Journal of Computer Science 2012, 8 (12), 2083-2097
ISSN 1549-3636

doi:10.3844/jcssp.2012.2083.2097 Published Online 8 (12) 2012 (http://www.thescipub.com/jcs.toc)

Matching LSI for Scalable Information Retrieval

[1]Rajagopal Palsonkennedy and [2]T.V. Gopal

[1]Department of Information Technology, PBCE,
Faculty of I and C, Anna University, India
[2]Department of CSE, Faculty of I and C,CEG, Anna University, Chennai-25, India

Received 2012-06-09, Revised 2012-08-28; Accepted 2012-08-28

ABSTRACT

Latent Semantic Indexing (LSI) is one of the well-liked techniques in the information retrieval fields. Different from the traditional information retrieval techniques, LSI is not based on the keyword matching simply. It uses statistics and algebraic computations. Based on Singular Value Decomposition (SVD), the higher dimensional matrix is converted to a lower dimensional approximate matrix, of which the noises could be filtered. And also the issues of synonymy and polysemy in the traditional techniques can be prevail over based on the investigations of the terms related with the documents. However, it is notable that LSI suffers a scalability issue due to the computing complexity of SVD. This study presents a distributed LSI algorithm MR-LSI which can solve the scalability issue using Hadoop framework based on the distributed computing model Map Reduce. It also solves the overhead issue caused by the involved clustering algorithm by k-means algorithm. The evaluations indicate that MR-LSI can gain noteworthy improvement compared to the other scheme on processing large scale of documents. One significant advantage of Hadoop is that it supports various computing environments so that the issue of unbalanced load among nodes is highlighted.Hence, a load balancing algorithm based on genetic algorithm for balancing load in static environment is proposed. The results show that it can advance the performance of a cluster according to different levels.

Keywords: SVD, K-Means Cluster, T-Dmatrix, Mapreduce, LSI, Information Retrieval

1. INTRODUCTION

Latent Semantic Indexing (LSI) has been broadly used in information retrieval due to its success in solving the problems of polysemy and synonymy. However, three negative aspects affect the performance of LSI. The first disadvantage is that LSI is notably a computationally rigorous method because of the computing complexities of singular value decomposition and filtering operations involved in the process. The second disadvantage is several studies show that the truncated SVD (Zha and Zhang 2000) can be lack of competence in processing large in identical text collections. The third disadvantage is for large datasets the SVD computation may be too expensive to be carried out on conventional computers. Also, the dense data structure of the truncated SVD (Zhang and Zha, 2001) matrices poses a huge challenge for both disk and memory spaces of conventional computers. One of the clustering algorithm k-means has been involved by Combining with k-means, the original dataset of documents can be clustered into several sub-clusters according to the similarities of topics of the documents. As a result, the dimension of the original T-D matrix formed from the inhomogeneous text collections is reduced. Also, the computing complexity and cost are reduced. However, it should be noted that the combined clustering algorithm k-means can also generate large overhead when it is dealing with large dataset. Thus to distribute the k-means combining with LSI is an efficient way to solve the above issue.This study presents a MapReduce based Distributed LSI algorithm (MR-DLSI) for high performance and scalable information retrieval (Bassu and Behrens, 2003; Dumais, 1995). MR-DLSI distributes k-means using Hadoop framework based on MapReduce computing model. Each mapper processes a

Corresponding Author: Rajagopal Palsonkennedy, Department of Information Technology, PBCE, Faculty of I and C, Anna University, India

data chunk which is separated from the original dataset by running k-means algorithm. After the dataset is clustered, a number of sub-clusters are output by reducer. And then, a number of mappers are started to do truncated SVD computation in each sub-cluster (Husbands *et al*., 2001). Finally, reducer outputs the final results into HDFS.

The performance of MR-DLSI is first evaluated in a small scale experimental environment. By partitioning the dataset into smaller subsets and optimizing the partitioned subsets across a cluster of computing nodes, the overhead of the MR-DLSI algorithm is reduced significantly while maintaining a high level of accuracy in retrieving documents of user interest. A genetic algorithm based load balancing scheme is also designed to optimize the performance of MR-DLSI in heterogeneous computing environments in which the computing nodes have varied resources.

1.1. Problem Statements

LSI suffers from scalability problems especially in processing massive document collections due to SVD which is considered to be computationally intensive.

Therefore, several techniques have been proposed to enhance the performance of LSI. Gao and Zhang (2003); Bassu and Behrens (2003) combined the clustering algorithm k-means (Steinbach *et al*., 2000) and LSI to reduce the overhead (large executing time consumed) of typical LSI. These approaches show enhancement in performances however the overhead of k-means with large document collection are not considered. An alternative approach is to distribute the computation of LSI among nodes in a cluster environment using the Message Passing Interface (MPI). Seshadri and Iyer (2010) proposed a parallel SVD clustering algorithm using MPI. Documents are split into a number of subsets. Each subset of the documents is clustered by a participating node in the cluster.

The MPI approaches mainly target on homogeneous computing environments with limited support for fault tolerance and incur large inter-node communication overhead when shipping large date across the cluster. Currently heterogeneous computing environments are increasingly being used as platforms for resource intensive distributed applications. One major challenge in using a heterogeneous environment is to balance the computation loads across a cluster of participating computer nodes.

1.2. The Design and Implementation of MR-DLSI

MR-DLSI employs k-means to group documents into a number of clusters of documents. To minimize the overhead of k-means in clustering documents, MR-LSI partitions the set of documents into a number of subsets of documents and distributes these subsets of documents among a number of processors in a MapReduce Hadoop environment. Each processor only clusters a portion of the documents and subsequently performs a truncated SVD operation on the generated document cluster.The details on the design of MR-LSI are given below:

$$\overline{M} = U_t \sum_t V_t^*$$

Let U represent the set of documents,
V represent the set of processors in a Hadoop cluster,

Each processor runs one map instance called mapper.Mm. Ms represent the set of mappers running in the Hadoop cluster.

In DLSI, the set of documents can be represented by a set of vectors denoted by, Vd

Each vector represents the frequencies of keywords that appear in document. The input of each mapper includes two parts. The first part is a centroid set of with initial centroids which are randomly selected from the vector set,. The second part of the input of a mapper is a portion of denoted by. The vector set is equally divided into portions according to the number of mappers. Each mapper runs on one processor calculating the Euclid distances between and which is denoted by, then:

$$d(p,q) = d(q,p)$$
$$= \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Let d represent the shortest distance between objects and, then.

Based on the shortest distance, the mapper selects the corresponding and to generate a key-value pair as one output record. The output pairs of all the mappers are fed into the reduce instance (called reducer). The reducer groups the values with the same key into a set of clusters denoted by C, where and for each the reducer calculates a new centroid denoted by X0=∑xi.

The reducer outputs a set of centroids denoted by x1, which will be fed into the mappers for computing another set of centroids until the values of the centroids in set are the same as those in then the reducer outputs the each of the jobs runs a mapper performing a truncated SVD operation in. In each, the vectors form a T-D matrix:

$$v_d = \left[w_1, d, w_2 d, ..., w_N, d \right]^T$$

Where:

$$w_{t,d} = tf_{t,d} . \log \frac{|D|}{|\{d' \in D\}| t \in d'|}$$

After performing a truncated SVD operation, the matrix can be represented by an approximate matrix, where, k is the rank of the matrix.

In LSI, for a submitted query, it is processed using Equation 1:

$$\begin{aligned} d_j &= \left(w_{1,j}, w_{2,j}, ..., w_{t,j} \right) \\ q &= \left(w_{1,q}, w_{2,q}, ..., w_{t,q} \right) \end{aligned} \quad (1)$$

The similarities of the query to the documents can be measured by calculating the cosine values of vector and the vectors of matrix using Equation 2:

$$\cos\theta = \frac{d_2 . q}{\|d_2\| \, \|q\|} \quad (2)$$

where, j represents the jth document in the clustered document set.

If the value of is larger than a given threshold, then the document will be a target document. Therefore the set of target documents can be represented as.

Finally, the reducer generates clusters of documents. For each cluster of documents, a truncated SVD operation is performed and targeted documents are retrieved.

1.3. Static Load Balancing Strategy for MR-LSI

A remarkable characteristic of the MapReduce Hadoop framework is its support for heterogeneous computing environments. Therefore computing nodes with varied processing capabilities can be utilized to run MapReduce applications in parallel. However, current implementation of Hadoop only employs First-In-First-Out (FIFO) and fair scheduling without support for load balancing taking into consideration the varied resources of computers. A genetic algorithm based load balancing scheme is designed to optimize the performance of MR-LSI in heterogeneous computing environments.

1.4. Algorithm Design

To solve an optimization problem, genetic algorithm solutions need to be represented as chromosomes encoded as a set of strings which are normally binary strings. However, a binary representation is not feasible as the number of mappers in a Hadoop cluster environment is normally large which will result in long binary strings. A decimal string to represent a chromosome in which the data chunk assigned to a mapper is represented as a gene is employed.

In Hadoop, the total time (Tt) of a mapper in processing a data chunk consists of the following four parts.

Data copying time (tc) 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 (tr) in processing a data chunk.

Intermediate data merging time (tm) in combining the output files of the mapper into one file for reduce operations. Buffer spilling time (tb) in emptying filled buffers using Equation 3:

$$Tb = Tc + Tr \quad (3)$$

Let 10MB be the size of the data chunk.

Ws1ms/1MB	=	The writing speed of hard disk in MB/second.
Nb	=	The network bandwidth in MB/second.
Ps	=	The speed of the processor running the mapper process in MB/second.
Bs	=	The size of the buffer of the mapper.
Ps/bs	=	The ratio of the size of the intermediate data to the size of the data chunk.
F1	=	The number of frequencies in processing intermediate data.
bBn	=	The number of times that buffer is filled up.
Vd	=	The volume of data processed by the processor when the buffer is filled up.
s be	=	The sort factor of Hadoop.

Therefore:

$$Vn = Ps/Bs(f1.bn) \quad (4)$$

Here depends on the available resources of hard disk and network bandwidth using Equation 4. 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:using Equation 5:

$$Map\ (c1,v1) \rightarrow list(c2,v2) \quad (5)$$

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. Thus the time to fill up a buffer can be computed by. 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 Equation 6:

$$V_b P_s - R_a \frac{Bf}{P_s - Hd} \quad (6)$$

The total amount of intermediate data generated from the original data chunk with a size of is D_m. Therefore the number of times for a buffer to be filled up can be computed using Equation 7:

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

The time for a buffer to be spilled once is, therefore the time for a buffer to be spilled for times is using Equation 8. Then we have:

$$t_b = N_b + Rf \int_{Hd} \quad (8)$$

The frequencies in processing intermediate data can be computed using Equation 9:

$$N_f = [q_b \mid s] - 1 \quad (9)$$

When the merging occurs once, the whole volume of intermediate data will be written into the hard disk causing an overhead of thus if the merging occurs times, the time consumed by hard disk IO operations can be represented by Tm:(given in Equation 10.):

$$t_m = \frac{D_m \; Ra \; N_f}{Hd} \quad (10)$$

The total time to process data chunks in one processing wave in MapReduce. Hadoop is the maximum time consumed by participating mappers, where:

$$T_{total} = \max(T_1, T_2 \ldots T_k) \ldots \quad (11)$$

According to divisible load theory (Othman, 2010; Robertazzi, 2003; Thysebaert *et al*., 2005) to achieve a minimum, it is expected that all the mappers to complete data processing at the same time:

$$T_1 = T_2 = \ldots T_k \quad (12)$$

Let Tp be the processing time for the mapper.

$\overline{T}$ be the average time of the mappers in data processing, $\overline{T} = \sum^k T_p / k$.

Based on Equation 11 and 12, the fitness function is to measure the distance between and gene. Therefore, the fitness function can be defined using Eq. 13 which is used by the genetic algorithm (Guo *et al*., 2010) in finding an optimal or a near optimal solution in determining the size for a data chunk:

$$f(t) = \sqrt{\sum_{i=1}^{k} (\overline{T} \_ T_p)^2} \quad (13)$$

1.5. Crossover

To maintain the diversity of the chromosomes, the algorithm needs functions of crossover. Crossover recomposes the homologous chromosomes via mating to generate new chromosomes or so called offspring. The generated offspring inherit the basic characteristics of their parents. Some of them may adapt to the fitness function better than their parents did, so they may be chosen as parents in next generation. Based on crossover, the algorithm can keep evolving until an optimal offspring has been found. In this algorithm, to gain the effective of design and operations, single-point crossover which refers to set only one crossover point randomly in the chromosome has been employed. The processes of crossover could be regarded as:

Randomly select pairs of the chromosomes (schedulers) as parents to mate. 2. In each pair, randomly select a position as crossover point. If the length of the chromosome is then there will be available points. k1k3. In each pair, the chromosomes change their parts which are after the crossover point with each other according to crossover probability. p

However in the algorithm simply crossing the chromosome may cause one problem.

As each gene is the value of the actual volume of data each Map instance takes, to change the members of genes may differentiate the original total volume of data. Assume the original total volume of data is and the volume of data after crossover is, then the difference should be considered and processed. In the algorithm is divided into parts. The size of each part is randomly assigned. And then these parts will be randomly added to or removed by ΔD:

$$\Delta D = \left| \sum_{1-1}^{k} D_i - \sum_{1-1}^{k} d_i \right|$$

From genes in the chromosome. Thus the total size of processed data in one wave could be guaranteed.

1.6. Mutation

To avoid the local optimum of the algorithm, mutation has been introduced into our algorithm. Mutation could mutate genes in a chromosome based on smaller probabilities. Moreover new individuals could be

generated. So that combined with crossover the information loss due to the selection could be avoided. Thus the validity of the algorithm could be guaranteed. The mutation contributes in two main aspects in our algorithm.

Improving the local search ability of the algorithm. The crossover operation could find a number of chromosomes with better adaptability from a global angle. These chromosomes are close to or helpful to gain the optimal solution. However crossover cannot execute local search in details. So using mutation to tune the values of certain genes from local detailed phase could make the chromosome much closer to the optimal solution. So the search ability is enhanced compare to that of only crossover involved.

Maintaining the diversity of the colony moreover preventing the premature convergence of the algorithm. Mutation replaces the original genes with newly mutated genes so that the structure of a chromosome could be significantly affected. The diversity of the colony could be maintained.

The algorithm mutates genes mainly based on simple mutation which refers that to mutate one or several genes in the chromosome based on mutation probability. There are two steps in the simple mutation.

Randomly select a gene to be the mutation point. Base on mutation probability to decide if the chromosome mutates.

If the probability decides the gene should mutate, then the value of the gene will be mutated which means a new value replaces the original value. As a result a new individual is generated.

However, it is quite similar to crossover processes that when the value of one gene mutates, the original total volume of data has been changed. Assume the original volume of the gene is and the volume after mutation is, then the difference. To solve issue, is divided into parts. The size of each part is randomly assigned. And then these parts will be randomly added to or removed from genes in the chromosome. Thus the total size of processed data in one wave could be guaranteed. Based on this design, the algorithm has a strong ability to change its searching direction to gain the optimal solution in a large search space.

iPip
iiPPpP

1.7. Experimental Results

To evaluate the performances of MR-LSI a small scale Hadoop cluster consisting four computer nodes has been set up. **Table 1** shows the configurations of the Hadoop cluster.

Table 1. The experimental environment

Number of Hadoop nodes:	4
Nodes' specifications:	Three Datanode: CPU Q6600@2.5G, RAM 3GB and running OS Fedora11. One namenode: CPU C2D7750@2.26,RAM2GB and running OS Fedora 12.
Number of mappers per node:	2
Number of reducer:	1
Network banwidth:	1000Gbps

To evaluate the performances of MR-LSI, 1000 papers were collected from the IEEE XPlore data source. For each paper selected, a T-D matrix will be constructed. In the tests, also two strategies Closest Distance Searching (CDS) and All Distances Searching (ADS) for clustering documents which are similar to the clustered strategies proposed in have been designed.

Processed by k-means, the original dataset is clustered into a number of sub-clusters. Within these sub-clusters, one or a few of them may be close to the query while the others are far away from the query. CDS calculates the distances between a query and the centroid of each sub-cluster. The closest sub-cluster to the query will have the highest probability in containing the target documents. A truncated SVD will only be performed on the closest sub-cluster. As CDS just retrieves information in one cluster, the time consumed for executing CDS is least. ADS calculates the distance between a query and the centroid of each sub-cluster and a truncated SVD will be performed on all the sub-clusters. As ADS retrieves information in all sub-clusters, the misclassified documents may have chance to be retrieved.

1.8. Evaluating MR-DLSI

MR-LSI was evaluated from the aspects of precision and recall in comparison with standalone LSI, standalone LSI combined with k-means using the CDS strategy and standalone LSI combined with k-means using the ADS strategy. From the results presented in **Fig. 1** and 2 it can be observed that the performance of MR-LSI is close to that of the standalone LSI. It is worth pointing out that the CDS strategy only works on the closest sub-cluster of documents related to a query. Compared with other algorithms, CDS retrieves a smaller number of documents which resulting in lower performance in recall.

There are a number of tests have been conducted to evaluate the overhead of MR-LSI in computation. The number of documents to be retrieved varied from 100-1000.

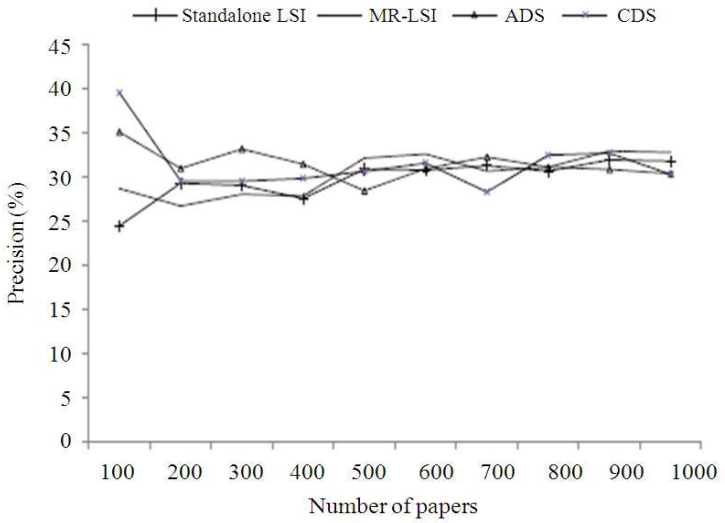

Fig. 1. The precision of MR-LSI

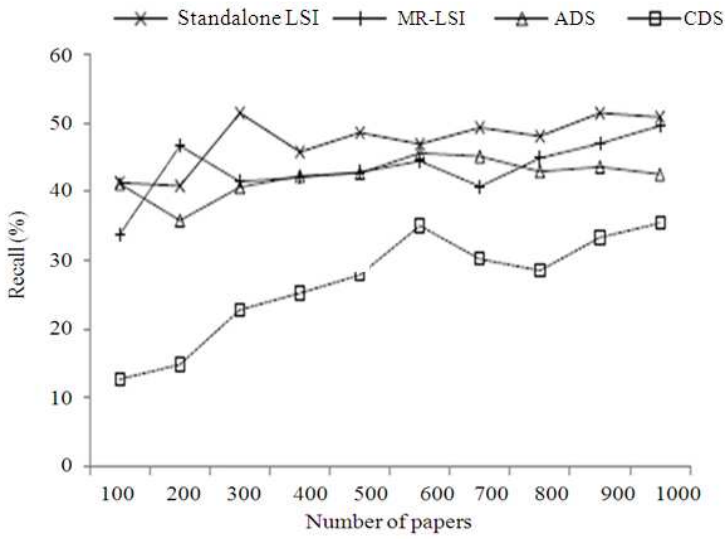

Fig. 2. The recall of MR-LSI

However, the size of the dataset was not large. From **Fig.** 3 and 1 it can be seen that MR-LSI consumed more time than other algorithms in processing the Number of papers dataset. This is mainly due to the overhead generated by the Hadoop framework which is effective in processing large scale data. Both the ADS and the CDS strategies perform faster than the standalone LSI indicating the effectiveness of a combination of LSI with k-means.

And also a number of additional tests have been as well conducted to further evaluate CDS and MR-LSI the overhead of MR-LSI in processing a large collection of documents.

The size of the document collection is increased from 5KB to 20MB and the overhead of MR-LSI with that of the CDS strategy is compared as CDS is faster than both the standalone LSI and the ADS strategy. From the results plotted in **Fig. 5** it can observed that when the data size is less than 1.25MB, the overhead of CDS is stable.

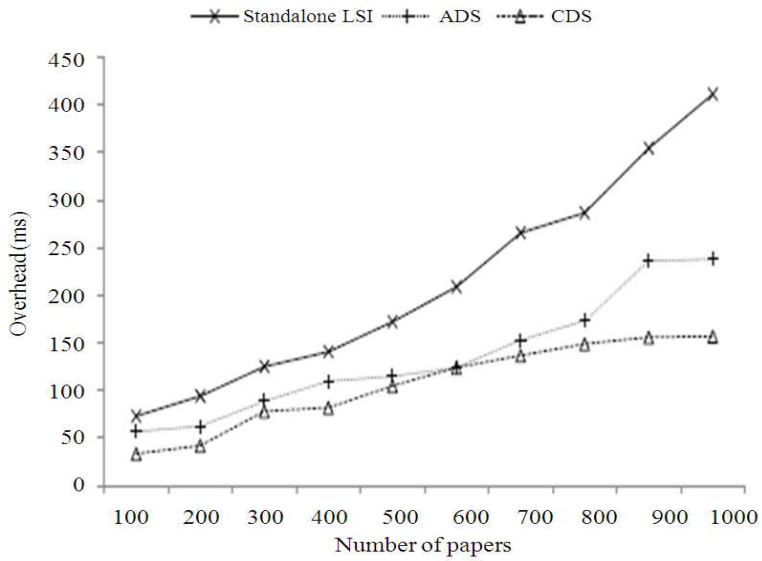

Fig. 3. The overhead of standalone LSI, ADS and CDS in computation

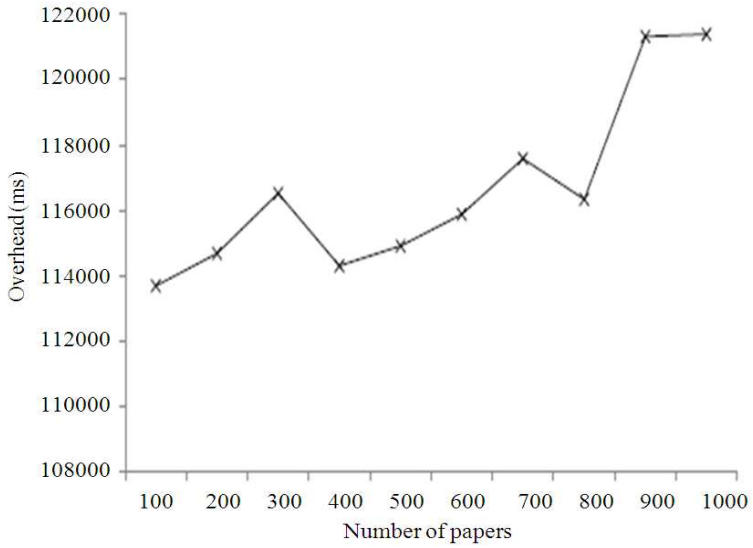

Fig. 4. The overhead of MR-LSI

However, the overhead of CDS starts growing when the size of dataset is larger than 2.5MB. When the size of data reaches to 10MB, the overhead of CDS increases sharply as shown in **Fig. 4**. Compared with CDS, the overhead of MR-LSI is highly stable with an increasing size of dataset shows its better scalability than the CDS strategy. It also should be mentioned that when the size of data increases higher than 20MB, the heap space exception occurs when CDS processes data due to the memory limitation of applications in a standalone node.

1.9. MR-DLSI Simulation Results

To further evaluate the effectiveness of MR-LSI in large scale MapReduce environments, HSim has been developed using pure JAVA programming language. This chapter accesses the performance of the MR-LSI in simulation environments.

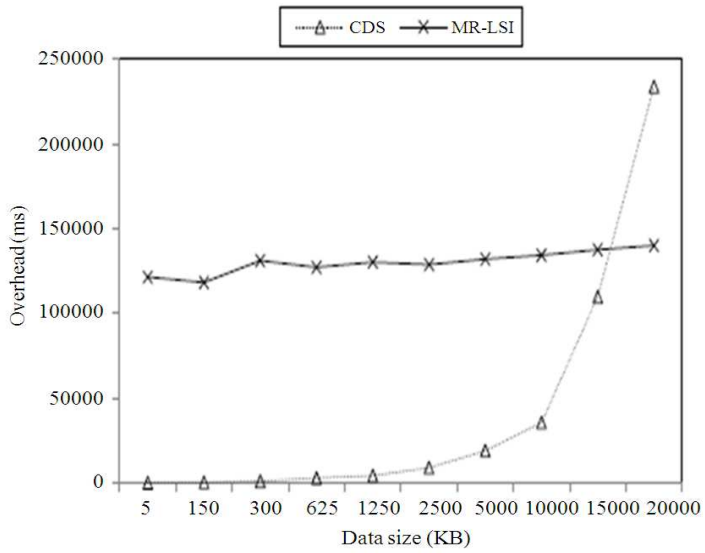

Fig. 5. Comparing the overhead of MR-LSI with CDS

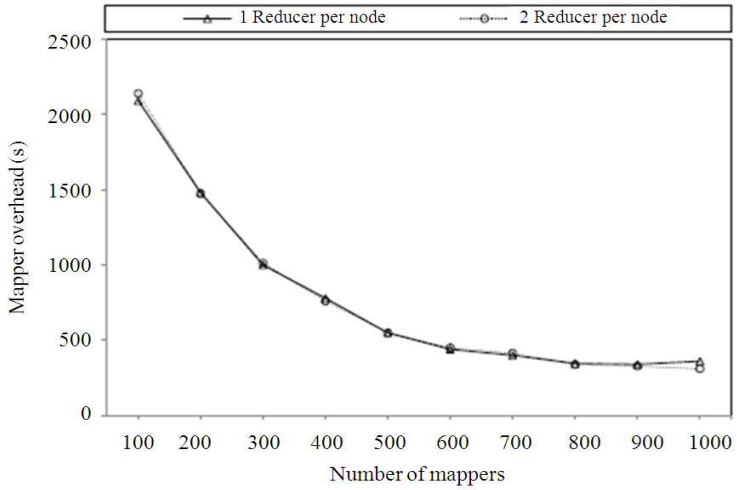

Fig. 6. The impact of the number of reducers on mapper performance

1.10. Simulation Results

To study the impacts of Hadoop parameters on performance of MR-LSI, a cluster has been simulated with the configurations as shown in Table 2. Each node has a processor with 1 cores. The number of mappers is equal to the number of processor cores. There are two mappers running on a single processor with two cores. The speeds of the processors were simulated in terms of the volume of data in MB processed per second. In the following sections, the impacts have been shown of a number of Hadoop parameters on the performance of MR-LSI.

1.11. Multiple Reducers in one Node

From **Fig. 6** it shows that the number of reducers does not affect the performance of mappers greatly. This is because mappers and reducers work almost independently in Hadoop environments. **Figure 7** shows the impact of the number of reducers on the overall overhead when processing a job. Allocating multiple reducers on one node increases results in the shared resources issue.

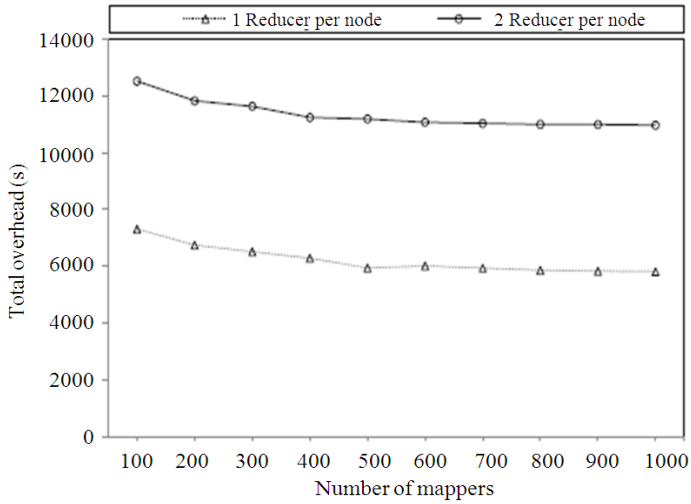

Fig. 7. The impact of the number of reducers on the total process

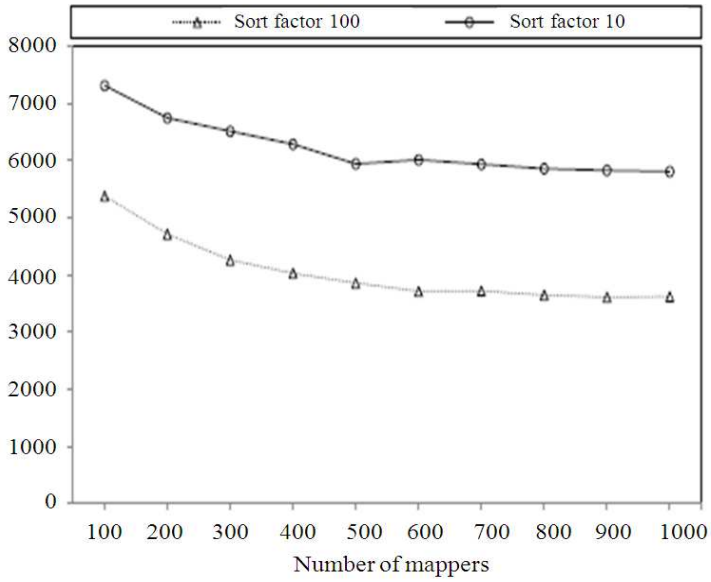

Fig. 8. The impact of sort factor

Table 2. The simulated environment

No. of simulated nodes	250
Data size:	100,000MB
CPU processing speed:	Up to 0.65MB/s
Hard drive reading speed:	80MB/s
Hard drive writing speed:	40MB/s
Memory reading speed:	6000MB/s
Memory writing speed:	5000MB/s
Network bandwidth:	1Gbps
Number of mappers:	4per node
Number of reducer:	1 or more

1.12. Sort Factor

In Hadoop, The parameter of sort factor controls the maximum number of data streams to be merged in one wave when sorting files. Therefore, the value of sort factor affects the IO performance of MR-LSI. From **Fig. 8** it can be observed that the case of using sort factor 100 gives a better performance than sort factor 10.

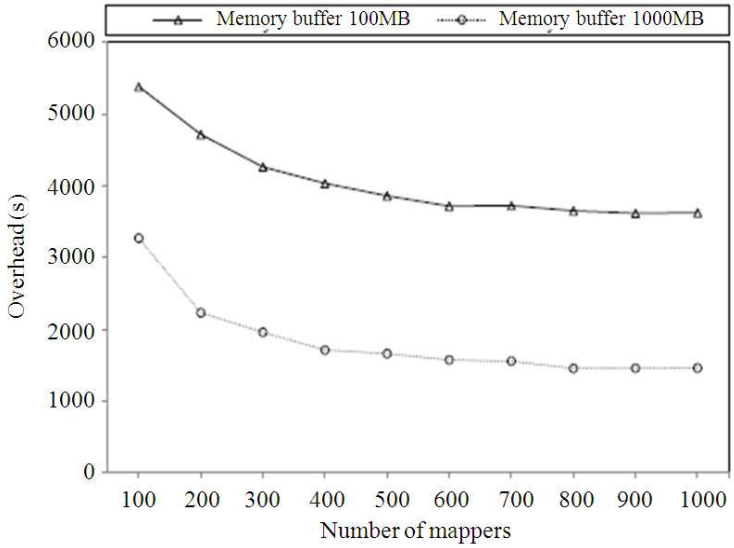

Fig. 9. The impact of buffer size

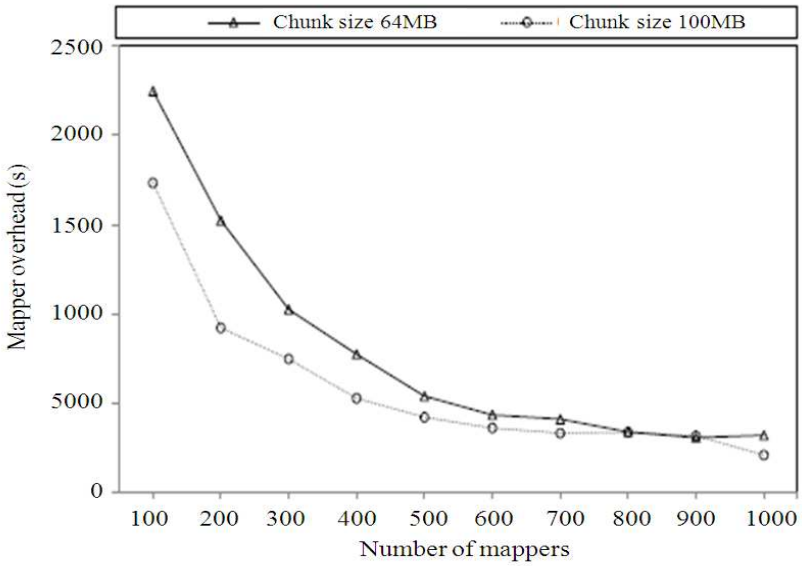

Fig. 10. The impact of data chunk size on the mappers in MR-LSI

When the value of sort factor is changed from 10 to 100, the number of spilled files will be increased which reduces the overhead in merging.

1.13. Buffer Size

The buffer size in Hadoop contributes to IO performance and it affects the performance of a processor. The default value of a buffer size is 100MB. The performance of MR-LSI with a data size of 1000MB is tested. As shown in **Fig. 9**, the mappers generate a small number of spilled files when using a large size buffer which reduces the overhead in merging. Furthermore, a large buffer size can keep the processor working without any blocking for a long period of time.

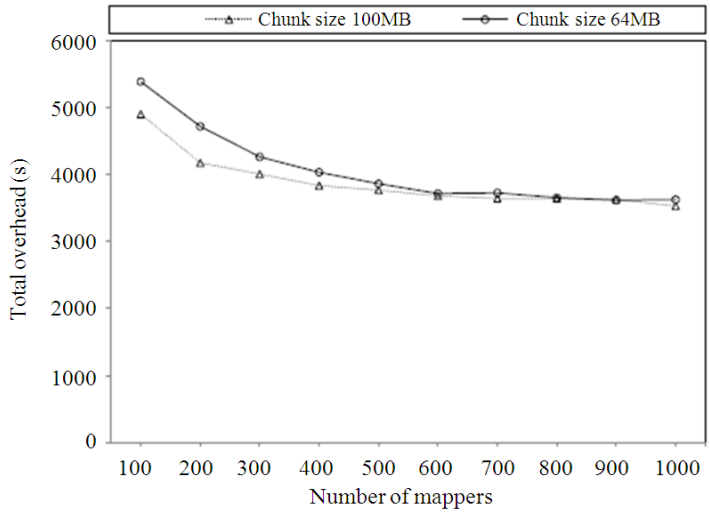

Fig. 11. The impact of data chunk size on MR-LSI

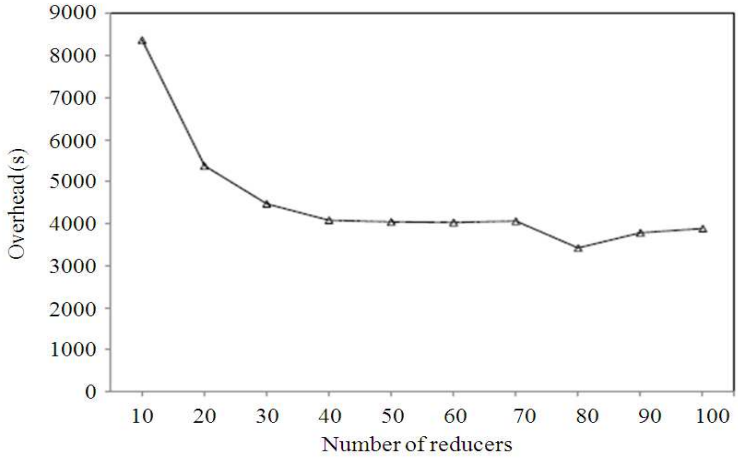

Fig. 12. The impact of different CPU processing speeds

Each mapper processes a data chunk at a time. Thus the size of data chunks highly affects the number of processing waves of mappers. From **Fig. 10** it can be observed that using a large size for data chunks reduces the overhead of mappers in processing and also reduces the total overhead of the process as shown in **Fig. 11**. However, both of the two chunk sizes produce the same performance when the number of mappers increases to 800 and 900 respectively. In the case of chunk size 61MB, to process 100,000MB data, using 800 mappers needs waves to finish the job. In the case of chunk size 100MB, using 800 mappers needs waves to finish the job. Similarly, using 900 mappers needs 2 waves to process the 100,000MB data in both cases. When the number of mappers reaches 1000, the performance of the two cases with different data sizes varies.

1.14. CPU Processing Speed

Figure 12 shows the impacts caused by different processing speed of processors. From the figure we can observe clearly that a faster processor can gain better performance compared to that of a slower processor.

1.15. Number of Reducers

Figure 13 shows that increasing the number of reducers enhances the performance of MR-LSI when the number of reducers is small.

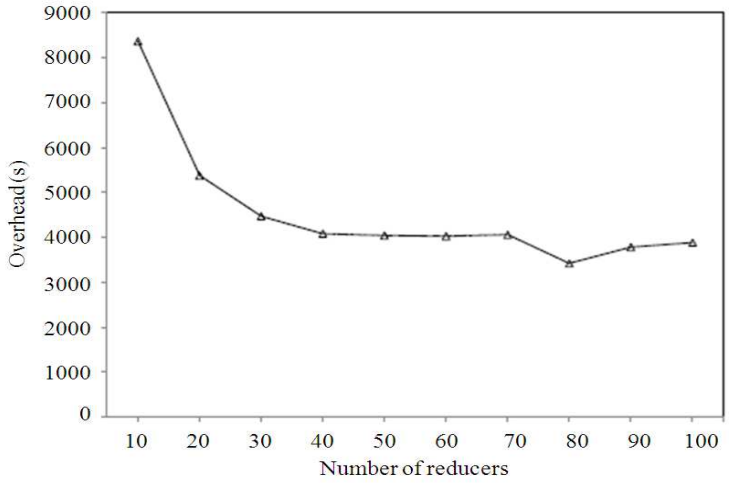

Fig. 13. The impact of reducers

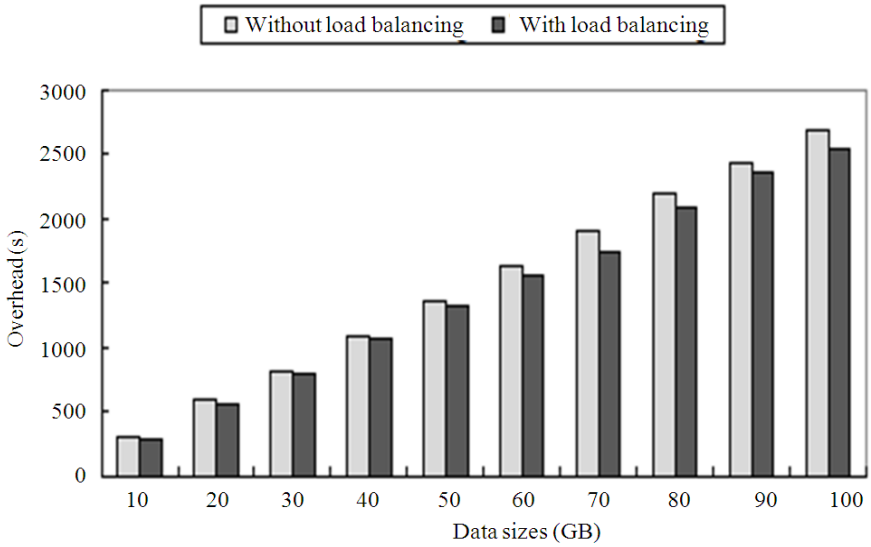

Fig. 14. The performance of the MR-LSI with difference sizes of data

More reducers are used more resources will need to be consumed due to Hadoop's management work on the reducers. In some cases multiple reducers need an additional job to collect and merge the results of each reducer to form a final result. This can also cause larger overhead.

1.16. Load Balancing Simulation Results

Table 3 shows the configurations of the simulated Hadoop environments in evaluating the effectiveness of the load balancing scheme of MR-LSI.

To evaluate the load balancing algorithm, a cluster with 20 computers has been simulated. Each computer has one processor with two cores. The number of mappers is equals to the number of processor cores. Therefore two mappers are running on a single processor with two cores. The speeds of the processors are generated based on the heterogeneities of the Hadoop cluster. In the simulation environments the total processing power of the cluster was where n represents the number of the processors employed in the cluster and represents the processing speed of the ith processor (Steinbach *et al*., 2000). For a Hadoop cluster with a total computing capacity, the levels of heterogeneity of the Hadoop cluster can be defined using Equation 11.

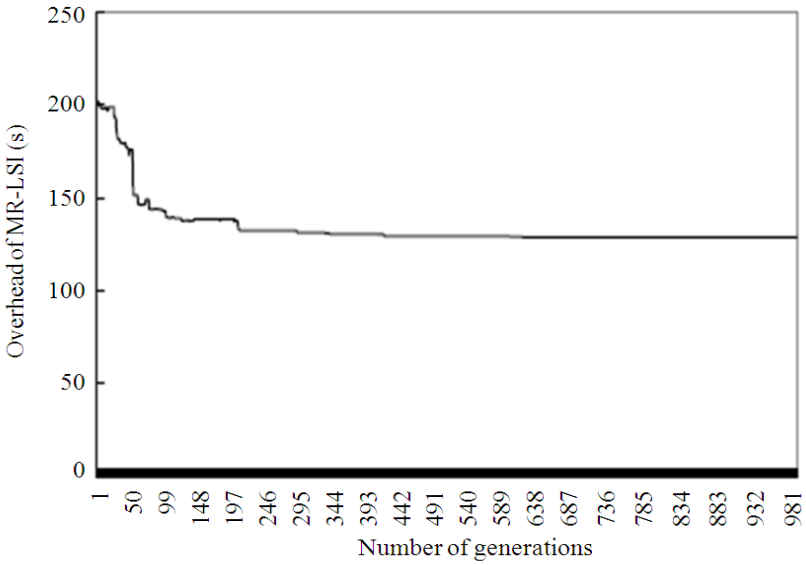

Fig. 15. The convergence of the load balancing scheme

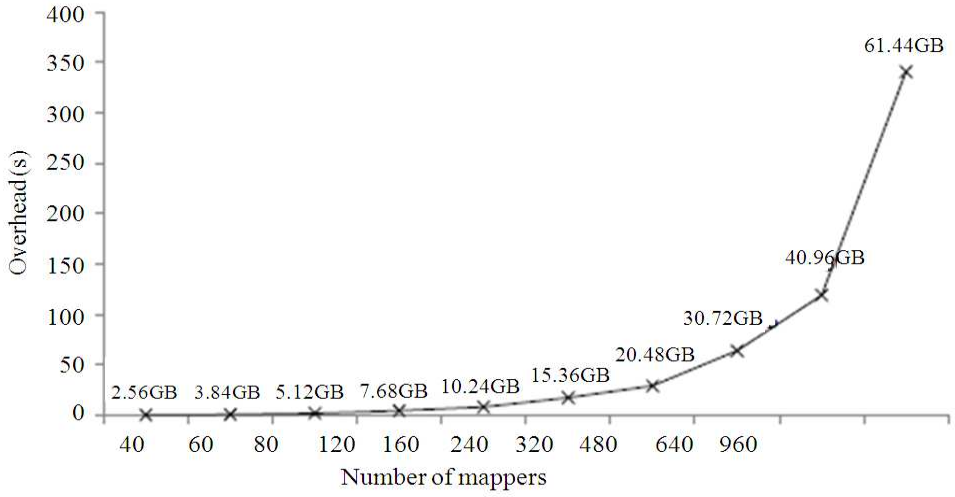

Fig. 16: The overhead of the load balancing scheme with different sizes of data

In the simulation, the value of heterogeneity was in the range of 0 and 2.28. The reading and writing speeds of hard disk were generated based on the real measurements from the experiments conducted.

Firstly 10GB data has been tested in the simulated cluster with different levels of heterogeneity. From **Fig. 11** it can be observed that when the level of heterogeneity is less than 1.08 which indicates a nearly homogeneous environment, the load balancing scheme does not make any difference to the performance of MR-LSI. However, the load balancing scheme reduces the overhead of MR-LSI significantly with an increasing level of heterogeneity.

The levels of heterogeneity are keeping the same in the tests but varied the size of data from 1GB to 10GB. This set of tests was used to evaluate how the load balancing scheme performs with different sizes of datasets. **Figure 14** shows that the load balancing scheme can always reduce the overhead of MR-LSI.

The load balancing scheme builds on a genetic algorithm whose convergence affects the efficiency of MR-LSI.

Table 3. Hadoop simulation configuration

Number of stimulated node	20
Number of processor in each node:	1
Number of cores in each processor:	2
Size of data:	Test 1: 10GB Test 2: 10-100GB
The processing speed of prpcessors:	Depending on heterogeneities
Hetrogeneities:	From 0-2.28
Number of hard diskin each node:	1
Reading speed of hard disk:	80MB/s
Writing speed of hard disk:	40MB/s
Number of Map instances:	Each node contributes 2Map instances.
Number of Reducer instances:	1
Sort factor:	100

To analyze the convergence of the genetic algorithm, the number of generations is varied and the overhead of MR-LSI in processing a 10GB dataset in the simulated Hadoop environment is measured. **Figure 15** shows that MR-LSI reaches a stable performance when the number of generations in the genetic algorithm reaches 300.

The load balancing scheme also produces some overhead during execution.

Figure 16 shows an increased overhead of the load balancing scheme when the number of mappers increases together with an increasing size of data. However the MR-LSI algorithm can still achieve benefit from load balancing algorithm. For example, for heterogeneity 2.08, the overhead of load balancing algorithm is 331s.

The time consumed for one processing wave of mappers is 363s with load balancing. The time consumed for one processing wave of mappers is 2256s without load balancing. Thus the performance is enhanced 69.2%. As in the static computing environment, the scheduler only needs to be computed once, thus it can be claimed that for a long-time processing job with proper heterogeneities, the load balancing algorithm can enhance performances greatly.

2. CONCLUSION

This study presents for scalable information retrieval. MR-LSI is effective when processing a large dataset due to high scalability of MapReduce in support of data intensive applications. Both experimental and simulation results have shown that the MR-LSI algorithm speeds up the computation process of SVD while maintaining a high level of accuracy in information retrieval. The simulating results also indicate that the load balancing strategy can enhance the performance of the Hadoop cluster when it is running a Hadoop application.

3. REFERENCES

Bassu, D. and C. Behrens, 2003. Distributed LSI: Scalable Concept-based information retrieval with high semantic resolution. Proceedings of the 3rd SIAM International Conference on Data Mining, May, 3-3, Telcordia Technologies, Inc., Morristown.

Dumais, S., 1995. Using LSI for Information Filtering: TREC-3 experiments. In: The Third Text Retrieval Conference (TREC3), D. Harman (Ed.), NIST Special Publication, Gaithersburg, ISBN-10: 0788129457, pp: 219-230.

Gao, J. and J. Zhang, 2003. Sparsification strategies in latent semantic indexing. Proceedings of the 2003 Text Mining Workshop, (TMW' 03), CiteSeerX, pp: 93-103.

Guo, P., Wang, X. and Y. Han, 2010. The enhanced genetic algorithms for the optimization design. Proceedings of the 3rd International Conference on Biomedical Engineering and Informatics, Oct. 16-18, IEEE Xplore Press, Yantai, pp: 2990-2994. DOI: 10.1109/BMEI.2010.5639829

Husbands, P., H. Simon and C. Ding, 2001. On the use of the Singular Value Decomposition for Text Retrieval. In: Computational Information Retrieval, Berry, M.W., (Ed.). SIAM, Philadelphia, ISBN-10: 0898715008, pp: 145-156.

Othman, M. (2010). Survey on divisible load theory and its applications. Proceedings of the 2009 International Conference on Information Management and Engineering, Apr. 3-5, IEEE Xplore Press, pp: 300-304. DOI: 10.1109/ICIME.2009.59

Robertazzi, T.G., 2003. Ten reasons to use divisible load theory. Comput., 36: 63-68. DOI: 10.1109/MC.2003.1198238

Seshadri, K. and K.V. Iye, 2010. Parallelization of a dynamic SVD clustering algorithm and its application in information retrieval. Software Practice and Experience, 40: 883-896. DOI: 10.1002/spe.v40:10

Thysebaert, P., M., Volckaert, B., De Turck, F. Dhoedt and P. Demeester, 2005. Using divisible load theory to dimension optical transport networks for grid excess load handling. Proceedings of the Joint International Conference on Networking and Services, Oct. 23-28, IEEE Xplore Press, Papeete, Tahiti, pp: 89-89. DOI: 10.1109/ICAS-ICNS.2005.97

Steinbach, M., G. Karypis and V. Kumar, 2000. A comparison of document Clustering Techniques. University of Minnesota.

Zha, H. and Z. Zhang, 2000. On matrices with low-rank-plus-shift structures: Partial SVD and latent semantic indexing. SIAM J. Matrix Analysis Appli., 21: 522-536.

Zhang, Z. and H. Zha, 2001. Structure and perturbation analysis of truncated svds for column-partitioned matrices. SIAM J. Matrix Anal. Applic., 22: 1245-1262. DOI: 10.1137/S0895479899357875