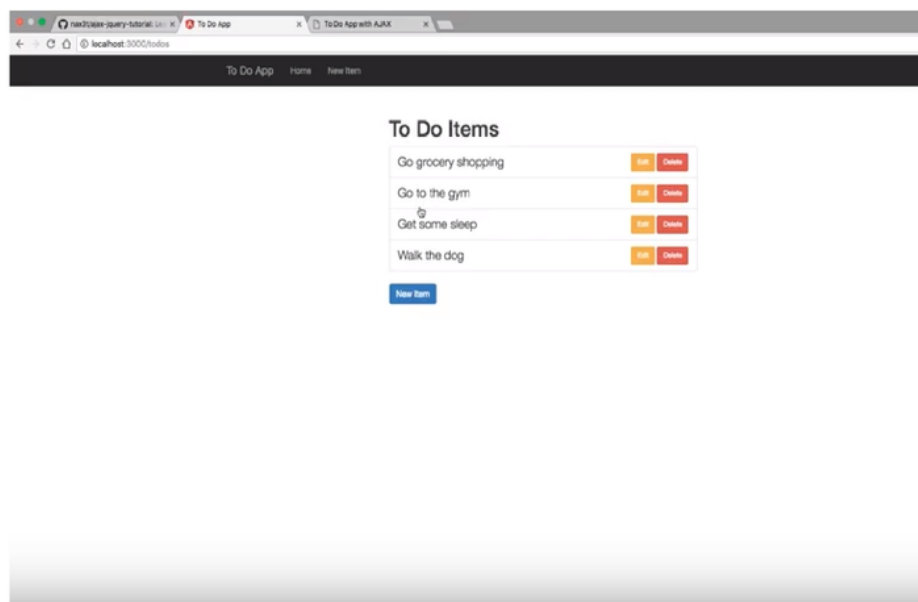
```
</form>
```

```
</div>
```

```
</div>
```

```
<% include partials/footer %>
```

The application



FINAL GRADE

42/100

GENERAL COMMENTS

Instructor

Big data application. (20) 13

Discussion of Hadoop technology (20) 18

Quality of website (Typescript) (20) 10

Single page application (20) 0

Dynamic content using AJAX (15) 0

Software testing (5) 1

The discussion of the four Vs was a bit cursory, you haven't really explained their importance.

Code only provided in the appendix.

The implementation is not written in typescript. The code provided executes in "node.js", although this is not made clear in the report. I will give you some marks for this, although you should have used Typescript.

No REST webservice.

No AJAX.

No evidence of testing, although there is a screenshot.

PAGE 4

PAGE 5

PAGE 6



Comment 1

Define these terms!!

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15



Comment 2

You really needed to provide some explanation for the JS libraries you are using.

PAGE 16

PAGE 17



Comment 3

There are no comments in this code

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22
