INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, e-ISSN 1841-9844, 14(2), 199-211, April 2019.

Optimal Data File Allocation for All-to-All Comparison in Distributed System: A Case Study on Genetic Sequence Comparison

L.X. Li, J. Gao, R. Mu

Leixiao Li
1. College of Computer and Information Engineering,
Inner Mongolia Agricultural University, Hohhot 010018, China
2. College of Data Science and Application,
Inner Mongolia University of Technology, Hohhot 010080, China
3. Inner Mongolia Autonomous Region Engineering & Technology
Research Center of Big Data Based Software Service, Hohhot 010080, China
llxhappy@126.com

Jing Gao*
College of Computer and Information Engineering,
Inner Mongolia Agricultural University, Hohhot 010018, China
*Corresponding author: gaojing@imau.edu.cn

Ren Mu
State key laboratory,
Beijing Jiaotong University, BeiJing 100044, China
568387304@qq.com

Abstract: In order to solve the problem of unbalanced load of data files in large-scale data all-to-all comparison under distributed system environment, the differences of files themselves are fully considered. This paper aims to fully utilize the advantages of distributed system to enhance the file allocation of all-to-all comparison between the data files in a large dataset. For this purpose, the author formally described the all-to-all comparison problem, and constructed a data allocation model via mixed integer linear programming (MILP). Meanwhile, a data allocation algorithm was developed on the Matlab using the *intlinprog* function of branch-and-bound method. Finally, our model and algorithm were verified through several experiments. The results show that the proposed file allocation strategy can achieve the basic load balance of each node in the distributed system without exceeding the storage capacity of any node, and completely localize the data file. The research findings can be applied to such fields as bioinformatics, biometrics and data mining.
Keywords: distributed system, all-to-all comparison, mix integer linear programming (MILP), file allocation, load balancing.

1 Introduction

The comparison between two random data files in a dataset is commonplace in bioinformatics, biometrics and data mining [1]. However, the all-to-all comparison between data files in a large dataset require special large scale computations. Previous solutions to such a problem mainly fall into two categories: those grounded on centralized computing or those using distributed computing based on centralized storage [2]. The former requires access to supercomputer resources [3], while the latter is bottlenecked by limited storage capacity and task delay caused by waiting for data transmission [4].

The distributed computing based on distributed storage provides an efficient, reliable and scalable solution to large-scale computing problems like all-to-all comparison. By distributed

computing, a large-scale problem is decomposed into several small problems, which are then handled separately on each node in a distributed system [5]. Nevertheless, the performance of distributed computing depends heavily on data allocation, task decomposition and task scheduling, and might be dampened by the irrational data allocation, poor data locality and computing load imbalance in the distributed system [6].

There are two data allocation strategies for all-to-all comparison of data files in a large dataset via distributed computing based on distributed storage, namely, allocating all input files to each computing node, and allocating a number of copies of each input files randomly to the system node (i.e. the Hadoop framework data allocation policy) [7–9]. Hadoop framework can not guarantee the load balancing of the comparison task calculation of each node. Storing all input files on each node is a common practice in centralized computing solutions, but it wastes storage and network resources [10] For the latter strategy, the computing performance is poor due to the frequency data movement between the nodes [11]. After all, Hadoop, as a general distributed computing framework, is not designed specifically for all-to-all comparison [1, 12].

To solve the above problems, this paper establishes a model to optimize the data file allocation optimization in all-to-all comparison of data files in a large dataset, puts forward a data file allocation algorithm and a task scheduling strategy, and verifies the proposed methods via experiments. The experimental results show that our model and algorithm can achieve data localization and load balancing of comparative files under the storage constraints of distributed system nodes, thus giving full play to the advantages of distributed system.

The remainder of this paper is organized as follows: Section 2 describes the all-to-all comparison problem and constructs a data file allocation model; Section 3 designs the data allocation algorithm; Section 4 performs the experimental verification and discusses the results; Section 5 wraps up this paper with several conclusions.

2 Problem description and data file allocation model

2.1 Problem description

All-to-all comparison refers to the multiple contrasts of all data files in a dataset. This problem can be illustrated by the graph in Figure 1, where each vertex is a data file to be compared and each edge is a comparison task between two data files. If m is the number of data files to be compared, then the total number of comparison tasks is $m(m-1)/2$. Hence, the all-to-all comparison problem can be expressed as a graph with m vertices and $m(m-1)/2$ edges.

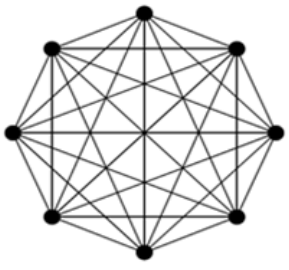

Figure 1: All-to-all comparison problem

In a distributed system with distributed storage, the all-to-all comparison of data files in a large dataset is implemented in two steps: First, all data files in the dataset are allocated to each computing node of the distributed system; then, the data files are compared in a pairwise manner.

Table 1: Symbol description

Number	Symbol	Symbolic description
1	m	Number of data files
2	n	Number of computing nodes in distributed system
3	s	M row 1 column matrix, representing the size of each file
4	$s_i, i = 1, 2, \ldots, m$	Size of i data file
5	u	N row 1 row matrix, representing the maximum storage limit of each node in a distributed system
6	$u_i, i = 1, 2, \ldots, n$	The storage capacity of computing nodesi
7	$w_{ij}, i = 1, 2, \ldots, m, j = 1, 2, \ldots, m$	The size of the task compares with file i and file j.
8	$c_{ij}, i = 1, 2, \ldots, m, j = 1, 2, \ldots, m$	Amount of calculation of the task compares with file i and file j.
9	$x_{kt}, k = 1, 2, \ldots, m(m-1)/2, t = 1, 2, \ldots, n$	whether or not allow the assignment of item K allocated to the T node
10	$W_{kt} = w_{ijt}$	Assign the size of item K to the T node (the file number corresponding to item K is i, j).
11	$C_{kt} = c_{ijt}$	The amount of computation assigned to item K of the T node (the file number corresponding to item K corresponds to i, j).
12	$taskno$	Task number
13	$av_$work	The average amount of tasks per computing node should be allocated in theory,
14	$deci$	Correspondence matrix between task and file
15	$result$	Task allocation result matrix
16	f	Objective function variable coefficient matrix
17	aeq	Coefficient matrix of equality constraint variables
18	beq	Equality constrained variable resource matrix
19	a	Inequality constraint coefficient matrix
20	b	Inequality constrained resource matrix
21	$intcon$	Integer variable subscript sequence number vector
22	LB	Lower limit of variable
23	UB	Upper limit of variable
24	$[X, Y]$	X is the best solution for obtaining the variable value. Y is the best solution.

Before allocating the data files, it is necessary to fully consider how the system performance is affected by node storage capacity, data transmission, network bandwidth, load balance and other factors. A desirable allocation strategy must satisfy the following conditions:(1) The data allocated to a node should not surpass the storage capacity of that node;(2) The two files to be compared on a node must be saved on that node to localize the data files for each comparison task;(3) The comparison tasks should be balanced among the computing nodes.

For simplicity, all the symbols used in this paper are listed in Table 1 below.

For better system speed and performance, our goal is to balance the comparison tasks among the computing nodes under the storage capacity of each node.

2.2 Data file allocation model

The above description shows that all-to-all comparison is a typical constrained optimization problem: maximizing or minimizing objective functions under multiple constraints. The most effective solution to constrained problem is linear programming (LP), which works well when the objective functions and constraints are all linear [13,14]. With a strong modelling ability, the LP is also a desirable way to tackle control and programming problems. The main idea of the LP is to find a control sequence that satisfies all constraints and minimizes the objective function [15].

In this paper, node storage is added to the constraints of the all-to-all comparison problem, and load balancing is treated as the objective function. All constraints were expressed as an equality or inequality. As mentioned before, our problem involves m data files and n nodes in a distributed system with distributed storage. For multiple comparisons of all data files, the m data files should be distributed rationally to those nodes to fulfill load balancing, data localization and node storage constraint.

As shown in Figure 1, the total number of comparison tasks of these data files can be expressed as:

$$C_m^2 = \frac{m(m-1)}{2} \tag{1}$$

Let $s_i(i = 1, 2, \cdots, m)$ be the size of each of the m data files. Then, the total size of the two files in each comparison task can be calculated as:

$$w_{ij} = s_i + s_j(i, j = 1, 2, \cdots, m, i < j) \tag{2}$$

Then, the computing load of the comparison task between file i and file j can be obtained as:

$$c_{ij}(i, j = 1, 2, \cdots, m, i < j) \tag{3}$$

Thus, the total number of multiple comparison tasks between the m data files can be described as:

$$\sum_{i=1}^{m} \sum_{j=i+1}^{m} c_{ij} \tag{4}$$

If the tasks are allocated equally to n nodes, then the theoretical mean number of tasks allocated to each computing node can be expressed as:

$$\frac{\sum_{i=1}^{m} \sum_{j=i+1}^{m} c_{ij}}{n} \tag{5}$$

Then, $x_{kt}(k = 1, 2, \cdots, \frac{m(m-1)}{2}, t = 1, 2, \cdots, n)$ was introduced to specify whether data file k is allocated to node n:

$$x_{kt} = 0 \text{ or } 1, k = 1, 2, \cdots, \frac{m(m-1)}{2}, t = 1, 2, \cdots, n \tag{6}$$

Since a comparison task can only be allocated to one node, we have:

$$\sum_{t=1}^{n} x_{kt} = 1, k = 1, 2, \cdots, \frac{m(m-1)}{2} \tag{7}$$

Let $u_j (j = 1, 2, \cdots, n)$ be the storage capacity of node n and $W_{kt} = w_{ijt}$ be the total size of the two files (file i and file j) in task k that are distributed to node t. Then, the total size of the files distributed to each node in the distributed system should not exceed the storage capacity of that node:

$$\sum_{k=1}^{\frac{m(m-1)}{2}} W_{kt} x_{kt} \leq u_t, t = 1, 2, \cdots, n \tag{8}$$

Let $C_{kt} = c_{ijt}$ be the computing load of the comparison task k between file i and file j that are distributed to node t. Then, the computing load distributed to each node in the distributed system can be expressed as:

$$\sum_{k=1}^{\frac{m(m-1)}{2}} C_{kt} x_{kt} \tag{9}$$

Then, the sum of the absolute difference between the actual and theoretical mean number of tasks actually allocated to each computing node can be expressed as:

$$\sum_{t=1}^{n} \left| \left(\sum_{k=1}^{\frac{m(m-1)}{2}} C_{kt} x_{kt} \right) - \frac{\sum_{i=1}^{m} \sum_{j=i+1}^{m} c_{ij}}{n} \right| \tag{10}$$

Under the constraints of equations (6), (7) and (8), an optimal task allocation model [16] was established to minimize the value of equation (10):

$$\min \sum_{t=1}^{n} \left| \left(\sum_{k=1}^{\frac{m(m-1)}{2}} C_{kt} x_{kt} \right) - \frac{\sum_{i=1}^{m} \sum_{j=i+1}^{m} c_{ij}}{n} \right|$$
$$\text{s.t.} \begin{cases} \sum_{t=1}^{n} x_{kt} = 1, k = 1, 2, \cdots, \frac{m(m-1)}{2} \\ \sum_{k=1}^{\frac{m(m-1)}{2}} W_{kt} x_{kt} \leq u_t, t = 1, 2, \cdots, n \\ x_{kt} = 0 \text{ or } 1, k = 1, 2, \cdots, \frac{m(m-1)}{2}, t = 1, 2, \cdots, n \end{cases} \tag{11}$$

If the objective function in Equation (11) contains nonlinear terms, then established model is a nonlinear programming model. In this case, new decision variables d_t^- and d_t^+ were introduced to Equation (11) to transform the model into the linear form. The variable d_t^- means the actual computing load allocated to a node is greater than the theoretical mean computing load to that node, while the variable d_t^+ has exactly the opposite meaning. The two variables are both numbers greater than or equal to zero. In other words, if the actual computing load allocated to a node is greater than the theoretical mean computing load to that node, the excess load d_t^- should be removed from the node; in the opposite scenario, the insufficient load d_t^+ should be added to the node. In this way, the objective function is changed into finding the minimum sum of $(d_t^- + d_t^+)$ for each node.

After introducing the new decision variables, Equation (11) can be transformed into a linear programming model below.

$$\min \sum_{t=1}^{n} (d_t^- + d_t^+)$$
$$\text{s.t.} \begin{cases} \sum\limits_{t=1}^{n} x_{kt} = 1, k = 1, 2, \cdots, \frac{m(m-1)}{2} \\ \sum\limits_{k=1}^{\frac{m(m-1)}{2}} W_{kt} x_{kt} \le u_t, t = 1, 2, \cdots, n \\ \left(\left(\sum\limits_{k=1}^{\frac{m(m-1)}{2}} C_{kt} x_{kt} \right) - \frac{\sum\limits_{i=1}^{m} \sum\limits_{j=i+1}^{m} c_{ij}}{n} \right) - d_t^- + d_t^+ = 0, t = 1, 2, \cdots, n \\ x_{kt} = 0 \text{ or } 1, k = 1, 2, \cdots, \frac{m(m-1)}{2}, t = 1, 2, \cdots, n \\ d_t^-, d_t^+ \ge 0, t = 1, 2, \cdots, n \end{cases} \tag{12}$$

Since the values of d_t^- and d_t^+ cannot be integers, the model of Equation (12) is a mixed integer linear programming (MILP) model.

3 Design of file allocation algorithm

The MILP model can be solved by commercial solvers like CPLEX [17] and Gurobi [18] and non-commercial approaches like branch and bound method, cutting plane method, branch-cutting plane method and heuristic method [19–21]. Here, the intlinprog function of branch-and-bound steps [16] in the Matlab is selected to solve the MILP in Equation (12). Table 2 lists the comparison tasks and data files in the problem. Among them, branch and bound method is an effective method to solve combinatorial optimization problems. It can get the optimal solution, and the average speed is very fast. Therefore, the idea of branch and bound method is adopted in this paper.

Table 2: Comparison tasks and data files

Task Number	**File1 Number**	**File2 Number**
1	1	2
2	1	3
...	...	...
$m-1$	1	m
M	2	3
$m+1$	2	4
...	...	...
$2*m-3$	2	m
...	...	...
$m(m-1)/2$	$m-1$	m

Then, a file allocation algorithm was designed to cover the following steps [16, 22, 23]:

4 Experimental verification

Four file allocation experiments were carried out in the environment of matlab 2018a [24,25], with the aim to verify our model and algorithm.

Algorithm 1 File allocation algorithm

1 step 1: define and initialize variables
2 define and initialize variables m, n, s, u : $m \leftarrow$ the total number of files, $n \leftarrow$ the total number of nodes, $s \leftarrow [s1, s2, \ldots, sm], u \leftarrow [u1, u2, \leftarrow, un]$;
if s.length==0 or u.length==0 **then**
  $s \leftarrow$ unit matrix whose values are all 1 with m rows and 1 column;
  $u \leftarrow$ unit matrix whose values are all infinite with n rows and 1 column.
end if
3 step 2: calculate the corresponding matrix deci between the tasks and the files
4 define task ordinal variables: $taskno \leftarrow 1$;
5
for all $i = 1$ to m **do**
  for all $j = i + 1$ to m **do**
    $deci(taskno, 1:4) \leftarrow [taskno, i, j, s(i) + s(j)]$;
    $taskno++$;
  end for
end for
6 calculate the theoretical mean number of tasks per node: av_work sum $(deci(,4))/n$;
7 Step 3: set the values for general form of MILP parameters
I) set the value for the objective function variable coefficient matrix f.
Define temporary variables i, j; $i \leftarrow$length$(deci(:,1)) * n; j \leftarrow 2 * n$;
$f \leftarrow$ the matrix with $i + j$ rows and 1 column, the former i row elements are 0, and the latter j row elements are 1.
II) set the value of the variable coefficient matrix$aeq$for the corresponding equality constraint.
for all $i = 1$ to length$(deci(:,1))$ **do**
  for all $j = 1$ to n **do**
    $aeq(i, (i-1) * n + j) \leftarrow 1$;
  end for
end for
for all $j = 1$ to n **do**
  for all $i = 1$ to length$(deci(:,1)$ **do**
    $aeq($length$(deci(:,1)) + j, (i-1) * n + j) \leftarrow deci(i, 4)$;
    $aeq($length$(deci(:,1)) + j, n*$length$(deci(:,1)) + (j-1) * 2 + 1) \leftarrow -1$;
    $aeq($length$(deci(:,1)) + j, n*$length$(deci(:,1)) + j * 2) \leftarrow 1$;
  end for
end for
III) set the value of the resource matrix beq for the corresponding equality constraint
Define temporary variable i; $i \leftarrow$ length$(deci(:,1))$;
$beq \leftarrow (1 + n$ row 1 row matrix, the former i row element is 1, and the latter j row element is av_work;
IV) set the value of the coefficient matrix a for the corresponding inequality constraint
Define temporary variable i; $i \leftarrow$length$(aeq(1,:))$;
$a \leftarrow$ Unit matrix whose values are all 0 with n rows and 1 column.
for all $i = 1$ to n **do**
  for all $j = 1$ to length$(deci(:,1))$ **do**
    $a(i, i + (j-1) * n) \leftarrow deci(j, 4)$;
  end for
end for
V) set the value of the resource matrix b for the corresponding inequality constraint: $b \leftarrow u$;
VI) set the value of the vector $intcon$ for integer variable subscript sequence:
$intcon \leftarrow$1: length$(deci(:,1)) * n$;
VII) set the value of LB for the lower bound and the value of UB for the upper bound. Define the temporary variable $i, j; i \leftarrow$length$(deci(:,1)) * n; j \leftarrow 2 * n$;
$LB \leftarrow$ unit matrix whose values are all 0 with i+j rows and 1 column;
$UB \leftarrow$ the matrix with $i + j$ rows and 1 column, where the former i row elements are 1, and the latter j row elements are + infinity.
8 use the branch and bound intlinprog function to solve the MILP problem, we can get the optimal solution:
$[X, Y] \leftarrow$ intlinprog(f, $intcon$, a, b, aeq, beq, LB, UB);
9 step 4: matrix of comparison task allocation result
Define temporary matrix variables sum, $sum \leftarrow$ unit matrix whose values are all 0 with n rows and 1 column.
for all i=1 to length$(deci(:,1))$ **do**
  for all $j = i + 1$ to n **do**
    if $X((i-1) * n + j) > 0.99999$ **then**
      $sum(j) \leftarrow sum(j) + 1$;
      result$(j, sum(j)) \leftarrow i$;
    end if
  end for
end for

(1) Experiment 1 (Same file size and equal distribution of comparison tasks)

Ten genetic sequence files of the same size (100M) were allocated to five nodes for sequence alignment. As shown in Figure 2, the comparison task distributed to each node has the same computing load and achieves full load balancing. Since $m = 10$ and $n = 5$, we have $m(m-1)\%(2*n) = 0$, that is, each node was distributed with the same number of tasks. The detailed results of experiment 1 are shown in Table 3 below.

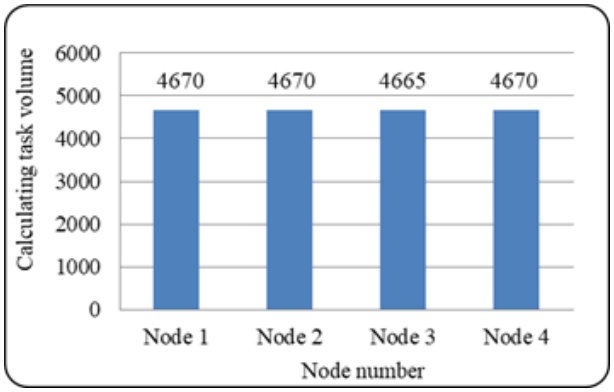

Figure 2: Load distribution to each node in experiment 1

Table 3: Detailed results of experiment 1

Node Number	Task Number	File Number	Task Amount	Calculation Amount
Node1	6, 7, 22, 27, 30, 34, 35, 41, 43	1, 3, 4, 5, 7, 8, 9, 10	9	1800
Node2	3, 9, 13, 16, 17, 25, 28, 31, 36	1, 2, 4, 5, 6, 7, 8, 9, 10	9	1800
Node3	2, 10, 11, 19, 20, 23, 26, 33, 37	1, 2, 3, 4, 5, 6, 8, 9	9	1800
Node4	1, 4, 12, 15, 18, 21, 29, 39, 42	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	9	1800
Node5	5, 8, 14, 24, 32, 38, 40, 44, 45	1, 2, 3, 5, 6, 7, 8, 9, 10	9	1800

(2) Experiment 2 (Same file size and unequal distribution of comparison tasks)

Ten genetic sequence files of the same size (100M) were allocated to four nodes for sequence alignment. Since $m = 10$ and $n = 4$, we have $(m-1)\%(2*n)! = 0$, that is, the nodes cannot achieve complete load balance. In this case, the load distribution to each node is shown in Figure 3. The experimental results (Table 4) show that three of the four nodes were distributed with 2200 tasks while the remaining one was distributed with 2400 tasks. The load between the nodes is basically balanced.

(3) Experiment 3 (Different file sizes and equal distribution of comparison tasks)

Ten genetic sequence files of different sizes (150M; 220M; 180M; 300M; 190M; 95M; 200M; 160M; 320M; 260M) were allocated to five nodes for sequence alignment. Since $m = 10$ and $n = 5$, we have $m(m-1)\%(2*n) = 0$. However, the nodes may not be able to achieve complete load balance due to the difference in file size. In this case, the load distribution to each node is shown in Figure 4. As shown in Table 5, four of the five nodes were distributed with 3735 tasks while the other two with 3775 tasks. The load between the nodes is basically balanced.

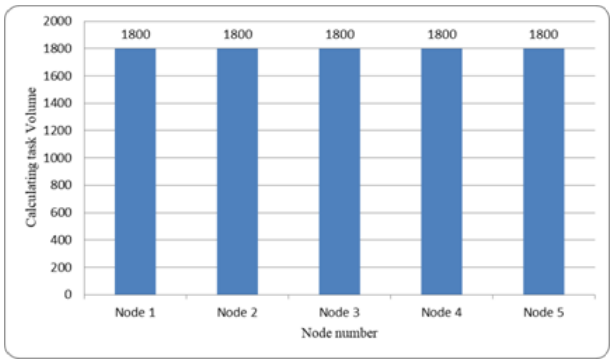

Figure 3: Detailed results of Experiment 2

Table 4: Detailed results of Experiment 2

Node Number	Task Number	File Number	Task Amount	Calculation Amount
Node1	3, 6, 9, 14, 17, 20, 23, 24, 29, 39, 45	1, 2, 3, 4, 6, 7, 9, 10	11	2200
Node2	7, 11, 13, 16, 18, 19, 21, 25, 27, 35, 37, 42	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	12	2400
Node3	4, 5, 8, 22, 31, 33, 34, 38, 41, 43, 44	1, 3, 4, 5, 6, 7, 8, 9, 10	11	2200
Node4	1, 2, 10, 12, 15, 26, 28, 30, 32, 36, 40	1, 2, 3, 4, 5, 6, 7, 8, 10	11	2200

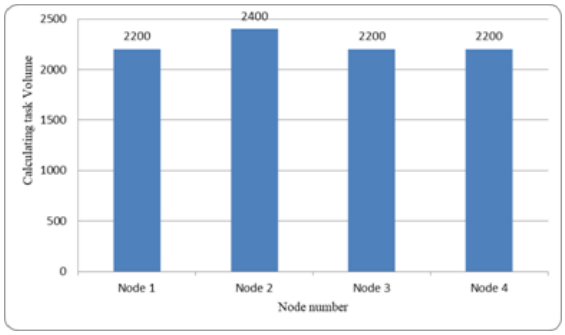

Figure 4: Load distribution to each node in experiment 3

(4) Experiment 4 (Different file sizes and unequal distribution of comparison tasks)

Ten genetic sequence files of different sizes (150M; 220M; 180M; 300M; 190M; 95M; 200M; 160M; 320M; 260M) were allocated to four nodes for sequence alignment. In this case, the load distribution to each node is shown in Figure 5. As shown in Table 6, three of the four nodes were distributed with 4670 tasks while the remaining one with 4665 tasks. The load between the nodes is basically balanced.

The results of the four experiments reveal that our algorithm can always reach load balance between the nodes, whether the files are of the same size, and ensure that the total size of the files distributed to a node never exceeds the storage capacity of that node. Even if load balancing is

Table 5: Detailed results of Experiment 3

Node Number	Task Number	File Number	Task Amount	Calculation Amount
Node1	2, 22, 26, 28, 32, 40, 41, 42, 43	1, 3, 4, 5, 6, 7, 8, 9, 10	9	3735
Node2	5, 10, 12, 13, 18, 20, 31, 35, 36, 45	1, 2, 3, 4, 5, 6, 7, 9, 10	10	3735
Node3	3, 4, 7, 16, 19, 21, 23, 25, 39	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	9	3775
Node4	1, 17, 24, 27, 29, 30, 34, 37	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	8	3735
Node5	6, 8, 9, 11, 14, 15, 33, 38, 44	1, 2, 4, 5, 6, 7, 8, 9, 10	9	3735

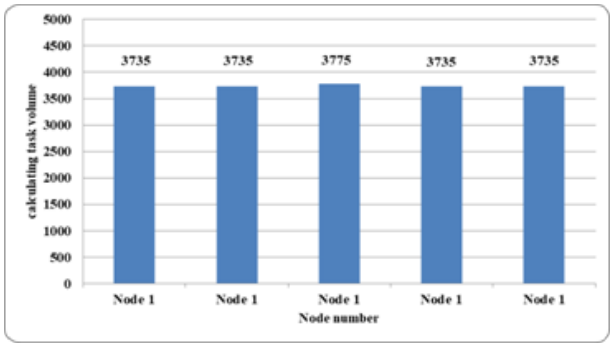

Figure 5: Load distribution to each node in experiment 4

Table 6: Detailed results of Experiment 4

Node Number	Task Number	File Number	Task Amount	Calculation Amount
Node1	1, 8, 11, 17, 19, 29, 32, 33, 41, 45	1, 2, 3, 4, 5, 7, 8, 9, 10	10	4670
Node2	4, 13, 16, 20, 24, 26, 30, 31, 35, 37, 39, 42	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	12	4670
Node3	3, 5, 10, 15, 18, 22, 27, 28, 34, 43, 44	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	11	4665
Node4	2, 6, 7, 9, 12, 14, 21, 23, 25, 36, 38, 40	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	12	4670

theoretically impossible, our algorithm can minimize the load difference between the nodes and approximate the load balance.

Compared with Hadoop-based data allocation strategy, this algorithm can ensure that the comparison tasks have 100% data locality, achieve load balancing between nodes, and improve storage saving and overall computing performance.

5 Conclusions

This paper probes deep into the all-to-all comparison between data files in a large dataset under the environment of a distributed system. After reviewing the problems existing in the existing methods, the author gives a formal mathematical description of the whole comparison problem. In order to achieve load balancing of each node in distributed system, a data file allocation model based on MILP is constructed by using the technology and method of mathematical modeling. Meanwhile, a file allocation algorithm was set up on the Matlab using the intlinprog function of branch-and-bound method. Finally, our model and algorithm were verified through several experiments. The results show that the proposed file allocation strategy can achieve the basic load balance of each node in the distributed system without exceeding the storage capacity of any node, and completely localize the data file. The research findings help to fully utilize the efficiency, stability and scalability of the distributed system to enhance the computing performance of all-to-all comparison.

Acknowledgements. Funding

The work is funded in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61462070, the Doctoral research fund project of Inner Mongolia Agricultural University under Grant No. BJ09-44 and the Inner Mongolia Autonomous Region Key Laboratory of big data research and application for agriculture and animal husbandry.

Author contributions. Conflict of interest

The authors contributed equally to this work. The authors declare no conflict of interest.

Bibliography

[1] Borodin, V.; Bourtembourg, J.; Hnaien, F., Labadie, N. (2018). COTS software integration for simulation optimization coupling: case of ARENA and CPLEX products, *International Journal of Modelling and Simulation*, (5), 1–12, 2018.

[2] Dai, Y.; Wu, W.; Zhou, H.B.; Zhang, J.; Ma, F.Y. (2018). Numerical Simulation and Oprimization of Oil Jet Lubrication for Rotorcraft Meshing Gears, *International Journal of Simulation Modelling*, 17(2), 318–326, 2018.

[3] Dai, Y.; Zhu, X.; Zhou, H.; Mao, Z.; Wu, W. (2018). Trajectory Tracking Control for Seafloor Tracked Vehicle By Adaptive Neural-Fuzzy Inference System Algorithm, *International Journal of Computers Communications & Control*, 13(4), 465–476, 2018.

[4] Deng, J. (2014). Research and Improvement of Mixed Integer Linear Programming Model for Unit Combination, *Nanning: Guangxi University*, 12–16, 2014.

[5] Gao, Y.J. (2017). Research on Data Allocation Strategy for All-to-all Comparison of Large Data Sets, *Taiyuan: Taiyuan University of Technology*, 5–10, 2017.

[6] Guo, J.W.; Li, Y.; Du, L.P.; Zhao, G.F.; Jiang, J.Y. (2014). Research on distributed data mining system based on hadoop platform, *Advances in Intelligent Systems and Computing*, 255, 629–636, 2014.

[7] He, H.; Du, Z.H.; Zhang, W.Z.; Chen, A. (2016). Optimization strategy of Hadoop small file storage for big data in healthcare, *Journal of Supercomputing*, 72(10), 3696–3707, 2016.

[8] Hess, M.; Sczyrba, A.; Egan, R.; Kim, T.W.; Chokhawala, H.; Schroth, G.; Luo, S.; Clark, D.S.; Chen, F.; Zhang, T.; Mackie, R.I.; Pennacchio, L.A.; Tringe, S.G.; Visel, A.; Woyke, T.; Wang, Z.; Rubin, E.M. (2011). Metagenomic discovery of biomass-degrading genes and genomes from cow rumen, *Science*, 331(6016), 463–467, 2011.

[9] Hu, S.R. (1991). Modern supercomputer system, *Journal of computer science*, (1), 47–56, 1991.

[10] Jiao, X.P.; Mu, J.J. (2013). Improved check node decomposition for linear programming decoding, *IEEE Communications Letters*, 17(2), 377–380, 2013.

[11] Liao, J.; Trahay, F.; Xiao, G.; Li, L.; Ishikawa, Y. (2017). Performing initiative data prefetching in distributed file systems for cloud computing, *IEEE Transactions on Cloud Computing*, 5(3), 550–562, 2017.

[12] Mu, R.; Wu, J.J.; Li, N. (2018). MATLAB and mathematical modeling, *Beijing: Science Press*, 63–78, 2018.

[13] MĂźller, E.R.; Carlson, R.C.; Junior, W.K. (2016). Intersection control for automated vehicles with MILP, *IFAC-PapersOnLine*, 49(3), 37–42, 2016.

[14] Nayahi, J.J.V.; Kavitha, V. (2017). Privacy and utility preserving data clustering for data anonymization and distribution on Hadoop, *Future Generation Computer Systems*, 74, 393–408, 2017.

[15] Pitty, S.S.; Karimi, I.A. (2008). Novel MILP models for scheduling permutation flowshops, *Chemical Product and Process Modeling*, 3(1), 35–42, 2008.

[16] Sun, J.Y. (2016). Simulation experiment of operation research model based on MATLAB, *Journal of Shenyang University (Natural Science Edition)*, 28(4), 337–339, 2016.

[17] Schulman, J.; Duan, Y.; Ho, J.; Lee, A.; Awwal, I.; Bradlow, H. (2014). Motion planning with sequential convex optimization and convex collision checking, *International Journal of Robotics Research*, 33(9), 1251–1270, 2014.

[18] Schmidt, B.; Hartmann, C. (2018). Wavepacket: a matlab package for numerical quantum dynamics. ii: open quantum systems, optimal control, and model reduction, *Computer Physics Communications*, 228, 229–244, 2018.

[19] Ubarhande, V.; Popescu, A.; González-Vélez, H. (2015). Novel Data-Distribution Technique for Hadoop in Heterogeneous Cloud Environments, *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, 217–224, 2015.

[20] Wang, L.Z.; Tao, J.; Ranjan, R.; Marten, H.; Streit, A.; Chen, J.Y.; Chen, D. (2013). G-Hadoop: MapReduce across distributed data centers for data-intensive computing, *Future Generation Computer Systems*, 29(3), 739–750, 2013.

[21] Yang, X.P.; Zhou, X.G.; Cao, B.Y. (2015). Multi-level linear programming subject to addition-min fuzzy relation inequalities with application in Peer-to-Peer file sharing system, *Journal of Intelligent and Fuzzy Systems*, 28(6), 2679–2689, 2015.

[22] Zhang, Y.F.; Tian, Y.C.; Fidge, C.; Kelly, W. (2016); Data-aware task scheduling for all-to-all comparison problems in heterogeneous distributed systems, *Journal of Parallel & Distributed Computing*, 93(C), 87–101, 2016.

[23] Zhang, Y.F.; Tian, Y.C.; Kelly, W.; Fidge, C. (2017). Scalable and efficient data distribution for distributed computing of all-to-all comparison problems, *Future Generation Computer Systems*, 67, 152–162, 2017.

[24] Zhang, Y.F.; Tian, Y.C.; Kelly, W.; Fidge, C. (2014). A distributed computing framework for All-to-All comparison problems, *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, 2499–2505, 2014.

[25] Zhou, J.X.; Shao, X.M.; Qiao, J.Y.; Zhang, Y.W. (2012). MATLAB from the introduction to proficiency (2nd edition), *Beijing: People's Post and Telecommunications Publishing House*, 35–92, 2012.