



Scalable and efficient data distribution for distributed computing of all-to-all comparison problems



Yi-Fan Zhang^a, Yu-Chu Tian^{a,b,*}, Wayne Kelly^a, Colin Fidge^a

^a School of Electrical Engineering and Computer Science, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia

^b College of Information Engineering, Taiyuan University of Technology, Taiyuan, Shanxi 030024, China

HIGHLIGHTS

- New insights into distributed computing of all-to-all comparison problems with big data.
- Formulation of data distribution of all-to-all comparisons as an optimization problem.
- A heuristic algorithm for scalable and efficient data distribution and task scheduling.
- Demonstration of the algorithm in improved storage saving, data locality and execution time.

ARTICLE INFO

Article history:

Received 11 October 2015

Received in revised form

27 May 2016

Accepted 6 August 2016

Available online 9 September 2016

Keywords:

Distributed computing

Big data

All-to-all comparison

Data distribution

ABSTRACT

All-to-all comparison problems represent a class of big data processing problems widely found in many application domains. To achieve high performance for distributed computing of such problems, storage usage, data locality and load balancing should be considered during the data distribution phase in the distributed environment. Existing data distribution strategies, such as the Hadoop one, are designed for problems with MapReduce pattern and do not consider comparison tasks at all. As a result, a huge amount of data must be re-arranged at runtime when the comparison tasks are executed, degrading the overall computing performance significantly. Addressing this problem, a scalable and efficient data distribution strategy is presented in this paper with comparison tasks in mind for distributed computing of all-to-all comparison problems. Specifically designed for problems with all-to-all comparison pattern, it not only saves storage space and data distribution time but also achieves load balancing and good data locality for all comparison tasks of the all-to-all comparison problems. Experiments are conducted to demonstrate the presented approaches. It is shown that about 90% of the ideal performance capacity of the multiple machines can be achieved through using the approach presented in this paper.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Data-intensive computing [1] with a very large number of big data sets can be a great challenge in service computing for both commercial applications and scientific investigations. For these big data problems, a huge amount of data needs to be stored, retrieved, analyzed and visualized within a period of time acceptable to the users [2,3]. This demands significant resources of computing power, memory and storage spaces, and network communications. While distributed computing systems provide

a general platform to solve large-scale computing problems, improving the computing performance for large-scale and data-intensive computing problems is an emerging requirement.

Among various big data processing problems, all-to-all comparison problems are widely found in many application domains such as machine learning, data mining, information management, bioinformatics and biometrics. A typical example is the calculation of covariance matrix for high-dimensional data in machine learning. In data mining, the computation of similarity matrix is a critical step for clustering and classification. It gives all pairwise similarities or dissimilarities between the objects under consideration [4]. The experiments by Perera [5] computed the cosine similarity between 8192 feature vectors. Mapping the ontologies Molecular Functions and Biological Processes from Gene Ontology with 10,000 and 20,000 concepts, respectively, involves comparisons of about 2×10^8 pairs of entities [6]. In bioinformatics,

* Corresponding author at: School of Electrical Engineering and Computer Science, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia.

E-mail address: y.tian@qut.edu.au (Y.-C. Tian).

phylogenetic relationships are inferred through comparing gene sequences of different species [7]. Sequence alignment, clustering analysis [8] and recently investigated global network alignment [9] are typical all-to-all comparison problems in computational biology and bioinformatics.

All-to-all comparisons represent a typical computing pattern, which focuses on processing a large number of large- or small-size data files. In a general formulation of the all-to-all comparison problems, each data item of a data file needs to be compared with all data items of the other data files in the data set. Therefore, the all-to-all comparison computing pattern is fundamentally different from the MapReduce computing pattern [10], which has been implemented in Hadoop with wide applications in big data processing [11]. If the number of data files in a data set or the number of data items in a data file is big, the scale of the required computation for all-to-all comparisons becomes massive [12].

Efforts have been made to address all-to-all comparison problems previously [13–18]. Some solutions have been proposed for special-purpose all-to-all comparison problems such as the popularly used BLAST [19] and ClustalW [20], or for different types of computing architectures with GPUs [16] or shared memory [21]. However, these solutions require that all data files be deployed to each of the nodes in the system. While distributing whole data sets everywhere is conceptually easy to implement, it causes significant time consumption and communication cost and demands a huge amount of storage space. Therefore, the data distribution strategy from these previous solutions is not scalable to the big data processing problems considered in this paper.

In addition to these specific solutions, some computing frameworks are also widely used to process big data problems in distributed environments [22–25]. Enabling distributed processing of large data sets across clusters of commodity servers, Hadoop [11] is designed to scale up from a single server to thousands of machines. With fault tolerance, it can achieve high computing performance for distributed computation that well matches the MapReduce computing pattern. However, the Hadoop distributed file system (HDFS) and its data distribution strategy are inefficient for all-to-all comparison problems due to the completely different computing pattern involved.

The main contribution of this paper is a scalable and efficient data distribution strategy for a class of all-to-all comparison problems, in which all input data files have the same or similar size and thus comparison tasks have the same or similar execution time. The strategy is developed with consideration of storage usage, data locality and load balancing of distributed computing tasks. This paper has substantially extended our preliminary studies in a conference paper [12] from four aspects: new insights into the challenges, reformulation of the problem, refinement of the data distribution strategy, and more experimental studies. This will be summarized at the end of Section 2 on Related Work.

The paper is organized as follows. Section 2 reviews related work and motivates the research. Section 3 formalizes all-to-all comparison problems and identifies the challenges of data distribution. This is followed by Section 4 to formulate the data distribution problem as an optimization problem. Providing a heuristic solution to the optimization problem, our new data distribution strategy is presented in Section 5, and is experimentally demonstrated in Section 6. Finally, Section 7 concludes the paper.

2. Related work and motivations

Several approaches have been developed to address specific all-to-all comparison problems in bioinformatics. All-to-all comparisons are a key calculation stage in Multiple Sequence Alignment (MSA) [26] and studying of phylogenetic diversity in protein families [27]. In general, the computing of these bioinformatics

problems includes the calculation of a cross-similarity matrix between each pair of sequences [28,29].

Data intensiveness describes those applications that are I/O bound or with a need to process large volumes of data [2,10]. Compared with computing-intensive problems, applications of all-to-all comparison problems devote most of the processing time to I/O and movement for a massive amount of data [30]. Therefore, to improve the performance of data-intensive problems, the distribution of all the data sets needs to be well considered.

To process all-to-all comparison problems, various distributed systems and runtime libraries have been used. Heitor and Guilherme [13] proposed a methodology to parallelize a multiple sequence alignment algorithm by using a homogeneous set of computers with the Parallel Virtual Machine (PVM) library. In their work, a detailed description of the modules was provided and a special attention was paid to the execution of the multiple sequence comparison algorithm in parallel. Macedo et al. [14] proposed an MPI/OpenMP master/slave parallel strategy to run the DIALIGN-TX algorithm in heterogeneous multi-core clusters. In their research, different task allocation policies were compared to determine the appropriate choice. All these approaches distribute all data to each of the computing nodes in the cluster, leading to inefficiencies in data distribution and storage. Our work in this paper aims to avoid such inefficiencies.

Meng and Chaudhary [15] presented a heterogeneous computing platform through a Message Passing Interface (MPI) enabled enterprise computing infrastructure for high-throughput biological sequence analysis. In order to achieve load balancing, they distributed the workload based on the hardware configuration. The whole database is split into multiple nearly-equal sized fragments; and then each of the computing nodes is assigned a number of database fragments according to its processing capacity. However, in practical computing of all-to-all comparison problems using their approach, data transmissions among the computing nodes cannot be avoided at runtime. This drawback will be overcome in our work presented in this paper.

Xiao et al. [16] proposed a design and optimization of the BLAST algorithm in a GPU–CPU mixed heterogeneous system. Due to the specific architecture of the GPU, their implementation can achieve a six-fold speed-up for the BLAST algorithm. GPUs were also used for a parallel implementation of MAFFT for MSA analysis [26]. In the work by Torres et al. [31], they were configured for exact alignment of short-read genetic sequences. To accelerate the next generation long read mapping, Chen et al. made use of FPGA hardware to speed up sequence alignment [32]. In comparison with all those hardware-dependent implementations, our work in this paper does not rely on specific hardware.

Several approaches were also developed for load balancing in distributed computing of all-to-all comparison problems. One version of parallel all-to-all comparison of genome sequences was carried out by Hill et al. [17]. The main intention of the work was to provide load balancing among the clusters by dividing the comparison matrix into rows and then dynamically assigning these rows to different nodes. Gunturu [18] proposed a load scheduling strategy, which depends on the length of the sequence and the number of processors in the network. They assumed that all the processors in the network already had both sequences to be compared in their local memory. Our work in this paper distributes data to the computing nodes with consideration of the computing tasks, thus avoiding this assumption.

Recently, efforts have been made to use computing frameworks for all-to-all comparison problems. All-pairs is an abstraction designed by Moretti et al. [33] for data-intensive computing on campus grids. It focuses on providing an abstraction for users to deal with all-to-all comparison problems. To give each comparison task the required data, a spanning tree method is proposed to deliver all data to every node efficiently.

Moreover, some methods have been developed on top of Hadoop and/or MPI for distributed computing of all-to-all comparison problems. Marc et al. [34] proposed a scalable complete genotyping system, which brings together the complete composition vector and affinity propagation algorithms needed to produce genotypes from a variety of genome data. This suite of tools was developed on top of Hadoop and MPI to support multiple computing nodes and large data sets. CloudBurst, proposed by Schatz [35], uses Hadoop for parallel short read-mapping. Experiments show that the running time of CloudBurst scales linearly with the number of reads mapped, and exhibits near linear speed-up as the number of processors increases. In all these methods, all the data sets are distributed through Hadoop's data distribution strategy, which is unsatisfactory for scalable computing of large-scale all-to-all comparison problems.

The inefficiency of Hadoop's data distribution in all-to-all comparison problems is mainly due to the following reasons: (1) Although the number of data replications can be manually set in HDFS, one does not know how to set it and thus tends to use the default number of three. Once set, this number becomes a constant regardless of the number of machines and the number of data files. (2) The location of each file is randomly determined in HDFS. This is not suitable for general all-to-all comparison problems for high performance that requires good data locality. (3) Increasing the number of data replications does not necessarily improve the overall computing performance much due to poor data locality unless replicating the data files to everywhere. These issues will be fixed in this paper by making data distribution aware of comparison tasks.

Despite significant developments in processing all-to-all comparison problems, technical gaps still exist in this area. The existing solutions mainly focus on parallelizing different all-to-all comparison algorithms and providing load balancing. None of them have paid a special attention to data distribution. Most previous methods have assumed that all data can be stored in each node, implying poor scalability for big data sets. While some of the existing methods use distributed file systems such as HDFS, the data strategy in use is nonetheless inefficient when processing large-scale all-to-all comparison problems. Many available approaches are designed for domain-specific applications or hardware-specific computing platforms, and thus do not provide a solution to general all-to-all comparison problems in general distributed computing platforms.

To address these challenges, this paper presents an approach to distributed computing of a class of all-to-all comparison problems in which all input data files have the same or similar size and consequently all comparison tasks have the same or similar execution time. The approach automatically determines the number of data replications with good data locality and load balancing for comparison tasks. Substantially extended our preliminary studies [12], new developments in this paper include: (1) new insights into the challenges of the all-to-all comparison problem (Section 3.2); (2) a more precise reformulation of the data distribution problem with additional computing performance considerations (Section 4.3); (3) a refined data distribution strategy (Section 5) from the reformulated problem; and (4) more case studies (Section 6). As Hadoop and its underlying MapReduce computing pattern are the most widely used solution to distributed computing of big data, the approach presented in this paper for distributed computing of all-to-all comparisons will be compared with Hadoop.

3. Problem statement and challenges

Let A , C and M denote the data set to be pairwise compared, the comparison function on pairs in A and the output similarity matrix of A , respectively. Characterized by the Cartesian product or cross

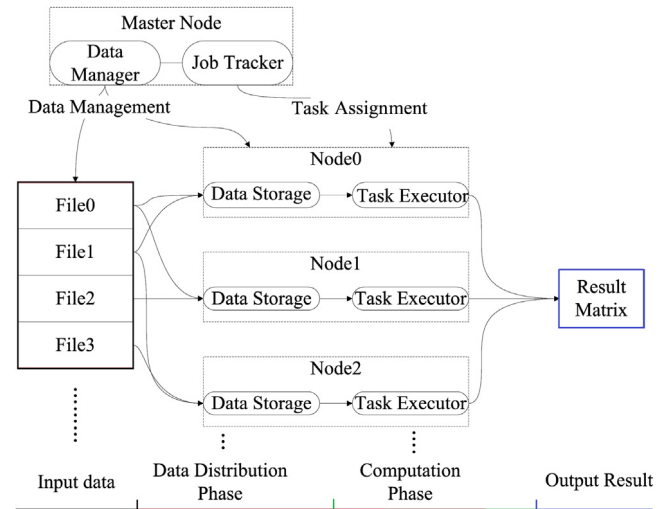


Fig. 1. General work flow for solving all-to-all comparison problems in distributed computing environments.

join of A , the all-to-all comparison problem discussed in this paper is mathematically stated as follows:

$$M_{ij} = C(A_i, A_j), \quad i, j = 1, 2, \dots, |A|, \quad i < j \quad (1)$$

where A_i represents the i th item of A , M_{ij} is the element of output matrix M resulting from the comparison between A_i and A_j , and $|A|$ means the size of A .

The original motivation for studying this problem came from our investigation into the performance improvement for bioinformatics, where the elements being pairwise compared are typically genes or genomes [7,29]. Later, it is realized that the same computing pattern also applies in many other domains, e.g., ontology mapping in information management [6] and similarity matrix computation for clustering and classification in data mining [4,5].

3.1. Principles for data distribution

A typical scenario for using distributed computing systems in all-to-all comparison problems is described as follows. In general, a data manager manages and distributes all the data to the worker nodes first. Then, computing tasks are generated and allocated by a job tracker to the nodes. Lastly, the tasks are executed by task executors to process related data sets. This work flow of this process is shown in Fig. 1. It is seen from Fig. 1 that to solve all-to-all comparison problems efficiently both data distribution and task computation phases need to be improved.

Two issues affect the overall computing performance of big data processing in distributed environments. They are data locality and computing task allocation.

- Data locality is a basic principle for processing big data problems. It means that the computing operation is generally more efficient when it is allocated to a worker node near the required data. Computation tasks which need to access remote data sets can be very inefficient due to the heavy network communications and data transmission.
- The distributed computing of big data problems is more efficient when all the worker nodes are allocated suitable numbers of computing tasks which match their processing capability. In that case, all the computing power of the system can be efficiently utilized.

Thus, data distribution for distributed computing of all-to-all comparison problems should enable (1) good data locality for comparison tasks, and (2) load balancing of comparison tasks through task allocation with good data locality.

Table 1

Three examples to show three challenges of the data distribution problem.

Example	Node ID	Data file IDs	Comparison tasks with local data	Tasks without local data	Load balanced
I: 6 files on 4 nodes	①	1, 5, 6	(1,5) (1,6) (5,6)	(1,3)	Likely
	②	1, 2, 5	(1,2) (2,5)	(1,4) (2,6)	
	③	2, 3, 4	(2,3) (2,4) (3,4)	(3,5)	
	④	3, 4, 6	(3,6) (4,6)	(4,5)	
II: 4 files on 3 nodes	①	1, 2, 3	(1,2) (1,3) (2,3)	None	No
	②	1, 2, 4	(1,4) (2,4)		
	③	3, 4	(3,4)		
III: 6 files on 8 nodes	①	1, 3, 5	(1,3) (1,5)	(1,2) (3,4) (3,6) (4,5) (5,6)	Likely
	②	1, 4, 6	(1,4) (1,6)		
	③	2, 3, 5	(2,3)		
	④	2, 4, 6	(2,4)		
	⑤	1, 3, 5	(3,5)		
	⑥	1, 4, 6	(4,6)		
	⑦	2, 3, 5	(2,5)		
	⑧	2, 4, 6	(2,6)		

3.2. Challenges of the data distribution problem

The Hadoop distributed file system (HDFS) provides a strategy to distribute and store big data sets. In HDFS, data items are randomly distributed with a fixed number of duplications among all storage nodes. While multiple copies of data items in HDFS enhance the reliability of data storage, the HDFS data strategy is inefficient for all-to-all comparison problems due to its poor data locality, unbalanced task load and big solution space for data distribution.

3.2.1. Poor data locality in HDFS

Hadoop's data strategy is designed for a trade-off between high data reliability and low read/write cost. From this design perspective, it does not consider the data requirements for comparison tasks that follow. For example, consider a scenario with 6 data items and 4 worker nodes. A possible solution for Hadoop's data strategy is shown in Example I in Table 1. It is seen from this example that although each of the 6 data items has 2 copies, there is no worker node that contains all the required data for certain comparison tasks, (1,3), (1,4), (2,6), (3,5) and (4,5), indicating poor data locality for these comparison tasks. In this case, runtime data movements among the nodes through network communications cannot be avoided. These will induce runtime storage costs and affect the overall computing performance of the all-to-all comparison problem. This problem becomes worse when the scale of the computing becomes bigger.

3.2.2. Unbalanced task load resulting from the HDFS

To show an unbalanced task load resulting from the HDFS data strategy, consider a scenario with 4 data items and 3 worker nodes. A possible Hadoop's data distribution solution is depicted in Example II in Table 1. Also shown in the table is a possible scheme for allocation of all 6 comparison tasks to the 3 worker nodes. With the HDFS, there is no way to ensure a balanced task allocation, which requires each of the 3 nodes be allocated 2 comparison tasks.

3.2.3. Inefficiency of increasing file replications in HDFS

In the HDFS data strategy, the number of data replications can be manually set by users. But there is no guidelines on how this number is set and thus users tend to use the default number of three. Once the number is set, it becomes a constant regardless of the number of machines to be used and the number of data files to be distributed. Moreover, the location of the data file replications is randomly determined in the sense of user's awareness. This causes poor data locality, leading to poor performance of the distributed computing. One might imagine that further increasing the data replications may help. However, unless replicating data

files to everywhere, increasing the data replications in HDFS does not improve the data locality performance much but results in significant computation costs due to the increased demand in network communications for data movement at runtime.

Consider a possible HDFS solution with 6 data items and 8 worker nodes shown in Example III in Table 1. It is seen that even if the number of data file replications is increased to 4, poor data locality (Section 3.2.1) and unbalanced task load (Section 3.2.2) still exist. The comparison tasks (1,2), (3,4) and (5,6) cannot be completed without remote data access at runtime.

3.2.4. Big solution space of the HDFS data distribution

Distributing data pairs also means allocating related comparison tasks. We aim to allocate Q distinguishable comparison tasks to N indistinguishable worker nodes. From combinatorial mathematics, the total number of feasible solutions is expressed by the Stirling number. Based on the discussion in our previous research paper [12], too many possible distribution solutions to access in practice even for a very simple case. This implies that it is generally impossible to evaluate all possible solutions to find the best answer in a reasonable period of time. Therefore, it is necessary to develop heuristic solutions for data distribution.

4. Formulation for data distribution

This section develops requirements of the computing framework for data distribution. It begins with overall considerations and assumptions. This is followed by formulating reduction of the storage usage and improvement of the computing performance. It ends with an overall optimization problem to specify the requirements for data distribution.

4.1. Overall considerations and assumptions

To solve all-to-all comparison problems by using the work flow shown in Fig. 1, the data distribution strategy needs to be developed. In the scenario we followed, the data distribution strategy will generate the solution of data distribution first. Then, all the data files are deployed based on the solution provided. In this paper, we do not consider the failure of worker nodes, which will be addressed in the future.

The following aspects need to be considered for data distribution:

- Storage usage of the distributed system. For all-to-all comparison problems with big data sets, distributing data sets among all nodes should consider not only the usage of storage space for each node within its capacity, but also keeping the total time spent on data distribution at an acceptable level.

- Performance of the comparison computation. For distributed computing, it is important to allocate comparison tasks in a way to make full use of all available computing power in the system. Also, good data locality for all comparison tasks can help improve the computing performance greatly. Hence, the data distribution strategy should be designed to allocate data items in a way to improve the performance of comparison computation.

In real-world scenarios, there exist a large number of applications in which all data files have the same or similar size and all comparison tasks have the same or similar execution time. Typical examples are covariance matrix computation, similarity calculation in clustering and classification, distance matrix estimation among a group of remotely sensed data or images, and many more. This paper addresses such a class of applications.

In the following, we analyze the requirements for storage usage of the distributed system and performance of the comparison computation.

4.2. Reducing the storage usage

Many factors contribute to the time spent on data distribution. Given the network bandwidth, network topology and the size of the data items, from the assumptions above, the time for data distribution is proportional to the number of data items to be distributed. Let \mathcal{D} represent all data files to be distributed. Then, the time for data distribution, $T_{distribution}$, can be expressed as:

$$T_{distribution} \propto |\mathcal{D}|. \quad (2)$$

As mentioned previously, the storage usage for each of the worker nodes must also be within its limitation. Given our assumptions, this can be achieved if all data sets are evenly distributed.

Now, we consider both data distribution time and storage limitation. Let $|D_i|$ denote the number of files allocated to worker node i . A data distribution strategy is expected to minimize the maximum of $|D_1|, \dots, |D_N|$, i.e.,

$$\text{Minimize } \max \{|D_1|, |D_2|, \dots, |D_N|\}. \quad (3)$$

Choosing to minimize the maximum number of data files in the worker nodes has the following benefits: (1) this target makes all the worker nodes have the similar number of data files. In the ideal case, the differences of the number of data files among the nodes are at most one, meaning a balance data storage usage for the distributed systems. (2) Considering the number of comparison tasks can be executed is proportional to the number of data files stored in the worker node, this target also makes all the worker nodes have the similar number of executable comparison tasks.

4.3. Improving the computing performance

In distributed computing of an all-to-all comparison problem, the overall computation time of the computing tasks executed is determined by the last finished worker node. To complete each of the comparison tasks, the corresponding worker node has to access and process the required data items.

Let K , $T_{comparison(k)}$ and $T_{accessdata(k)}$ represent the number of comparison tasks allocated to the last finished worker node, the time for comparison operations for task k and the time for accessing the required data for task k , respectively. The total elapsed time of executing comparisons tasks, $T_{comparison}$, is then expressed as:

$$T_{comparison} = \sum_{k=1}^K (T_{comparison(k)} + T_{accessdata(k)}). \quad (4)$$

From Eq. (4), our data distribution strategy reduces the total execution time T_{total} by meeting two constraints: load balancing

for comparison tasks on the worker nodes, and good data locality for all pairwise comparison tasks.

For load balancing, the maximum number J of the comparison tasks allocated to the last finished worker nodes can be minimized. Let T_i denote the number of pairwise comparison tasks performed by worker node i . For a distributed system with N worker nodes and M data files, a total number of $M(M-1)/2$ comparison tasks need to be allocated the work nodes. Minimizing the value of K can be expressed as follows:

$$\forall T_i \in \{T_1, T_2, \dots, T_N\}, \quad T_i \leq \left\lceil \frac{M(M-1)}{2N} \right\rceil, \quad (5)$$

where $\lceil \cdot \rceil$ is the ceiling function. Because $N > 1$ (for distributed computing) and T_i , the number of tasks, is a positive integer, Eq. (5) implies that

$$M(M-1)/2 \geq N, \quad M > 2. \quad (6)$$

Good data locality can also be mathematically formulated. If all required data for a comparison task are stored locally in the node that performs the task, the task will not need to access data remotely through network communications. Good data locality implies a minimized value of $T_{accessdata(k)}$ with its lowest possible value of 0. Let (x, y) , T , T_i and D_i represent the comparison task for data x and data y , the set of all comparison tasks, the set of tasks performed by worker node i , and the data set stored in worker node i , respectively. Good data locality for all comparison tasks can be expressed as follows:

$$\forall (x, y) \in T, \exists i \in \{1, \dots, N\}, \quad x \in D_i \wedge y \in D_i \wedge (x, y) \in T_i. \quad (7)$$

In other words, there must be at least one node i capable of performing each comparison (x, y) with local data only.

Data allocation discussed above becomes trivial for $N = 2$. In this case, at least one node must store all data files from the requirements in Eqs. (5) and (7). Therefore, a well-defined data distribution problem requires

$$N > 2. \quad (8)$$

4.4. Optimization for data distribution

Considering both storage usage and computing performance as discussed above, a data distribution strategy is expected to meet the target in Eq. (3) and also satisfy the constraints in Eqs. (5) and (7). When the target in Eq. (3) is achieved, the storage usage for all work nodes can be reduced greatly, thus reducing the time spent on distributing all data sets ($T_{distribution}$). Meeting the constraints in Eqs. (5) and (7) means that the overall comparison time $T_{comparison}$ can be minimized. As a result, the following total time elapsed for data distribution and task execution is effectively reduced:

$$T_{total} = T_{distribution} + T_{comparison}. \quad (9)$$

Therefore, the data distribution problem can be expressed as a constrained optimization problem:

$$\begin{cases} \text{Minimize } \max \{|D_1|, |D_2|, \dots, |D_N|\} \\ \text{s.t. Eqs. (5)–(8).} \end{cases} \quad (10)$$

As discussed previously in Section 3, the large number of combinations of data and related comparison tasks makes the optimization problem for data distribution difficult to solve in practical applications. In the next section, a heuristic algorithm will be developed to address this challenge.

5. Heuristic data distribution strategy

This section starts with discussions on heuristic rules for data distribution. Then, our data distribution algorithm is presented








	1	2	3	4	p	d
1		*	*	*	*	*	*	
2			*	*	*	*	*	
3				*	*	*	*	
4					*	*	*	
⋮						*	*	
⋮							*	
p								
d								

Fig. 2. Additional comparison tasks introduced by adding new data d to node i .

with detailed steps. Our data distribution strategy is further analyzed through an example.

5.1. Heuristic rules for data distribution

A way to derive a feasible solution to the distribution problem formulated in Eq. (10) is to meet the constraints in Eqs. (5) and (7). It is seen from the constraint in Eq. (7) that if we can determine the location for a specific comparison task (x, y) , the location of the required data x and y can also be determined. Thus, we will allocate all comparison tasks to the worker nodes in a way to meet the constraints in Eqs. (5) and (7). From this task allocation, a feasible solution to the data distribution problem is also obtained.

Consider the scenario shown in Fig. 2. The right column of the comparison matrix in the figure shows additional comparison tasks that can be allocated to a specific node i when a new data file d is distributed to a node with p data files already stored. Therefore, if we can allocate as many comparison tasks as possible to node i for each new data file d , the total number of data files needing to be distributed can be minimized.

The additional comparison tasks introduced by adding new data d to worker node i include those that have never been allocated before (marked by circles in Fig. 2) and those that have already been allocated (marked by triangles in Fig. 2). The related rules for distributing data are developed as follows:

Rule 1: For those comparison tasks that have never been allocated before, a data distribution strategy can be designed to allocate as many of these tasks as possible to node i by following Constraint (5).

Rule 2: For those comparison tasks that have already been allocated, the data distribution strategy can be designed to re-allocate each of these tasks by following Constraint (5). For instance, if a comparison task t has already been allocated to worker node q , the strategy compares the numbers of allocated comparison tasks between node i and node q . If worker node i has fewer comparison tasks, task t is re-allocated to node i .

From these heuristic rules, an algorithm with detailed steps can be developed for actual data distribution.

5.2. Data distribution algorithm

Algorithm 1 shows our heuristic and task-driven data distribution strategy. It is self-explained.

Algorithm 1: Data Distribution Algorithm

Initial : Set \mathbb{U} of all unallocated pairwise comparison tasks;
Variable: Variable data set \mathbb{D} and node set \mathbb{C} , which are initially empty;

```

1 while Set  $\mathbb{U}$  of unallocated tasks is not empty do
2    $\mathbb{D} \leftarrow \phi$ ;  $\mathbb{C} \leftarrow \phi$ ; /* empty  $\mathbb{D}$  and  $\mathbb{C}$  */
3   Find all data files needed for these unallocated tasks;
4   Put these data files into set  $\mathbb{D}$ ;
5   For each file in set  $\mathbb{D}$ , count the unallocated tasks that
   require this file;
6   Sort set  $\mathbb{D}$  in descending order in terms of the counting
   numbers;
7   while Node set  $\mathbb{C}$  is empty do
8     Choose the first data file in set  $\mathbb{D}$ . Let  $d$  denote this file;
9     for All worker nodes in the system do
10      Find the set of nodes without file  $d$ ;
11      Put these nodes into set  $\mathbb{C}$ ;
12      for All nodes in set  $\mathbb{C}$  do
13        Find and mark those nodes with the fewest
        allocated tasks;
14      Remove all marked nodes from  $\mathbb{C}$ ;
15      for All nodes in set  $\mathbb{C}$  do
16        Find and mark those nodes with the fewest
        allocated files;
17      Remove all marked nodes from  $\mathbb{C}$ ;
18      if Set  $\mathbb{C}$  becomes empty then
19        Remove file  $d$  from set  $\mathbb{D}$ ;
20      for Each worker node  $i$  in set  $\mathbb{C}$  do
21        if node  $i$  is empty then
22          Distribute data file  $d$  to this node  $i$ ;
23          break; /* get out of this for loop */
24        else
25          Calculate the number of new comparison tasks
          that can be allocated to this node  $i$  by adding the
          file  $d$  (Rule 1);
26      if Data file  $d$  has not been distributed then
27        Sort  $\mathbb{C}$  in descending order in terms of the numbers
        from Line 25;
28        Distribute data file  $d$  to the first node in set  $\mathbb{C}$ ;
29        Allocate to this node all new tasks found in Line 25
        for this node;
30        Update set  $\mathbb{U}$  of unallocated tasks;
31      Reallocation: For comparison tasks introduced by adding
      data file  $d$  above but already allocated to other nodes
      before, re-allocate these tasks by following Rule 2.

```

In comparison with Hadoop's data distribution strategy, our solution has the following advantages. Firstly, Hadoop randomly distributes data items without considering the requirements from computation tasks. In this case, a large number of data files has to be relocated at runtime for the computing tasks, resulting in massive data movement and performance degradation (Section 3.2). However, our solution presented in this paper is computing tasks aware. It distributes data items in a way that all data items can be processed locally, indicating good data locality for all computation tasks. Secondly, static system load balancing is promised by using our data distribution strategy, which will be shown later in Section 5.4. Contrarily, Hadoop does not provide static task allocation solutions. To achieve a load balancing in Hadoop, again a huge amount of data has to be moved around among multiple machines.

5.3. Theoretic results

Theoretic analysis is conducted to derive a lower bound d_{\max} for $\max\{|D_1|, \dots, |D_N|\}$ as well as an insight to data availability and system reliability. These theoretic results are summarized in two theorems.

Theorem 1. For the constrained optimization problem defined in Eq. (10) for distributing M data files to N computing nodes, a solution $\max\{|D_1|, \dots, |D_N|\}$ has a lower bound d_{\max} as:

$$\max\{|D_1|, |D_2|, \dots, |D_N|\} \geq \frac{1}{2} \left(1 + \frac{\sqrt{4M^2 - 4M + N}}{\sqrt{N}} \right) \triangleq d_{\max}. \quad (11)$$

Proof. For M data files, the total number of comparison tasks in an all-to-all comparison problem is $M(M-1)/2$. Consider the following extreme scenario: if each worker node is allocated no more than $\lceil M(M-1)/(2N) \rceil$ comparison tasks (Eq. (5)), at least how many data items are needed to complete all comparisons. As each comparison task needs two different data items as characterized in Eq. (7), we have the following relationship:

$$\left(\frac{\max\{|D_1|, |D_2|, \dots, |D_N|\}}{2} \right) = \left\lceil \frac{M(M-1)}{2N} \right\rceil. \quad (12)$$

Solving the equation gives the result in Eq. (11). \square

Remark 1. The lower bound d_{\max} given in Eq. (11) of Theorem 1 is useful. It implies that a solution will not be lower than this lower bound. In other words, it is the theoretically lowest possible number of data files that need to be deployed to each of the nodes in order to meet the constraints of the data distribution problem. The actual solution obtained from an algorithm may be higher than this lower bound.

If a data file has only one copy in a distributed system, it will become inaccessible from anywhere if the node where the data file is stored fails. In this case, the distributed computing system for the all-to-all comparison problem is unreliable. Therefore, it is an essential reliability requirement to have at least two data copies stored at different nodes for each data file. The following theorem shows that this system reliability requirement is guaranteed by the data distribution approach presented in this paper.

Theorem 2. For the constrained optimization problem defined in Eq. (10) for distributing M data files to N computing nodes, the distribution strategy presented in this paper gives a solution that promises at least two data copies stored in different nodes for each data file.

Proof. If a data item stored in one worker node is not duplicated on another node, in order to meet the constraint in Eq. (7), all the other data items have to be stored in the same worker node. This implies that the worker node stores all data items. Considering the target in Eq. (10) is to save storage by making each node only store part of the whole data, storing all data in one node can be avoid. Therefore, at least two copies stored in different nodes are guaranteed for each data file. This completes the proof. \square

Remark 2. Hadoop provides three duplications for each of the data items. This is a heuristic to address the data availability and system reliability requirements. It has not been theoretically demonstrated to be an optimal setting for a specific objective function. It is also noted that users may change the default three replications to a different number, e.g., four or five, according to the requirements of specific systems and/or applications. The approach presented in this paper gives a solution that provides at least two replications of data items through our theoretical optimization framework. For

Table 2

Distribution of 6 files to 4 nodes. (Theoretic lower bound $d_{\max} = 4$ from Theorem 1.)

Node	Distributed data files	Allocated comparison tasks
A	0, 2, 3, 4	(0, 3) (0, 4) (2, 4) (3, 4)
B	0, 1, 4, 5	(0, 1) (1, 4) (1, 5) (4, 5)
C	0, 2, 3, 5	(0, 2) (0, 5) (2, 5) (3, 5)
D	1, 2, 3	(1, 2) (1, 3) (2, 3)

a specific distributed system or a specific application, if more than two copies of data are more appropriate, one may still use our theoretical optimization framework and implementation algorithm while adding an additional constraint on the lower bound of the number of data replications to make it more than two, e.g., three or four. This is also similar to Hadoops general user interface for configuring the number of data duplications.

5.4. Analysis of the data distribution strategy

To analyze the data distribution strategy, consider an example with 6 data files {0, 1, 2, 3, 4, 5} and 4 worker nodes {A, B, C, D}. Our data distribution algorithm presented above gives the solution shown in Table 2.

In the result shown in Table 2, each worker node is allocated similar number of comparison tasks, which indicates a good static load balancing for the distributed system. Moreover, only part of the whole data sets is stored in each worker node for storage saving.

6. Experiments

This section conducts experiments to demonstrate the effectiveness of our data distribution strategy. It includes both simulation studies and experiments in a real distributed computing system. The section begins with development of evaluation criteria. Then, it evaluates the behavior and performance of our data distribution strategy against these criteria.

6.1. Evaluation criteria and experimental design

Three criteria are used to evaluate our data distribution strategy: storage usage, task execution performance and scalability.

Storage Saving is one of the objectives of our data distribution strategy. The storage saving of our data distribution strategy is compared with that of the Hadoop one. The storage space required by Hadoop is variable as the number of data replications is adjustable. Thus, it is not a good baseline for storage performance comparisons. This paper evaluates storage savings against the storage required by distributing all data to everywhere.

Data Locality reflects the quality of data distribution and is an indicator of the computing performance. As comparison tasks are allocated based on the data distribution in our approach, the number of comparison tasks with good data locality can be measured after the data distribution. To show different levels of data locality between our and Hadoop's data strategies, different numbers of worker nodes and Hadoop data replications are tested in the experiments.

Execution Performance, characterized by the total execution time T_{total} of an all-to-all comparison problem, is the ultimate goal subject to the storage usage and other constraints. As shown in Eq. (9), T_{total} includes $T_{\text{distribution}}$ and $T_{\text{comparison}}$, which are affected by data distribution and storage savings. Comparisons are made on these time performance metrics between our and Hadoop's data distribution strategies.

Scalability is significant for large-scale distributed computing of all-to-all comparison problems with big data sets. Various scenarios will be investigated under different numbers of processors on the worker nodes in a real distributed system. This demonstrates the scalability of data distribution strategy.

Table 3

Storage and data locality of our approach and Hadoop with 3 data replications for $M = 256$ files on N nodes.

	N	4	8	16	32	64
$\max\{ D_1, \dots, D_N \} : \underline{d}_{\max}$ (Theorem 1)		129	91	65	46	33
	This paper	192	152	117	85	59
	Hadoop(3)	192	96	48	24	12
Storage saving (%):	This paper	25	41	54	67	77
	Hadoop(3)	25	63	81	91	95
Data locality (%):	This paper	100	100	100	100	100
	Hadoop(3)	56	48	28	14	7

6.2. Performance in storage saving and data locality

Consider a scenario with 256 data files and a set of storage nodes with the number of nodes ranging between 1 and 64. From the optimization problem in Eq. (10) for data distribution, the maximum value of the numbers of data items distributed to the worker nodes is used to characterize the storage usage from our data distribution strategy. The theoretic lower bound \underline{d}_{\max} in Theorem 1 is calculated for each of the N values.

6.2.1. Comparison with Hadoop's strategy with default 3 data replications

The first group of experiments compares our data distribution strategy with Hadoop's one with the default 3 data replications. The experimental results are tabulated in Table 3, which shows storage usage, storage saving and data locality for both our and Hadoop's data distribution strategies. The storage saving is calculated against the storage space required when distributing all data files to every node as many existing approaches do.

In comparison with the data distribution strategy to distribute all data to all nodes, it is seen from Table 3 that both our and Hadoop's data distribution strategies have significant storage savings for large-scale all-to-all comparison problems while the Hadoop one saves even more space. This implies less data distribution time, especially when the number of nodes becomes big. For a cluster of 64 nodes, the storage saving reaches three quarters (77%) from our data distribution strategy and even as high as 95% from the Hadoop one.

Though with a lower storage saving, our data distribution strategy achieves 100% data locality for all computing tasks, as clearly shown in Table 3. In comparison, the higher storage savings from Hadoop are achieved with a significant sacrifice of data locality. For example, for a cluster of 64 nodes, the data locality from Hadoop is as low as 7% compared to 100% from our data distribution strategy. Good data locality is particularly important for large-scale all-to-all comparison problems. It will reduce data movements among nodes at runtime for comparison task execution. Thus, it benefits the overall computing performance of the all-to-all comparison problem.

6.2.2. Comparison with Hadoop's strategy with increased data duplications

One may argue that manually increasing the number of data replications may solve the data locality problem in Hadoop's data distribution. However, there are no guidelines on how this number is set for an all-to-all comparison problem in a given distributed environment. Also, once set, this number becomes constant in the deployed environment, leading to inflexibility for a range of other all-to-all comparison problems. Furthermore, even if this number can be manually tuned every time, it does not fundamentally solve the data locality problem. In comparison, our data distribution strategy automatically determines the number of data replications with 100% data locality.

Table 4

Storage and data locality of our approach and Hadoop(variable x) for $M = 256$ files on N nodes, where the number (x) of data replications for Hadoop has to be tuned manually for each case to achieve a similar maximum number of files on a node.

	N	4	8	16	32	64
Setting of x in Hadoop(x)		3	6	9	12	15
$\max\{ D_1, \dots, D_N \} : \underline{d}_{\max}$ (Theorem 1)		129	91	65	46	33
	This paper	192	152	117	85	59
	Hadoop(x)	192	192	144	96	60
Storage saving (%):	This paper	25	41	54	67	77
	Hadoop(x)	25	25	44	64	77
Data locality (%):	This paper	100	100	100	100	100
	Hadoop(x)	56	52	38	20	26

To support these claims, the second group of experiments are conducted, in which we manually tune the number of data replications for Hadoop's data distribution strategy to the value that gives a similar maximum number of data files on a node to that of our distribution strategy. Therefore, a data file is replicated 6, 9, 12 and 15 times for a distributed system with 8, 16, 32 and 64 data nodes, respectively, as shown in Table 4. With these manually settings, the experimental results in Table 4 show that our data strategy behaves with better storage saving performance than the Hadoop one. With more storage usage than our data strategy, the Hadoop one still demonstrates very poor data locality. For example, for 64 data nodes, Hadoop's data distribution only achieves 26% data locality compared to 100% from our data locality.

6.3. Performance in execution time T_{total}

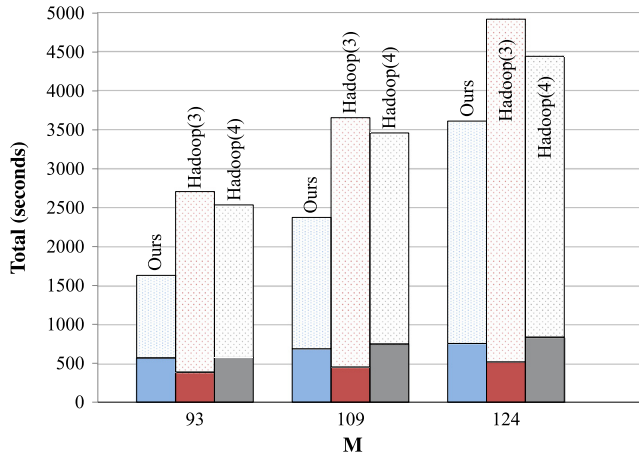
T_{total} is used to measure the execution performance of processing an all-to-all comparison. As shown in Eq. (9), it includes the time $T_{distribution}$ for data distribution and the time $T_{comparison}$ for comparison computations. All these time metrics are evaluated, and compared with those from Hadoop-based data distribution and distributed computing.

The settings for our experiments are as follows:

- A distributed system in real machines. A homogeneous Linux cluster was built with nine physical servers interconnected through 1 Gps Ethernet network. Among the nine servers, one node acts as the master and the remaining eight are worker nodes. All nine nodes use Intel(R) Xeon E5-2609 and 64 GB memory. They all run 64-bit Redhat Enterprise Linux.
- Experimental application. As a typical and significant all-to-all comparison problem in bioinformatics [7], the CVTree problem is chosen for our experiments. The computation of the CVTree problem has been recently investigated in single computer platforms [28,29]. It is further studied in this paper in a distributed computing environment. The problem has been re-programmed in our experiments by using the Application Programming Interfaces (APIs) provided by our distributed computing system. For comparison, a sequential version of the CVTree program is also developed for our experiments. As CVTree is a significant technique in bioinformatics, an additional benefit of using CVTree as an experimental example is to demonstrate a significant real-world application of all-to-all comparisons.
- Experimental data. A set of dsDNA files from the National Center for Biotechnology Information (NCBI) is chosen as the input data. The size for each file is around 150 MB and the total amount of data is over 20 GB.
- Experimental cases. To show that arbitrarily increasing the number of data replications is inefficient in achieving high computing performance for all-to-all comparison problems (discussed in Section 6.2), the Hadoop data strategy with

Table 5The CVTree problem for $N = 8$ nodes and different numbers of input data files (M).

M	$\max\{ D_1 , \dots, D_8 \}$	This paper	Hadoop(3)	Hadoop(4)
	d_{\max}			
93	34	53	35	59
109	39	63	41	69
124	45	71	47	78

**Fig. 3.** Comparisons of the T_{total} performance between our strategy and Hadoop's strategies (lower solid-filled part of the bars: $T_{distribution}$; upper dot-filled part of the bars: $T_{comparison}$).

different data replication numbers are used to compare with our approach. Table 5 clearly shows that when each data has 4 copies for Hadoop's data distribution (Hadoop(4)), the Hadoop data strategy distributes more data files to each worker node than our approach.

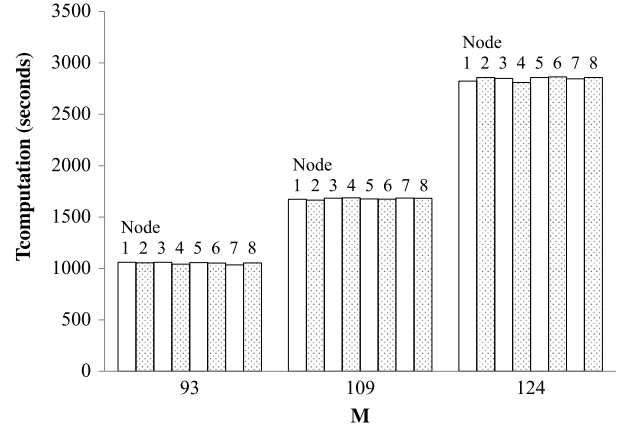
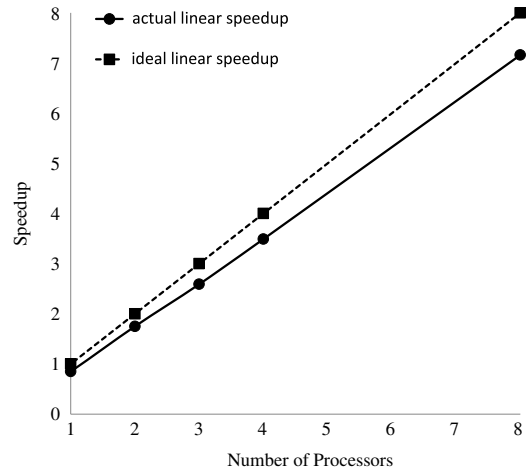
Let us consider the execution time of data distribution and distributed computing. For a given all-to-all comparison problem with 3 different sizes of data sets, both the data distribution time ($T_{distribution}$) and computation time ($T_{comparison}$) are measured, respectively. Adding up these two time measurements gives the total execution time T_{total} .

For comparison of the execution time performance between our and Hadoop-based approaches, it is clear that the Hadoop MapReduce computing framework is not suitable for supporting the all-to-all comparison computing pattern directly. Thus, our distributed computing task scheduling is integrated with Hadoop's data distribution strategy when the execution time performance is evaluated for Hadoop-based distributed computing.

Fig. 3 shows T_{total} for three different data distribution strategies: Ours, Hadoop(3) and Hadoop(4), under different M values. For each bar plot of T_{total} , the bottom and upper parts represent $T_{distribution}$ and $T_{comparison}$, respectively.

It is clearly seen from Fig. 3 that our data distribution strategy achieves much better T_{total} performance than the Hadoop one. It also confirms that for Hadoop's data distribution strategy, simply increasing the number of data replications from 3 to 4, which generates more data files on a node than our approach, does not improve much the T_{total} performance. It is noted from Fig. 3 that the $T_{distribution}$ performance resulting from our data distribution strategy is slightly poorer than that from Hadoop(3) but better than that from Hadoop(4). This is because our distribution strategy has less storage savings than Hadoop(3) but more storage savings than Hadoop(4) as shown in Table 5. More storage savings mean less time for data distribution.

To show the good load balancing from our data distribution strategy, Fig. 4 depicts $T_{comparison}$ performance measurements for each of the eight worker nodes under different M values. It is seen

**Fig. 4.** $T_{comparison}$ performance from our data distribution strategy for each of the worker nodes under different M values.**Fig. 5.** Speed-up achieved by the data distribution strategy and computing framework of this paper.

from the figure that for the same M value, $T_{comparison}$ for each of the worker nodes is very similar and well within the load balancing requirement from Eq. (5). The balanced tasks on each node all use local data without the need for data movements among the nodes through network communications.

6.4. Scalability

To support processing all-to-all comparison problems with big data sets, scalability is an important ability for our data distribution strategy. It is evaluated in the following experiments by using the speed-up metric.

Shown in Fig. 5 for up to eight worker nodes (plus a manager node) in our lab, this linear speed-up dotted line can be considered as an ideal speed-up.

Also shown in Fig. 5 is the actual speed-up achieved from our data distribution strategy. It shows that with the increase of the number of processors, our data distribution strategy behaves with a linear speed-up. This implies good scalability of the overall distributed computation. It is worth mentioning that although all-to-all comparison problems incur inevitable costs in network communications, extra memory demand and disk accesses, the data distribution strategy presented in this paper can achieve about 89.5% of the performance capacity of the ideal linear speed-up. The is measured by $7.16/8 = 89.5\%$ from the results shown in Fig. 5.

7. Conclusion

To address distributed computation of all-to-all comparison problems with big data, a scalable and efficient data distribution strategy has been presented in this paper. Driven by comparison task allocation, it is designed to minimize and balance storage usage in the distributed worker nodes while still maintaining load balancing and good data locality for all comparison tasks. Experiments have been conducted in real machines to demonstrate the proposed data distribution strategy for all-to-all comparison problems.

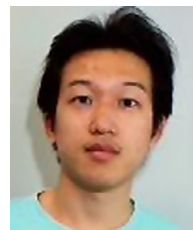
Contributions of the authors: Y.-F. Zhang conducted detailed research and experiments, and wrote the draft version of the paper. Y.-C. Tian designed the project, supervised the research, and re-wrote the paper. W. Kelly provided guidance on problem formulation and algorithm development. C. Fidge co-supervised the project and polished up the paper.

Acknowledgments

This work is supported in part by Shanxi Provincial Government of China under the International Collaboration Projects Scheme (Grant No. 2015081007), and the Talent and Technological Innovation Projects Grant Scheme (Grant No. 201605D211021), both to author Y.-C. Tian.

References

- [1] D. Gil, I.-Y. Song, Editorial: Third international workshop on modeling and management of big data (MoBiD14), *Future Gener. Comput. Syst.* 63 (2016) 96–99.
- [2] M. Gokhale, J. Cohen, A. Yoo, W. Miller, A. Jacob, C. Ulmer, R. Pearce, Hardware technologies for high-performance data-intensive computing, *Computer* 41 (4) (2008) 60–68.
- [3] M.L. Huang, T.-H. Huang, X. Zhang, A novel virtual node approach for interactive visual analytics of big datasets in parallel coordinates, *Future Gener. Comput. Syst.* 55 (2016) 510–523.
- [4] A. Skabar, K. Abdalgader, Clustering sentence-level text using a novel fuzzy relational clustering algorithm, *IEEE Trans. Knowl. Data Eng.* 25 (1) (2013) 62–75.
- [5] D.G. Perera, K.F. Li, Parallel computation of similarity measures using an fpga-based processor array, in: 22nd International Conference on Advanced Information Networking and Applications, Okinawa, 2008, pp. 955–962.
- [6] F. Giza-Belciug, S.-G. Pentiu, Parallelization of similarity matrix calculus in ontology mapping systems, in: RoEduNet International Conference – Networking in Education and Research, RoEduNet NER, 2015 14th, Craiova, 2015, pp. 50–55.
- [7] B. Hao, J. Qi, B. Wang, Prokaryotic phylogeny based on complete genomes without sequence alignment, *Modern Phys. Lett. B* 27 (4) (2013) 14–15.
- [8] A.K.C. Wong, E.-S.A. Lee, Aligning and clustering patterns to reveal the protein functionality of sequences, *IEEE/ACM Trans. Comput. Biol. Bioinform.* 11 (3) (2014) 548–560.
- [9] F.E. Faisal, H. Zhao, T. Milenkovic, Global network alignment in the context of ageing, *IEEE/ACM Trans. Comput. Biol. Bioinform.* 12 (1) (2015) 40–52.
- [10] Z. Ren, J. Wan, W. Shi, X. Xu, M. Zhou, Workload analysis, implications, and optimization on a production Hadoop cluster: A case study on Taobao, *IEEE Trans. Serv. Comput.* 7 (2) (2014) 307–321.
- [11] C. Doug, C. Mike, 2005. <http://hadoop.apache.org> (accessed: 26.05.16).
- [12] Y.-F. Zhang, Y.-C. Tian, C. Fidge, W. Kelly, A distributed computing framework for all-to-all comparison problems, in: The 40th Annual Conference of the IEEE Industrial Electronics Society, IECON'2014, Dallas, TX, USA, 2014, pp. 2499–2505.
- [13] S. Heitor, L. Guilherme, A distributed approach for a multiple sequence alignment algorithm using a parallel virtual machine, in: Proc. of the 25rd Eng. in Medicine and Biology Annual Conference, IEEE Engineering in Medicine and Biology Society, Shanghai, China, 2005, pp. 2843–2846.
- [14] E. Macedo, A. Melo, G. Pfisterer, A. Boukerche, Hybrid MPI/OpenMP strategy for biological multiple sequence alignment with DIALIGN-TX in heterogeneous multicore clusters, in: Proceedings of the IEEE International Parallel and Distributed Workshops and PhD Forum, IPDPSW, Shanghai, 2011, pp. 418–425.
- [15] X. Meng, V. Chaudhary, A high-performance heterogeneous computing platform for biological sequence analysis, *IEEE Trans. Parallel Distrib. Syst.* 21 (9) (2010) 1267–1280.
- [16] S. Xiao, H. Lin, W. Feng, Accelerating protein sequence search in a heterogeneous computing system, in: IEEE International Parallel and Distributed Processing Symposium, Anchorage, AK, 2011, pp. 1212–1222.
- [17] J. Hill, M. Hambley, T. Forster, M. Mewissen, T.M. Sloan, F. Scharinger, A. Trew, P. Ghazal, SPRINT: A new parallel framework for R, *BMC Bioinformatics* 9 (1) (2008) 558–559.
- [18] S. Gunturu, X. Li, L. Yang, Load scheduling strategies for parallel DNA sequencing applications, in: Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications, Seoul, 2009, pp. 124–131.
- [19] NCBI, Blast: Basic local alignment search tool, 1990. <http://blast.ncbi.nlm.nih.gov/Blast.cgi> (accessed: 26.05.16).
- [20] K. U. B. Center, Multiple sequence alignment by clustalw, 2007. <http://www.genome.jp/tools/clustalw/> (accessed: 26.05.16).
- [21] U. Catalyurek, E. Stahlberg, R. Ferreira, T. Kurc, J. Saltz, Improving performance of multiple sequence alignment analysis in multi-client environments, in: Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS02, Washington, DC, USA, 2002, p. 183.
- [22] L. Jiang, L. Xu, H. Cai, Z. Jiang, F. Bu, B. Xu, An IoT-oriented data storage framework in cloud computing platform, *IEEE Trans. Ind. Inf.* 10 (2) (2014) 1443–1451.
- [23] L. Barolli, F. Xhafa, Jxta-overlay: A P2P platform for distributed, collaborative, and ubiquitous computing, *IEEE Trans. Ind. Electron.* 58 (6) (2011) 2163–2172.
- [24] E. Elnikety, T. Elsayed, H. Ramadan, iHadoop: Asynchronous Iterations for MapReduce, in: Third IEEE International Conference on Cloud Computing Technology and Science, Athens, 2011, pp. 81–90.
- [25] L. Neumeyer, B. Robbins, A. Nair, A. Kesari, S4: Distributed stream computing platform, in: IEEE International Conference on Data Mining Workshops, ICDMW, Sydney, NSW, Australia, 2010, pp. 170–177.
- [26] X. Zhu, K. Li, A. Salah, L. Shi, K. Li, Parallel implementation of MAFFT on CUDA-enabled graphics hardware, *IEEE/ACM Trans. Comput. Biol. Bioinform.* 12 (1) (2015) 205–218.
- [27] O. Trelles, M. Andrade, A. Valencia, E.L. Zapata, J.M. Carazo, Computational space reduction and parallelization of a new clustering approach for large groups of sequences, *Bioinformatics* 14 (5) (1998) 439–451.
- [28] A.P.D. Krishnajith, W. Kelly, R. Hayward, Y.-C. Tian, Managing memory and reducing I/O cost for correlation matrix calculation in bioinformatics, in: Proc of the IEEE Symp on Computational Intelligence in Bioinformatics and Computational Biology, Singapore, 2013, pp. 36–43.
- [29] A.P.D. Krishnajith, W. Kelly, Y.-C. Tian, Optimizing I/O cost and managing memory for composition vector method based correlation matrix calculation in bioinformatics, *Curr. Bioinform.* 9 (3) (2014) 234–245.
- [30] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, M.J. West, Scale and performance in a distributed file system, *ACM Trans. Comput. Syst.* 6 (1) (1988) 51–81.
- [31] J.S. Torres, I.B. Espert, A.T. Dominguez, V.H. Garcia, I.M. Castello, J.T. Gimenez, J.D. Blazquez, Using GPUs for the exact alignment of short-read genetic sequences by means of the burrows-wheeler transform, *IEEE/ACM Trans. Comput. Biol. Bioinform.* 9 (4) (2012) 1245–1256.
- [32] P. Chen, C. Wang, X. Li, X. Zhou, Accelerating the next generation long read mapping with the fpga-based system, *IEEE/ACM Trans. Comput. Biol. Bioinform.* 11 (5) (2014) 840–852.
- [33] C. Moretti, H. Bui, K. Hollingsworth, B. Rich, P. Flynn, D. Thain, All-pairs: An abstraction for data-intensive computing on campus grids, *IEEE Trans. Parallel Distrib. Syst.* 21 (2010) 33–46.
- [34] E.C. Marc, W.P. Matthew, M. Scott, H. Lynette, Nephel: genotyping via complete composition vectors and MapReduce, *Source Code Biol. Med.* 6 (2011) 13.
- [35] M. Schatz, Cloudburst: highly sensitive read mapping with MapReduce, *Bioinformatics* 25 (2009) 11.



Yi-Fan Zhang received the Ph.D. degree in information technology in 2016 from Queensland University of Technology, Brisbane QLD, Australia. He is a research assistant at the School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane QLD, Australia. His research interests include distributed computing, big data processing, and reliability of safety-critical software systems. Contact him at ivanbuaa@hotmail.com.



Yu-Chu Tian received the Ph.D. degree in computer and software engineering in 2009 from the University of Sydney, Sydney NSW, Australia, and the Ph.D. degree in industrial automation in 1993 from Zhejiang University, Hangzhou, China. He is currently a Professor of Computer Science at the School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane QLD, Australia. His current research interests include big data processing, distributed and cloud computing, real-time computing, computer networks, and control theory and engineering. He has published a monograph and more than 200 refereed journal and conference papers, and is the holder of a patent. He is the editor-in-chief of Springer's book series *Handbook of Real-Time Computing*, and an Associate Editor for Elsevier's *Information Sciences* journal and Wiley's *Asia-Pacific Journal of Chemical Engineering*. Contact him at y.tian@qut.edu.au.



Wayne Kelly received the Ph.D. degree in computer science in 1996 from the University of Maryland at College Park, Maryland, USA. He is currently a Senior Lecturer at the School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane QLD, Australia. His research interests include parallel computing, cluster computing, grid computing, high-performance computing, peer-to-peer (P2P) computing and Web services. Contact him at w.kelly@qut.edu.au.



Colin Fidge received the Ph.D. degree in computer science in 1990 from Australian National University, Canberra ACT, Australia. He is currently a Professor at the School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane QLD, Australia. His research interests include information security evaluation of computer software, security engineering of industrial control systems, and risk-aware business process modeling and analysis. He is an associate editor for a number of journals. Contact him at c.fidge@qut.edu.au.