A MapReduce based Parallel K-Means Clustering for Large Scale CIM Data Verification

Chuang Deng, Yang Liu*, Lixiong Xu, Jie Yang, Junyong Liu
School of Electrical Engineering and Information, Sichuan University, 610065, Chengdu, China

Maozhen Li
Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, UB8 3PH, UK

Abstract

The Common Information Model (CIM) has been heavily used in electric power grids for data exchange among a number of auxiliary systems such as communication systems, monitoring systems and marketing systems. With an rapid deployment of digitalized devices in electric power networks, the volume of data continuously grows which makes verification of CIM data a challenging issue. This paper presents a parallel K-means for large scale CIM data verification based on the MapReduce computing model which has been widely taken up by the community in dealing with data intensive applications. By distributing the CIM data into a number of computers in a MapReduce cluster environment, the computation in CIM data verification is significantly improved. Furthermore, a load balancing scheme is designed to balance the workloads among the heterogeneous MapReduce computing nodes for a further improvement in computation efficiency. The performance of the parallel K-means clustering in CIM data verification is first evaluated in a small scale experimental MapReduce cluster and subsequently evaluated in a large scale simulation environment.

Keywords: *CIM verification, stochastic sampling, clustering, MapReduce, load balancing*

1. Introduction

An electric power grid is a complex system which consists of a large number of supporting information and communication systems. The past few years have witnessed the employment of Common Information Model (CIM) as a standard way to facilitate data and information exchange among these heterogeneous systems in power networks. CIM can model components in power networks with defined properties, protocols and parameters. Not only can the power system itself be described using CIM but also the supporting systems can be described by the model. Each system generates a different set of data and transfers the dataset among the communication networks for further utilization such as monitoring, controlling and maintenance. In addition, the integration of data from these supporting systems can significantly benefit the power network.

CIM is an abstract model which not only can describe the electric power objects such as facilities like generators, switches, transmission lines, transformers, but also can describe the information of systems. To facilitate the design and usage of the model, CIM follows the syntax of Extended Markup Language (XML). As a result, being a kind of XML based data model, CIM now has become a widely accepted standard technology for data exchange and interoperation among multiple systems. However, only well-formed and strictly-validated CIM data is considered as standard modeled data in dealing with data heterogeneity among different systems. One CIM model may be incorrectly interpreted by different systems due to model misunderstanding, or the attribution tags are incorrectly selected by the data encapsulating applications. As a result, CIM data must be verified to ensure data to be interpreted correctly. As CIM is a type of XML based data model, XML verification techniques can be employed to verify CIM data using a tool such as Dom4J [24, 25, 27]. At present, the most frequently used verification approaches are based on the concept of schema. Basically schema can be considered as an abstract skeleton of data, which contains data attributes such as types, values, dependences. A schema based verification process can detect data inconsistency. However, with the rapid development of a variety of computer systems in the electric power grid, it has become a challenging issue to ensure a large amount of CIM data to be correct and consistent all the time.

MapReduce has become a de facto standard computing model in support of data intensive applications [11]. The MapReduce computing model facilitates a number of important functions such as

* Corresponding Author: Yang.Liu@scu.edu.cn

partitioning the input data, scheduling MapReduce jobs across a cluster of participating nodes, handling node failures, and managing the required network communications [35]. We have implemented a MapReduce based parallel K-means clustering for scalable information retrieval [33]. The parallel K-means segments the original data into a number of sub-clusters which can be processed in parallel. In this paper, we employ the MapReduce based parallel K-means for CIM data verification which distributes the computation into a number of computer nodes in a cluster environment. It should be pointed out that a random selection of the initial centroids would cause K-means unstable in clustering. To enhance the stability of the MapReduce based parallel K-means, we use Stochastic Sampling and Merging (SSM) technique to generate the initial centroids for K-means to stabilize in clustering. Although the MapReduce model supports heterogeneous environments, it does not have a sufficient load balancing scheme to utilize the resources with a variety of computing capabilities. For this purpose we have implemented a genetic algorithm based load balancing scheme to enhance the utilization of the computing resources of a MapReduce cluster in support of large scale information retrieval [33] and image annotation [34]. In this paper, the load balancing scheme is employed to enhance the scalability of the parallel K-means clustering for CIM data verification in heterogeneous MapReduce computing environments.

The performance of the parallel K-means clustering in CIM data verification is evaluated in a small scale experimental MapReduce computer cluster. The scalability of the parallel K-means is further evaluated in large scale simulated MapReduce environments. Both the experimental and simulation results show that the parallel K-means reduces the CIM data verification time significantly compared with a standalone sequential K-means clustering while generating a high level of precision in data verification.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 presents the algorithm design in detail including the parallel K-means and SSM for selection of the initial k and centroids in K-means. Section 4 presents a load balancing scheme based on genetic algorithm to enable the parallel K-means to better utilize computing resources in heterogeneous MapReduce environments. Section 5 evaluates the performance of the parallel K-means and analyzes the experimental results. Section 6 concludes this paper.

2. Related Work

CIM has been widely used in many fields due to its comprehensive, consistent and object oriented description ability. Memon et al [16] proposed an extensible information service (CIS) with an underlying unified information model. The CIS model consumes data from sources and formats it according to CIM. It can deliver the data against XQuery requests. Memon et al successfully implemented a service discovery approach based on the CIS model. However, the authors pointed out that the CIS model has not been verified in a large-scale computing environment. Field et al [17] focused on one characteristic of CIM which can describe the relationships between entities within the electric grid infrastructure along with their semantics. A data model enables consumers of information to efficiently find the information they require and ensures the meanings of the information to be consistent with that produced by producers. The authors argued that CIM facilitates data exchange in heterogeneous systems. Ranzhe1 [18] discussed the necessity of CIM in enterprise environments which encompass information related to products, resources and management.

In 1990s, the US Electric Power Research Institute started a project on CIM for power systems. A number of studies have been done on information modeling and structuring. Wang et al [19] extended CIM concepts to electrical power distribution networks. The authors stated that the electric power grid is an extremely large enterprise system with many computer systems and applications to complete both business and engineering functions, which motivates the development of efficient information exchanging and sharing infrastructure. This work created a common data model based on the extensions of standard CIM. In their small-size data tests, they claimed that their work successfully modeled the electrical power distribution. However, the authors admitted that the power distribution network has a large number of objects which can generate a large size of data which makes CIM data verification a challenging issue. Hargreaves et al [5] developed a novel metadata model repository which is used to represent knowledge of enterprise power system resources. The repository leverages the value of model namespaces and resource description framework technology in providing contexts for multiple identities referring to common power system resources. The metadata model offers a more realistic understanding of the network reality by merging a number of metadata models. There are a

number of research efforts on CIM not only for modeling various information elements but also for data consistency verification [6][7][8]. Works presented in [1][2][3] focus on verifying the consistency of CIM data. Mao et al [4] defined the consistency of the data based on the OSI model. This work further showed the importance and necessity of verifying CIM data. Sinz et al [20] developed a CIM verifier to guarantee their CIM based model to work consistently in Apache Web-Server configurations. Although a number of studies have been conducted on CIM data verifications, Field et al [17] pointed out that currently the most widely used verification approach is based on data schema. Nevertheless, speeding up the computation process in CIM verification has attracted a significant research effort with a continuous growth in the datasets defined in CIM. For example, Ming et al [21] proposed an extended version of CIM to describe the features of a micro-power grid and focused on optimizing and designing a real-time database to enhance data retrieval process. Summarizing, CIM can greatly help to model complex systems such as power systems.

MapReduce has been taken up by the community in dealing with data intensive applications [10, 31]. However, MapReduce only offers simple job scheduling schemes which may deteriorate the performance significantly in heterogeneous computing environments [23, 26, 30, 31]. Fan et al [28] proposed a reducer-phase based load balancing algorithm, however, mappers are highly time consuming as computational tasks are usually executed by mappers. Considering disk utilization rate, the work presented in [29] modeled both disk utilization and service blocking rate on each datanode. This work does not other factors such as processor powers, memory space and network bandwidth which can impact the performance on MapReduce. This paper employs genetic algorithm to optimizes balance the workloads in MapReduce environments.

3. Algorithm Design

Considering a CIM file, it describes a certain type of data consisting of a number of attributes. These attributes can be represented by a series of special keywords such as { *transmission line, transformer, …, switch* } which describes electrical devices. Let v represent a CIM file, λ_i represent keywords describing the file, n represent the number of keywords, then we have $v = \{\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_n\}$.

K-means is a clustering algorithm based on the distances between centroids and points (vectors) denoted by Eq.(1).

$$d(x, y) = \left|\left|x - y\right|\right| = \sqrt{\sum_{i=1,j=1}^{n,k} (x_i - y_j)^2} \qquad (1)$$

where

- x_i represents ith points,
- y_j represents the jth centroids,
- n represents the number of points,
- k represents the number of centroids.

3.1 Parallelizing K-means with MapReduce

K-means involves a number of loosely coupled iterations [10] which can be fully parallelized in the MapReduce cluster environment. MapReduce has two primary functions. One is the map operation (mapper) and the other is the reduce operation (reducer). All the data chunks in MapReduce are processed as Key (K)-Value (V) pairs. A mapper processes a {K1, V1} pair and generates an intermediate list {K2, V2} pairs. A reducer takes all the values represented by the same key in the intermediate list generated by a mapper and processes them accordingly, generating a final new list {V2}. All the map and reduce operations can run independently in parallel. There are a number of MapReduce implementations including Mars [14], Phoenix [15] and Hadoop framework [13]. Hadoop has been widely taken up by the community due to its open source feature.

The Hadoop framework is an open source implementation of MapReduce mainly developed in Java. The framework greatly enables the distributed processing of large data sets across a computer cluster using commodity computers. Within a Hadoop computer cluster, individual computers share a Hadoop Distributed File System (HDFS), in which the computers are logically cataloged into one Namenode and a number of Datanodes. The Namenode manages metadata of the cluster in which a Jobtracker frequently locates to dispatch data for the whole cluster. A Datanode is actual a processing node

running Map and Reduce functions of MapReduce model. When a job is submitted into the Hadoop cluster, the data is firstly divided into small chunks and saved in the HDFS. In terms of data integrity, each data chunk has replications according to the cluster configuration. When a job starts, the execution file is copied to each Datanode, afterwards the Jobtracker running on the Namanode uses heartbeat to contact Tasktrackers on a Datanodes. Based on data locality mappers either copy data from a remote node or from a local node. The final result will be sorted, merged and generate by reducers to HDFS. Fig.1 shows the architecture of a typical Hadoop cluster.

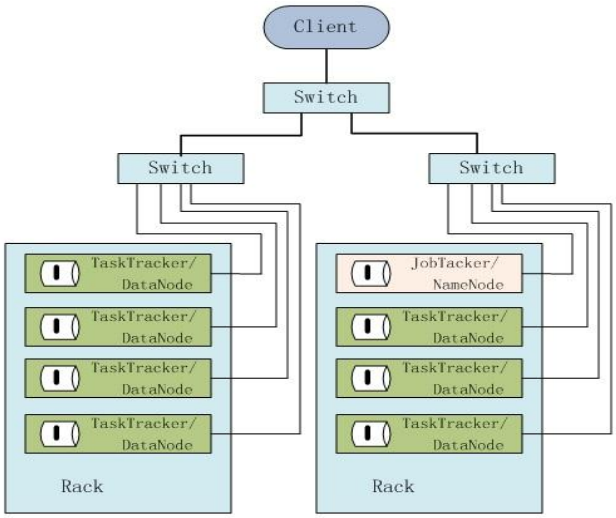


Fig.1. The architecture of Hadoop cluster.

To reduce the computation overhead of K-means in clustering CIM files, the MapReduce based parallel K-means groups CIM data into a number of categories. The Hadoop HDFS can save the whole dataset using data chunks and process these data chunks in a distributed way. Therefore, each computer needs to process a small portion of the original data. In such a process, each mapper computes the distance between each point and centroids, which enables each point to be clustered into a certain cluster. Then, a reducer copies all the outputs of the mappers and merges the points belonging to the same cluster. Moreover the reducer computes the new centroids for all clusters. The iteration repeats until the centroids computed by the reducer become stable. Finally the reducer outputs the final clusters with clustered points back into HDFS.

Let

- $\boldsymbol{D}$ represent the set of p points, $\boldsymbol{D} = \{d_1, d_2, d_3, \dots, d_p\}$
- $\boldsymbol{P}$ represent the set of m processors, $\boldsymbol{P} = \{p_1, p_2, p_3, \dots, p_m\}$, each processor runs one mapper.
- $\boldsymbol{M}$ represent the set of m mappers, $\boldsymbol{M} = \{map_1, map_2, map_3, \dots, map_m\}$

The data set $\boldsymbol{D}$ can be represented by a set of vectors denoted by $\boldsymbol{V}$:

$$\boldsymbol{V} = \{v_1, v_2, v_3, \dots, v_p\}$$

Each vector v_i represents the frequency of keywords that appear in one CIM data d_i. The input of each mapper includes two parts. The first part is a centroid set of $\boldsymbol{C}$ with k initial centroids which are randomly selected from the vector set $\boldsymbol{V}$:

$$\boldsymbol{C} = \{c_i \in V | c_1, c_2, c_3, \dots, c_k\}$$

The second part of the input of a mapper is a portion of $\boldsymbol{V}$ denoted by $\boldsymbol{V}_i$. The vector set $\boldsymbol{V}$ is equally divided into m portions according to the number of mappers. Thus V_i satisfies:

$$\bigcup_{i=1}^{m} \boldsymbol{V}_i = \boldsymbol{V}$$

Each mapper m_i runs on one computer p_i calculating the Euclid distances between $v_{ij} \in \boldsymbol{V}$ and $\boldsymbol{C}$ which is denoted by d_{ij}, then

$$d_{ij} = \left\| v_{ij} - c_q \right\|,\ j = 1, 2, \ldots, \frac{p}{m},\ \ q = 1, 2, \ldots, k$$

Let d_{min} represent the shortest distance between v_{ij} and $\boldsymbol{C}$, thus:

$$d_{min} = min\ (d_{i1}, d_{i2}, d_{i3}, \ldots, d_{i\frac{p}{m}})$$

Based on the shortest distance, the mapper selects the corresponding c_i and v_{ij} to generate a *key-value* pair as one output record. The output pairs of all the mappers are ultimately fed into the reducer. The reducer groups the values with the same key c_i into a set of clusters denoted by $\boldsymbol{Cluster}_i$:

$$\boldsymbol{Cluster}_i = \{v'_1, v'_2, v'_3, \ldots, v'_{ai}\}$$

For each $\boldsymbol{Cluster}_i$ the reducer calculates a new centroid denoted by c'_i:

$$c'_i = \frac{\sum_{j=1}^{a_i} v'_j}{a_i}$$

And then it outputs a set of centroids denoted by C':

$$C' = \{c'_1, c'_2, c'_3, \ldots, c'_k\}$$

Finally $\boldsymbol{C}'$ is fed into the mappers for computing another set of centroids $\boldsymbol{C}''$again, until the values of the centroids in set $\boldsymbol{C}'$ are the same as those in $\boldsymbol{C}''$. In this case the reducer outputs the finally clustered sub-clusters denoted by $\boldsymbol{Cluster}_i$.

3.2 The Selection of Initial k with Stochastic Sampling and Merging

As each CIM data can be represented by $\boldsymbol{V}$, the parallel K-means considers each point $\boldsymbol{V}$ as a centroid of one cluster, which only contains the point itself. Stochastic Sampling and Merging (SSM) is used to define a threshold distance *r* to determine if the distance of two points from different clusters is less than *r*, then the two clusters can be merged. Therefore in order to measure the distances of two points, Euclid distance (*d*) is used. Based on the following equations SSM determines if two clusters contain two points that should be merged.

If $d \leqslant r$, *merge*

If $d > r$, *ignore*

The selection iteration process continues working until the distance *d* between any two points is less than the threshold *r*, the iteration is terminated. The un-clustered points can be clustered based on the minimum distance between each point and the centroid of a certain cluster using Euclid distance. The centroid *c* of a cluster is represented by Eq.(2).

$$\boldsymbol{c} = \{\frac{\sum_{i=1}^{m} v_{1i}}{m}, \frac{\sum_{i=1}^{m} v_{2i}}{m}, \frac{\sum_{i=1}^{m} v_{3i}}{m}, \ldots, \frac{\sum_{i=1}^{m} v_{ni}}{m}\} \qquad (2)$$

where *m* represents the number of points in the cluster and *n* represents the length of point.

The computation process in k selection can compute the number of clusters and the centroids of each cluster. However, the computation is a time consuming process as it needs to traverse all the points for calculating the distances. To speed up the computational process, SSM also uses sampling to reduce the size of data. Therefore, SSM can determine the initial values of *k* and centroids using Eq. (2).

The algorithm terminates when:

$$|\boldsymbol{C}_{n-i} - \boldsymbol{C}_{n-i-1}| \leq \boldsymbol{\varepsilon}$$

where $\boldsymbol{\varepsilon}$ is threshold for terminating the algorithm.

3.3 Parallelizing CIM Verification

Based on the algorithm proposed above, the CIM data can be clustered into a number of sub-clusters, in which contains the points highly related to each other. This design firstly can facilitate the usage of categorized CIM data for different types of application systems. And secondly it potentially reduces the loading times of schema files for data verification since ideally only one kind of schema is needed for one cluster minimally.

In each sub-cluster, the verification works in the following way. Let D_i represent the data size of CIM data with similar points in cluster i. Therefore each mapper of a Hadoop cluster can take each D_i as input. And also in terms of verification, a pre-designed schema file which can verify D_i is fed into the mapper too. As a result, the input data is read and verified using the corresponding schema file. The intermediate output of mappers is in the form of <*point ID, verification result*> key-value pair. At last reducer merges all the outputs copied from mappers and outputs the final results using <*point ID, verification result*> pairs in a final result list. There is one exception that if one CIM point is incorrectly clustered, the algorithm starts loading the other schema files and tries to verify it. Among the extra loaded schemas, if one schema verifies the CIM point to be correct, the verification result is labeled as correct. Contrarily, if no schema file reports the correctness of the CIM point, then the result will be labeled with an *incorrect* tag. Fig.2 shows the dataflow of the parallel CIM verification process.

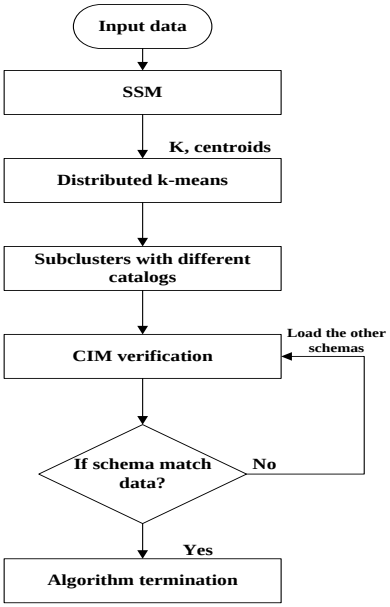


Fig.2. The workflow of parallel CIM verification.

4. Improving the Parallel *K*-Means Performance with Load Balancing

The CIM verification can be conducted in parallel in a Hadoop MapReduce cluster. One critical issue with Hadoop MapReduce is its lack of sufficient load balancing schemes in utilizing heterogeneous computing resources. This section presents a load balancing scheme to further improve the performance of the parallel K-means in computation [33].

The MapReduce based verification involves a number of mappers, however, only one reducer is needed as all results are generated in one file. And also the most time consuming verifying computations are carried out by the mappers. Therefore load balancing is mainly focused on the

mapper processing phase. In Hadoop framework the total time (T) of a mapper consists of the following four parts:

- Data copying time (t_c) in copying a data chunk from Hadoop distributed file system to local hard disk. It depends on the available network bandwidth and the writing speed of hard disk.
- Processor running time (t_p) in processing a data chunk.
- Intermediate data merging time (t_m) in combining the output files of the mapper into one file for reduce operations.
- Buffer spilling time (t_b) in emptying filled buffers.

Therefore:

$$T = t_c + t_p + t_m + t_b \quad (3)$$

Let

- D_m be the size of the data chunk.
- H_d be the writing speed of hard disk in MB/second.
- B_w be the network bandwidth in MB/second.
- P_r be the speed of the processor running the *mapper* process in MB/second.
- B_f be the size of the buffer of the *mapper*.
- R_a be the ratio of the size of the intermediate data to the size of the data chunk.
- N_f be the number of frequencies in processing intermediate data.
- N_b be the number of times that buffer is filled up.
- V_b be the volume of data processed by the processor when the buffer is filled up.
- s be the sort factor of Hadoop.

We have:

$$t_c = \frac{D_m}{\min(H_d, B_w)} \quad (4)$$

Here t_c depends on the available resources of hard disk and network bandwidth. The slower one of the two factors will be the bottleneck in copying data chunks from Hadoop distributed file system to the local hard disk of the mapper.

$$t_p = \frac{D_m}{P_r} \quad (5)$$

When a buffer is filling, the processor keeps writing intermediate data into the buffer and in the mean time the spilling process keeps writing the sorted data from the buffer to hard disk. Therefore the filling speed of a buffer can be represented by $P_r \times R_a - H_d$. Thus the time to fill up a buffer can be computed by $\frac{B_f}{P_r \times R_a - H_d}$. As a result, for a buffer to be filled up, the processor will generate a volume of intermediate data with the size of V_b which can be computed using Eq.(6).

$$V_b = P_r \times R_a \times \frac{B_f}{P_r \times R_a - H_d} \quad (6)$$

The total amount of intermediate data generated from the original data chunk with a size of D_m is $D_m \times R_a$. Therefore the number of times for a buffer to be filled up can be computed using Eq.(7).

$$N_b = \frac{D_m \times R_a}{V_b} \quad (7)$$

The time for a buffer to be spilled once is $\frac{B_f}{H_d}$, therefore the time for a buffer to be spilled for N_b times is $\frac{N_b \times B_f}{H_d}$. Then we have:

$$t_b = \frac{N_b \times B_f}{H_d} \quad (8)$$

The frequencies in processing intermediate data N_f can be computed using Eq.(9).

$$N_f = \left\lfloor \frac{N_b}{s} \right\rfloor - 1 \qquad (9)$$

When the merging occurs once, the whole volume of intermediate data will be written into the hard disk causing an overhead of $\frac{D_m \times R_a}{H_d}$. Thus if the merging occurs N_f times, the time consumed by hard disk IO operations can be represented by $\frac{D_m \times R_a \times N_f}{H_d}$. We have

$$t_m = \frac{D_m \times R_a \times N_f}{H_d} \qquad (10)$$

The total time T_{total} to process data chunks in one processing wave in MapReduce Hadoop is the maximum time consumed by k participating *mappers*, where $T_{total} = \max(T_1, T_2, T_3, \ldots, T_k)$

According to divisible load theory, to achieve a minimum T_{total}, it is expected that all the *mappers* to complete data processing at the same time: $T_1 = T_2 = T_3 = \ldots = T_k$

In terms of solving the functions above, genetic algorithm is employed. For measuring the differences of T_i, we use mean square errors to build the fitness function.

Let

- T_i be the processing time for the i^{th} *mapper*.
- $\bar{T}$ be the average time of the number of k *mappers* in data processing, $\bar{T} = \frac{\sum_{i=1}^{k} T_i}{k}$

Therefore the fitness function can be represented by Eq.(11).

$$f(T) = \sqrt{\sum_{i=1}^{k} (\bar{T} - T_i)^2} \qquad (11)$$

5. Experimental Results

To evaluate the performance of the parallel K-means clustering, a Hadoop cluster was built consisting of 3 computer nodes. The details of the experimental environments are presented in Table 1.

Table 1: Hadoop cluster.

Standalone node	CPU Corei3-2120@3.3GHz MEM 8GB HDD 500GB OS Fedora12
Namenode	CPU Core2-P7750@2.6GHz MEM 5GB HDD 160GB OS Fedora12
Datanode1	CPU Corei3-2120@3.3GHz MEM 8GB HDD 500GB OS Fedora12
Datanode2	CPU Corei7-4770K@3.5GHz MEM 32GB SSD 256GB OS Fedora12

5.1 Evaluating Sequential K-Means

The Iris data which is a published machine learning benchmark dataset [32] was used to evaluate the precision, stability and efficiency of the SSM based sequential K-means in clustering.

5.1.1 Precision

SSM uses stochastic sampling and a threshold *r*, tests were conducted to find out how the volume of the sampling data and the value of *r* impact performance of the standalone K-means in clustering. The clustering precision *p* can be computed using $p = \frac{d'}{d}$, where *d'* represents the number of correctly clustered data and *d* represents the total number of correct data. In the first test we duplicated the volume of data from 4KB to 312MB. The sampling rate was set to 10% and *r* was set to 0.79. Fig.3 shows that when the data size is small, the precision is low using sampling. However, The precision increases with an increasing size of data. When the data size is larger than 2500kB, the precision stays at 70%. As shown in Fig.4, a larger sampling rate can generate a higher precision in clustering.

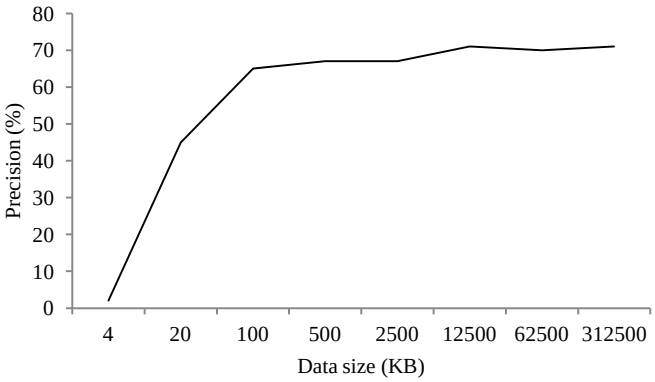


Fig.3. The precision of SSM.

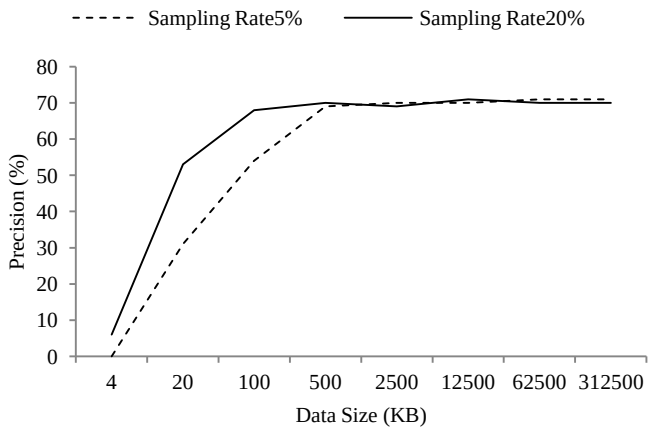


Fig.4. The impacts of sampling rates.

Fig.5 shows the impact of *r* on the precision. In the two tests the values of *r* were respectively set to 0.30 and 3.00 and sampling rate was set to 10%. Firstly at the point of 20KB, it can be seen that the precision sharply jumps from 4% to 31%. When the volume of sampled data is not large, the distances of sampled points may not be close enough, thus a larger *r* can help to cluster points with large distances. As the sampled data becomes saturated, a larger *r* may merge more points which should have been separated.

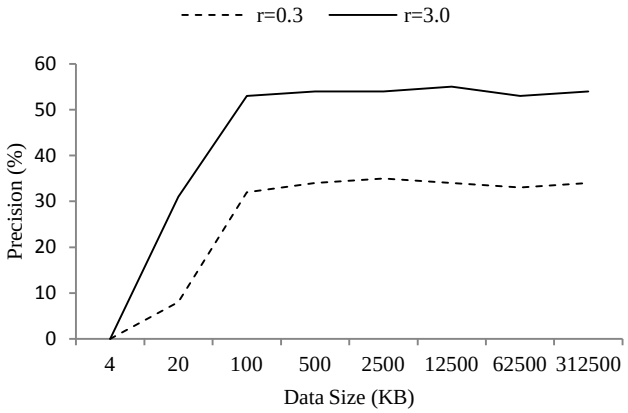


Fig. 5. The impact of *r*.

5.1.2 Stability

SSM is used to select the initial centroids and number of clusters. SSM was evaluated in comparison with standard K-means and K-medoids. As there are 3 clusters in the original data set, thus k was set to 3 for the two algorithms. For SSM, the sampling rate was set to 10% and *r* was set to 0.79. The testing data was duplicated to 100KB. The experimental results are shown in Fig.6. It can be observed that due to the randomly initial centroids and medoids selection, both standard K-means and K-medoids algorithms fluctuate in performance with varied precisions. Contrarily, when a proper value of *r* is given, SSM is stable in all the 10 execution runs.

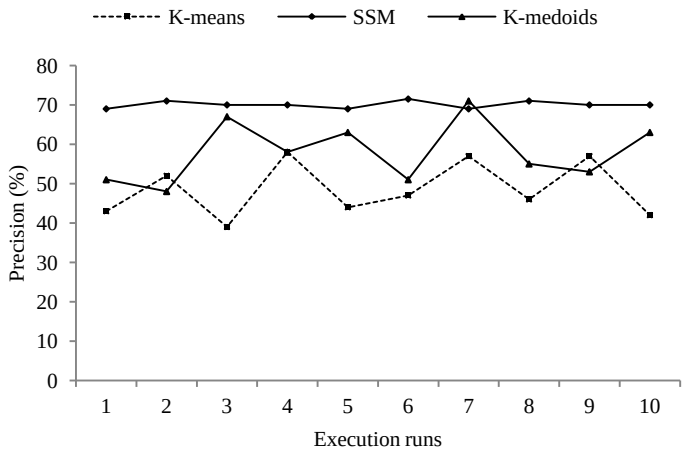


Fig.6. The stability of the clustering algorithms.

SSM involves iterations and samples to find the target centroids, Fig.7 shows the convergence of SSM. This test evaluated the merging speed of SSM in K-means clustering. It shows that during 5-execution runs, the number of clusters can converge to a stable number which can successfully determine the number of sub-clusters.

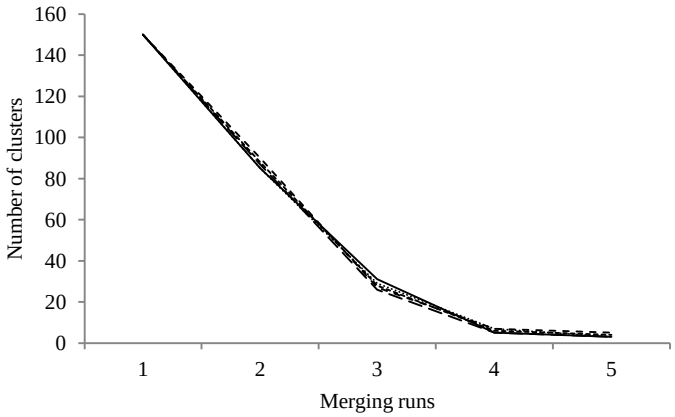


Fig.7: The convergence of SSM.

5.1.3 Computation Efficiency

A number of tests were conducted to evaluate the efficiency in computation, this test conducted a series of figures to show the algorithm performances. Fig.8 shows the increasing overhead with an increasing data size. It also indicates that the overhead of SSM increases with a sampling rate increasing from 1% to 10% whilst *r* was set to 0.79.

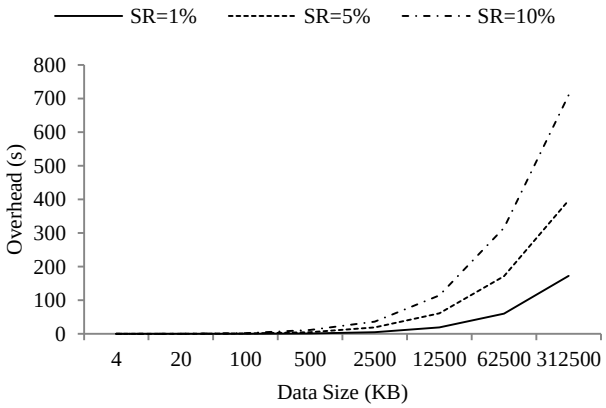


Fig.8. SSM efficiency in computation.

As discussed, theoretically SSM needs to traverse all the points within clusters to measure if two clusters need to be merged, which causes its large computational overhead. However, in the actual execution, when the distance of two points belonging to two clusters is close enough, the two clusters are merged. Therefore the overhead is reduced. From Fig.9 it can be observed that a larger data size contrarily has a lower ratio in decreasing overhead.

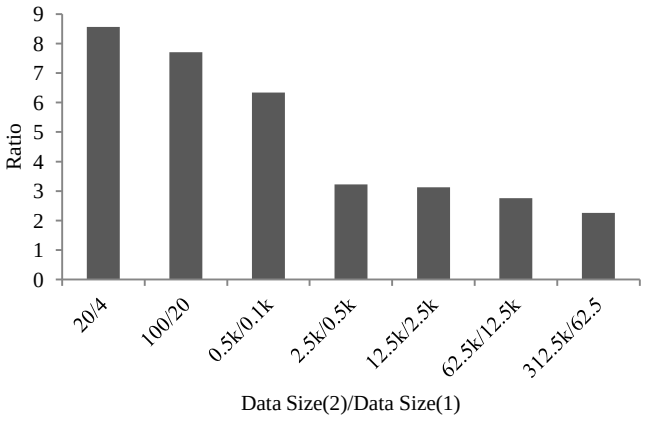


Fig.9. Decreasing ratio.

Fig.10 shows SSM is slower than both standard K-means and k-medoids due to the time consumed in traversing the clusters for merging.

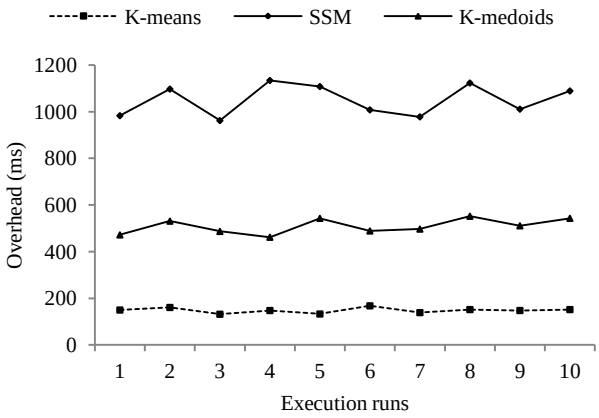


Fig.10. A comparison in computation overhead.

5.2 Evaluating Parallel K-means in CIM Verification

In this section, we evaluate 3 CIM data verification methods. The first one is the MapReduce based parallel K-means verification. The second one is the standalone K-means verification using a single computer. The last one is the standalone CIM verification without clustering and the verification was carried out on a single computer as well. Two CIM dataset were designed. The instances in the first dataset belong to 3 obviously separable clusters. Contrarily the instances in the second dataset belong to 3 clusters but two of them are not obviously separable from each other. The sizes of both datasets were varied from 16MB to 512MB.

Fig.11 shows the efficiency of the 3 CIM data verification methods in dealing with the obviously separable dataset. It indicates that the parallel K-means verification achieves the best performance in computation whereas the CIM verification without clustering is the slowest one in computation. Due to less IO operations, the standalone K-means for CIM verification is faster than the CIM verification without clustering. Fig.11 also shows that the MapReduce based parallel CIM verification cannot achieve the highest efficiency when the data size is less than 128MB. The reason is due to the initialization overhead of the Hadoop framework. The parallel CIM verification starts increasing speed in computation when the size of data gets large.

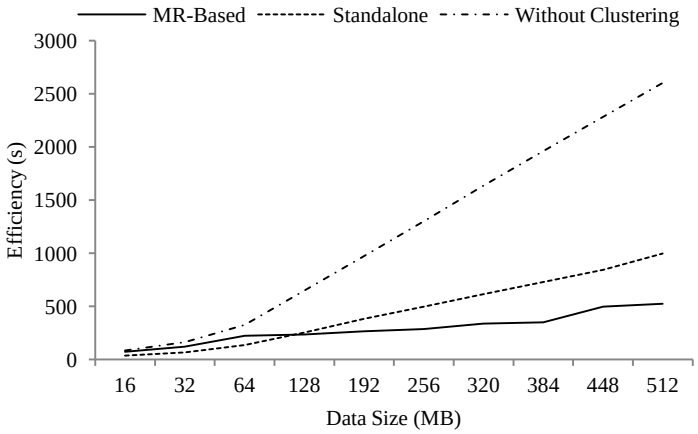


Fig.11. Computation efficiency on the first dataset.

Fig.12 shows the computation efficiency of the 3 algorithms in dealing with the dataset in which 2 clusters are not obviously separable. As expected, the MapReduce based parallel verification performs similarly on different types of data.

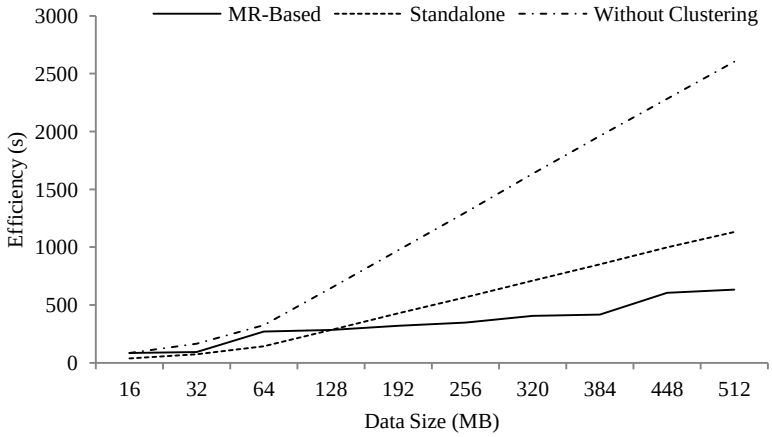


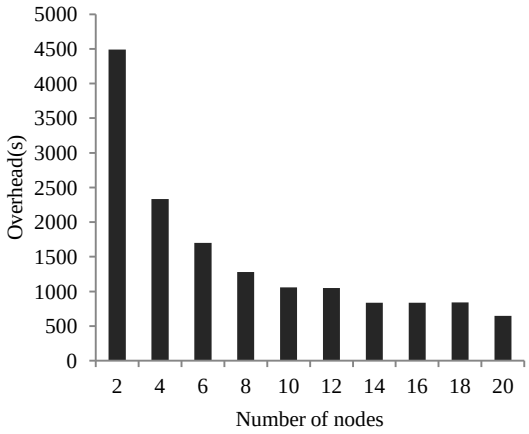
Fig.12. Computation efficiency on the second dataset.

5.3 Simulation Results

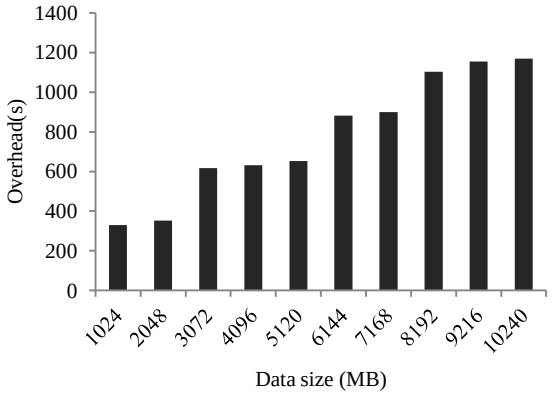
We have developed HSim [9], a Hadoop simulator for simulating large scale Hadoop cluster environments. Him was used to further evaluate the performance of the MapReduce based parallel K-means clustering in CIM data verification from the aspects of scalability and load balancing.

5.3.1 Scalability

In order to evaluate scalability, two sets of simulations were designed for homogeneous Hadoop environments. In the first set of simulations, the data size was set to 5120MB while the number of Hadoop nodes was varied. In the second set of simulations, the number of Hadoop nodes was set to 20 with a varied size of data. In the simulations, each node was configured with 2 mappers and 1 reducer. The simulation results are shown in Fig.13a and Fig.13b.



a. The impact of the number of nodes.



b. The impact of data size.

Fig.13. Algorithm scalability.

Fig.13a clearly indicates that the algorithm gains better efficiency with an increasing number of Hadoop nodes. It is worth noting that when the number of nodes increases to a certain value, the efficiency cannot be improved greatly. The reason is that when the number of nodes is small, a small number of mappers are assigned which causes a large number of processing waves. Contrarily, when the number of nodes is large, a large number of mappers can be assigned which results in a small number of processing waves. For example, consider using 2 nodes with 4 mappers (each mapper deals with 64MB data once) to process 512MB data. Theoretically $\frac{512}{64\times4} = 2$ waves are needed. However, if 4 nodes with 8 mappers are offered, the wave number equals $\frac{512}{64\times8} = 1$. Moreover, if 6 nodes with 12 mappers are used to process 512MB data, due to 8 out of 12 mappers are enough for dealing with the data so that still one wave is needed whilst 4 mappers are kept running idle. Fig.13b shows the computation overhead increases linearly with an increasing size of data.

5.3.2 Load Balancing

To evaluate the load balancing algorithm, a cluster with 20 nodes was simulated. Each node had one processor with two cores, simultaneously the number of *mappers* equals to the number of processor cores. The speeds of the processors were generated based on the heterogeneities of the Hadoop cluster.

Considering the total processing power of the cluster is $P = \sum_{i=1}^{n} p_i$ where n represents the number of the processors employed in the cluster and p_i represents the processing speed of the i^{th} processor. For a Hadoop cluster with a total computing capacity P, the levels of heterogeneity of the Hadoop cluster can be defined using Eq.(12).

$$Heterogeneity = \sqrt{\sum_{i=1}^{k} (\bar{p} - p_i)^2} \qquad (12)$$

The heterogeneity of the first simulation was 0.27. The nodes' details are listed in Table 2.

Three scenarios were simulated including homogeneous environment, heterogeneous environment with load balancing and without load balancing. The simulation results are shown in Fig.14 which indicates that the unbalanced algorithm works the worst. When the data size reaches to 2GB, the executing time sharply increases above 6000 seconds. The reason is that in a simulated Hadoop cluster with a heterogeneity of 0.27, some extremely slow nodes may exist in the cluster. When they are assigned with tasks, these slow nodes may take an extremely long period of time. If the data size is not enough for example 1GB in this case, the slow nodes may not have chance to join the processing thus the job may finish with a high efficiency. This phenomenon greatly impacts the cluster performance.

Table 2 : Heterogeneity specification

Nodes	Simulated speed(MB/s)
Node 1	0.065
Node 2	0.582
Node 3	0.757
Node 4	0.651
Node 5	0.058
Node 6	0.099
Node 7	0.819
Node 8	0.221
Node 9	0.183
Node 10	0.010
Node 11	0.570
Node 12	0.012
Node 13	0.189
Node 14	0.027
Node 15	0.496
Node 16	0.638
Node 17	0.163
Node 18	0.043
Node 19	0.103
Node 20	0.205

Fig.14 also indicates that the parallel K-means works best in the homogeneous environment as the load balancing scheme itself also causes some overhead in computation.

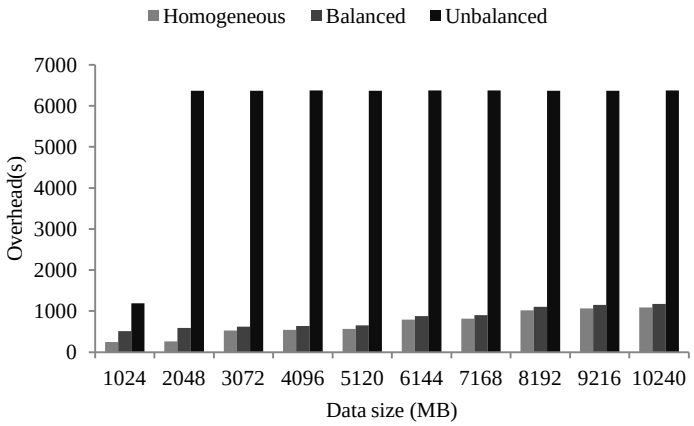


Fig.14. Load balancing among 20 nodes.

Fig.15 shows the convergence of the genetic algorithm. The result turns stable around 500 iterations. Fig.16 reveals the impact of heterogeneity on the performance of the parallel K-means. It can be observed that when the heterogeneity is small, the balanced verification algorithm takes more time because of the overhead of the genetic algorithm. However, the balanced algorithm outperforms the unbalanced one with an increasing value in heterogeneity.

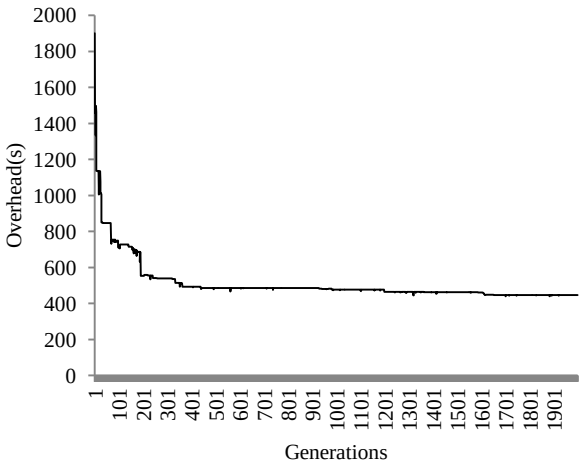


Fig.15. Convergence of genetic algorithm.

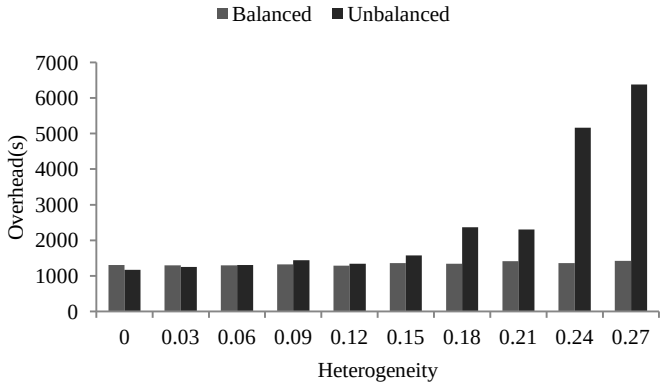


Fig.16. The impact of heterogeneity.

5. Conclusion

In this paper we have presented a MapReduce based parallel K-means for large scale CIM data verifications. By partitioning the CIM data into smaller subsets, the parallel K-means clustering reduces the computation time significantly compared with a standalone sequential K-means clustering method. We introduced a genetic algorithm based load balancing scheme to balance the workloads among MapReduce computing nodes to optimize the performance of the parallel K-means in CIM data verification.

Acknowledgment

The authors would like to appreciate the support from National Science Foundation of China (No. 51437003).

References

[1] EMS-API Work Group of National Power System Control and Communication Committee, “Introduction of the 4th EMS-API interoperability test,” Automation of Electric Power System, vol. 28(16) , pp.1-3, 7, 2004.

[2] Y. Pan and J. Zhou, "Interoperability test based on common information model," Power System Technology, vol. 10, pp. 25-28, 2003.

[3] M. Ding, Z. Zhang and R. Bi, "Distributed Generation System Oriented CIM Extension," Automation of Electric Power Systems, vol.32(20), pp.83-87, 2008.

[4] P. Mao, Y. Li, J. Li, Y. Chen and L. Shuai, "Research on framework of CIM conformance test based on OSI architecture," Advanced Technology of Electrical Engineering and Energy, vol. 32(2),pp. 8-71, 2013.

[5] N. B. Hargreaves, S. M. Pantea, A. Carter and G. A. Taylor, "Foundations of a Metamodel Repository for Use With the IEC Common Information Model," IEEE Transactions on Power Systems, vol. 28, no. 4, pp. 4752-4759, 2013.

[6] H. B. Sun, L. Peng, B. Zhang and W. Wu, "Design of A Hierarchical Network Remodeling System based on IEC61970 for Electrical Power Control Centers in China," in Proceedings of Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, pp. 1-6, 2008.

[7] G. Ravikumar, Y. Pradeep and S. A. Khaparde, "Graphics Model for Power Systems Using Layouts and Relative Coordinates in CIM Framework," IEEE Transactions on Power Systems, vol. 28, No. 4, pp. 3906-3915, 2013.

[8] A. Mercurio, A. D. Giorgio and P. Cioci, "Open-Source Implementation of Monitoring and Controlling Services for EMSSCADA Systems by Means of Web Services— IEC 61850 and IEC 61970 Standards," IEEE Transactions on Power Delivery, vol. 24, No. 3, pp. 1148-1153, 2009.

[9] Y. Liu, M. Li, N. K. Alham and S. Hammoud, "HSim: A MapReduce simulator in enabling Cloud Computing," Future Generation Comp. Syst. vol. 29, no. 1, pp. 300-308, 2013.

[10] Y. Liu, M. Li, S. Hammoud, N. K. Alham and M. Ponraj, "A MapReduce based distributed LSI," in: Proceedings of the 7th International Conference on Fuzzy Systems and Knowledge Discovery, pp. 2978–2982, 2010.

[11] J. Dean, S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, pp.107–113, 2008.

[12] Apache Hadoop, http://hadoop.apache.org (last accessed: 10.01.2014).

[13] J. Venner, Pro Hadoop, Springer, USA, 2009.

[14] B. He, W. Fang, Q. Luo, N. K. Govindaraju and T. Wang, "Mars: a MapReduce framework on graphics processors," in: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT, pp. 260–269, 2008.

[15] K. Taura, K. Kaneda, T. Endo and A. Yonezawa, "Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources," ACM SIGPLAN Notices, vol.38 (10), pp.216–229, 2003.

[16] A. S. Memon, M. S. Memon, P. Wieder and B. Schuller, "CIS: An Information Service based on the Common Information Model," Third IEEE International Conference on e-Science and Grid Computing, pp. 465-472, 2007.

[17] L. Field, S. Andreozzi, "Grid Information System Interoperability The Need For A Common Information Model," Fourth IEEE International Conference on eScience, pp.501-507, 2008.

[18] J. Ranzhe1, "Research Framework on Enterprise Common Information Model oriented Network Cooperation," 2008 International Symposium on Computer Science and Computational Technology, pp. 422-426, 2008.

[19] X. F. Wang, N. N. Schulz and S. Neumann, "CIM Extensions to Electrical Distribution and CIM XML for the IEEE Radial Test Feeders," IEEE Transactions on Power Systems, vol. 18, No. 3, pp. 1021-1028, 2003.

[20] C. Sinz, A. Khosravizadeh, W. Kuchlin and V. Mihajlovski, "Verifying CIM Models of Apache Web-Server Configurations," in Proceedings of the Third International Conference On Quality Software, pp. 290-297 2003.

[21] M. Ding, T. Xie and L. Wang, "Research of Real-Time Database System for Microgrid," 2nd IEEE International Symposium on Power Electronics for Distributed Generation Systems, pp. 708-712, 2010.

[22] N. K. Alham, M. Li and Y. Liu, "Parallelizing multiclass support vector machines for scalable image annotation," Neural Comput & Applic. vol. 24(2), pp. 367-381, 2014.

[23] Y. Liu, M. Li, N. K. Alham, S. Hammoud and M. Ponraj, "Load Balancing in MapReduce Environments for Data Intensive Applications," FSKD 2011, pp. 2675-2678, 2011.

[24] X. Fan, H. Zhang, W. Hao and S. Zhao, "Metadata of Database Based Pre-Process For Watermarking," 4th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), pp.232 - 234, 2011..

[25] DOM4J, http://www.dom4j.org/, last accessed: 11/03/2014

[26] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian and J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," IEEE International Symposium on parallel & distributed processing symposium, pp. 1-9, 2010.

[27] http:// dom4j.sourceforge.net, last accessed: 11/03/2014

[28] Y. Fan, W. Wu, H. Cao and H. Zhu, "LBVP: A load balance algorithm based on Virtual Partition in Hadoop cluster," Cloud Computing Congress (APCloudCC), pp. 37-41, 2012.

[29] K. Fan, D. Zhang, H. Li, and Y. Yang,"An Adaptive Feedback Load Balancing Algorithm in HDFS," 5th International Conference on Intelligent Networking and Collaborative Systems (INCoS), pp. 23-29, 2013.

[30] Y. Wei, X. Yuan and B. Malin, "Scalable and robust key group size estimation for reducer load balancing in MapReduce," IEEE International Conference on Big Data, pp. 156-162, 2013

[31] Y. Liu, M. Li, M. Khan and M. Qi, "A Mapreduce Based Distributed Lsi For Scalable Information Retrieval," Computing and Informatics, vol. 33, no. 2, pp. 259-280, 2014.

[32] The Iris Dataset, https://archive.ics.uci.edu/ml/datasets/Iris

[33] Y. Liu, A Resource Aware Distributed LSI Algorithm for Scalable Information Retrieval, PhD Thesis, Brunel University, UK, 2011.

[34] N. K. Alham, Parallelizing Support Vector Machines for Scalable Image Annotation, PhD Thesis, Brunel University, UK, 2011.

[35] N. K. Alham, M. Li, Yang Liu, S. Hammoud, M. Ponraj, "A distributed SVM for scalable image annotation", Proceedings of the 8th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 2655-2658, 2011.