ClientNet Cluster an Alternative of Transferring Big Data Files by Use of Mobile Code

Waseem Akhtar Mufti(✉)

Alborg University, Aalborg, Denmark
wmufti@gmail.com

Abstract. Big Data has become a nontrivial problem in the field of business as well as in scientific applications. It becomes more complex with the growth of data and scaling of data entry points. These points refer to the remote and local sources where huge data is generated within tiny slots of time. This may also refer to the end user devices including computers, sensors and wireless gadgets. As far as scientific applications are concerned, for example, Geo Physics applications or real time weather forecast requires heavy data and complex mathematical computations. Such applications generate large chunks of data that needs to transfer it through conventional computer networks. Problem with Big Data applications emerges when heavy amount of data is transferred or downloaded (files or objects) from remote locations. The results drawn in real-time from large data files/sets become obsolete due to the fact data keeps on adding new data into the files and the downloading by remote machines remains slower as compared to file growth. This paper addresses this problem and provides possible solution through ClientNet Cluster of remote computers, Specialized Cluster of Computers, as one of the alternative to deal with real-time data analytics under the hard constraints of network. The idea is moving code, for analytic processing, to the remotely available big size files and returning the results to distributed remote locations. The Big Data file does not need to move around network for uploading or downloading whenever the processing is required from distributed locations.

Keywords: Big data · Mobile code · File transfers · Distributed clients

1 Introduction

1.1 Big Data Sources

Computer is efficient at processing highly complex algorithms. CPU gives a correct and fastest output to the problems if these problems are programmed. The best algorithms available for sorting and searching are still used by many day to day applications involving considerably large amounts of data and higher level of problem complexity. The limitations of computing devices are realized especially when a very large data are processed by a simple program. Problem becomes worse when there are multiple keys given for simultaneous searching of trillions of data items scattered around different data structures or memory locations. Sorting them and then searching infinitely many data items in the scenario when multiple organizations are connected for some common goals of business. The problem of handling Big Data [1–3] is even impossible if the output is required in real-time. The decision making has to be automatic because it is not possible for a human to analyze and assess higher volumes of data in real-time and making business decisions within limited time duration. Geo physical applications involve computations of a number of physical properties of earth layers, wind densities on different altitudes and the computations of tides under water to be the main reason for tsunamis, earth quacks and sea storms.

Initially Big data emerged due to Web logs and machine logs recording for user behavior analysis on internet and offline [8]. The size of web logs evolved with increase in number of computer and internet users, what went to be considered as big volume of data and for the analytics of user behaviors. The volume of big data further exploded as number of internet based applications introduced e.g. social networks, cellular gadgets, E-commerce, cloud computing and all those devices and human sources involved putting inputs to the systems and generate outputs on communication network.

This paper introduces the first version of distributed **ClientNet** cluster developed in Java. It provides the solution as an alternate of transferring very large files to the points of processing. Files are stored on fixed locations whereas processing code, it may be for data analytic purpose, is transferred to the file locations. The code transfer algorithm used in ClientNet is inspired by the visitor design pattern [4] one of the classic software design technique for object oriented software.

1.2 Proposed System

In this paper map computing is implemented in ClientNet cluster system for a given large text files as an example of Big Data. The system consists of 3 clients, 1 coordinator, 1 data server and 1 executor server; all are connected as remote machines. Very large text files to compute the maps are physically stored on the data server. Since the files are large sized therefore do not need to be transported to remote networks for data analytics. Instead, the client codes that can compute the map are transported in parallel to process files on destination node and the results are sent back to the remote clients. This saves network bandwidth, time for real-time data analytics coping with real-time growth of remote data files. The map compute for big files is an example of a scenario to demonstrate mobile code method to deal with Big data. ClientNet distributed cluster is completely developed in Java with built-in RMI and Map compute classes.

The system is in early stage and does not support distributed file system of its own or of any other cluster. It also does not support advanced techniques for in-memory processing of very big data sets which is the focus of my next paper. However, the system provides an efficient coordination among local hosts simulating real scenario of remote map compute job. The system is flexible enough that clients can add as many jobs as they want by adding Java classes for each job. All client jobs execute in parallel using multithreaded Java model of concurrency. It has used the power of Java classes and objects that provides an early version for in-memory processing avoiding the frequent read/write accesses on physical storage. The system is less complex and is tailored directly to serve the processing of remote jobs as compared to Apache Hadoop [5] and Apache Spark [6]. ClientNet is platform independent and fully demonstrates independent functioning of all components of clustered computers distributed over remote locations. This paper is composed of Big Data concepts, its architectures and challenges; introduces the first version of ClientNet cluster system as programming solution for Big Data transfers and data analytics by map computing as an example, its design model and concurrency model.

2 Big Data Concepts and Architectures

2.1 Foundations

According to O'Relly Media [7], who first coined the word Big Data in 2005, they define it as "*Big data is data that exceeds the processing capacity of conventional database systems. The data is two big, moves too fast, or doesn't fit the structures of your database architectures. To gain value from this data you must chose an alternative way to process it*". Almost everywhere in the literature the size of big data is not termed as fixed to designate it to be Big Data. I would consider a big data when it is not possible to handle it with traditional relational database tools and techniques. This includes the size of data must be large enough that cannot be accommodated in database tables, or unstructured enough to extract its meaning, or a continuously growing data that is not possible to be placed in a database container of fixed size. Big data can be considered if it is large enough that available searching and sorting algorithms cannot be applied as they are used for conventional systems. This intrinsically poses the possibility of creation of new techniques and algorithms to target for the typical nature of data. For example, if data is large enough that cannot be passed to remote computers (one of the main focus of this paper) or the processing is possible only through data mining techniques that lead to data clusters and mapping techniques.

More specifically Big data is characterized based on its specific properties known as V's of Big data [1, 2]. Since this paper is not dedicated to the survey and detailed definitions of big data and its available clusters, therefore I have limited its text to focus on the contribution of paper in addition to the brief descriptions.

Volume: It refers to the size that would be equal or beyond the maximum of its size present at the time of writing this article. The volume may possibly go beyond of multiples of petabytes. While considering the distributed big data scenario then it is possible the volume would cross thousands of petabytes. This would lead it to infinitely big enough to measure the size of continuously growing files and the only way to consider it would be by using partitioning algorithms. One can imagine the difficulties of searching and sorting that would need specialized and context bound techniques to achieve goals for every special scenario.

Variety: As given above one of the distinct factors of big data is the variety of data due to which it requires advanced techniques for processing. Types of data may be unstructured data generated through social media in form of random tweets and file attachments produced in multiple contexts of conversation. Unstructured or unformatted data normally cannot be used if data filtering is not applied. However the other formats of data e.g. media files: videos, images, different texts: doc, xml, pdf, txt, etc. are not difficult to maintain in available databases. In this case if the data is continuously being generated to build enormously big size then new techniques would require for real-time data analytics.

Velocity: Rate of growth of data is velocity which is the most crucial factor to deal. This is the biggest challenge that has pushed computing professionals to device new algorithms and high capacity storage devices and high speed computer networks.

Veracity: It is the incomplete or noisy data that makes analytics more difficult. Data is periodically monitored preventing it garbage data. For this purpose data filtering is applied or manually the developers filter it before analytic process begins.

Validity: This refers data must be valid and consistently available in a distributed system of computers. The replicated data must be taken care of its validity before extracting its meaning at different locations in real-time. If data is not valid at all points of processing then the extracted meaning would not be valid as well and results would be inconsistent. This is also the basic property of conventional databases.

Volatility: This is one of the difficult tasks in big data that continuously keeps evolving into different volumes and variety. It refers to the data that is no longer relevant must be discarded or not to store it in valid data containers. To save the space from unnecessary data the volatility processes must be monitored continuously as garbage collection is performed in different systems and languages.

2.2 Big Data Architectures and Technologies

After that the paper moves on the actual goal defined in the title. Big data architecture framework [9] defines several of its models, formats, management methods, analytics methods, infrastructure (storage, methods of accessing, processing and routing of data) and security. Big data technology includes programming tools that provide the solution to the big data architecture framework. This involves technological framework e.g. Hadoop and Spark clusters which are also called big data architectures. These tools are the collection of several components that provide data processing, analytics, storage and distribution solution along with powerful compute engine. The overall data lifecycle and the architecture is called big data ecosystem [9–11]. These large scale clusters can process big data in parallel through unified framework of components for each service. For example, Hadoop contains Hadoop Distributed File System (HDFS) for managing very large data files into partitions spanned over thousands of remote nodes, MapReduce, Mahout, HBase, OoZie, Pig, Flume, Zookeeper, Hive, Sqoop, Whirr, etc. As shown into the well known diagram these tools are integrated in Hadoop tool framework (Fig. 1).

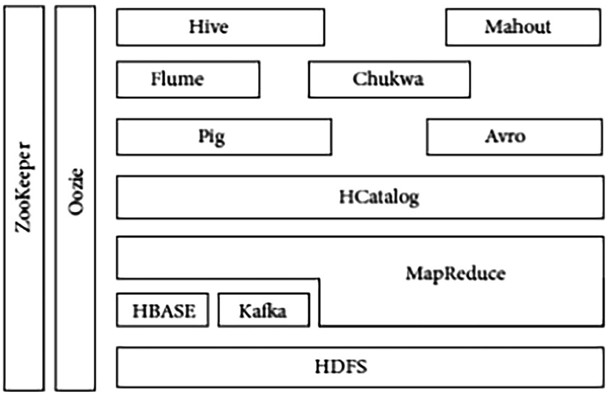

Fig. 1. Hadoop tool suite [12]

One of the famously known tools is MapReduce which is collection of programs written in Java and Scala. This system is used to transform raw text data into counted words from collection of text files so that the output can be queried or analyzed automatically; reduces the text into words and their occurrence as given in the following diagram. Originally MapReduce was developed by Google since then it has been used by others as well. This is one of the big data analytic tools provided by collection of Java classes used in this paper as an example to demonstrate ClientNet cluster. So far it is the solution used only for text processing on large scale distributed systems. For instance it counts words or users and their behavior on Web. Further details are given after the section of Spark.

Mahout is developed by Apache used for distributed collaborative filtering, clustering and classification of data. It is written in Java and Scala therefore supports both languages. The latest version of Mahout is a distributed linear algebra framework designed to let mathematicians, statisticians and data scientists quickly implement their own algorithms (Fig. 2).

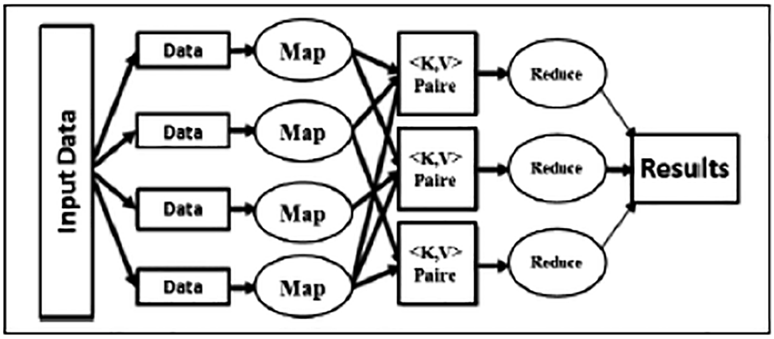

Fig. 2. MapReduce [13]

Microsoft's Azur HDInsight [14] was initially known as Windows Azure uses popular open source frameworks including Apache Hadoop, Spark and Kafka. Azur is cost-effective and provides enterprise-grade service for open analytics. It integrates seamlessly with components of open source echo system of above mentioned clusters with global scale. "*It is a cloud computing platform, designed by Microsoft to successfully build, deploy and manage applications and services through a global network of datacenters*" [15].

For the purpose of simplicity and focus I have deferred more details on other clusters such as Spark, one of highly significant big data clusters. It is more famously known for its in-memory computing and scheduling of availability of large memory objects. The most significant work of Spark is it supports both object oriented programming and functional.

3 ClientNet Cluster

3.1 Introduction

ClientNet is set of Java programs running on a cluster of computers connected by communication network. The first version of it contains 3 clients, 1 coordinator, 1 data server and 1 executor engine. It is scalable to a number of similar clusters providing solution to remote clients by cascading the design to multiple physical locations. Since it is non-commercial and in the early stage of its development therefore currently the ClientNet is simple and limited to minimum number of computers communicating via passing messages through Java Remote Method Invocation protocol as shown in the following diagram. The messages are passed by invoking remote functions and passing objects as parameters.

To keep it simple and working I have implemented the map compute using built-in Java **`HashMap`** [16] for a large size text files that takes long time if transmitted to remote computers. Taking the advantage of message passing, the computing code that is responsible of map analytics is passed from parallel clients to the remote data server where big size text file is physically available. Since the map compute code is light weight therefore it is easier to pass around remote network and execute at the remote server side. After completion, the analysis results (map entries) are sent back to the remote clients as Java objects which are lighter enough for smooth data communication. The process involves coordination of all computers running Java clients and server programs. Since the Java objects are memory residents therefore map compute does not executes frequent reads and writes on hard drive.

3.2 ClientNet Architecture and Programming Model

The cluster system consists of a group of computers coordinating via programming model based on Java RMI [17] as given in the following (Fig. 3).

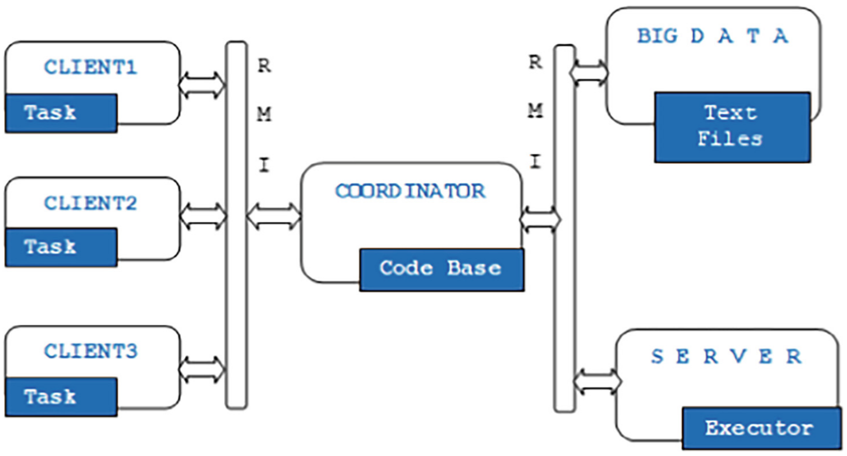

Fig. 3. ClientNet cluster connections diagram.

3.2.1 Connectivity and Message Passing

All machines are running Microsoft Windows operating system. The next release of ClientNet would adopt heterogynous platform. Each computer participating in cluster needs to get connected on communication network before it runs any program. For this purpose each computer registers to RMI registry with computer name recognizable on the network and its port number where it receives and sends messages to other computers as given in the following Java code:

```
Registry DSregistry = LocateRegistry.createRegistry(1099);

DSregistry.rebind("Data", new Data());
```

The RMI registry comes with Java distribution and it must execute on each machine with JVM before the start of any Java Client-Server program. The computer name, e.g. "`Data`", is name string that would be globally known to all of computers in the cluster. Since the big text file does not transfers to other computers therefore Data computer does not send messages to other computers rather it only receives requests from Coordinator for file read access. Once the machine is available on the network it publishes its I.P address, port number and name string to show its global availability. When the Client1 machine, for instance, sends messages to other computer in the cluster group it executes the following lines of Java code before actual message passing.

```
DSregistry = LocateRegistry.getRegistry(6000);

coordinator = (CoordinatorInteface) DSregistry.lookup("127.0.0.1");
```

This means **Client1** looks for other computer's I.P address and its port number by accessing RMI registry function `lookup()`; Here it finds **Coordinator** machine of port number 6000 and localhost identified by I.P: 127.0.0.1. After creating link the

Client1 creates **reference** named `coordinator` to the Coordinator's remote object on Client1 local disk. Since it has Coordinator's reference therefore it can send and receive messages to remotely available Coordinator by invoking interface functions on `coordinator` as given below:

```
plainTextFile = coordinator.getTextFile();
```

The `getTextFile()` is remote interface function of remote Coordinator. The class diagram of ClientNet cluster is given in the following Fig. 6. This function returns the remote reference, to the Coordinator, of type `File` of the big data text file that resides on remote Data server. For security purpose the Coordinator represents Data server for all clients. Everywhere in the cluster the communication takes place among all computers as discussed above example. Remote references are first obtained after creating RMI connections.

3.2.2 Data Transmission and Filtering

Each client is a Java class contains inner class `WordCount` which contains a thread that runs anywhere where it is invoked at destination. The inner class instantiates map object of type `HashMap`. It is built-in Java utility for map computing jobs, records a count for each word entry e.g. `map.put(word, 1)`. Here the `put()` function invokes on **map** object and receives two parameters i.e. `word` text token and its occurrence **1**. The `HashMap` class does not provide filtering functionality by default therefore I have added extra function `applyfiterWord()` that filters out each scanned word from 5 MB text file and truncates the special characters attached to the word. For instance if the given text is:

"**Among the highest living standard cities of the world are Switzerland, Oslo and Copenhagen etc. Switzerland is the most beautiful as compared to Oslo or Copenhagen.**"

The returned map of **ClientNet** program after applying filter function is: {Switzerland = 2, Copenhagen = 2, Oslo = 2, …}, where as the original Java map would return it like this: {Switzerland = 1, Copenhagen = 1, Switzerland**,** = 1, Oslo = 2, Copenhagen**.** = 1, …}. The original Java map computes it as: "**Switzerland,**" and "**Switzerland**" two different words and "**Copenhagen**" and "**Copenhagen.**" as two different words. It is because Java map includes last punctuation marks (, and .) attached to the word as the part of that word. The function `applyfilterWord()` truncates the punctuations marks attached and considers it the same word if the word has appeared before.

Therefore filter function prevents and it does not count the new word occurrence. Each time a word is added after filtering unless the complete text is processed. The text filtering function is lengthy enough to present in this paper therefore for clarity purpose one of its checking conditions is provided in the following diagram. This condition filters for surrounded words, e.g.: "`(Apple)`…" Or "`[Apple]`…" Or

"`(Apple, Orange)`..." and generates only two tokens "**`Apple`**" and "**`Orange`**" as two distinct clean words.

The following peace of code given in Fig. 4 is one of the several **`if`** conditions that filters out some of the special characters and returns a clean word that becomes part of the map entries. The condition checks if the special characters occur at first and last position of a word, that kind of word is a surrounded word e.g. "(Apple)" that is surrounded by two round braces. This condition may further be extended by adding more checks of all special characters. After the condition is evaluated, program copies characters other than braces into another array **`surroundedTail`** and finally converts into String object **`cleanWord`**. The program segment is part of a loop which continues for all scanned words.

```
   if(chars.length > 1){
       if((chars[0]   ==   '*'   &&   chars[chars.length-1]   ==
'*')||(chars[0]    ==    '('    &&    chars[chars.length-1]    ==
')')||(chars[0]    ==    '['    &&    chars[chars.length-1]    ==
']')||(chars[0]    ==    '{'    &&    chars[chars.length-1]    ==
'}')||(chars[0]    ==    '\''    &&    chars[chars.length-1]    ==
'\'')||(chars[0]   ==   '-'   &&   chars[chars.length-1]   ==   '-
')||(chars[0] == '<' && chars[chars.length-1] == '>')||(chars[0]
==  '('  &&  chars[chars.length-1]  ==  ',')||(chars[0]  ==  '['  &&
chars[chars.length-1]    ==    ',')||(chars[0]    ==    '{'    &&
chars[chars.length-1] == ',')){
char[] surrounded = new char[chars.length-2];
for(int x=1; x < chars.length-1; x++){
    surrounded[x-1] = chars[x];
    cleanWord = new String(surrounded);
  }
  return cleanWord;
 }
}
```

Fig. 4. One of the filtering condition taken from the function `applyfilterWord()`.

The ClientNet does not support shuffling of intermediate map results because at this stage it does not support distributed file system which has been left for next version. The system assumes final piece of file and begins map compute when it receives the file. All three clients contain the similar functionality which can be changed at anytime with new data analytics functions. It should be noted that data analytics is not the main focus of this paper.

After that the Coordinator packs all the clients' source codes as binary code in form of objects into the Java array list data structure which is called **Code Base**. The code base contains all threads ready to execute wherever the array list is to be accessed. The code base (array lists) is then transferred from Coordinator to Executor server through

RMI protocol. As the number of client requests will increase the size of code base will also increase. The following peace of code as given in the Fig. 5 shows how the Executor runs each client and finally traverses the list and executes each client thread by the function **`startClientThread()`**.

The Executor program segment traverses the list, accesses each of its remote objects and type casts each object to **`ClientCommonInterface`** by creating local reference **`client`** to the related remote Client object. Once the object reference is created it is then used to invoke remote function on it. The invoked function executes the embedded thread of inner class object of the remote client; this is just to remind that data analytic jobs are embedded into the inner class of each Client.

```
   private void runClients() throws InterruptedException, Re-
moteException{

      int x = 0;
      // process remote clients
      while(x < coordinator.getCollection().size()){

         client      =      (ClientCommonInterface)      coordina-
tor.getCollection().get(x++);

   client.startClientThread();
   gc();
         }
     }
```

Fig. 5. Processing of clients threads at executor server

It is assumed that Coordinator and Executor Server are physically near the location of Data server. Therefore for clients the data access becomes easier as compared if the big files would have been accessed by remote clients independently. The final **computed map** is small size text file that contains map entries. This file can easily be sent to anywhere for knowledge extraction or can be transformed into transactional data by inserting into conventional database tables for query processing.

3.2.3 ClientNet Java Classes

The **class diagram** of full system is given in the following Fig. 6. All clients have similar connections as shown in the diagram which shows only one client. All classes are connected through their related RMI interfaces which are shown on top of the class diagram. Since the text file is accessed only for reading purposes therefore there is no concurrency issues are raised. However all clients are parallel and disjoint because of each one is remotely available and physically independent. The **plus** connector of Client1 shows its relation with its inner class WordCount. The Client1 class is shown empty because analytic jobs are nothing but the inner classes of their related client classes. In this paper only one job is given which is embedded into single inner class. All classes and interfaces boxes contain functions written in their body. The **diamond** connector means the reference to remote object and the normal arrow means inheritance among classes or implementation of interfaces. Class diagram may become more complex if more jobs are added. The sequence in which the system must execute is as follows: First of all run the **start** (connectivity) programs of the Data server, then all Clients, then the Coordinator. Finally Executor server runs its application client that executes its processing engine.

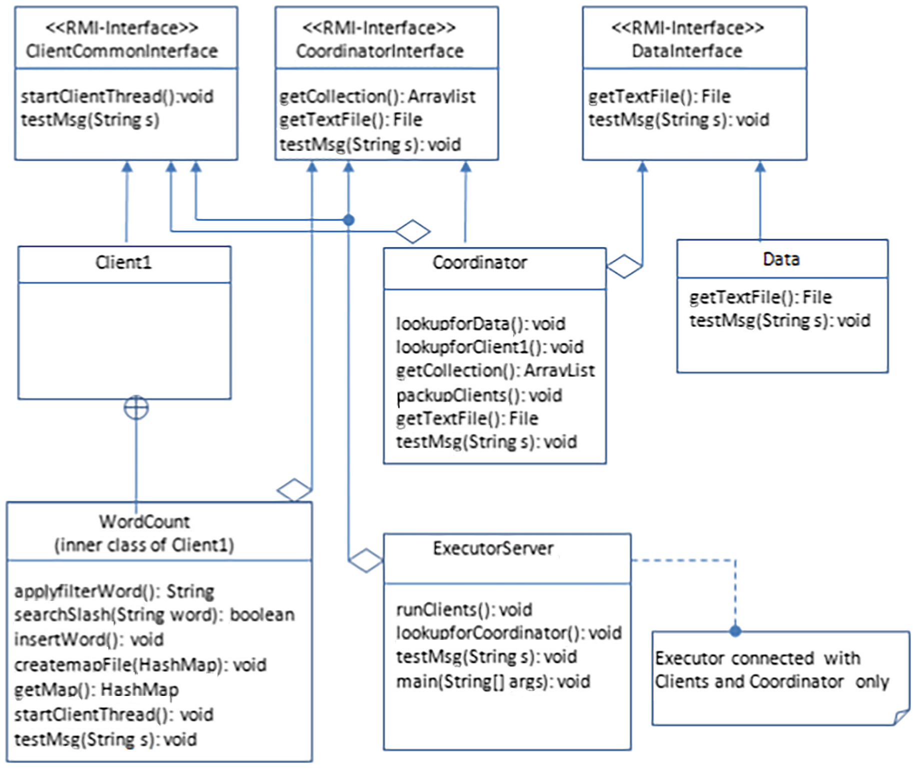

Fig. 6. ClientNet UML classes

At this stage ClientNet is light weight and allows limited functionality of map compute. The uniqueness of this cluster is that the data analytic code, for instance map compute, does not need to be available on the node; rather the node would be passed the source code when analytics is required. This allows light weight and less complex nodes. Therefore every time the data analytics would be performed as a client request. This will allow parallel processing of analytics on loosely coupled resources. The mobile code would enhance mobility of soft resources within the cluster as it is naturally provided by the cloud architecture. It does not support or use any available distributed file system and also does not scales over multiple servers to provided availability of remote resources. I believe if the system can successfully show the basic functioning it can easily be scaled on a number of machines with advanced analytics functionality. The most difficult tasks that would arise are possibly memory management of large objects, scheduling of remotely available storage and processing resources.

4 Conclusion

The cluster successfully implements mobile code technique in Java. ClientNet cluster is scalable to number of clients distributed across remote locations. More work is required to add data analytics of different domains, memory management for large number of objects of big size. Distributed file system and the availability of remote processing nodes is yet to be developed in its next versions. Since the data analytics is not covered. ClientNet can also successfully perform concurrent business transactions with accurate number computation from remote clients in an online scenario. This functioning of the cluster is not present in this paper because banking data transactions is not the subject of this paper. The transactions component of ClientNet cluster implements mutual exclusion to obtain correct calculations by using many of built-in concurrent data structures and functions of Java.

References

1. Watson, H.J.: Tutorial: big data analytics: concepts, technologies and applications. Commun. Assoc. Inf. Syst. **34**, Article no. 65 (2014)
2. Mary, A.J., Arockiam, L.: A study on basic concepts of big data. Int. J. Emerg. Trends Comput. Commun. Technol. **1**, Article no. 3 (2015)
3. Wang, Y., Kung, L.A., Byrd, T.A.: Technol. Forecast. Soc. Change **126**, 3–13 (2018)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, 1st edn. Addison-Wesley Professional, Boston (1995)
5. Polato, I., Goldman, A., Re, R., Kon, F.: A comprehensive view of Hadoop research – a systematic literature review. J. Netw. Comput. Appl. **46**, 1–25 (2014)
6. Apache Software Foundation. Apache Spark Survey 2016 Report, DATABRICKS (2016)
7. Dontha, R.: Big Data. www.digitltransformationpro.com
8. Mohanty, H.: Big data: an introduction. In: Mohanty, H., Bhuyan, P., Chenthati, D. (eds.) Big Data. SBD, vol. 11, pp. 1–28. Springer, New Delhi (2015). https://doi.org/10.1007/978-81-322-2494-5_1
9. Demchenko, Y., Membrey, P.: Defining architecture components of the Big Data Ecosystem. In: International Conference on Collaboration Technologies and Systems (CTS) 2014. IEEE, Minneapolis, MN, USA (2014)
10. Oussous, A., Benjelloun, F.-Z., Lahcen, A.A., Belfkih, S.: Big data technologies: a survey. J. King Saud Univ. Comput. Inf. Sci. **30**, 431–448 (2018)
11. Joseph, C.P., Thulasi, B.S., Susmitha, V.: Big data – concepts, analytics, architectures – overview. Int. J. Eng. Technol. (IRJET) **5**(2), 125–129 (2018)
12. Khan, N., et al.: Big data: surveys, technologies, opportunities, and challenges. Sci. World J. **2014**, 18 (2014)

13. Zerhari, B., Mouline, S., Lahcen, AA.: Big data clustering: algorithms and challenges. In: International Conference on Big Data, Cloud and Applications BDCA 2015, Morocco (2015)
14. https://azure.microsoft.com/en-us/services/hdinsight/
15. Microsoft Azure Tutorial. www.tutorialspoint.com
16. https://www.javatpoint.com/java-hashmap
17. https://www.javatpoint.com/RMI