A Distributed Computing Framework for All-to-All Comparison Problems

Yi-Fan Zhang, Yu-Chu Tian, Wayne Kelly and Colin Fidge
School of Electrical Engineering and Computer Science
Queensland University of Technology
GPO Box 2434, Brisbane, QLD 4001, Australia.
Email: y.tian@qut.edu.au.

***Abstract*—Distributed computation and storage have been widely used for processing of big data sets. For many big data problems, with the size of data growing rapidly, the distribution of computing tasks and related data can affect the performance of the computing system greatly. In this paper, a distributed computing framework is presented for high performance computing of All-to-All Comparison Problems. A data distribution strategy is embedded in the framework for reduced storage space and balanced computing load. Experiments are conducted to demonstrate the effectiveness of the developed approach. They have shown that about 88% of the ideal performance capacity can be achieved in multiple machines through using the approach presented in this paper.**

I. Introduction

Computing applications with very large data sets can be a great challenge. For these big data problems, huge amounts of data need to be stored, accessed and analysed within a limited period of time, demanding significant computation and storage resources. While distributed computing systems can provide the ability to solve large scale applications in general, new techniques need to be developed to improve the performance for storage and processing of big data sets.

Among various big data problems, All-to-All Comparison Problems represent a typical computation pattern with the focus on processing a huge number of large- or small-size data files. In a general formulation of an all-to-all comparison problem, each file within a data set needs to be compared with all the others. Such comparisons are important in different domains such as bioinformatics, biometrics and data mining. For example, in bioinformatics, many researchers such as Wang [1] and Hao et al. [2] focus on inferring phylogenetic relationships by comparing gene sequences of different species. In biometrics, a typical type of problem is to identify people's physical characteristics by pairwise comparisons of different data stored in biometrics databases [3].

There have been several efforts to address All-to-All Comparison Problems [4, 5, 6, 7, 8, 9]. A main approach is to provide solutions for special-purpose All-to-All Comparison Problems only. Implementations of these solutions are dependent on specific runtime libraries, system architectures and comparison algorithms, and are applicable to a limited scale of data sets only. This makes the solutions and their implementations too complicated to be reused, implying a lack of flexibility to support big data processing.

Another approach is to use computing frameworks designed for distributed computing [10, 11, 12, 13]. Providing simple programming interfaces for users to develop their applications, these frameworks hide implementation issues such as data distribution, load balancing, data locality and fault tolerance. They enable programmers to focus on the application domain without the need to consider complicated parallel computing details. Among the existing frameworks, Hadoop [14] is popularly used to support the MapReduce computation pattern, but is inefficient in processing of All-to-All Comparison Problems. All-pairs [15] is designed for All-to-All Comparison Problems in a Campus Grid, but its application range is limited due to its brute-force data storage strategy, which stores all the data on all the worker nodes.

In this paper, we present a new framework for processing All-to-All Comparison Problems with large data sets. The main contributions of this paper include: 1) A data distribution strategy is developed, which considers disk utilization, data balancing and load balancing; and 2) A computing framework embodying the above data distribution strategy is also developed. Experimental studies are designed to evaluate the performance of the data distribution strategy and the distributed computing framework.

The paper is organized as follows. Section II discusses related work and motivations. Section III describes the All-to-All Comparison Problem and its challenges. A distributed computing framework and a data distribution strategy are presented in Sections IV and V, respectively. Experiments are conducted in Section VI to demonstrate our approach. Finally, Section VII concludes the paper.

II. 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) and also in studying phylogenetic diversity in protein families [16]. In general, big data processing in these problems include calculation of a cross-similarity matrix between each pair of the data sequences [17, 18]. This calculation is followed by several data grouping stages.

Various distributed computing systems and runtime libraries have been used to process All-to-All Comparison Problems. Heitor and Guilherme [4] have proposed a methodology for parallelizing 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 is provided and a special attention is paid to to execution of this multiple sequence comparison algorithm in parallel. Macedo et al. [5] have 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 are compared to show the appropriate choice. All these approaches assume that each of the nodes in the cluster stores all the data, leading to inefficiency that our work aims to avoid.

A heterogeneous computing platform is presented by Meng and Chaudhary [6] for high-throughput biological sequence analysis through a Message Passing Interface (MPI) enabled enterprise computing infrastructure. To achieve load balancing, the workload is distributed based on the hardware configuration. The whole database is split into multiple nearly-equal sized fragments, and each computing node is assigned a number of database fragments depending on its processing capacity. Although each node stores only part of the data set, data transmissions among different nodes cannot be avoided at runtime.

A method of design and optimization for the BLAST algorithm has been presented in a GPU-CPU mixed heterogeneous computing system [7]. Due to the specific GPU architecture, the implementation of the method has shown a six-fold speed-up for the BLAST program. In comparison, the approach to be presented in this paper does not rely on special hardware.

Load balancing has also been addressed in different ways. Parallel All-to-All Comparisons of genome sequences are considered by Hill et al. [8]. The main focus of this work is on load balancing among the clusters by dividing the comparison matrix into rows and assigning these rows to different nodes dynamically. Gunturu [9] has proposed a load scheduling strategy, which depends on the length of the sequence and the number of the processors in the network. These approaches have assumed that all the processors in the network already have both sequences to be compared in their local memory.

Recently, efforts have been made to develop computing frameworks for All-to-All Comparison Problems. All-pairs is presented by Moretti et al. [15] for data-intensive computing on campus grids. It provides an abstraction to users to deal with All-to-All Comparison Problems. Giving each comparison task the required data, this method employs a spanning tree method to deliver all the data to every node efficiently.

A scalable and complete genotyping system is proposed by Marc et al. [19] to produce genotypes from a variety of genome data. This suite of tools is built on top of Hadoop and MPI to support multiple nodes and large data sets. CloudBurst [20] uses Hadoop for parallel short read-mapping, which is used in a variety of biological analyses including SNP discovery, genotyping, and personal genomics. The running time of CloudBurst is shown to scale linearly with the number of reads mapped, and with 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.

Despite significant developments in processing All-to-All Comparison Problems, technical gaps exist in this area. Existing solutions mainly focus on parallelizing different All-to-All Comparison algorithms and providing load balancing, but none of them have paid special attention to data distribution. Most previous works have assumed that all data can be stored in each worker node, implying poor scalability for big data sets. While some of them use distributed file systems such as HDFS in Hadoop, the data distribution strategies are nonetheless inefficient for All-to-All Comparison Problems. To address these challenges, this paper presents a new computing framework for All-to-All Comparison Problems. A specific data distribution strategy is embedded into the framework to support efficient big data processing.

III. Problem Statement

Let A, C and M denote the data set to be pairwise compared, the comparison function on A and the output similarity matrix of A, respectively. Characterized by the Cartesian product or cross join of the data set A, the All-to-All Comparison Problem discussed in this paper is mathematically stated as follows:

$$M[i,j] = C(A[i], A[j]),\ i,j = 1,2,...,|A| \tag{1}$$

where $|A|$ means the size of A.

The original motivation of studying this problem came from bioinformatics, where the elements being pairwise compared are typically genes or genomes [1, 2]. The same comparison pattern also applies in many other domains, e.g., comparing image, video and audio in biometrics and data mining [3].

A. Data Distribution for All-to-All Comparisons

To design our computing framework with data distribution for all-to-all comparisons, some assumptions are made as follows, which are either system configurations or features of the computing problem:

1) The computing framework running on the distributed cluster has a master node and multiple worker nodes with the same computing power.
2) Each of the comparison operations can be performed independently from the others.
3) The framework is designed to deal with the general scenario that all the data should be pre-deployed to the system first and then all the comparison tasks are executed by all the worker nodes.
4) For the large number of data files involved in the All-to-All Comparison Problems, each of the files needs to be processed as a whole and thus is not split in this paper. All the files are assumed to have the same size.

Consider a scenario with M data files in the data set and N worker nodes in the system. There are $Q = M(M-1)/2$ comparison tasks. The following question must be answered: *How to distribute the M data files and schedule Q comparison tasks to N worker nodes so as to minimize data movement and maximize data locality while achieving good load balance?*

Before introducing our approach, we first outline two relatively simple and extreme data distribution solutions, which have been widely used in various applications: Strategy I: Pre-distribute every input file to every available worker node; and Strategy II: pre-distribute each input file to a (relatively small) fixed number of worker nodes.

In developing our approach for data distribution and distributed computing, we consider the following issues: 1) the time required to pre-distribute input files and the amount of network traffic generated by this process; 2) the amount of storage space required on each worker node; 3) the number of communicating machines required for the comparisons; and 4) the load balance when executing the comparison tasks.

Strategy I behaves worse than any other strategies in terms of the time consumption and network traffic during the pre-distribution, and the amount of storage space required on each worker node. However, it does not require communications between machines for the comparison tasks, i.e., each comparison task can be performed on any machine. The comparison tasks can be easily allocated evenly for load balancing.

Strategy II is basically the Hadoop strategy, which replicates each file a small number of times (typically 3) for fault tolerance. The time consumption and network traffic generated during the pre-distribution, and the amount of storage space required on each worker node, are smaller than other strategies. But this strategy makes it impossible to schedule comparison tasks to worker nodes such that all the required input files are presented locally. Thus, considerable communications between workers will be required during the comparison phase.

It can be seen from the above analysis that it is important to reduce the storage usage and keep load balancing. Firstly, for a distribution system with limited network bandwidth, reducing the storage usage will save the time spent on distributing data sets greatly. Secondly, storing fewer data files on each worker node will support distributed processing of very large data sets, enabling a wider range of applications of data distribution strategy. Thirdly, system load balancing means the computing power of all the worker nodes can be fully utilized.

Instead of the above Strategies I and II, a new data distribution strategy is developed in this paper. It has the advantages of both extreme Strategies I and II. It pre-distributes files to worker nodes such that the usage of storage space for each worker node is balanced and minimized. Meanwhile, all comparison tasks can be performed without the need of communications between the worker nodes. Moreover, all the comparison tasks can also be allocated to the worker nodes to ensure a good load balance.

For N worker nodes in the distributed system, let S_k represent the usage of storage space of node k. The objective of our data distribution strategy can be expressed as follows:

$$\text{Minimize} \max\{S_1, S_2, ..., S_N\}. \tag{2}$$

It is assumed that all data files have the same size s. Let D_k and $|D_k|$ denote the set and size of files distributed to worker node k, respectively. We have $S_k = |D_k|\, s$. Thus, Equation (2) means to minimize the number of files stored in each node:

$$\text{Minimize} \max\{|D_1|, |D_2|, ..., |D_N|\}. \tag{3}$$

As a constraint to this optimization, a data distribution strategy is required for data set A such that any two elements x and y appear on at least one worker node k:

$$\forall x, y \in A, \exists k, x \in D_k \wedge y \in D_k. \tag{4}$$

The above mentioned Strategy I meets this condition. So, in addition, a data distribution strategy is sought to minimize the maximum usage of storage space for each worker node.

When scheduling the comparison task that compares data items x and y, let set P include all the possible choices of worker node to do a certain comparison task:

$$P_{xy} = \{k \mid x \in D_k \wedge y \in D_k\}. \tag{5}$$

This set will always contain at least one element due to Condition (4). If it contains more than one element, then our distribution strategy needs to choose one of them.

Let $C(x, y)$ denote the comparison task for data x and data y, T represents all comparison tasks and T_k and $|T_k|$ denote the set and number of tasks performed by worker node k, respectively. Then, good data locality for all comparison tasks can be expressed as follows:

$$\forall C(x, y) \in T, \exists i, x \in D_i \wedge y \in D_i \wedge C(x, y) \in T_i. \tag{6}$$

As mentioned previously, for a scenario with N worker nodes and M data files, there are $Q = M(M-1)/2$ comparison tasks in total, which need to be scheduled on the N nodes. Our strategy is designed to achieve this in a manner of good load balance. If all comparison tasks are assumed to consume the same amount of time t and all the worker nodes have the same computing power, load balancing can be represented statically by a requirement to allocate comparison tasks to nodes in a way that satisfies the following bound:

$$\forall T_k \in \{T_1, \cdots, T_N\}, |T_k| \le \lceil M(M-1)/(2N) \rceil. \tag{7}$$

With the overall aim in Equation (3), our data distribution strategy enables actual comparison through Equations (6) and (7) as constraints. The constraints in Equations (6) and (7) highlight the requirements of good data locality and load balance, respectively. With consideration of both storage usage and related comparison tasks, our data distribution strategy provides a solution to distributing data sets and allocating comparison tasks at the same time.

B. Challenges of the Data Distribution Problem

The problem of distributing data and related comparison tasks can be treated as a classic combinatorial problem: to place M objects into N boxes. This distribution problem has the following characteristics:

1) All the comparison tasks are distinguishable. For All-to-All comparison problems, each comparison task is different for processing different data pairs.

2) All the worker nodes are indistinguishable. Generally, for homogeneous distributed computing systems as discussed in this paper, all worker nodes are assumed to have the same processing power and storage space, and thus can be treated as indistinguishable.

We aim to allocate $Q = M(M-1)/2$ distinguishable comparison tasks to N indistinguishable worker nodes. From combinatorial mathematics, the total number of solutions is expressed by the Stirling number [21]:

$$S(Q,N) \quad = \text{the number of task distribution solutions.} \tag{8}$$

The Stirling number of the second kind $S(Q,N)$ counts the number of ways to partition a set of Q distinguishable elements into N non-empty subsets. It has the following properties for $Q \geq 1$ and $N \geq 1$:

$$S(0,0) = 1, S(Q,0) = 0, S(0,N) = 0,$$
$$S(Q,N) = NS(Q-1,N) + S(Q-1,N-1).$$

For a special and simple case of $N = 2$, we have

$$S(Q,2) = 2^{Q-1} - 1. \tag{9}$$

This is graphically shown in Figure 1. The trend depicted in Figure 1 indicates too many possible distribution solutions even for a very simple case of $S(Q,2)$. This implies that it is generally impossible to evaluate all possible solutions to find the best answer in a reasonable period of time. Therefore, development of heuristic solutions is necessary.

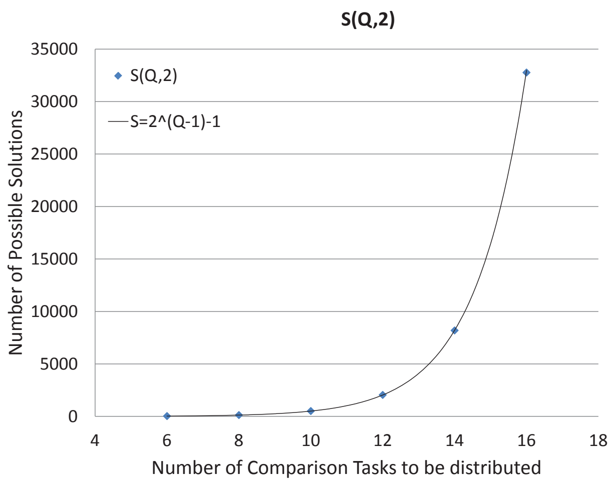

Fig. 1. The growth trend of solution space for $S(Q,2)$.

IV. Computing Framework Design

Before designing the data distribution strategy, a distributed computing framework is presented in Figure 2. The data distribution strategy is integrated into the framework to support processing of All-to-All Comparison Problems.

The computing framework has three main components: data manager, job tracker and job executor. The Data Manager calculates the data distribution solution based on the data distribution strategy. Different data sets are pre-deployed to the data storage of each worker node before computations begin. The Job Tracker collects all the data information from the data manager and schedules each comparison task to the worker node that stores all the necessary data files. The Job Executor executes the allocated comparison tasks and processes data files stored in local data storage. The comparison results are collected in a result matrix for further processing.

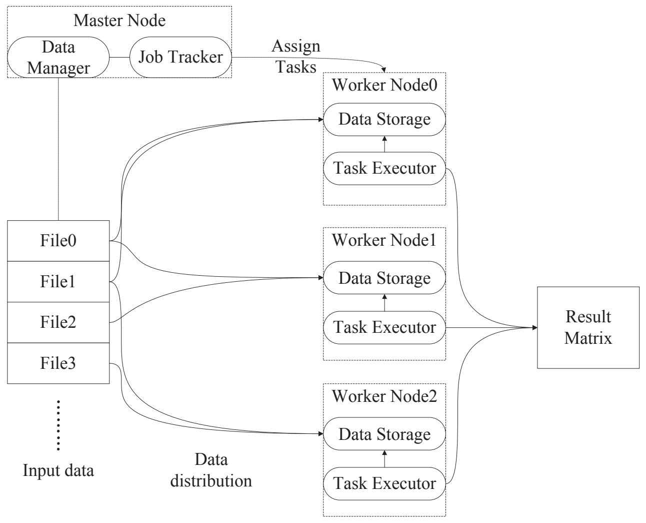

Fig. 2. Architecture of the Framework for All-to-All Comparison Problems.

V. Data Distribution Strategy

This section presents a data distribution strategy to achieve the objective in Equation (3) subject to the constraints shown in Equations (6) to (7).

A. *Description of Data Distribution Strategy*

According to the analysis in Section III, the data distribution strategy should satisfy the following requirements:

1) To minimize and balance the usage of storage space for all the worker nodes. For the distributed system, the number of data files stored in each worker node should be minimized to reduce the disk utilization and data distribution time. To balance the storage usage, each node should store similar numbers of data files. To meet these requirements, Equation (3) must be satisfied.
2) To balance the comparison tasks among different nodes in the distributed system. Avoiding extra data transmission during computation is an effective way to shorten the overall execution time. In order for each comparison task to have good data locality, Equation (6) needs to be met. In addition, to achieve load balance for the distributed system before the computation, the data distribution strategy should allocate each node a similar number of comparison tasks as shown in Equation (7).

Meeting the above requirements, our data distribution strategy delivers a number of benefits. The benefits include reduced time for data distribution, minimized number of data files in each node, balanced data and load in the distributed system, and avoided extra data transmission between different nodes when executing comparison tasks.

B. Data Distribution Algorithm

From the above descriptions, Algorithm 1 was developed to implement our data distribution strategy. The basic idea behind this algorithm is that when a data file d is distributed to a worker node k that already has data files $D = \{1, 2, \cdots, p\}$, the worker node k becomes available to execute comparison tasks $T = \{(d, u) | u \in D\}$. For a data set of M items and N worker nodes, the number of comparison tasks is $Q = M(M-1)/(2N)$.

To achieve the minimization in Equation (3), the algorithm applies the following rules during the distribution process:

- A data file d is distributed to a worker node that has the least number of data files and the least number of comparison tasks;
- The data file d is distributed to a specific worker node k only when at least one comparison task c involving d can be allocated to this node;
- A comparison task c is allocated to an available worker node with the least number of comparison tasks.

Application of these three rules enables all worker nodes to get a balanced and minimized number of data files while keeping the constraints of Equations (6) and (7).

C. Data Distribution Strategy Design

For a data set D of n items and m storage nodes from set S, we can get all the comparison tasks needed to be allocated U (line 6) and the maximum number of comparison tasks Q for each node (line 5). The set D_i is initialized to save all the data files stored in node i (line 3) and the set T_i is initialized to save all the comparison tasks allocated to node i (line 4).

The data distribution strategy consists of the following steps:

1) Choose a data file d that is needed for the unallocated pairwise comparison tasks (line 8).
2) Choose a set of storage nodes C that store the least number of data files and also have been allocated the least number of comparison tasks (lines 10 and 11).
3) Allocate data file d to a storage node m in set C (lines 10-29). The newly added data file d will generate new comparison tasks (line 17) and at least one new comparison task will be allocated to that node (line 21). Besides this, if the chosen node m is empty, allocate data file d to it directly (line 32).
4) If data file d cannot be distributed to any of the nodes in set C, choose another data file in set I and repeat steps 1 to 3.
5) Repeat steps 1 to 4 until all the pairwise comparison tasks in set U are fully allocated.

VI. Experiments

This section describes experiments conducted to evaluate the performance of the data distribution strategy and the distributed computing framework proposed in this paper.

Algorithm 1 Data Distribution Strategy

```
Input:
1:  $m$, the number of storage nodes;
2:  $n$, the number of all data files;
Process:
3:  $D_i = \{\}, i = 1, 2, ..., m$;
4:  $T_i = \{\}, i = 1, 2, ..., m$;
5:  $Q = \lceil n * (n-1)/(2 * m) \rceil$;
6:  $U = \{(i, j) \mid \forall (i, j), i < j, i, j = 1, 2, ..., n\}$;
7:  while $|U| > 0$ do
8:      $I = \{i \mid \exists j, (i, j) \in U, j = 1, 2, ..., n\}$;
9:      for each $d \in I$ do
10:         $S = \{i \mid |D_i| = \min |D_j|, j = 1, 2..., m\}$;
11:         $C = \{i \mid i \in S \wedge (|T_i| = \min |T_j|) \wedge d \notin D_i, j \in S\}$;
12:         if $|C| > 0$ then
13:             for each $m \in C$ do;
14:                 if $|D_m| > 0$ then
15:                     Findnode = false;
16:                     $G = \{(d, i) \mid i \in D_m\}$;
17:                     for each $t \in G$ do
18:                         if $|T_m| < Q \wedge t \in U$ then
19:                             $T_m = T_m \cup \{t\}$;
20:                             $U = U - \{t\}$;
21:                             Findnode = true;
22:                     if Findnode then
23:                         if $|T_m| < Q$ then
24:                             for each $t \in G$ do
25:                                 if $|T_m| < Q \wedge t \notin T_m$ then
26:                                     if $t \in T_k \wedge |T_k| > |T_m|$ then
27:                                         $T_k = T_k - \{t\}$;
28:                                         $T_m = T_m \cup \{t\}$;
29:                     $D_m = D_m \cup \{d\}$;
30:                     break;
31:                 else
32:                     $D_m = D_m \cup \{d\}$;
33:                     break;
```

A. Job Scheduling Strategy

For All-to-All Comparison Problems, this paper proposes to place the input data sets into storage nodes before the comparison computing begins. After the data sets are distributed to the nodes, a job scheduling strategy is required to choose suitable worker nodes for all the comparison tasks.

In our computing framework, the data distribution strategy designed in Section V has already considered static load balancing. An initial allocation of a balanced number of tasks to each of the worker nodes enables each node in the distributed computing system to process a balanced number of comparison tasks with good data locality. For example, for 4 data files $\{0, 1, 2, 3\}$ and 3 worker nodes $\{A, B, C\}$, our data distribution strategy gives a solution shown in Table I.

TABLE I
Distribution of 4 data files to 3 worker nodes.

Worker node	Distributed data files	Allocated comparison tasks
A	0,1,3	(0,1) (0,3)
B	1,2,3	(1,2) (1,3)
C	0,2,3	(0,2) (2,3)

B. Performance of the Data Distribution Strategy

In this experiment, we consider a data set with 256 files and a number of storage nodes between 2 and 64. To evaluate the performance of the data distribution strategy, Strategies I and II discussed in Section III are compared with ours presented in this paper. The results appear in Tables II and III. Table II shows how much storage space our data distribution strategy can save with the growth of the number of the storage nodes, while Table III shows how many comparison tasks cannot have good locality for these three strategies.

TABLE II
STORAGE USAGE: THE NUMBER OF FILES STORED IN THE SYSTEM.

No. of storage nodes	Strategy I	This paper	Strategy II
2	512	512	512
4	1024	780	768
8	2048	1344	768
16	4096	2240	768
32	8192	3552	768
64	16384	5312	768

TABLE III
THE NUMBER OF TASKS WITHOUT GOOD DATA LOCALITY.

No. of storage nodes	Strategy I	This paper	Strategy II
2	0	0	0
4	0	0	0
8	0	0	5686
16	0	0	16559
32	0	0	23990
64	0	0	28211

In terms of the storage usage, it can be seen from Table II that the data distribution strategy proposed in this paper requires much less storage space than Strategy I does, which distributes all data to every node. This is particularly evident when the number of storage nodes becomes big.

As shown in Table II, the data distribution strategy proposed in this paper consumes more storage space than the Hadoop-based Strategy II does as expected. However, it maintains good data locality as shown in Table III, implying good computing performance in comparison computation because no data movements and communications are required over the network during the computation.

The space saving from Strategy II comes with a significant sacrifice of data locality. Table III shows that the number of comparison tasks without data locality is huge in Strategy II. For a system with 64 nodes, 28,211 tasks do not have data locality when Strategy II is applied. Comparison tasks without data locality will require data movements and communications over networks during the computation, leading to degraded overall performance of the computing problem.

C. Performance of the Distributed Computing Framework

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

Let $time(n, x)$ denote the time required by an n-processor system to execute a program to solve a problem of size x. Then, $time(1, x)$ represents the time required by a sequential version of the program, the speed-up is measured as [22]:

$$speedup(n, x) = time(1, x)/time(n, x). \tag{10}$$

Generally, if we do not consider the communication overhead, load imbalance and extra computation, the system can get linear speed-up [23]. This linear speed-up is shown in Figure 3, and can be considered as an ideal speed-up.

We have conducted our experiments on a homogeneous Linux cluster with 5 servers: 1 as master node and the remaining 4 as worker nodes, which all run 64-bit Redhat Enterprise Linux. All 4 worker nodes are limited to use one core and 20GB RAM.

As a typical All-to-All Comparison Problem in bioinformatics [2], the CVTree problem is chosen for our experiments. Its computation has been recently investigated for single computer platforms [17, 18]. It is re-programmed for our experiments in this paper by using the Application Programming Interfaces (APIs) provided in our distributed computing framework. A sequential version of the CVTree program was also developed for our experiments.

A set of dsDNA virus files from the National Center for Biotechnology Information (NCBI) [24] was chosen as input data. The results of our experiments are shown in Figure 3.

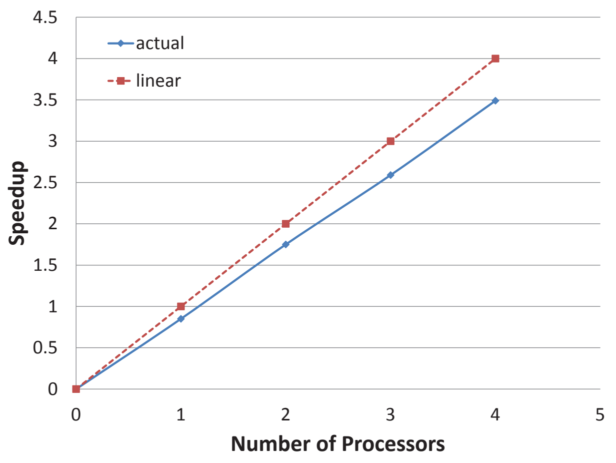

Fig. 3. Speed-up of our data distribution strategy and computing framework.

Figure 3 shows that with the number of processors growing, our computing framework has a linear speed-up, implying good scalability. Moreover, although All-to-All comparison problems incur inevitable costs in network communications, extra memory demand and disk access, the computing framework presented in this paper can achieve about 88% of the performance capacity of the ideal linear speed-up.

VII. Conclusion and Future Work

A new distributed computing framework has been presented for All-to-All Comparison Problems with big data. It is designed with an embedded data distribution strategy for minimizing and balancing the usage of storage space for the whole system while still maintaining good data locality for all comparison tasks. With our data distribution strategy, the computing framework further allocates comparison tasks to worker nodes with consideration of load balancing. Experiments have been conducted to demonstrate the effectiveness of the approach presented in this paper for All-to-All Comparison Problems with big data.

In the future, we will extend the data distribution strategy to support heterogeneous distributed computing systems, in which the worker nodes may have different computing power. Furthermore, dynamic job scheduling of comparison tasks will also be considered. In addition, comprehensive experiments of the computing framework will be conducted on large-scale distributed computing systems.

References

[1] W. Wang, "Composition vector methods for phylogeny," Ph.D. dissertation, The Chinese University of Hong Kong, 2009.

[2] B. Hao, J. Qi, and B. Wang, "Prokaryotic phylogeny based on complete genomes without sequence alignment," *Modern Phy. Lett.*, vol. 2, no. 4, pp. 14–15, 2003.

[3] P. Phillips, P. Flynn, T. Scruggs, K. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, "Overview of the face recognition grand challenge," in *Computer Society Conf on Computer Vision and Pattern Recognition (CVPR05)*, vol. 1, 2005, pp. 947 – 954.

[4] S. Heitor and 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 Ann Conf*, Shanghai, China, 2005, pp. 2843–2846.

[5] E. Macedo, A. Melo, G. Pfitscher, and A. Boukerche, "Hybrid MPI/OpenMP strategy for biological multiple sequence alignment with DIALIGN-TX in heterogeneous mluticore clusters," in *Proceedings of the IEEE International Paraeel and Distributed Workshops and PhD Forum (IPDPSW)*, 2011, pp. 418–425.

[6] X. Meng and V. Chaudhary, "A high-performance heterogeneous computing platform for biological sequence analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1267–1280, 2010.

[7] S. Xiao, H. Lin, and W. Feng, "Accelerating protein sequence search in a heterogeneous computing system," in *IEEE Int Parallel and Distributed Processing Symposium*, 2011, pp. 1212 – 1222.

[8] J. Hill, M. Hambley, T. Forster, M. Mewissen, T. M. Sloan, F. Scharinger, A. Trew, and P. Ghazal, "SPRINT: A new parallel framework for R," *BMC Bioinformatics*, vol. 9, no. 1, pp. 558–559, 2008.

[9] S. Gunturu, X. Li, and L. Yang, "Load scheduling strategies for parallel DNA sequencing applications," in *Proc of the 11th IEEE Int Conf on High Performance Computing and Communications*, 2009, pp. 124–131.

[10] L. Jiang, L. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu, "An iot-oriented data storage framework in cloud computing platform," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, May 2014.

[11] L. Barolli and F. Xhafa, "Jxta-overlay: A P2P Platform for Distributed, Collaborative, and Ubiquitous Computing," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 6, June 2011.

[12] E. Elnikety, T. Elsayed, and H. Ramadan, "iHadoop: asynchronous iterations for MapReduce," in *The 3rd IEEE Int Conf on Coud Computing Tech and Sci (CloudCom)*, Athens, Greece, N29 Nov - 1 Dec 2011.

[13] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: distributed stream computing platform," in *The 10th IEEE Int Conf on Data Mining Workshops (ICDMW 2010)*, Sydney, Australia, 14 Dec 2010.

[14] Hadoop, http://hadoop.apache.org, accessed: 3 July 2014.

[15] C. Moretti, H. Bui, K. Hollingsworth, B. Rich, P. Flynn, and D. Thain, "All-pairs: An abstraction for data-intensive computing on campus grids," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 33–46, 2010.

[16] O. Trelles, M. Andrade, A. Valencia, E. Zapata, and J. Carazo, "Computational space reduction and parallelization of a new clustering approach for large groups of sequences," *Bioinformatics*, vol. 14, no. 5, pp. 439–451, 1998.

[17] A. P. D. Krishnajith, W. Kelly, R. Hayward, and 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, 16-19 April 2013, pp. 36–43.

[18] A. P. D. Krishnajith, W. Kelly, and Y.-C. Tian, "Optimizing I/O cost and managing memory for composition vector method based correlation matrix calculation in bioinformatics," *Current Bioinformatics*, vol. 9, no. 3, pp. 234–245, 2014.

[19] E. C. Marc, W. P. Matthew, M. Scott, and H. Lynette, "Nephele: genotyping via complete composition vectors and mapreduce," *Source Code for Biology and Medicine*, vol. 6, p. 13, 2011.

[20] M. Schatz, "Cloudburst: highly sensitive read mapping with mapreduce," *Bioinformatics*, vol. 25, p. 11, 2009.

[21] H. W. Gould, "The q -stirling numbers of first and second kinds," *Duke Mathematical Journal*, vol. 28, no. 2, pp. 281–289, 1961.

[22] D. Mark, "What is scalability?" *ACM SIGARCH Computer Architecture News*, vol. 18, no. 4, pp. 18–21, 1990.

[23] K. Li, Y. Pan, H. Shen, and S. Zhang, "A study of average-case speedup and scalability of parallel computations on static networks," *Mathematical and Computer Modelling*, vol. 29, no. 9, pp. 83–94, May 1999.

[24] NCBI, "National Center for Biotechnology Information," http://www.ncbi.nlm.nih.gov/, accessed: 3 July 2014.