Volume 4, Issue 6, June 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Parallel Power Iteration Clustering for Big Data using MapReduce in Hadoop

Ankit Darji
CSE Department,
Parul Institute of Technology
India

Dinesh Waghela
CSE Department,
Parul Institute of Technology
India

Abstract-In today's life Distributed Data Mining is most popular topic in research area because as data are increasing in day to day life there are so many problems occurs to handle them and there are also a solutions for that but still they are not as per expectation, still there are some issue already there in the Distributed Data Mining, among them mainly we are focus in this papers that about reducing computational cost. PIC (Power Iteration Clustering) is simple, fast and relatively scalable and uses matrix vector Multiplication. PIC is the combination of two things one is PI that is Power Iteration and remaining C for Clustering. In Power Iteration it finds the largest Eigenvector and then after it apply K-mean algorithm for clustering. Here this algorithm is fast and capable of finding good clusters in a less time but it is not capable for handling large data, so here another is parallel approach for PIC called Parallel Power Iteration Clustering and implementations for minimizing communication cost. Due to its parallel approach different nodes are also their so for make our system more robust and avoid node failure we use Hadoop platform using Map Reduce. Hadoop provide the environment to implement application in distributed environment and it can capable of handling node failure. In Hadoop each DataNode contains the replica of other Datablocks of other DataNode and location of that Datablocks detail store into Namenode. Due to that when any node will goes down or fails at this moment we can use Replica of that Crash node from the other node.

Keywords—Distributed Data Mining, Parallel Clustering, K-Mean Clustering, Threading, Hadoop, NameNode, DataNode,

I. INTRODUCTION

Clustering is a process of organizing data into groups within which the elements are similar in some way. As an unsupervised learning technique mainly for discovering natural groups or underlying structure of a given dataset, clustering has been an active research subject in many fields including statistical analysis, image analysis, pattern recognition, machine learning, and data-mining. Clustering applications also span wide range of domains, including text-mining[4,5] , social network analysis[7] , bioinformatics[6] , market research[8], and scientific and engineering analysis[28], to name a few.. Traditional clustering methods include hierarchical methods (e.g., single link) and partition methods (e.g., k-means). Although those traditional methods are still popularly used, they generally lack robustness and suffer from the so-called ''curse of dimensionality''. Most of those methods are also computational expensive especially when applied to large data. Numerous new clustering algorithms have been introduced as a means to address these issues, for example density-based methods (e.g.DBSCAN)

The paper is organized as follows: In Section 1, Contains introduction, Section 2, Main Method Power Iteration Clustering, Section 3 Contains existing system parallel power iteration clustering. Section 4, Hadoop introduction (HDFS), Section 5 contains Comparison of Performance output. Section 6 contains conclusion.

II. POWER ITERATION CLUSTERING

In 2010,F. LinPresent [3], It represents PIC finds a very low-dimensional embedding of a dataset using truncated power iteration on a normalized pair-wise similarity matrix of the data. PIC is very fast on large datasets which are running over 1,000 times faster than an NCut implementation based on the state-of-the-art IRAM eigenvector computation technique. In spectral clustering the embedding is formed by the bottom eigenvectors of the Laplacian of a similarity matrix. In PIC the embedding is an approximation to an eigenvalue-weighted linear combination of al l the eigenvectors of a normalized similarity matrix. The main focus of PIC is its simplicity and scalability. They demonstrate that a basic implementation of this method is able to partition a network dataset of 100 million edges within a few seconds on a single machine, without sampling, grouping, or other preprocessing of the data.

The equation for finding the Similarity between two data points as below

$$s(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}\right)$$

By using this equation we can find the distance between two data points which is given as an input.

Table 1-Notations and abbreviations used in the paper.

S(xi,xj)	***Similarity function***
A	*Similarity Matrix*
D	*Diagonal Matrix with D(i,i)=ΣjAij*
W	*Normalized Similarity Matrix ,* *W=D-1A*
R	*Row Sum vector*
V0	*Initial vector*
Vt-1,Vt	*Vector at iteration t-1 and t respectively*
y	*Normalized constant*

Steps of PIC Algorithm [10]

```
Input: A dataset x=(x1, x2, x3…..xn) and similarity function S (xi, xj)
//Similarity matrix calculation and normalization
    1)  Construct  similarity matrix (A)
    2)  Normalize similarity matrix by dividing each element by its row sum, $W=D^{-1}A$
//Iterative matrix-vector multiplication
    3)  Generate initial vector, v0= R/||R||1,where R is row sum of $W$
    4)  Repeat the following
    5)  Calculate new vector and new velocity, $v^t=ywv^{t-1}$
    6)  Until stopping criterion is met, $|\delta^t- \delta^{t-1}|\approx 0$
//Clustering
    7)  Use K-Mean to cluster point on vt
Output: Clusters C1, C2 ...Ck.
```

Perhaps one of the greatest advantages of PIC lies in its scalability. Space-wise it needs only a single vector of size n for v^t and two more of the same to keep track of convergence. Speed-wise, power iteration is known to be fast on sparse matrices and converges fast on many real-world datasets; yet PIC converges even more quickly than power iteration, since it naturally stops when v^t is no longer accelerating towards convergence of the dataset points

III. PARALLEL POWER ITERATION CLUSTERING (P-PIC)

In this section they describe methods in detail the parallel implementation of the PIC algorithm. They start this section with a brief discussion of different parallel programming frameworks. The message passing interface (MPI) is a message passing library interface and is the defacto standard for performing communications in parallel programming environments. It has been traditionally used and is still the dominant model for high performance computing. OpenMP as well as POSIX Threads (Pthreads) are other parallel programming strategies, but both of these only work well on shared-memory multiprocessor systems. Due to its efficiency and performance for data communications in distributed cluster environments, they choose POSIX thread (Threading of processes)[9] as the programming model for implementing the parallel PIC algorithm

Steps for Master thread

```
1) Determine # splits of the data to clusters and get the starting and end indices of the cases
2) Broadcast indices to each of the slave child processors
3) Read in one case at a time and broadcast to all    processors.
4) Receive $R_{i,}$ i=1,2,......p from all Slave child processors and concatenate them to obtain
   overall row sum,R
5) Obtain the initial vector,$v^0=R/||R||_1$
6) Broadcast the vector,$v^0$,to all child thread processors
7) Receive sub vector $v_i^t$,i=1,2,..p from all child thread processors and concatenate to obtain
   new vector $v^t$, and normalized it by its element sum
8) Check the stop criteria, $|\delta^t- \delta^{t-1}| \approx 0$
9)If  not, go to step 6
10)If yes, stop
```

we use the cosine similarity as the similarity function for PIC. Let r_1i and r_2i be the starting and the ending indices assigned to the ith slave thread processor. Then the calculations of the similarity sub-matrix calculation and its normalization performed on the ith slave thread processor are shown above. The master processor then collects all row sums from all slave processors and concatenates them into a global row sum, R(n-by-1). The master processor also generates the initial vectorV0 and broadcasts it to all slave processors. Each slave processor updates its vector by performing matrix–vector multiplication. The master collects all updated vectors from slave processors and determines if the stopping criterion is met. The master processor will repeat this process until the stopping criterion is met.

Steps for Slave threads

1) Get starting and end indices from the Master thread
2) Read in chunks of case data and also get case broadcasted from Master
3) Calculate Similarity Sub-Matrix,A_i,n/t-by-n matrix
4) Calculate row sum,R_i,of the sub matrix and send it to Master
5) Normalized sub-matrix by the row sum, $W_i=D_i^{-1}A_i$
6) Receive the initial vector from the Master,v^{t-1}
7) Obtain sub-vector by performing matrix-vector-multiplication$v_i^t=y_iw_iv^{t-1}$
8) Send sub-vector to ,v^t_i,to master

As we can see from the pseudo code, each processor only stores two cases in memory at a time. The amount of memory used can be reduced by implementing a blocking mechanism if necessary. That would also help reduce the computational complexity by exploiting spatial locality. For distributed matrix multiplication, there are generally three basic forms of implementation strategies. They are row-wiseblock striped, column-wise block striped, and checkerboard block decomposition. Choosing these implementation strategies is application-dependent. As described in the algorithm steps, for the p-PIC algorithm the similarity matrix is split row-wise and stored at the T thread in a distributed manner. Thus in this paper we chose the row-wise block stripped matrix–vector multiplication strategy. The overall complexity of row-wise block stripped matrix vector multiplication. This paper describes implementation of a parallelization of power iteration clustering–a recently developed clustering algorithm in this paper, they address the memory issue for storing the similarity matrix by splitting the data into small chunks and distributing these small chunks of data to two threads and we can see that we get the proper clusters of the our pendigit Dataset which is unstructured in manner.

Figure 1:Output with Distributed Data points in P-PIC

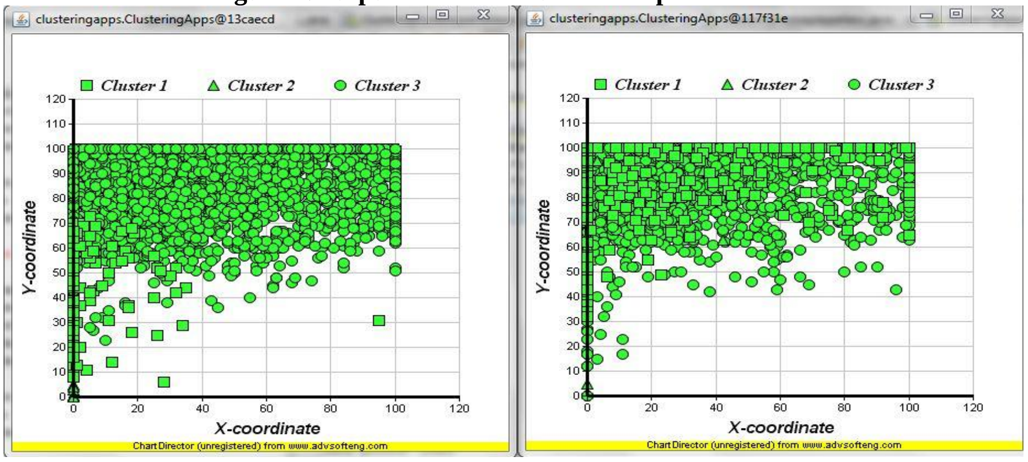
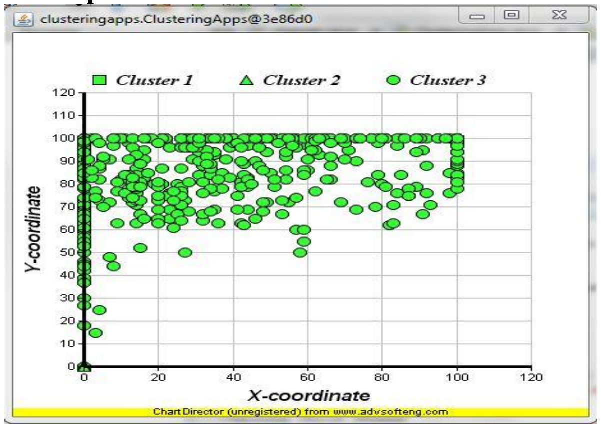


Figure 2:Output with failure of thread and loss of data in P-PIC



As we can see in figure 1, it gives proper output in the form of clusters of our given input dataset. So here we can see that as we apply dataset in distributed way and run them parallel than there is failure of one thread result in loss of data as we can see in figure 2. So if there will not any mechanism to recover our data then it would be difficult to work with large data, that’s why here we are going to use Hadoop technology which can easily handles failure of thread and there will be no loss of data even failures will occurs.

IV. Hadoop Distributed File System[9]

Apache Hadoop is an open-source software framework that supports data-intensive distributed applications. It supports the running of applications on large clusters of commodity hardware. Hadoop was derived from Google's MapReduce and Google File System (GFS) papers.

Both map/reduce and the distributed file systems are designed so that node failures are automatically handled by the framework. It enables applications to work with thousands of computation-independent computers and petabytes of data.

Hadoop is written in the Java programming language and is an Apache top-level project. Hadoop and its related projects (Hive, HBase, Zookeeper, and so on) have many contributors from across the ecosystem. Though Java code is most common, any programming language can be used with "streaming" to implement the "map" and "reduce" parts of the system. The file system uses the TCP/IP layer for communication. Clients use Remote procedure call (RPC) to communicate between each other. HDFS stores large files (typically gigabytes to terabytes), across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts. With the default replication value, 3, Data is stored on three nodes: two on the same rack, and one on a different rack. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high. HDFS is not fully POSIX[3]compliant, because the requirements for a POSIX file system differ from the target goals for a Hadoop application.[9]

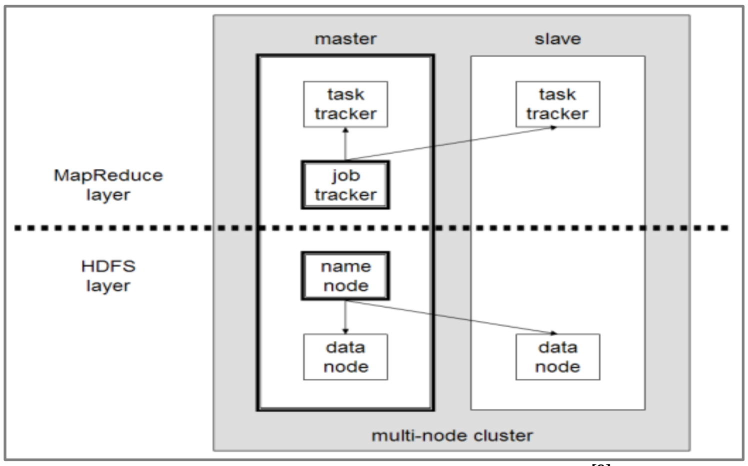


Figure 3: A multi-node Hadoop cluster [9]

A small Hadoop cluster will include a single master and multiple worker nodes. The master node consists of a Job Tracker, TaskTracker, NameNode and DataNode. A slave or worker node acts as both a DataNode and Task Tracker, though it is possible to have data-only worker nodes and compute-only worker nodes. These are normally used only in nonstandard applications. Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard start-up and shutdown scripts require Secure Shell (ssh) to be set up between nodes in the cluster.

[10]

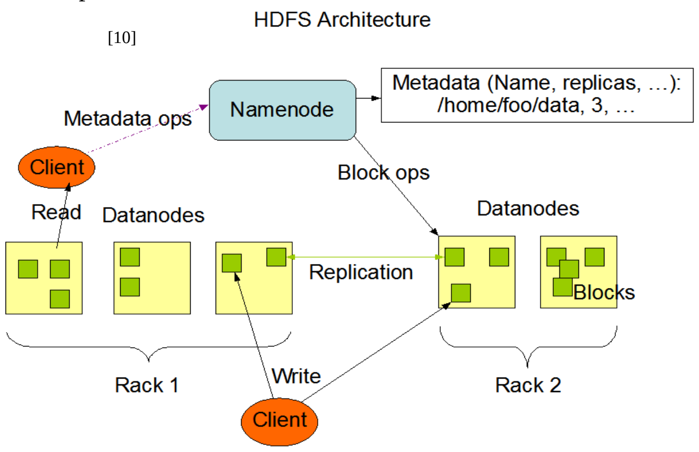


Figure 4: HDFS Architecture[10]

An HDFS cluster is comprised of a NameNode which manages the cluster metadata and DataNodes that store the data. Files and directories are represented on the NameNode by inodes. Inodes record attributes like permissions, modification and access times, or namespace and disk space quotas. The file content is split into large blocks (typically 128 megabytes), and each block of the file is independently replicated at multiple DataNodes. The blocks are stored on the local file system on the DataNodes. The Namenode actively monitors the number of replicas of a block. When a replica of a block is lost due to a DataNode failure or disk failure, the NameNode creates another replica of the block. The NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in RAM. The NameNode does not directly send requests to DataNodes. It sends instructions to the DataNodes by replying to heartbeats sent by those DataNodes. The instructions include commands to: replicate blocks to other nodes, remove local block replicas, re-register and send an immediate block report, or shut down the node. 'MapReduce' is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) Computational processing can occur on data stored either in a file system(unstructured) or in a database (structured). MapReduce can take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted.

"Map" step: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

"Reduce" step: The master node then collects the answers to all the sub-problems and combines them in some way to form the output – the answer to the problem it was originally trying to solve.

MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative.

While this process can often appear inefficient compared to algorithms that are more sequential, MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours. The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one Mapper or reducer fails, the work can be rescheduled – assuming the input data is still available. Another way to look at MapReduce is as a 5-step parallel and distributed computation:[11]

Prepare the Map() input – the "MapReduce system" designates Map processors, assigns the K1 input key value each processor would work on, and provides

1. That processor with all the input data associated with that key value.
2. Run the user-provided **Map() code** – Map() is run exactly once for each K1 key value, generating output organized by key values K2.
3. "Shuffle" the Map output to the **Reduce processors** – the MapReduce system designates Reduce processors, assigns the K2 key value each processor should work on, and provides that processor with all the Map-generated data associated with that key value.
4. Run the user-provided **Reduce () code** – Reduce () is run exactly once for each K2 key value produced by the Map step.
5. Produce the **final output** – the MapReduce system collects all the Reduce output, and sorts it by K2 to produce the final outcome.

Figure 5:Datanode block report to the Namenode

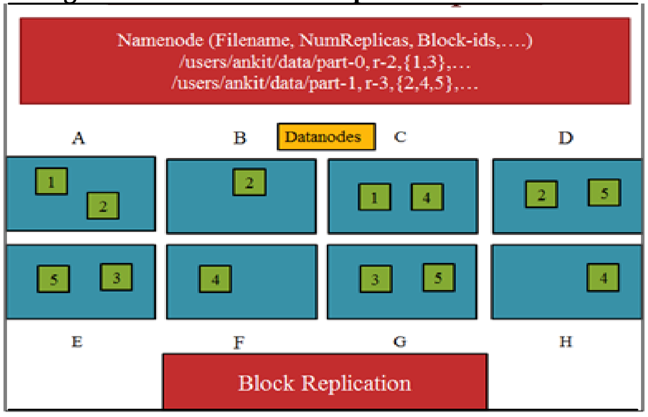
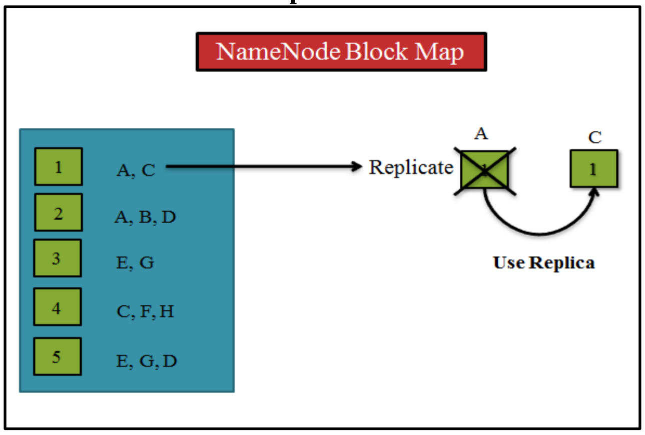

Namenode (Filename, NumReplicas, Block-ids,….)
/users/ankit/data/part-0, r-2,{1,3},…
/users/ankit/data/part-1, r-3,{2,4,5},…
A
B
Datanodes
C
D
1
2
2
1
4
2
5
5
3
4
3
5
4
E
F
G
H
Block Replication

Figure 6:Namenode Block Map that detail sent from theDatanode



As described in above figure here Master and Slave concept are there so in that Master(Namenode) contains the all DataNodes block report as shown in above figure means at which position particular Namenode is. So as shown in above figure 5, we can see that block 1 is replicated in DataNode A,C and block 2 is replicated in A,B and D in this way further all blocks are replicated in DataNodes and these information's are stored in Namenode as Block report. As shown in figure 6, Block Map is stored in Namenode sent by DataNode. So as block 1 which is replicated in A and C, if one of DataNode goes fail or not working than we can continue our work from another replica means if DataNode A will fail than we can work from the DataNode C.

Figure 7:Result of Hadoop Implementation with job failure

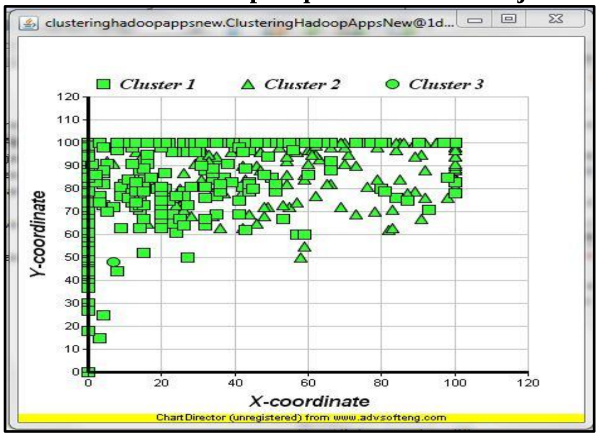
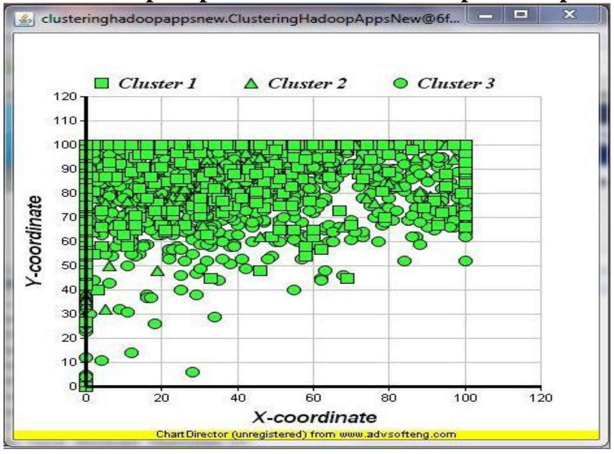
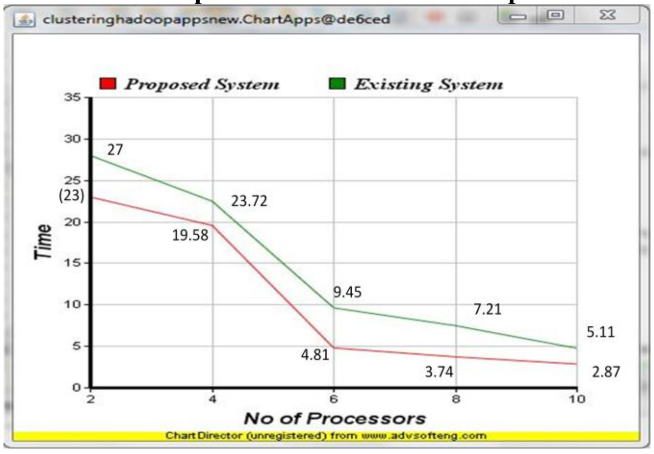


Figure 8: Result of Hadoop Implementation with complete output due to Replica



As we can see in above figure it describe the output of the our PPIC algorithm in Hadoop distributed system and we can see that if we are distributing our data in Hadoop environment so as per the our above theory of HDFS(Hadoop Distributed File System) in parallel processing if there will be some failure occurs so Hadoop can easily handle it so as we implement our PPIC algorithm in this environment and suppose there will be failure occurs and one job will be fail as shown in figure 7 than also due to Hadoop facility of create replica there are no loss of data and even though there is failure of node we get proper output of our data and due to replica there will be no loss of any data and we get good clusters of our given input data as shown in figure 8.So as the advantage of Hadoop platform the issue of Parallel Power Iteration Clustering of failure of node during execution that can be easily handle by Hadoop platform and there is no loss of data.

V. Comparison of Performance output.



As shown in above figure of performance between existing system that is Parallel Power Iteration Clustering(PPIC) and Proposed system that is Implementing PPIC using Map Reduce in Hadoop technology ,that performance graph describes that as we are increasing no. of processors than our proposed system gives good performance in time as compared to existing system. In our implementation we use Net beans 7.0 version and we put 0.0001 stopping criteria and we get good result. Here we also use Hadoop-0.20.6.jar,Hadoop-0.20.2-core.jar,commons-logging-1.0.4.jar,commons-httpclient-3.0.1.jar,commons-cli-1.2.jar,com.springsource.org.apache.log4j-1.2.16.jar.

VI. CONCLUSION

Parallel Power Iteration Clustering is implement for minimizing communication cost, Due to its parallel approach different nodes are also their so for make our system more robust and avoid node failure here we use Hadoop platform using Map Reduce. Hadoop provide the environment to implement application in distributed environment and it can capable of handling node failure. Due to creating replica we solve the issue of node failure of existing system. In future we can use another approach for clustering the data and check the performance by implementing it in Hadoop and also we can work on . Hadoop system in which (1)If Namenode will fail than how to take care of data(2) Try to develop append write model in Hadoop(3)Extend the capacity of NameNode because if A namespace with an extremely large number of files exceeds NameNode capacity to maintain.

ACKNOWLEDGEMENT

I thanks to Assistant Professor Dinesh Waghela sir for the inspiring me for doing such work and due to their support here I am able to write this research paper and he also provide me good knowledge about big data and Guide me for doing this work in Hadoop platform.

REFERENCES

[1] F. Lin, W.W. Cohen, “Power iteration clustering”,27th International Conference on Machine Learning, pp. 655–662, 2010,

[2] WeizhongYana, Umang Brahmakshatriyaa, YaXuea, MarkGilderb, BowdenWisec“p-PIC: Parallel power iteration clustering for big data” J.ParallelDistrib.Comput. 73(2013)352–359

[3] POSIX threads programming, High Performance Computing: High PerformanceComputing, Web, 30 June2011.https://computing.llnl.gov/tutorials/pthreads/

[4] D. Cai, X. He, J. Han, Document clustering using locality preserving indexing,IEEE Transactions on Knowledge and Data Engineering 17 (12) (2005)1624–1637

[5] I. Dhillon, S. Mallela, R. Kumar, Enhanced word clustering for hierarchical text classification, in: Proceedings of the 8th ACM SIGKDD, Edmonton, Canada,2002, pp. 191–200.

[6] A. Enright, C. Ouzounis, GeneRAGE: a robust algorithm for sequence clustering and domain detection, Bioinformatics 16 (2000) 451–457.

[7] S. Sharma, R.K. Gupta, Improved BSP clustering algorithm for social network analysis, International Journal of Grid and Distributed Computing 3 (3) (2010)67–76.

[8] A.J. Vakharia, J. Mahajan, Clustering of objects and attributes for manufacturing and marketing applications, European Journal of Operational Research 123 (3)(2000) 640–651.

[9] http://en.wikipedia.org/wiki/Apache_Hadoop

[10] http://hortonworks.com/hadoop/hdfs/

[11] http://en.wikipedia.org/wiki/MapReduce