Computing and Informatics, Vol. 33, 2014, 259–280

A MAPREDUCE BASED DISTRIBUTED LSI FOR SCALABLE INFORMATION RETRIEVAL

Yang Liu

School of Electrical Engineering and Information
Sichuan University, China
e-mail: `yang.liu@scu.edu.cn`

Maozhen Li

School of Engineering and Design, Brunel University
Uxbridge, UB8 3PH, UK
&
The Key Laboratory of Embedded Systems and Service Computing
Tongji University, China
e-mail: `Maozhen.Li@brunel.ac.uk`

Mukhtaj Khan

School of Engineering and Design, Brunel University,
Uxbridge, UB8 3PH, UK
e-mail: `Mukhtaj.Khan@brunel.ac.uk`

Man Qi

Department of Computing, Canterbury Christ Church University
Canterbury, Kent, CT1 1QU, UK
e-mail: `man.qi@canterbury.ac.uk`

Abstract. Latent Semantic Indexing (LSI) has been widely used in information retrieval due to its efficiency in solving the problems of polysemy and synonymy. However, LSI is notably a computationally intensive process because of the computing complexities of singular value decomposition and filtering operations involved in the process. This paper presents MR-LSI, a MapReduce based distributed LSI algorithm for scalable information retrieval. The performance of MR-LSI is first evaluated in a small scale experimental cluster environment, and subsequently evaluated in large scale simulation environments. By partitioning the dataset into smaller subsets and optimizing the partitioned subsets across a cluster of computing nodes, the overhead of the MR-LSI 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 designed to optimize the performance of MR-LSI in heterogeneous computing environments in which the computing nodes have varied resources.

Keywords: Information retrieval, latent semantic indexing, MapReduce, load balancing, genetic algorithms

1 INTRODUCTION

Latent Semantic Indexing (LSI) [6, 7, 8, 11, 22, 24, 29, 36] has been widely used in information retrieval [9, 10, 12]. It is based on the concept that latent structures exist among a number of documents. Building on Vector Space Model (VSM), LSI generates a Term-Document (T-D) matrix. LSI employs a truncated Singular Value Decomposition (SVD) [4, 17] to convert the keywords domain of the original document corpus to a conceptual domain so that the latent semantic relationships among the words and documents can be highlighted and the problems of polysemy and synonymy can be solved. However, it has been widely recognized that LSI suffers from scalability problems in processing massive document collections. The reason is that SVD is computationally intensive due to its computing complexity which can be represented by $O(m \times r^2)$ where m is the number of documents and r is the rank of T-D matrix [13, 26].

A number of approaches have been proposed in speeding up LSI process in computation [3, 14, 16, 19, 21, 25]. However, the scalability of these approaches still remains a challenging issue because of the lack of an effective load balancing scheme in utilization of heterogeneous computing resources.

This paper presents MR-LSI, a distributed LSI for high performance and scalable information retrieval. MR-LSI improves current approaches by focusing on three aspects. First, MR-LSI employs k-means to cluster documents into a number of subsets of documents to reduce the complexity of SVD in computation [18, 20, 37]. Second, MR-LSI builds on MapReduce [2, 5, 23, 33, 35] to distribute the computation of LSI among a number of computers of which each computer only processes a subset of documents. MapReduce has become a major enabling technology in support of data intensive applications. MapReduce has built-in fault tolerance and handles I/O operations effectively which reduces communication overhead significantly. Finally, a resource aware load balancing scheme is designed to optimize the performance of the MapReduce based MR-LSI in heterogeneous computing environments.

The performance of MR-LSI is first evaluated in a small scale experimental MapReduce environment from the aspects of both accuracy and efficiency. Subsequently, a MapReduce simulator is implemented to evaluate the effectiveness of the resource aware MR-LSI algorithm in large scale simulated MapReduce environments. Both experimental and simulation results show the effectiveness of MR-LSI in speeding up LSI computation while maintaining a high level of accuracy.

The rest of the paper is organized as follows. Section 2 discusses some related work on LSI speedup in computation. Section 3 presents the design of the distributed MR-LSI algorithm. Section 4 introduces a resource aware load balancing scheme for optimizing the performance of MR-LSI in heterogeneous computing environments. Section 5 evaluates the performance of MR-LSI in experimental MapReduce environments, and Section 6 evaluates the performance of MR-LSI in large scale simulated MapReduce environments. Section 7 concludes the paper and points out some future work.

2 RELATED WORK

The current research efforts in speeding up LSI computation generally fall into two approaches. One approach combines LSI with clustering algorithms such as k-means [30, 31] to cluster a set of documents into a number of smaller subsets and process each subset of documents individually to reduce the complexity of SVD in computation [3, 14, 16, 19]. One representative work of this approach is presented in [14] in which three clustering schemes are introduced, i.e. non-clustered retrieval (NC), full clustered retrieval (FC) and partial clustered retrieval (PC). The NC scheme employs a truncated SVD to pre-process the original data without any clustering. The FC scheme fully clusters data with a k-means algorithm, and then makes use of SVD to approximate the matrix of the document vectors in each cluster. The PC scheme only works on a few clusters that are closely related to a given query for high efficiency.

Another approach distributes the computation of LSI among a cluster of computers using the Message Passing Interface (MPI). For example, Seshadri and Iyer [28] proposed a parallel SVD clustering algorithm using MPI. Documents are split into a number of subsets of which each subset of the documents is clustered by a participating computer. Experimental results have shown that the overhead in LSI computation is significantly reduced using a number of processors.

Although the two aforementioned approaches are effective in a certain way in speeding up LSI computation, a number of challenges still remain. For example, the k-means approach does not consider the overhead incurred in clustering documents which can be high when the size of document collection is large. The MPI approach is restricted to homogeneous computing environments without any support for fault tolerance. It should be noted that modern computing infrastructures are mainly heterogeneous computing environments in which computing nodes have a variety of resources in terms of processor speed, hard disk and network bandwidth. As a result, distributing LSI computation in a heterogeneous computing environment with MPI can cause severe unbalanced workload in computation which leads to poor performance.

3 DESIGN AND IMPLEMENTATION

MR-LSI builds on MapReduce for distribution of LSI in computation. We first give a brief description of the MapReduce programming model followed by a detailed description of the MR-LSI algorithm.

3.1 MapReduce

MapReduce is a distributed programming model for data intensive tasks which has become an enabling technology in support of data intensive applications. The basic function of MapReduce model is to iterate over the input, compute key/value pairs from each part of input, group all intermediate values by key, then iterate over the resulting groups and finally reduce each group. The model efficiently supports parallelism. Figure 1 presents an abstraction of a typical MapReduce framework.

Map is an initial transformation step, in which individual input records are processed in parallel. The system shuffle and sort the map outputs and transfer them to the reducers. *Reduce* is a summarization step, in which all associated records are processed together by a single entity. Popular implementations of the MapReduce model include Mars [15], Phoenix [32], Hadoop and Google's implementation [1]. Among them, Hadoop has become the most popular one due to its open source feature.

3.2 MR-LSI

MR-LSI 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. Let

- D represent the set of p documents, $D = \{d_1, d_2, d_3, \ldots, d_p\}$.
- P represent the set of m processors in a Hadoop cluster, $P = \{p_1, p_2, p_3, \ldots, p_m\}$. Each processor runs one *map* instance called *mapper*.
- M represent the set of m *mappers* running in the Hadoop cluster,

$$M = \{map_1, map_2, map_3, \ldots, map_m\}.$$

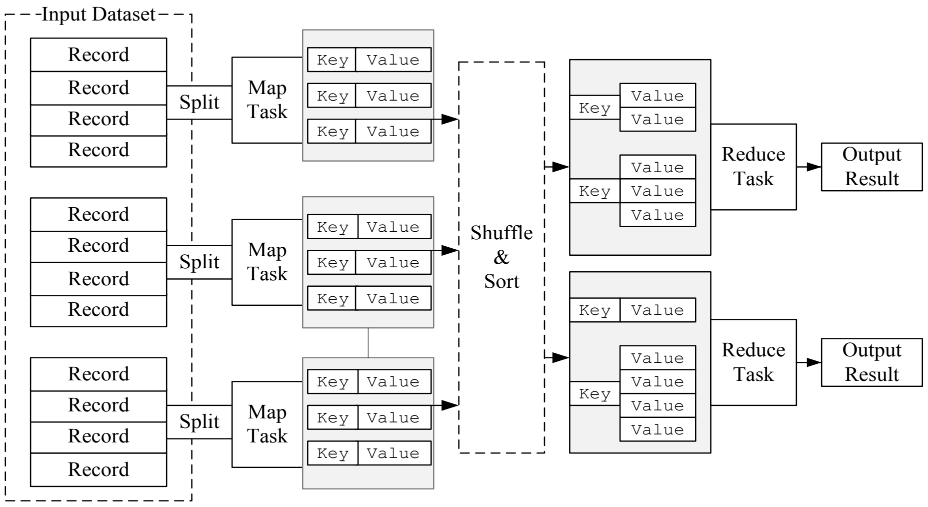


Figure 1. The MapReduce model

In LSI, the set of D documents can be represented by a set of vectors denoted by V, $V = v_1, v_2, v_3, \dots, v_p$. Each vector v_i represents the frequencies of keywords that appear in document d_i. The input of each mapper includes two parts. The first part is a centroid set of C with k initial centroids which are randomly selected from the vector set V, $C = \{c_i \in V | c_1, c_2, c_3, \dots, c_k\}$. The second part of the input of a mapper is a portion of V denoted by V_i. The vector set V is equally divided into m portions according to the number of mappers. Thus V_i satisfies $\bigcup_{(i=1)}^{m} V_i = V$.

Each mapper m_i runs on one processor p_i calculating the Euclidean distances between $v_{ij} \in V_i$ and C which is denoted by d_{ij}, then

$$d_{ij} = \| v_{ij} - c_q \|, \quad j = 1, 2, \dots, \frac{p}{m}, q = 1, 2, \dots, k.$$

Let d_{min} represent the shortest distance between v_{ij} and C, then $d_{min} = \min\Big(d_{i1}, d_{i2}, d_{i3}, \dots, d_{i\frac{p}{m}}\Big)$. 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 fed into the *reduce* instance (called *reducer*). The *reducer* groups the values with the same key c_i into a set of clusters denoted by $Cluster_i$, $Cluster_i = v'_1, v'_2, v'_3, \dots, v'_{ai}$, where $i = 1, 2, 3, \dots, k$ and $\sum_{i=1}^{k} a_i = p$.

For each $Cluster_i$, the *reducer* calculates a new centroid denoted by c'_i, $c'_i = \frac{\sum_{j=1}^{a_i} v'_j}{a_i}$. The *reducer* outputs a set of centroids denoted by C', $C' = c'_1, c'_2, c'_3, \dots, c'_k$ which will be fed into the *mappers* for computing another set of centroids C'' until the values of the centroids in set C' are the same as those in C'', then the *reducer* outputs the $Cluster_i$. Each of the k jobs runs a *mapper* performing a truncated SVD operation in $Cluster_i$. In each $Cluster_i$, the vectors v'_{ai} form a T-D matrix A, where $A = U\Sigma V^T$. After performing a truncated SVD operation, the matrix A can be represented by an approximate matrix A_k, where $A_k = U_k\Sigma_k V_k$, k is the rank of the matrix.

In LSI, for a submitted query q, it is processed using Equation (1).

$$q = q^T U_k \Sigma_k^{-1}. \tag{1}$$

The similarities of the query to the documents can be measured by calculating the cosine values of vector q and the vectors of matrix V_k using Equation (2).

$$\cos\theta_j = \frac{q_v \cdot D_j}{\| q_v \|_2 \| D_j \|_2}, \tag{2}$$

where j represents the j^{th} document in the clustered document set.

If the value of $\cos\theta_j$ is larger than a given threshold τ, then the document D_j will be a target document. Therefore the set of target documents D can be represented as $D = d_j | \cos\theta_j = \cos(q_{v'} D_j) \geq \tau$. Finally, the *reducer* generates k clusters of documents. For each cluster of documents, a truncated SVD operation is performed and targeted documents are retrieved.

4 LOAD BALANCING

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.

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. We employ a decimal string to represent a chromosome in which the data chunk assigned to a *mapper* is represented as a gene.

In Hadoop, the total time (T) of a *mapper* in processing a data chunk 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.

$$T = t_c + t_p + t_m + t_b. \tag{3}$$

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

$$t_c = \frac{D_m}{\min(H_{d'}, B_w)}. \tag{4}$$

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*.

$$t_p = \frac{D_m}{P_r}. \tag{5}$$

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 Equation (6).

$$V_b = P_r \times R_a \times \frac{B_f}{P_r \times R_a - H_d}. \tag{6}$$

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 Equation (7).

$$N_b = \frac{D_m \times R_a}{V_b}. \tag{7}$$

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

$$t_b = \frac{N_b \times B_f}{H_d}. \tag{8}$$

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

$$N_f = \left\lfloor \frac{N_b}{s} \right\rfloor - 1. \tag{9}$$

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

$$t_m = \frac{D_m \times R_a \times N_f}{H_d}. \tag{10}$$

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

$$T_{total} = \max(T_1, T_2, T_3, \ldots, T_k). \tag{11}$$

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:

$$T_1 = T_2 = T_3 = \ldots = T_k. \tag{12}$$

Let

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

Based on Equations (11) and (12), the fitness function is to measure the distance between T_i and $\bar{T}$. Therefore, the fitness function can be defined using Equation (13) which is used by the genetic algorithm in finding an optimal or a near optimal solution in determining the size for a data chunk.

$$f(T) = \sqrt{\sum_{i=1}^{k} (\bar{T} - T_i)^2}. \tag{13}$$

5 EXPERIMENTAL RESULTS

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

Number of Hadoop nodes:	4
Nodes' specifications:	Three Datanodes: CPU Q6600@2.4 G, RAM 3 GB and running OS Fedora 11. One Namenode: CPU C2D7750@2.26 G, RAM 2 GB and running OS Fedora 12.
Number of mappers per node:	2
Number of reducer:	1
Network bandwidth:	1 000 Gbps

Table 1. The experimental environment

To evaluate the performances of MR-LSI, 1 000 papers were collected from the IEEE Xplore data source. For each paper selected, a T-D matrix will be constructed. In the tests, we also designed two strategies for clustering documents which is similar to the clustered strategies proposed in [14]. One strategy is Closest Distance Searching (CDS) and the other one is All Distances Searching (ADS). CDS calculates the distances between a query q and the centroid of each sub-cluster. The closest sub-cluster to the query q will have the highest probability in containing the target documents. A truncated SVD will only be performed on the closest sub-cluster. 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.

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 Figure 2 and Figure 3, we can observe 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 result in lower performance in recall.

We conducted a number of tests to evaluate the overhead of MR-LSI in computation. The number of documents to be retrieved varied from 100 to 1 000. However, the size of the dataset was not large. From Figure 4 and Figure 5, we can observe that MR-LSI consumed more time than other algorithms in processing the 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.

We also conducted a number of additional tests to further evaluate the overhead of MR-LSI in processing a large collection of documents. We increased the size of the document collection from 5 KB to 20 MB and compared the overhead of MR-LSI with that of the CDS strategy as it is faster than both the standalone LSI and the ADS strategy. From the results plotted in Figure 6, we can observe that when the data size is less than 1.25 MB, the overhead of CDS is stable. However, the overhead of CDS starts growing when the size of dataset is larger than 2.5 MB. When the size

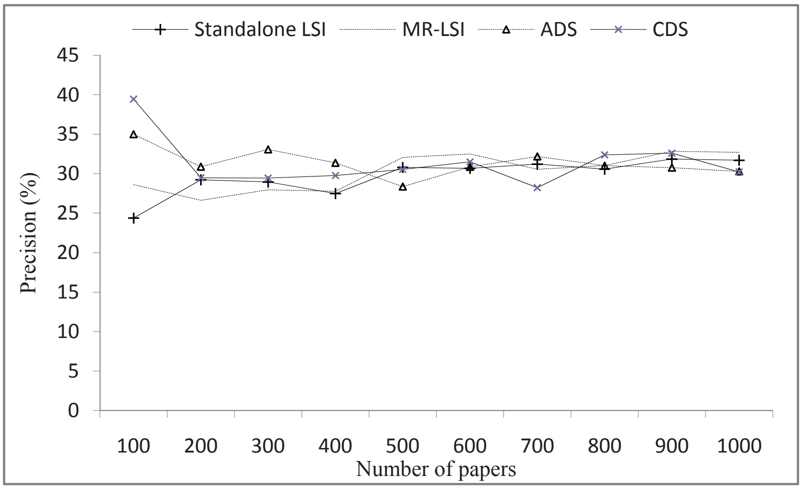


Figure 2. The precision of MR-LSI

of data reaches 10 MB, the overhead of CDS increases sharply. Compared with CDS, the overhead of MR-LSI is highly stable with an increasing size of dataset; this shows its better scalability than the CDS strategy.

6 SIMULATION RESULTS

To further evaluate the effectiveness of MR-LSI in large scale MapReduce environments, we have implemented HSim [39], a MapReduce Hadoop simulator using the Java programming language. In this section, we assess the performance of the MR-LSI in simulation environments. Using HSim, we simulated a number of Hadoop environments and evaluated the performance of MR-LSI from the aspects of scalability, the effectiveness in load balancing and the overhead of the load balancing scheme.

To study the impacts of Hadoop parameters on the performance of MR-LSI, we simulated a cluster with the configurations as shown in Table 2. Each node had a processor with 4 cores. The number of *mappers* is equal to the number of processor cores. We run two *mappers* 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, we show the impacts of a number of Hadoop parameters on the performance of MR-LSI.

6.1 Scalability

From Figure 7, we observe that the overhead of MR-LSI in computation is reduced with an increasing number of mappers showing a high scalability of the MR-LSI

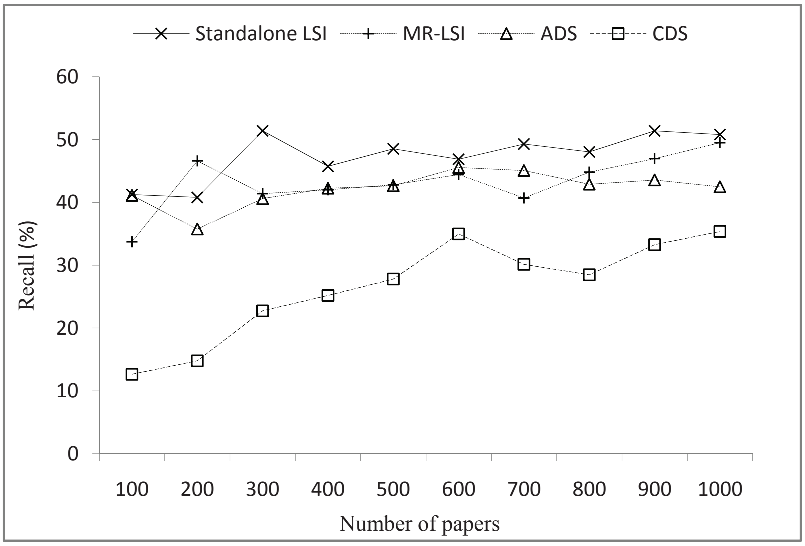


Figure 3. The recall of MR-LSI

Number of simulated nodes:	250
Data size:	100 000 MB
CPU processing speed:	Up to 0.65 MB/s
Hard drive reading speed:	80 MB/s
Hard drive writing speed:	40 MB/s
Memory reading speed:	6 000 MB/s
Memory writing speed:	5 000 MB/s
Network bandwidth:	1 Gbps
Number of mappers:	4 per node
Number of reducers:	1 or more

Table 2. The simulated environment

algorithm. It is worth noting that the number of *reducers* on a computing node does not contribute much to the overhead of MR-LSI due to the fact that the overhead of MR-LSI is mainly caused by the *mappers* involved.

6.2 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 Figure 8, we can observe that the case of using sort factor 100 gives a better performance than sort factor 10. 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.

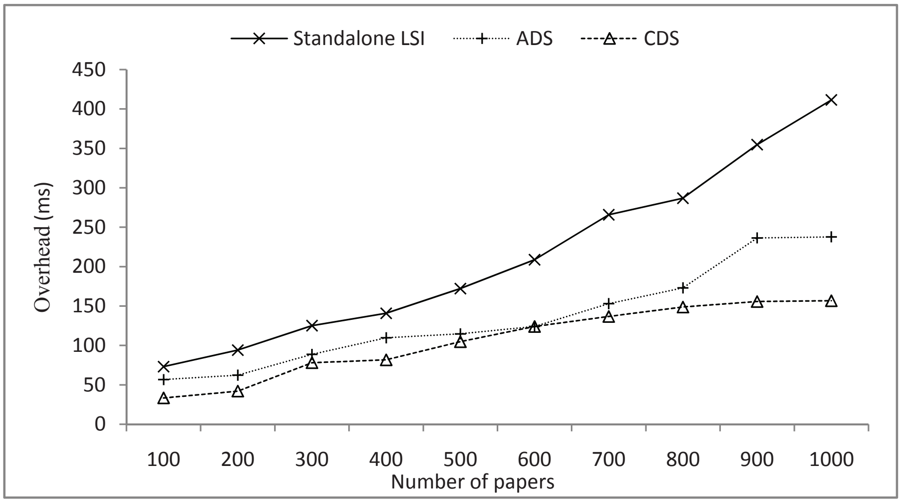


Figure 4. The overhead of standalone LSI, ADS and CDS in computation

6.3 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 100 MB. We tested the performance of MR-LSI with a data size of 1 000 MB. As shown in Figure 9, the *mappers* generate a small number of spilled files when using a large size buffer

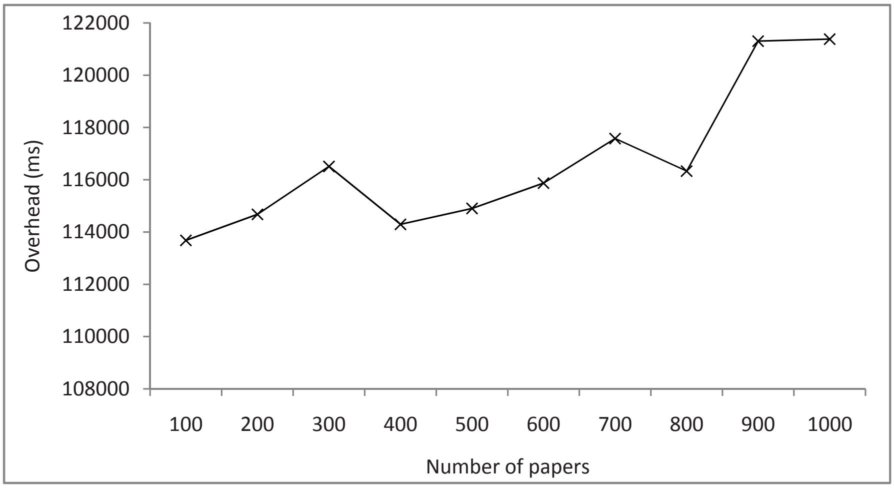


Figure 5. The overhead of MR-LSI

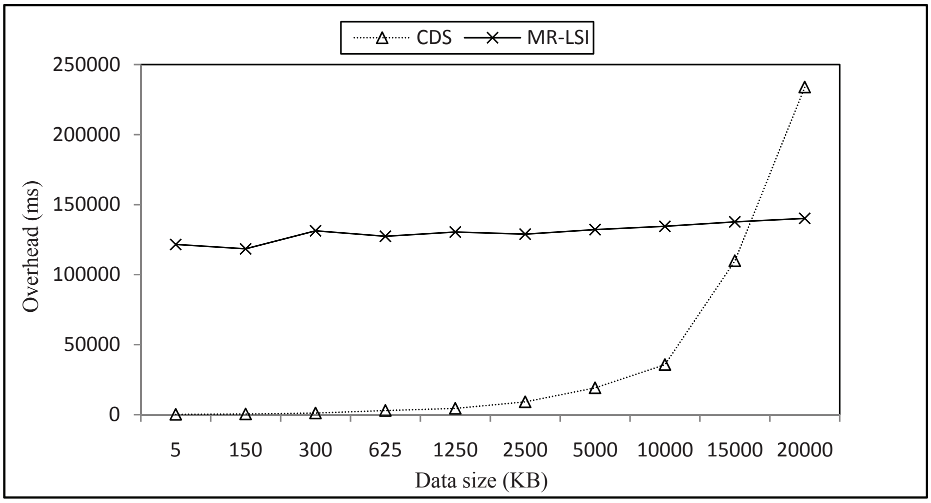


Figure 6. Comparing the overhead of MR-LSI with CDS

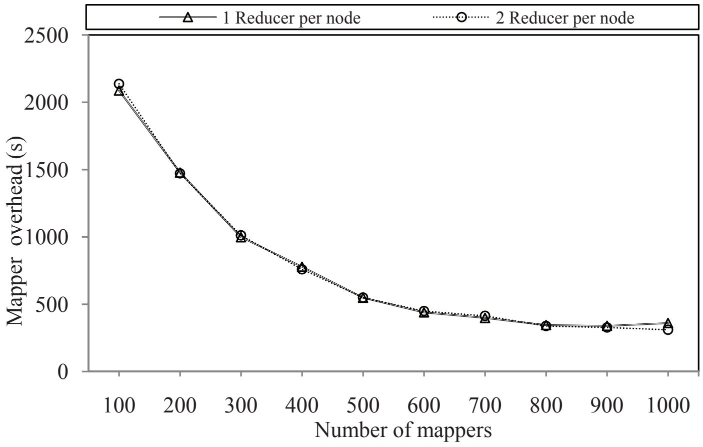


Figure 7. Scalability of MR-LSI

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.

6.4 Chunk Size

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 Figure 10, we can observe that using a large size for data chunks reduces the overhead of map-

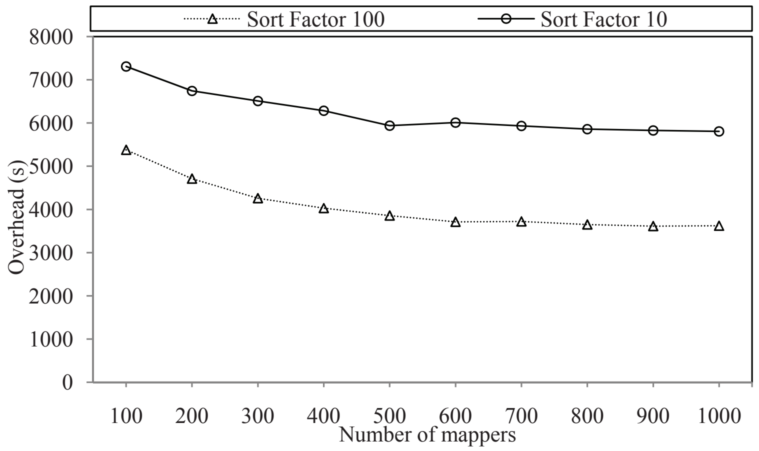


Figure 8. The impact of sort factor

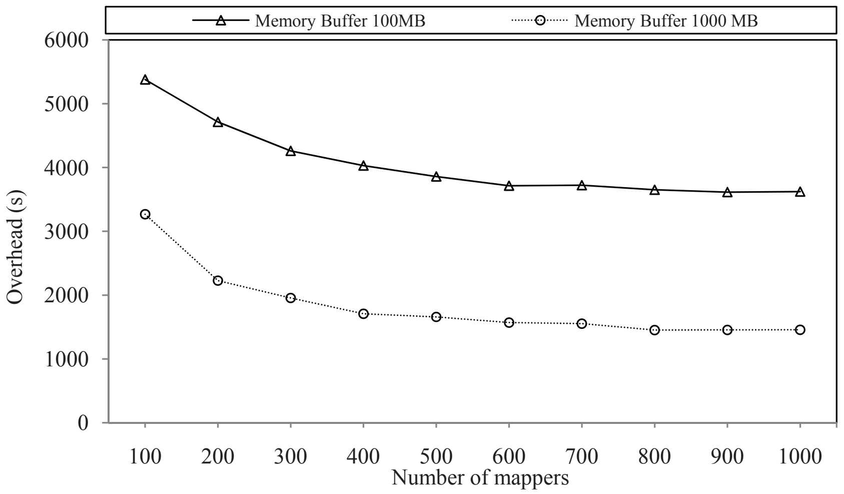


Figure 9. The impact of buffer size

pers in processing, and also reduces the total overhead of the process as shown in Figure 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 64 MB, to process 100 000 MB data, using 800 *mappers* needs $\left\lceil \frac{100\,000\,\mathrm{MB}}{800 \times 64\,\mathrm{MB}} \right\rceil = 2$ waves to finish the job. In the case of chunk size 100 MB, using 800 *mappers* needs $\left\lceil \frac{100\,000\,\mathrm{MB}}{800 \times 64\,\mathrm{MB}} \right\rceil = 2$ waves to finish the job. Similarly, using 900 *mappers* needs 2 waves to process the 100 000 MB data in both cases. When the number of *mappers* reaches 1 000, the performance of the two cases with different data sizes varies.

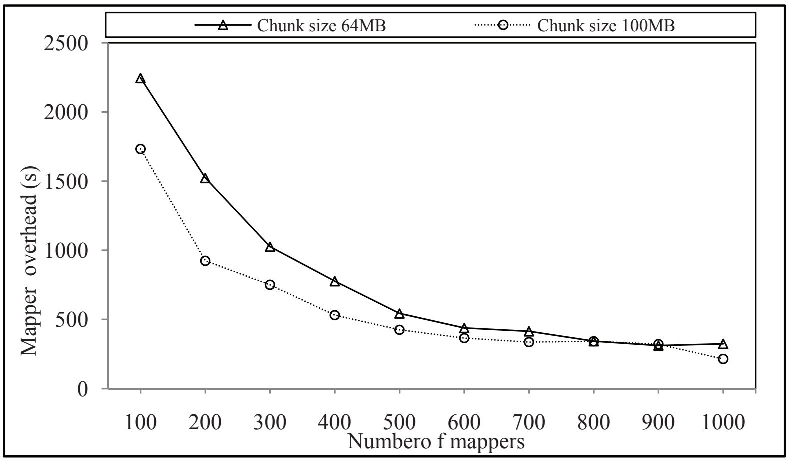


Figure 10. The impact of data chunk size on the *mappers* in MR-LSI

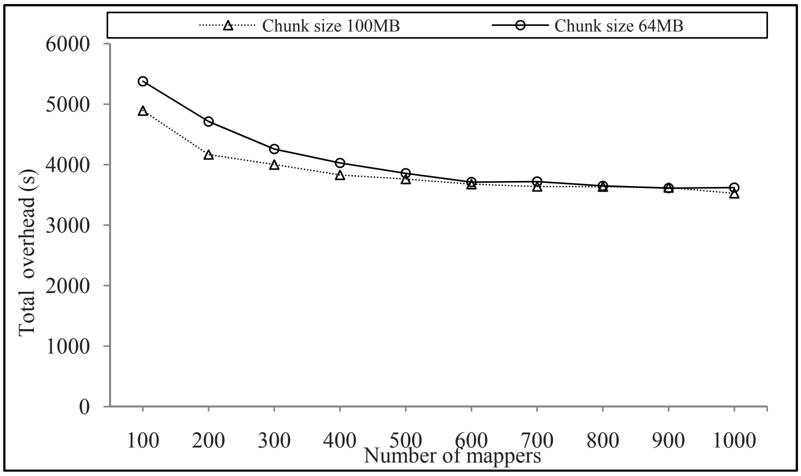


Figure 11. The impact of data chunk size on MR-LSI

6.5 Number of Reducers

Figure 12 shows that increasing the number of *reducers* enhances the performance of MR-LSI when the number of *reducers* small. When 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 large overhead.

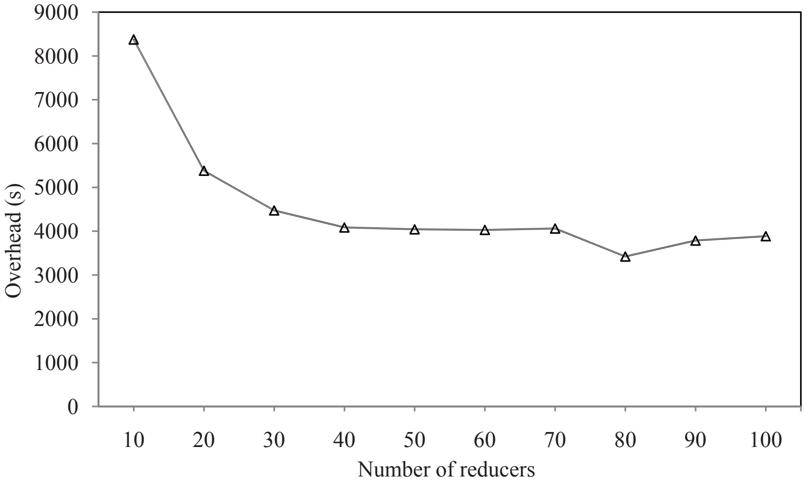


Figure 12. The impact of reducers

6.6 Load Balancing

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

Number of simulated nodes:	20
Number of processors in each node:	1
Number of cores in each processor:	2
Size of data	Test 1: 10 GB, Test 2: 10 GB to 100 GB
The processing speeds of processors:	Depending on heterogeneities
Heterogeneities:	From 0 to 2.28
Number of hard disk in each node:	1
Reading speed of Hard disk:	80 MB/s
Writing speed of Hard disk:	40 MB/s
Number of Map instances	Each node contributes 2 Map instances
Number of Reduce instances	1
Sort factor:	100

Table 3. Hadoop simulation configuration

To evaluate the load balancing algorithm we simulate a cluster with 20 computers. Each computer has one processor with two cores. The number of *mappers* is equal to the number of processor cores. Therefore we run two *mappers* 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 $P = \sum_{i=1}^{n} p_i$ 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 Equation (14).

$$Heterogeneity = \sqrt{\left(\sum_{i=1}^{k}(\bar{p} - p_i)^2\right)}. \tag{14}$$

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

We first tested 10 GB data in the simulated cluster with different levels of heterogeneity. From Figure 13, it can be observed that when the level of heterogeneity is less than 1.08 which indicates a 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.

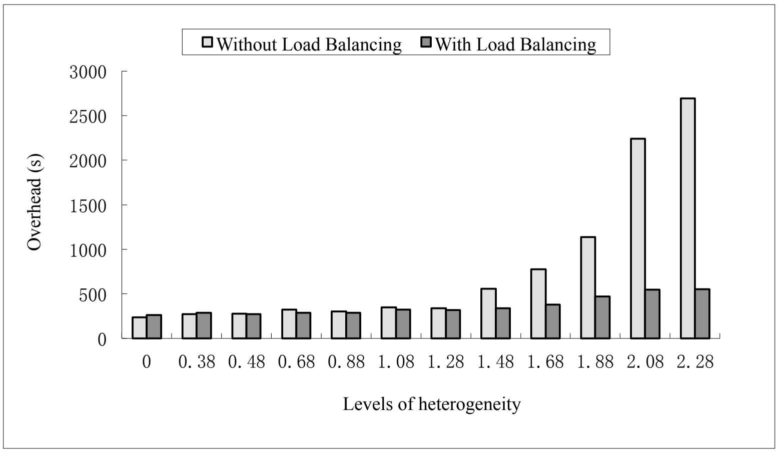


Figure 13. The performance of the load balancing scheme

The load balancing scheme builds on a genetic algorithm whose convergence affects the efficiency of MR-LSI. To analyze the convergence of the genetic algorithm, we varied the number of generations and measured the overhead of MR-LSI in

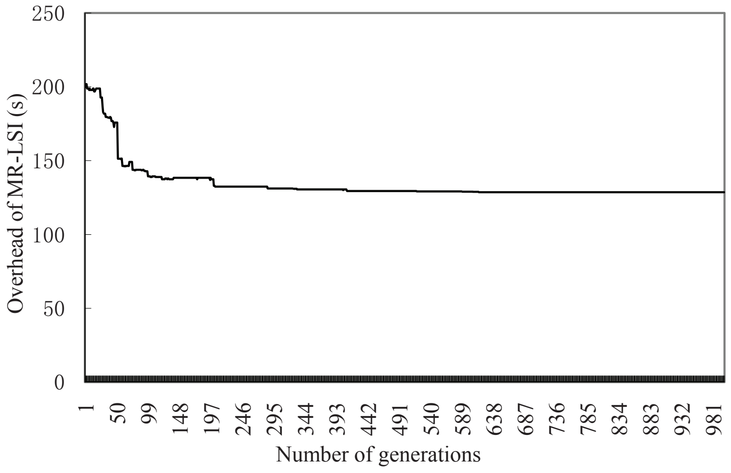


Figure 14. The convergence of the load balancing scheme

processing a 10 GB dataset in the simulated Hadoop environment. Figure 14 shows that MR-LSI has a quick convergence process in reaching a stable performance.

7 CONCLUSION

In this paper we have presented MR-LSI 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.

For future work, we are considering Amazon EC2 Cloud resources [38] for evaluating the performance of MR-LSI in large scale real Hadoop clusters.

Acknowledgement

This research is partially supported by the 973 project on Network Big Data Analytics No. 2014CB340404.

REFERENCES

[1] Aarnio, T.: Parallel Data Processing with Mapreduce. TKK T-110.5190, Seminar on Internetworking, 2009. Availaible on: `http://www.cse.tkk.fi/en/publications/B/5/papers/Aarnio_final.pdf`.

[2] Apache Hadoop! Availaible on: `http://hadoop.apache.org/` [Accessed 20 July 2012].

[3] Bassu, D.—Behrens, C.: Distributed LSI: Scalable Concept-Based Information Retrieval with High Semantic Resolution. In Proceedings of Text Mining 2003, workshop held in conjunction with the Third SIAM Int'l Conference on Data Mining.

[4] Berry, M. W.: Large Scale Singular Value Computations. International Journal of Supercomputer Applications and High Performance Computing, Vol. 6, 1992, pp. 13–49.

[5] Dean, J.—Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In Proc. of OSDI '04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA.

[6] Deerwester, S.—Dumais, S. T.—Furnas, G.: Indexing by Latent Semantic Analysis. Journal of the American Society For Information Science, Vol. 41, 1990, pp. 391–407.

[7] Ding, C.: A Similarity-Based Probability Model for Latent Semantic Indexing. In Proc. of 22nd ACM SIGIR Conference 1999, pp. 59–65.

[8] Du, L.—Jin, H.—de Vel, O. Y.—Liu, N.: A Latent Semantic Indexing and WordNet based Information Retrieval Model for Digital Forensics. In Proc. of Intelligence and Security Informatics, ISI 2008, IEEE International Conference, Taipei, pp. 70–75.

[9] Dumais, S.: LSI Meets TREC: A Status Report. In: D. Harman (Ed.): The First Text Retrieval Conference (TREC1), NIST Special Publication 500-207, 1993, pp. 137–152.

[10] Dumais, S.: Latent Semantic Indexing (LSI) and TREC-2. In: D. Harman (Ed.): The Second Text Retrieval Conference (TREC2), NIST Special Publication 500-215, 1994, pp. 105–116.

[11] Dumais, S.: Using LSI for Information Filtering: TREC-3 Experiments. In: D. Harman (Ed.): The Third Text Retrieval Conference (TREC3), NIST Special Publication, 1995, pp. 219–230.

[12] Foltz, P.—Dumais, S.: Personalized Information Delivery: An Analysis of Information Filtering Methods. Communications of the ACM, Vol. 35, 1992, pp. 51–60.

[13] Gao, J.—Zhang, J.: Sparsification Strategies in Latent Semantic Indexing. In Proceedings of the 2003 Text Mining Workshop, San Francisco, CA, 2003, pp. 93–103.

[14] Gao, J.—Zhang, J.: Clustered SVD Strategies in Latent Semantic Indexing. Information Processing and Management, Vol. 41, 2005, pp. 1051–1063.

[15] He, B.—Fang, W.—Luo, Q.—Govindaraju, N. K.—Wang, T.: Mars: A MapReduce Framework on Graphics Processors. In PACT '08: Proceedings of the 17th international Conference on Parallel Architectures and Compilation Techniques 2008, pp. 260–269.

[16] Hotho, A.—Maedche, A.—Staab, S.: Text Clustering Based on Good Aggregations. In Proc. of the 2001 IEEE International Conference on Data Mining, IEEE Computer Society, San Jose, CA, pp. 607–608.

[17] Husbands, P.—Simon, H.—Ding, C.: On the Use of Singular Value Decomposition for Text Retrieval. In Computational Information Retrieval, Philadelphia, PA, pp. 45–156.

[18] Jain, A. K.—Murty, M. N.—Flynn, P. J.: Data Clustering: A Review. ACM Computing Surveys, Vol. 31, 1999, pp. 264–323.

[19] Jiménez, D.—Ferretti, E.—Vidal, V.—Rosso, P.—Enguix, C. F.: The Influence of Semantics in IR Using LSI and K-Means Clustering Techniques. In Proceedings of the 1st International Symposium on Information and Communication Technologies 2003.

[20] Karypis, M. S. G.—Kumar, V.: A Comparison of Document Clustering Techniques. Technical Report 00-034, Department of Computer Science and Engineering, University of Minnesota 2000.

[21] Koller, D.—Sahami, M.: Hierarchically Classifying Documents Using Very few Words. In Proceedings of the Fourteenth International Conference on Machine Learning 1997.

[22] Kumar, C. A.—Srinivas, S.: Latent Semantic Indexing Using Eigenvalue Analysis for Efficient Information Retrieval. International Journal of Applied Mathematics and Computer Science, Vol. 16, 2006, pp. 551–558.

[23] Lammel, R.: Google's MapReduce programming model – Revisited. Sci. Comput. Program., Vol. 68, 2007, pp. 208–237.

[24] Majavu, W.—van Zyl, T.: Classification of Web Resident Sensor Resources Using Latent Semantic Indexing and Ontologies. Proceedings of the IEEE International Conference on Man, Systems and Cybernetics 2008.

[25] Oksa, G.—Becka, M.—Vajtersic, M.: Parallel SVD Computation in Updating Problems of Latent Semantic Indexing. In Proceedings of ALGORITMY 2002 Conference on Scientific Computing 2002, pp. 113–120.

[26] Park, H.—Elden, L.: Matrix Rank Reduction for Data Analysis and Feature Extraction. Tech. Rep., Dept. Computer Science and Engineering, University of Minnesota 2003.

[27] Pavlo, A.—Paulson, E.—Rasin, A.—Abadi, D. J.—DeWitt, D. J.—Madden, S.—Stonebraker, M.: A Comparison of Approaches to Large-Scale Data Analysis. In: Proceedings of the 35th SIGMOD International Conference on Management of Data, New York, NY, USA 2009.

[28] Seshadri, K.—Iye, K. V.: Parallelization of a Dynamic SVD Clustering Algorithm and Its Application in Information Retrieval. Software: Practice and Experience, Vol. 40, 2010, No. 10, 2010, pp. 883–896.

[29] Song, W.—Park, S.: Analysis of Web Clustering Based on Genetic Algorithm with Latent Semantic Indexing Technology. In Proc. of Advanced Language Processing and Web Information Technology 2007, pp. 21–26.

[30] Steinbach, M.—Karypis, G.—Kumar, V.: A Comparison of Document Clustering Techniques. KDD-2000 Workshop on Text Mining, Boston, MA, USA, 2000.

[31] Tarpey, T.—Flury, B.: Self-Consistency: A Fundamental Concept in Statistics. Statistical Science, Vol. 11, 1996, pp. 229–243.

[32] Taura, K.—Kaneda, K.—Endo, T.—Yonezawa, A.: Phoenix: A Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources. SIGPLAN Not., Vol. 38, 2003, pp. 216–229.

[33] Venner, J.: Pro Hadoop. 1st ed. Springer New York 2009.

[34] Wang, G.—Butt, A. R.—Pandey, P.—Gupta, K.: Using Realistic Simulation for Performance Analysis of Mapreduce Setups. In LSAP '09: Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance.

[35] White, T.: Hadoop: The Definitive Guide. 2nd ed. CA: O'Reilly Media 2009.

[36] Yan, B.—Du, Y.—Li, Z.: The New Clustering Strategy and Algorithm Based on Latent Semantic Indexing. In Proc. of Fourth International Conference on Natural Computation (ICNC '08), Vol. 1, pp. 486–490.

[37] Yates, R. D.—Neto, B. R.: Modern Information Retrieval. 1st ed. US: Addison-Wesley, 1999.

[38] Amazon EC2 Cloud, Availaible on: `http://aws.amazon.com/ec2/`.

[39] Liu, Y.—Li, M.—Alham, N. K.—Hammoud, S.: HSim: A MapReduce Simulator in Enabling Cloud Computing. Future Generation Computer Systems, DOI:10.1016/j.future.2011.05.007 g.

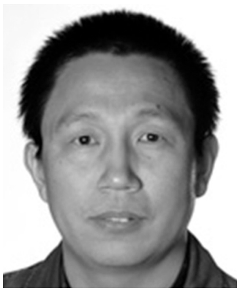
[40] CAI web site. Availaible on: `http://www.cai.sk`.

[41] Akl, S. G.—Bruda, S. D.: Parallel Real-Time Optimization: Beyond Speedup. Parallel Processing Letters, Vol. 9, 1999, No. 4, pp. 499–509.

[42] Hintikka, J.: Knowledge and Belief: An Introduction to the Logic of Two Notions. Cornell University Press, Ithaca, NZ 1962.

[43] Marek, W.—Truszczinski, M.: Relating Autoepistemic and Dolomit Logic. In: R. Brachman and H. Levesque (Eds.): Principles of Knowledge Representation and Reasoning, Proceedings of the First International Conference, KR '89, Toronto 1989, pp. 276–288.

Yang Liu is an Associated Professor in the School of Electrical Engineering and Information, Sichuan University, China. He received his Ph. D. in the School of Engineering and Design at Brunel University, UK. His research interests include big data analytics for smart grids, parallel computing technologies for large scale power system analysis, data mining and information retrieval.



Maozhen Li is a Professor in the School of Engineering and Design at Brunel University, UK. He received the Ph. D. from Institute of Software, Chinese Academy of Sciences in 1997. He was a post-doctoral scholar in the School of Computer Science and Informatics, Cardiff University, UK in 1999–2002. His research interests are in the areas of high performance computing (grid and cloud computing) and intelligent systems. He is on the Editorial Boards of Computing and Informatics journal and journal of Cloud Computing: Advances, Systems and Applications. He has over 100 research publications in these areas. He is a Fellow of the British Computer Society.

Mukhtaj Khan received his M. Sc. in Mobile Computer System from Staffordshire University, UK in 2006. He is currently a Ph. D. student in the School of Engineering and Design at Brunel University, UK. The Ph. D. study is sponsored by Abdul Wali Khan University Mardan, Pakistan. His research interests are focused on high performance computing and intelligent systems and networks.

Man Qi is a Senior Lecturer in Department of Computing at Canterbury Christ Church University, UK. Her research interests are in the areas of computer graphics, computer animation, multimedia and applications. She is a Fellow of the British Computer Society and also a Fellow of the Higher Education Academy.