International Journal of Grid and Distributed Computing
Vol. 9, No. 2 (2016), pp.145-156
http://dx.doi.org/10.14257/ijgdc.2016.9.2.13

A Data Placement Algorithm for Data Intensive Applications in Cloud

Qing Zhao[1], Congcong Xiong[2], Kunyu Zhang[3], Yang Yue[4] and Jucheng Yang[5*]

[1,2,5]*Tianjin University of Science and Technology, Tianjin China, 300222*
[3,4]*TROILA Development CO.,Ltd, Tianjin China,300133*
[1]*zhaoqing@tust.edu.cn,* [2]*Xiongcc@tust.edu.cn,* [3]*zhangkunyu@troila.com*
[4]*yueyang@troila.com,* [5]*jcyang@tust.edu.cn*

Abstract

Data layout is an important issue which aims at reducing data movements among data centers to improve the efficiency of the entire cloud system. This paper proposes a data-intensive application oriented data layout algorithm. It is based on hierarchical data correlation clustering and the PSO algorithm. The datasets with fixed location have been considered, and both the offline strategy and the online strategy for data layout have been given. As this proposed strategy is aimed at reducing the global amount of data transmissions, and the special permission of the datasets has been introduced, the cost of data transmission can be measured more reasonable. Simulation results show that compared with two classical strategies, our algorithm can reduce the amount of data transmission more effectively.

Keywords*: Cloud computing, Data correlation, Hierarchical correlation clustering*

1. Introduction

A cloud system [1] always consists of multiple data centers. Data transfers among data centers are unavoidable when running applications. With the arrival of Big Data, the very time-consuming data movements have become the biggest bottleneck of improving the efficiency of the entire cloud.

Complex dependencies often exist among the tasks and datasets in cloud. Some datasets are required by multiple tasks, while some tasks need several datasets, but the storage capacity and computation power of data centers are limit. In fact, data placement is a NP-hard problem, since it could be reduced to the Knapsack Packing Problem. So how to distribute these data items reasonably is a challenging issue. In literature, many previous research results[2-4] have shown that data layout algorithms are very important for overall performance of the cloud platform. Hence, every cloud system used for data intensive applications should automatically and intelligently allocate the data to ensure a low level of global data transmission amount. In traditional distributed systems, Karger *et al*. proposed the consistent hashing algorithm [5], which is still frequently used in current cloud systems. This method maps the datasets and devices using the same hashing function, in order to distribute datasets into data centers uniformly. However, this method only considered the balance condition, without considering the dependencies between data sets. Currently, many famous cloud, such as Google App Engine [6], Amazon EC2 [7] and Hadoop [8], have automatic mechanisms of allocating user's data, but are not designed for scientific workflow. Take Hadoop as an example, when the user load a big file to the Hadoop file system, it will segment the file into small file-chunks with equal sizes and randomly allocate them to the servers. So it also does not considered the factor of data dependencies. However, for a large cloud system built on Internet, time consumed by data transfer is often more than that used for computation, so designing data distribution algorithms according to data dependencies is very essential. [9] has proposed

Jucheng Yang is the corresponding author.

ISSN: 2005-4262 IJGDC

a data placement strategy for Hadoop in heterogeneous environments. The basic idea of their method is to distribute the fragments of the input data into heterogeneous Hadoop nodes based on their computation capabilities. But obviously, this solution is not suitable for the workflow applications, since the size of the data items is different, and there is no obvious relationship between the tasks' computing load and their data involved, as well as complex dependencies exist among a scientific workflow. In [10], Dong Yuan proposed a matrix based k-means clustering strategy for data placement. It can effectively reduce data movement of the workflow cloud systems, but it is only for the target of lowering the number of data movements, rather than the amount of data transfer since the sizes of datasets are quite different.

In this paper, we have taken a hierarchical clustering algorithm on the datasets according to the dependencies between datasets. In deducing the dependency matrix, the size factor of datasets has been introduced, which make reducing the amount of data movements become the direct optimization objective; and in partitioning the matrix, an improved dichotomy algorithm has been issued, which make the hierarchical clustering more reasonable. Besides this, we have also applied a data allocation method based on the PSO algorithm to optimize the mapping between the data groups and the servers.

The remainder of the paper is organized as follows. Section 2 gives the calculation process of the data correlation. Section 3 shows the offline data placement algorithm at initial stage. Section 4 introduces the online re-distribution strategy of the intermediate datasets for the runtime stage. Section 5 shows the simulation results. Finally, Section 6 states the conclusions and future works.

2. Data Correlation Calculation

First, the applications in our system are abstracted as scientific workflows. Figure 1 is an example of typical workflow. The rings in the figure marked as t_1, t_2 and *etc*. represent the tasks, and the squares tagged d_1, d_2 and *etc*. represent the datasets. Besides these, there are a number of arrows in the Figure 1, which describe the data flows in it. For example, both d_1 and d_2 have an arrow pointing to t_1, which means for running the task t_1, two datasets d_1 and d_2 should be loaded first. And another example, task t_1 arrows to task t_2 means an intermediate data item d_3 outputted by task t_1 is required by task t_2 as input.

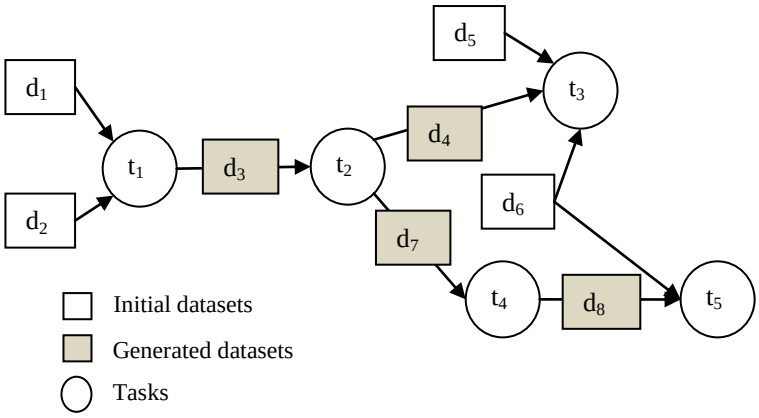


Figure 1. A Simple Instance of Science Workflow

As can be seen from the Figure 1, some tasks, such as t_1, t_3 and t_5, need more than 1 input datasets; while some datasets, like d_6, are required by more than 1 task. Accordingly, complex data dependencies always exist in a workflow system, hence, placing the datasets with close relationship on the same center as possible as you can, can effectively reduce data transfers. Therefore, we build a data correlation matrix, which records the correlation of every two data items, and then according to these dependencies

the data sets will be divided recursively into clusters, until all of them can be saved to one of the data centers. it should be noted that the data correlation calculation algorithms are different with that of other existing works, such as the classical method in [10], since our correlation computation method are based on the volume of data transferred, rather than the frequency of data movements.

Taking into account that some datasets may only be stored in a specific data center, we denote the set consisting of all these datasets that have a fixed storage location as D_{fix}. A data item $i \in D_{fix}$ and must be placed on the j th data center can be expressed as $fix(i) = j$, and a data item with no storage position requirement is expressed as $fix(i) = 0$. In detail, the data correlation calculation is illustrated as follows, and according to whether fixed-position datasets exist, the calculation method can be divided into two cases.

Case 1: There is no fixed-position dataset, *i.e.* $D_{fix} = \phi$

In this case, first, considering the tasks required only 2 input data items, the dependency gain from these tasks can be deduced as:

$$c_{i,j} = \left|T_i \bigcap T_j\right| \times \min\left\{Size_{d_i}, Size_{d_j}\right\}, \ d_i, \ d_j \in D_{in}(t) \tag{1}$$

Here, $\left|T_i \bigcap T_j\right|$ is the number of tasks which need the data sets d_i and d_j as its whole input data; $\min\left\{Size_{d_i}, Size_{d_j}\right\}$ is the size of the smaller data set between $d_i \ and \ d_j$. This is because migrating the smaller file to the storage location of the bigger one is the least cost strategy.

And then, for the tasks required more than 2 input data, the dependency gain can be derived as:

$$\begin{aligned} c_{i,j} &= \max\{\{Size_k \mid d_k \in D_{in}(t), k \neq i \neq j\} \cup \{Size_i + Size_j\}\} \\ &\quad - \max\{Size_m \mid d_m \in D_{in}(t)\} \end{aligned} \tag{2}$$

The dependency between two data items is also defined as how much data transfer amount would be increased when put them together. For example, a task t requires 3 data sets d_1, d_2 and d_3 which sized 5G, 7G, and 10G, respectively. Considering the beginning stage in which d_1 and d_2 are stored together, d_3 is placed on another node, then10G data-movement is the least amount of data transmission for running this task. And then consider another situation that the three data sets are stored on three different nodes, the least amount of data movement for this task is 5+7=12. This is because moving d_1 and d_2 to the computing center where d_3 (the largest dataset among the three) places on is the optimal schedule approach. Therefore, the correlation gain of d_1 and d_2 for this situation is 12-10=2. We then calculate the correlation gain of d_i and d_j where

$$\begin{aligned} c_{i,j} &= \Big(\sum_{d_m \in D_{in}(t)} Size_m - \max\{Size_m \mid d_m \in D_{in}(t)\}\Big) - \\ &\Big(\sum_{d_m \in D_{in}(t)} Size_m - \max\{\{Size_k \mid d_k \in D_{in}(t), k \neq i \neq j\} \cup \{Size_i + Size_j\}\}\Big) \end{aligned} \tag{3}$$

And this formula can be simplified into Equation 2.

Case 2: There are some fixed-position datasets, *i.e.* $D_{fix} \neq \phi$

The fixed-position data items need to be stored on the specific nodes in terms of some authority restriction. Hence, the datasets have the same storage position requirement can be grouped together first. That means we have

$G_j = \{d_i \mid d_i \in D_{fix} \ and \ fix(i) = j\}$. If there is no dataset that must be placed on the k th Server, then $G_k = \phi$. Therefore, in this case, the groups $G_j (1 \le j \le m)$ can be seen as a big dataset, and the data correlation can be calculated between these big datasets and the initial data items. The detail derivation method is shown as follows:

$$c_{A,B} = \begin{cases} 0, both\ A\ and\ B\ are\ fixed - position data\ groups \\ \sum_{(i,j)\in\{A\times B\}} c_{i,j}, \ others \end{cases} \quad (4)$$

Here, A and B can be either a single data item or a fixed-position data group, and $A \times B$ denote the Cartesian product of the two sets A and B. $c_{i,j}$ is the data correlation between two data items and calculated according to Equation 1 and Equation 2. For example, suppose $G_j = \{d_2, d_3, d_4\}$, hence the data correlation between G_j and d_1 is calculated as $c_{d_1,G_j} = c_{1,2} + c_{1,3} + c_{1,4}$. And take another instance, that $G_j = \{d_2, d_3, d_4\}$ and $G_k = \{d_5, d_7\}$, since G_j and G_k have different storage position, the correlation c_{G_j,G_k} can be valued as 0 so as to be separated at the early stage of clustering process. The detail clustering method is given in Section 3.

3. Offline Data Placement Algorithm at Initial Stage

Each item in the above correlation matrix denotes a correlation value between two data items, or between a data item and a fixed-position data group. We utilized the Bond Energy Algorithm (BEA) to transform the correlation matrix CM, and this method is also used in our previous work [11]. After applying the BEA, the items with similar values in the matrix collect together. A demonstration of BEA transformation is present in Figure 2.

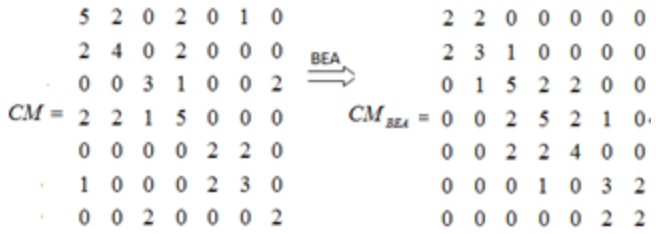


Figure 2. BEA Transformation of Correlation Matrix

After the above BEA operation, recursive partition will be performed. The partitioning algorithm has a better performance than the existing ones, such as that used in [10]. For each partition of the matrix, a division position p is determined to maximize the following objective function in Equation 5.

$$CR = (\sum_{i=1}^{p}\sum_{j=1}^{p} c'_{ij} + \sum_{i=p+1}^{n}\sum_{j=p+1}^{n} c'_{ij}) / (\sum_{i=1}^{p}\sum_{j=p+1}^{n} c'_{ij}) \quad (5)$$

The denominator of the whole CR expression is the total correlation reserved by this partitioning, while the molecule represents the correlation broken. Figure 3 has given a simple example of the recursive dichotomy. The advantage of this partition algorithm is that it has a reasonable equivalent-division preference, neither too high nor too low. Since excessive equivalent-division tendency may cause the data items with high dependency to be separated prematurely; while the consequences of too low equivalent-division tendency are the binary-tree formatted may be too unbalanced.

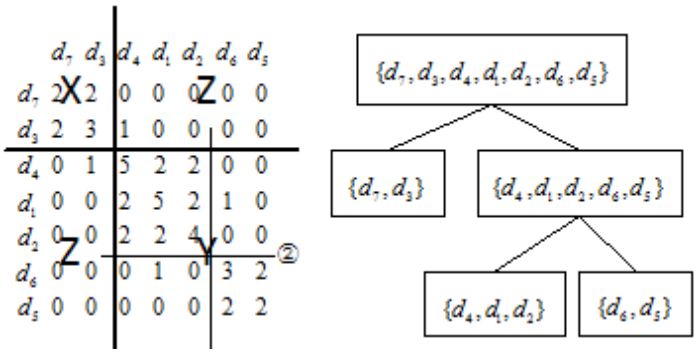


Figure 3. The Recursive Partition of the Matrix

These same partition operations are performed iteratively on each sub-tree until both of the two following constraints are satisfied.

Constraint 1: There is at most only one fixed-position data group in the leaf node.

The fixed-position data items have been grouped in advance in terms of their storage requirement, in the other words, each fixed-position data group has a unique storage position. Therefore, every leaf node should include no more than one fixed-position group, otherwise, the storage location conditions cannot be met.

Constraint 2: There are some servers that can accommodate all of the datasets in the leaf node.

After each partition, the newly generated sub-clusters (or leaf nodes) will be tried to place on one of the servers in cloud. If there is a fix-position data group in this node, it will be tried to allocate to that specified server. If the remainder available space of this server is more than the total size of datasets in the leaf node, this allocation succeeds. Otherwise, the node will be divided again. And for the leaf node that there is no fix-position data item in it, these datasets in the leaf will be attempted to distribute one of the servers, and the node with the largest remaining storage space have the priority. If no node can accommodate, it will be divided again. So the partition operations will be done recursively until all of the data groups have been allocated in accordance with their storage requirement.

4. PSO-Based Strategy for Data-Group Allocation

In the data placement method proposed in Section 3, the data groups generated from correlation clustering are randomly distributed into data centers as long as there is enough storage space in the servers. In this section, another data-group allocation strategy is given, which is based on the PSO algorithm. In the previous literature, the PSO algorithm is used for data placement or task scheduling [12-14], with the increase of the number of the datasets or the tasks, execution efficiency has become an important problem for the evolutionary computing. In this method, because the datasets are first clustered into groups according to their correlation, the objective of this PSO algorithm is simplified into "how to map the data-groups into data centers so as to optimize the amount of data transmission".

In this method, the data correlation matrix is partitioned iteratively until the total data sizes of all the leaf nodes in the data tree are less than a certain threshold, and in the same time the "only one fixed-position" constraint should be satisfied.

PSO was derived from complex adaptive systems (CAS) [15]. Compared with genetic algorithm (GA), PSO has several advantages: simple, easy to implement, and there are not many parameters need to be adjusted. The basic PSO algorithm speed formula is as follows [15]:

$$V_{id}(t+1) = V_{id}(t) + r_1 \cdot c_1 \cdot (P_{id}(t) - X_{id}(t)) + r_2 \cdot c_2 \cdot (P_{gd}(t) - X_{id}(t)) \quad (6)$$

$$X_{id}(t+1) = V_{id}(t) + X_{id}(t) \tag{7}$$

where $V_{id}(t+1)$ denotes the speed value of the dimension d of particle i in the t+1 generation, $r_1 r_2$ is random number in [0, 1], c_1, c_2 is a constant velocity coefficient, P_{id} is the individual current best location, P_{gd} is the current global best position; and $X_{id}(t+1)$ is the position of the dimension d of particle i in the t+1 generation. In this algorithm, the dimension number of the particles is the number of data-groups after correlation clustering, and for each data-group, the candidate storage positions are the whole data centers in the cloud platform. The objective of the proposed PSO algorithm is reducing the global amount of data transmissions, therefore the fitness function is:

$$Fit = \frac{1}{\sum_{t_i \in T} minDataTransmissionAmount_{t_i}} \tag{8}$$

where $minDataTransmissionAmount_{t_i}$ denotes the minimum amount of data movements required to running the task t_i. It is calculated based on the rule that always moving the smaller dataset to the storage location of the bigger one.

5. Online Data Placement Algorithm at Runtime Stage

For the data-intensive applications running on cloud, large amount of intermediate data may be generated during execution, and they are needed by one or more subsequent tasks. As shown in Figure 1, the datasets d_3, d_4, d_7 and d_8 are all newly generated data. In some situations, these newly generated data items could be very large; therefore, the data correlations between them and other pre-existing data are valuable, since distributing these newly generated data reasonably can further reduce the runtime data transmission.

In this paper, newly generated datasets will be saved to a data center that can minimize these data items' associated subsequent data movements. The data movement amount associated with these intermediate data consists of two parts: 1) a latter task requires the intermediate data as input, and not all of the task's input datasets are stored in the same node, hence data movements are unavoidable; 2) the re-distribution of the intermediate data itself increase a certain amount of data movements.

Here, the first part data transmission amount is also measured by the data correlation, especially, it is the correlation between a dataset i and a data center j, where

$$c_{i,dc_j} = \sum_{d_k \in dc_j} c_{i,k} \tag{6}$$

And the second part is an important supplement to the first part, since the intermediate data may be large size and the re-layout itself of a new intermediate data may cause large data movements. Therefore, the correlation between an intermediate dataset and a server should be modified as

$$c'_{i,dc_j} = c_{i,dc_j} + cost_{re-distribution}$$
$$cost_{re-distribution} = \begin{cases} 0, Initial\_Position_i = dc_j \\ Size_i, Initial\_Position_i \neq dc_j \end{cases} \tag{7}$$

Then the data center which has the maximum dependency and also has sufficient storage capacity will be selected to store this dataset.

6. Simulation and Results Analysis

We use Visual Studio 2010 to program the simulation application. 100 random workflows are generated, including initial datasets and intermediate datasets. And the input data set and the output data set of each task include 1-4 and 1-2 random generated data items sized from 1G to 100G respectively. Interdependencies between tasks are determined by the randomly generated input and output data sets of each task. This would make the evaluation results independent of any specific applications. Our initial stage algorithm is used for the initial datasets, and our runtime stage algorithm is applied to the newly generated datasets. Then for each workflow, the simulation software will calculate the total data movement amount on the datasets layout. According to whether considered the fixed-position datasets, the experiments can be divided into 2 groups: no fixed-position data and 10% fixed-position data.

The same workflows and cloud environments are also simulated on other contrast experiments that use the consistent hashing algorithms and DongYuan's algorithms respectively. In these two experiments, the nearest layout principle is used for running stage. After running 100 workflows for every data placement algorithms, the average performance indicators are calculated and used for comparative performance analysis.

Experiment 1: Taken No Fixed-Position Data into Consideration

In this experiment, we suppose that there are no fixed-position dataset in the workflow model. The associated simulation results are shown in Figure 4 and Figure 5.

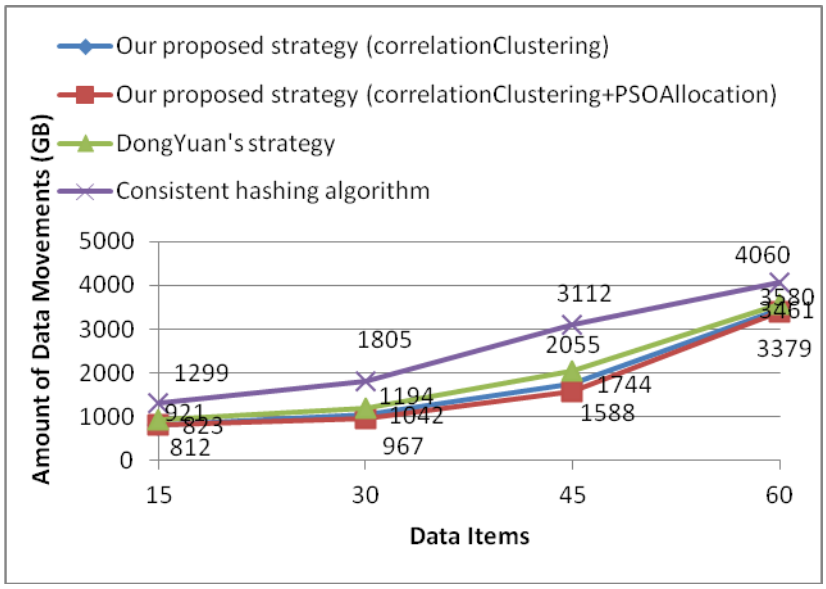


Figure 4. Simulation Results with Different Numbers of Datasets and without Fixed-Position Datasets

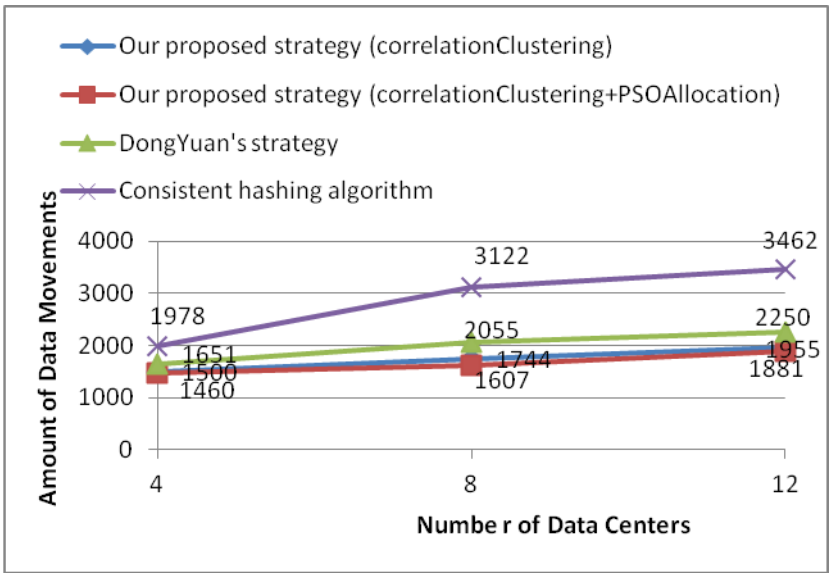


Figure 5. Simulation Results with Different Numbers of Data Centers and without Fixed-position Datasets

In Figure 4, the number of data centers is selected to 8. From this figure we can see, contrast with the consistent hashing algorithm and DongYuan's algorithm, our proposed strategy without PSO allocation can reduce data movement amount of 29.8% and 8.8% on average, and the best performance occurs at 45-datasets, *i.e.* reduced 44.0% and 15.1% respectively than the consistent hashing's and DongYuan's. By introduction of the PSO allocation, 34.3% and 13.0% amount of data movements can be reduced compared with the consistent hashing's and DongYuan's method. Then, we fix the number of data items as 45, and experiment 3 cases with different numbers of data centers 4, 8 and 12, the result is shown in Figure 5. It can be seen that our approach is always better than the performance of the other two methods at different data center numbers. On average, compared with the consistent hashing algorithm and DongYuan's algorithm, 39.3% and 12.7% data amount of movement have declined respectively if only using the proposed correlation clustering strategy, and after applied the PSO allocation strategy, 42.2% and 16.9% are reduced on average compared with the consistent hashing algorithm and DongYuan's algorithm.

Experiment 2: 10% Fixed-Position Datasets

Then we changed 10% of the input datasets to fixed-location datasets so as to see whether our strategy can be applied to the condition of existing fixed-position datasets. The results are shown in Figure 6 and Figure 7.

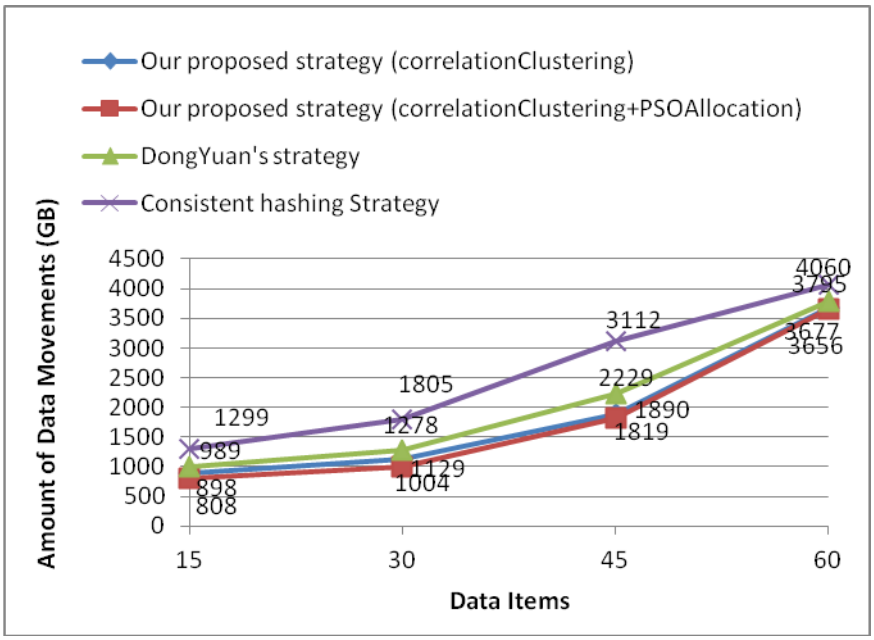


Figure 6. Simulation Results with Different Numbers of Datasets and with 10% Fixed-Position Datasets

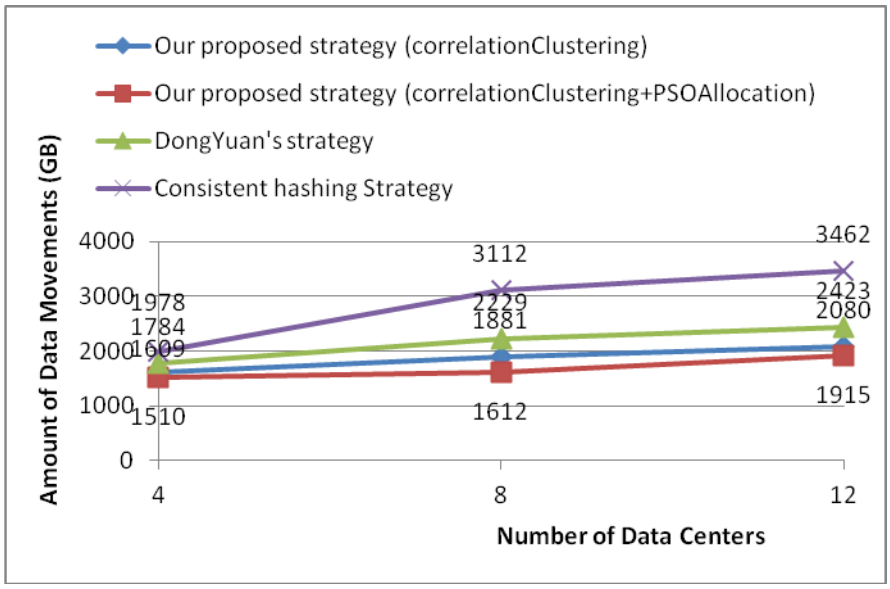


Figure 7. Simulation Results with Different Numbers of Data Centers and with 10% Fixed-Position Datasets

The experiment conditions in Figure 6 and Figure 7 remain the same. In Figure 6, the number of data centers is fixed to 8. From this figure we can see, contrast with the consistent hashing algorithm and DongYuan's algorithm, our proposed strategy can reduce data movement amount of 26.1% and 8.4% on average. By applied the PSO allocation strategy, these decrease percentages of data movements can be increased to 29.1% and 12.1% respectively on average compared with the consistent hashing algorithm and DongYuan's algorithm. Similar result is shown in Figure 7, on average, compared with the consistent hashing algorithm and DongYuan's algorithm, 34.9% and 13.5% amount of data movement have declined respectively. And after applied the PSO allocation strategy, the amount of data movements declines in the percentage of 41.1%

and 21.7% respectively. Therefore, we can draw the conclusion that with 10% fixed location datasets added to the system, our algorithms can still work very well.

7. Conclusions and Future Work

In this paper, a data placement algorithm based on data correlation hierarchical clustering has been proposed, and both the offline data placement strategy and the online strategy have been given. Simulation results indicate that, compared with the two classical strategies, *i.e*. the consistent hashing algorithm and Dong Yuan's strategy, our strategy can reduce the amount of data movements more effectively. And even we set the percentage of fixed location datasets to 10%, our proposed algorithms can still reduce the data movements obviously.

In the future work, heterogeneous network structure is the focus of research, since the transfer time consumption not only depends on the data amounts of movements, but also relies on the network bandwidth between the data centers. Therefore making the frequent data transfers happen on high-speed channels may be a good strategy. And besides network conditions, when we distribute the datasets into servers, some other factors are also very important, such as server's computation capacity and computing load balance, as the efficiency of the cloud system could be further improved. By integrated with these factors, our proposed algorithm can have wider adaptability.

Acknowledgements

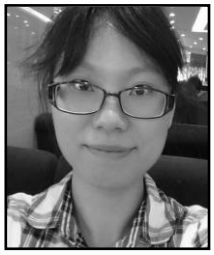
This work is supported by the National Natural Science Foundation of China (61272509), the National Natural Science Foundation of China (61402332) and the Natural Science Foundation of Tianjin University of Science and Technology (20130124).

References

[1] A. Weiss, Computing in the Cloud, ACM Networker, **(2007)**, vol. 11, pp. 18-25.
[2] T. Kosar and M. Livny, A framework for reliable and efficient data placement in distributed computing systems, Journal of Parallel and Distributed Computing, **(2005)**, vol. 65, pp. 1146-1157
[3] T. Kosar and M. Livny, Stork: making data placement a first class citizen in the grid, Proceedings of the 24th International Conference on Distributed Computing Systems, **(2004)** March 23-26; Tokyo, Japan
[4] H. Liu and D. Orban, GridBatch: Cloud Computing for Large-Scale Data-Intensive Batch Applications. Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid, **(2008)** May 19–22; Lyon, France
[5] D. Karger, E. Lehman and T. Leighton, Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. Proceedings of the 29th annual ACM symposium on Theory of computing, **(1997)** May 4-6; El Paso, TX, USA
[6] Google App Engine. http://code.google.com/appengine/.
[7] Amazon Elastic Computing Cloud. http://aws.amazon.com/ec2/.
[8] Hadoop. http://hadoop.apache.org/.
[9] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, Improving mapreduce performance through data placement in heterogeneous Hadoop clusters. Proceedings of the International Symposium on Parallel Distributed Processing, Workshops and Phd Forum, **(2010)** April, pp. 19-23.
[10] D. Yuan, Y. Yang, X. Liu and J. Chen, A data placement strategy in scientific cloud workflows, J. Future Generation Computer Systems, 26, 8 **(2010)**
[11] Q. Zhao, C. Xiong, X. Zhao, C. Yu and J. Xiao, A Data Placement Strategy for Data-Intensive Scientific Workflows in Cloud, Proceedings of IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, **(2015)** May 4-7, Shenzhen, China
[12] X. Li, Y. Wu, F. Ma, E. Zhu, F. Wang, L. Wu, and Y. Yang, A New Particle Swarm Optimization-Based Strategy for Cost-Effective Data Placement in Scientific Cloud Workflows, J. Future Information Technology, 309, 115 **(2014)**
[13] B. Xu, Z. Peng, F. Xiao, A. Gates, J. Yu, Dynamic deployment of virtual machines in cloud computing using multi-objective optimization, J. Soft Computing, **(2014)**
[14] Z. Wu,Z. Ni, L. Gu, X. Liu, A Revised Discrete Particle Swarm Optimization for Cloud Workflow scheduling, Proceedings of the International Conference on Computational Intelligence and Security (CIS), **(2010)** December, pp. 11-14; Nanning, China

[15] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on micro machine and human science, **(1995)** October, pp. 4-6; Nagoya, Japan

Authors



Qing Zhao, She is a currently lecturer in the department of Computer Science, Tianjin University of Science and Technology. She received her PhD degree of Computer Application Technology from Tianjin University, China in 2010. The author has been teaching in Tianjin University of Science and Technology for 3 years. Her primary research interest includes parallel computing, cloud computing and intelligent computing.

Jucheng Yang, He is a full professor in College of Computer Science and Information Engineering, Tianjin University of Science and Technology. He is a Specially-appointed Professor of Tianjin City and Haihe Scholar. He received his B.S. degree from South-Central University for Nationalities, in 2002, MS and PhD degrees from Chonbuk National University, Republic of Korea in 2004 and 2008. His research interests include image processing, biometrics, pattern recognition, and neural networks

Kunyu Zhang, He is the chairman and general manager of Tianjin Troila Technology Development Co. Ltd. He received his B.S. degree of Materials Science and Engineering from Hebei University of Technology, China, in 2008. Recently, he is mainly engaged in Cloud Infrastructure Software and Virtualization Technology.

Yang Yue, He is the senior vice president of Tianjin Troila Technology Development Co. Ltd. He received his MS degree of Software Engineering from NanKai University, China, in 2010. He has received the PMP certification, and recently, his primary research includes Cloud Technology Accumulation and Project Management.