Research Article

Heuristic Data Placement for Data-Intensive Applications in Heterogeneous Cloud

Qing Zhao,¹ Congcong Xiong,¹ and Peng Wang²

¹*Tianjin University of Science and Technology, Tianjin 300222, China*

²*Tianjin House Fund Management Center, Tianjin 300222, China*

Correspondence should be addressed to Congcong Xiong; xiongcc@tust.edu.cn

Received 1 December 2015; Accepted 12 April 2016

Academic Editor: Hui Cheng

Copyright © 2016 Qing Zhao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Data placement is an important issue which aims at reducing the cost of internode data transfers in cloud especially for data-intensive applications, in order to improve the performance of the entire cloud system. This paper proposes an improved data placement algorithm for heterogeneous cloud environments. In the initialization phase, a data clustering algorithm based on data dependency clustering and recursive partitioning has been presented, and both the factor of data size and fixed position are incorporated. And then a heuristic tree-to-tree data placement strategy is advanced in order to make frequent data movements occur on high-bandwidth channels. Simulation results show that, compared with two classical strategies, this strategy can effectively reduce the amount of data transmission and its time consumption during execution.

1. Introduction

With the arrival of Big Data, many scientific research fields have accumulated vast amounts of scientific data. Cloud computing provides vast amounts of storage and computing resources; however, because the cloud is architected in a distributed environment, a large number of I/O operations can be very time-consuming. There are different correlations between the datasets, and the network conditions between servers are also different. Hence, distributing these datasets intelligently can decrease the cost of data transfers efficiently.

In literature, many works [1–4] show that data placement is very crucial for the overall performance of the cloud system, due to the datasets on it being usually distributed enormously and their tasks having unavoidable complex dependencies [5]. Hence, every cloud platform should automatically and intelligently place the data into nodes to ensure that it can be accessed efficiently. Therefore, both the data correlation and the heterogeneous hardware condition of data centers should be taken into consideration for a reasonable data layout.

In this paper, a heuristic data layout method is proposed. First, both the datasets and the cloud system are abstracted as tree-structured models, respectively, according to data correlations and network bandwidth. And then, a heuristic data allocation method is advanced, so as to make frequent data movements occur on high-bandwidth networks. Therefore, the global time consumption of data communication can be reduced effectively.

The remainder of the paper is organized as follows. Section 2 presents related works. Section 3 illustrates the dataset model and the cloud platform model. Section 4 gives the details of the heuristic data allocation strategy. Section 5 presents and analyzes the simulation results. Finally, Section 6 addresses conclusions and future work.

2. Related Works

Cloud computing [6] offers a promising alternative for data-intensive scientific applications. As cloud platform is distributed and often architected on internet, the strategy of data placement is very significant. Many successful cloud systems, such as Google App Engine [7], Amazon EC2 [8],

and Hadoop [9], have automatic mechanisms for storage of user's data but have not given much consideration to data dependencies. Recently, [10] has proposed a data placement strategy for Hadoop in heterogeneous environments, but it is also based on the specific cloud environment and has not considered the heterogynous network conditions. Another research is the Pegasus system which has proposed some data placement strategies [11, 12] based on the RLS system for workflows. These strategies can effectively reduce the overall execution time but only for the runtime stage. Furthermore, in [2], the authors issued a data placement strategy for the distributed systems. It guarantees the reliable and efficient data transfer with different protocols but is not aiming at reducing the total data movement of the whole system.

To reduce the total data movement, modeling the dependencies between datasets is usually the first step. In some literature [13–15], the DAG graph is used to model the data dependencies. And, in addition to the problem of storage resources allocation, other kinds of cloud resources including computing resources also use the similar approaches. For example, in [16], the Tabu Search approach is used for DAG figure partitioning, in order to get size balanced and net-cut minimized scheme of figure partitioning. However, this work is only for the method of graph division; no resource allocation algorithm has been issued. In addition, it only considers how to reduce the frequency of data transmission but did not consider the data amount of transmission. Therefore, it is not suitable for massive amounts of data transmission in Big Data era.

In [17], a matrix based k -means clustering strategy for data placement has been proposed. It groups the data items into k groups in terms of these data dependencies and the value of k is equal to the number of data centers in the cloud. This method can greatly reduce the data movement; nevertheless, it is more suitable for an isomorphic environment. This is because all the differences in the size of datasets, the storage capacities of the servers, and the network transmission speed have not been incorporated. On the other hand, their method is only aiming at decreasing the data transmission frequency, not the data amount of transmissions, or the time consumption by data transmissions.

And in these above methods, after data clustering, the resource allocation algorithms are relatively simple, such as that based on a random distribution principle, and the structure of network is not taken into consideration. In [18], data placement strategy based on a genetic algorithm has been proposed. This method is intended to reduce data movement while balancing the loads of data centers. However, it also lacks the consideration of how to utilize the heterogeneous network conditions more effectively. In our previous work [19, 20], some data placement algorithms have been proposed. In this work, we suppose that the datasets have no fixed-position storage requirements, and the data allocation strategy is also not network-aware. In this paper, the problem of fixed-position data has been considered, and, after data dependency clustering, the clustered datasets are allocated following a principle: frequent data movements occur on high-bandwidth network channels.

FIGURE 1: Tasks and datasets.

3. Data Model and Cloud Platform Model

3.1. Tree-Structure Modeling of the Datasets. In this paper, each application is modeled at a coarse granularity, that is, it is presented as an atomic operation, since data layout is our most significant concern, and the job scheduling can be simplified. Figure 1 provides an example illustration of the tasks and datasets.

Clearly, placing some data items together can decrease data transfer amount. Therefore, the data dependency $c_{i,j}$ between datasets d_i and d_j is defined as how much data transfer amount will be increased if these two datasets are placed on different data centers. This dependency computation method we proposed has improved the method of Yuan [17]. The detail computation algorithm of each dependency item $c_{i,j}$ of the dependency matrix is simply given as follows. This has already been given in our previous work [19, 20], but we have made some improvements.

Because of the problem of different ownership, some of the datasets may have fixed storage location. Here, a symbol D_{fix} is used as the set consisting of all these datasets that have a fixed storage location. For example, any data item i where $i \in D_{\text{fix}}$ must be stationary on a fixed position, and if this fixed position is the j th data center, then this can be expressed as $\text{fix}(i) = j$. Similarly, a data item i with no fixed storage position is expressed as $\text{fix}(i) = 0$. In detail, according to whether fixed-position datasets exist, the calculation method of the data dependency can be divided into two cases.

Case 1 (none of datasets has a fixed storage requirement, i.e., $D_{\text{fix}} = \emptyset$). We assume that the smaller data item is always moved to the node where the bigger one is stored so as to minimize the cost of data transmission. First, considering the tasks with 2 data items as input, the dependency gain from these tasks is

$$c_{i,j} = |T_i \cap T_j| \times \min \{ \text{Size}_{d_i}, \text{Size}_{d_j} \}, \quad (1)$$

$$d_i, d_j \in D_{\text{in}}(t).$$

Here, $|T_i \cap T_j|$ is the number of tasks which need the datasets d_i and d_j as its whole input data; $\min \{ \text{Size}_{d_i}, \text{Size}_{d_j} \}$ is the size of the smaller dataset between d_i and d_j .

And then, for the tasks with more than 2 input data items, the dependency gain is

$$c_{i,j} = \max \left\{ \{ \text{Size}_k \mid d_k \in D_{\text{in}}(t), k \neq i \neq j \} \right. \\ \left. \cup \{ \text{Size}_i + \text{Size}_j \} \right\} - \max \{ \text{Size}_m \mid d_m \in D_{\text{in}}(t) \}. \quad (2)$$

In this case, the dependency gain between two data items is also defined as how much data transfer amount would be increased when they are put in the same position. For example, a task t requires 3 datasets d_1 , d_2 , and d_3 sized 3 G, 15 G, and 17 G, respectively. At the beginning, d_1 and d_2 are stored together, d_3 is placed on another node, and then 17 G data transmission at least is needed for running this task. And then consider another situation that the three datasets are stored on three different nodes; the least amount of data movement for this task is $15 + 3 = 18$. This is because moving d_1 and d_2 to the computing center where d_3 is placed on is the optimal approach. Therefore, the dependency gain of d_1 and d_2 for this situation is $18 - 17 = 1$. Therefore, the dependency gain of d_i and d_j can be calculated as

$$c_{i,j} = \left(\sum_{d_m \in D_{\text{in}}(t)} \text{Size}_m - \max \{ \text{Size}_m \mid d_m \in D_{\text{in}}(t) \} \right) \\ - \left(\sum_{d_m \in D_{\text{in}}(t)} \text{Size}_m \right. \\ \left. - \max \{ \{ \text{Size}_k \mid d_k \in D_{\text{in}}(t), k \neq i \neq j \} \right. \right. \\ \left. \left. \cup \{ \text{Size}_i + \text{Size}_j \} \right\} \right). \quad (3)$$

And this formula can be simplified into (2).

Case 2 (some of datasets have fixed storage requirements, i.e., $D_{\text{fix}} \neq \phi$). We group the datasets with the same storage position requirement first. Therefore, we have $G_j = \{d_i \mid d_i \in D_{\text{fix}} \text{ and } \text{fix}(i) = j\}$. And 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 ($1 \leq j \leq m$) can be seen as a big dataset, and the data dependency can be calculated between these big datasets and the initial data items:

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

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 dependency between two data items and calculated according to (1) and (2). For example, suppose $G_j = \{d_2, d_3, d_4\}$; hence the data dependency 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 where $G_j = \{d_2, d_3, d_4\}$ and $G_k = \{d_5, d_7\}$, since G_j and G_k have different

storage position, the dependency c_{G_j, G_k} between them values 0, so they will be separated at an early stage in clustering process.

Based on this dependency derivation, a dependency matrix CM can be generated. After that, BEA transformation is performed on this matrix so as to collect the similar valued items together. And then, recursive partitioning operations are done. For each division, the division point is selected where the following formula reaches its max value:

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

The denominator is the total dependency reserved by this partitioning, while the molecule represents the broken dependency. These same partition operations are performed iteratively on each subtree until this following constraint is satisfied: there is at most only one fixed-position data group in the leaf node.

3.2. Tree-Structure Modeling of the Cloud Platform. We assume that there are m geographically distributed servers in the cloud platform. The servers are heterogeneous, have different storage capacities, and also have different network bandwidth among them. For the purpose of reducing the time consumption on data movements, the clustered data groups should be placed to the servers according to the following principles: It is better to place the close related data items on the same server so as to decrease the number of data transfers; and if these data cannot be saved together for the reason of limit storage capacities, they should be allocated to closer nodes with high network bandwidth; therefore, most of the data transmissions can obtain high efficiency benefiting by the high-speed channels. In this section, we build a tree-structure model for the cloud system, since it is suitable for the following data allocation stage.

For the case of the cloud architected on LAN, the platform structure can be easily abstracted to a tree structure based on the server's physical connection structure. Figure 2(a) shows an example illustration of a simple cloud platform. According to its topological structure, it can be directly abstracted as tree structure which is illustrated in Figure 2(b). Otherwise, for the cloud based on WAN, we first build a network condition matrix B and then do BEA transformation and recursive dichotomy on this matrix B . Hence an approximate tree structure can be abstracted. The network bandwidths and the distance between servers will be reflected in $b_{x,y}$ values. And, in order to perform BEA transformation, the items on the diagonal, marked by equivalent indices on the two dimensions, like $b_{1,1}$, $b_{2,2}$, and so forth, can be simply calculated as the sum of all the other items in this row; that is, $b_{i,i} = \sum_{j \neq i} b_{i,j}$.

4. Data Distribution

The clustered data items are allocated onto the tree-structured cloud servers based on the following idea: for each allocation, try to allocate the highest level subtree in the data tree to the

Function Name: DataAllocation
 Input: dtNode: The root of the data item binary-tree
 ctNode: The root of the data center binary-tree
 Output: whether the data tree can be allocated to the server tree
 (01) /*First, find the smallest server sub-tree that its total storage capacity is more than the total data size of the sub-tree
 dtNode, suppose its root node is r */
 (02) $r = \text{nextSmallestServerTree}(\text{totalStorageRequirement}(\text{dtNode}))$;
 (03) while ($r \neq \text{Null}$) {
 (04) if ($\text{DataPlacement}(\text{dtNode}, r) == \text{True}$) return True; //DataPlacement is a recursive function
 (05) else $r = \text{nextSmallestServerTree}(\text{totalStorageRequirement}(\text{dtNode}))$;
 (06) }
 (07) Return False;

PSEUDOCODE 1: The pseudocode of the heuristic data allocation method: DataAllocation.

FIGURE 2: An example of a simple cloud platform and its tree-structured model.

lowest level subtree in the server tree, as long as the storage space can accommodate. Therefore, the highly associated data items could be assigned into the same node as much times as possible, and if these items could not be stored together due to some storage limits, they could be placed on closer nodes of the tree-structured environment, since these nodes have high-speed network. The detail of the heuristic data allocation method is as follows.

First, we select the highest layer data subtree in a top-down manner to perform the data assignment as higher priorities, and that could retain the data dependency in the group and reduce the frequency of data transmission as much as possible. On the other hand, a bottom-up strategy (assign the data to the lowest level subtree) is adopted for storage location selection and to make sure the storage requirements should be satisfied. This data allocation strategy can effectively guarantee the high-bandwidth requirements, and it can also save resources to facilitate the subsequent data assignments. It is worth noting that the storage space condition must be met during allocation; otherwise it is not a feasible solution. The detailed constraints that should be tested are as follows.

Storage Space Constraint: $RSS_\beta \geq ASS_\alpha$. ASS_α is total Available Storage Space of the subtree α in the server tree, and RSS_β denotes total Required Storage Space of the subtree β in the data tree. These two parameters of every subtree are

recorded in advance, and their detailed computation methods are, respectively, illustrated as follows:

$$\begin{aligned} ASS_\alpha &= \sum_{s_k \in S_\alpha} SS_k \cdot \lambda, \\ RSS_\beta &= \sum_{d_k \in D_\beta} \text{Size}_k. \end{aligned} \quad (6)$$

Here, Size_k is the size of the dataset d_k , which is a member of the data subtree β and SS_k is the total storage space of the server s_k which belongs to the server set of the subtree α .

For data-intensive applications, besides the initial data, the generated data can also be very large. We therefore cannot fill the data center with their maximum storage in the build-time stage. Otherwise, newly generated data at the runtime will have no space to be stored. Accordingly, an experiential parameter λ is introduced, which denotes the allowable initial usage of the centers' storage space.

This is the overall strategy of the data allocation method. An allocation example is demonstrated in Figure 3, and the pseudocode is shown in Pseudocodes 1 and 2. In Figure 3(a), the "tree-to-tree" data placement task is described. The data tree (on the left) with a size of 102 GB in all should be allocated into the data center tree (on the right) with the total storage space of 136 GB. Figure 3(b) gives the detailed allocation steps of this heuristic data placement strategy. The allocation steps are numbered as "1", "2", ..., "9" in a top-down order.

```

Function Name: DataPlacement (a recursive function)
Input: dtNode: The root of the data sub-tree prepared to be allocated
      ctNode: The root of the server sub-tree prepared to allocate into
Output: whether this data placement success
(01) if (ctNode.LeftNode == Null && ctNode.RightNode == Null) //leaf node!
(02)   If (the remainder storage space >= the total size of the current data tree) {
(03)     //Allocate success! And record the allocation result
(04)     for (every node in the tree rooted by ctNode)
(05)       dataPlace[node.number] = ctNode.number;
(06)       update ctNode's Ccr and size; //update the remainder computation and storage value
(07)     Return True; }
(09) /* Bottom-up server sub-tree selection strategy. First, try to allocate to the left node, and then the right */
(10) else if (dtNode.size < ctNode.LeftNode.size)
(11)   if (DataPlacement (dtNode, ctNode.LeftNode) == True) return True; //allocated!
(12) else if (dtNode.size < ctNode.RightNode.size) //try to allocate to the right node
(13)   if (DataPlacement (dtNode, ctNode.RightNode) == True) return True; //allocated!
(14) /* Top-down data tree selection principle. If the overall current data tree cannot be allocated to any server sub-tree,
then try to allocate its left sub-tree and right-subtree*/
(15) Bool LeftSuccess, RightSuccess;
(16) if (max (dtNode.LeftNode.size, dtNode.RightNode.size) < min (ctNode.LeftNode.size,
(17) ctNode.RightNode.size)) //the two methods of distributing subtrees (left to left, right to
(18) //right; or left to right, right to left) are both possible
(19) /* First try to allocate the data sub-tree with more storage requirements*/
(20)   If (dtNode.LeftNode.size >= dtNode.RightNode.size)
(21)   if (DataPlacement (dtNode.LeftNode, ctNode.LeftNode) == True) LeftSuccess = True;
(22)   else if (DataPlacement (dtNode.LeftNode, ctNode.RightNode) == True) LeftSuccess = True;
(23)   if (LeftSuccess == True ) allocate the reminder data sub-tree to the other server sub-tree, if success,
RightSuccess = True, else:
(24) //Bottom-up server sub-tree selection strategy. Try to allocate the data sub-tree to the parent node
(the higher level sub-tree)
(25)   if (DataPlacement (dtNode.RightNode, ctNode) == True) RightSuccess = True;
(26)   else if (DataPlacement (dtNode.LeftNode, ctNode) == True) LeftSuccess = True;
(27) /* Else, the right node has more storage requirements, allocate it */
(28)   else   if (dtNode.RightNode.size > dtNode.LeftNode.size)
(29)     if (DataPlacement (dtNode.RightNode, ctNode.LeftNode) == True) RightSuccess = True;
(30)     else if (DataPlacement (dtNode.RightNode, ctNode.RightNode) == True) RightSuccess = True;
(31)     if (RightSuccess == True ) allocate the reminder data sub-tree to the other server sub-tree, if success,
LeftSuccess = True, else:
(32)   if (DataPlacement (dtNode.LeftNode, ctNode) == True) LeftSuccess = True;
(33)   else if (DataPlacement (dtNode.RightNode, ctNode) == True) RightSuccess = True;
(34) if (LeftSuccess == True && RightSuccess == True) return True;
(35) else return False;

```

PSEUDOCODE 2: The pseudocode of the heuristic data allocation method: DataPlacement.

When the allocation arrives at the bottom leaf node, the judgement of success or nonsuccess will be made. Here, a sign of smiling face means the allocation is feasible. After that, this judgement procedure will trace back to its upper allocation, such as steps (04) and (05).

5. Simulation and Results Analysis

100 random data-task test datasets are generated, and the input dataset of each task includes 1–4 random generated data items sized from 1 G to 100 G, respectively. The dependencies between datasets and tasks are also generated randomly. After running 100 groups of test data for every data placement algorithm, the average performance indicators are calculated and used for comparative performance analysis.

The same test dataset and cloud environments are simulated on other contrast experiments which use the random placement strategy, Yuan's strategy [13], and our previous strategy in CCGrid [20].

In the 1st experiment, we suppose that there are no fixed-position dataset in the model. The simulation results are shown in Figures 4 and 5. In Figure 4, the number of data centers is 8. From this figure we can see that, in contrast to the random algorithm, Yuan's algorithm, and our previous method in CCGrid, both the data movement amount and the time consumption can be reduced by our proposed method. The average reduction percentages of data movement frequency are, respectively, 32.2%, 11.6%, and 1.49% compared with the random strategy, Yuan's strategy, and our previous method in CCGrid. Therefore, we can

FIGURE 3: An example of the heuristic data allocation algorithm.

FIGURE 4: Total data movement amount without fixed-position datasets.

believe that the data dependency has been utilized properly to decrease data movements. And, on the other hand, from Figure 5, the average reduction percentages of transfer time consumption are 33.2%, 16.3%, and 2.8% compared with the random strategy, Yuan's strategy, and our previous method. This indicates that our heuristic data placement method can indeed make many frequent data transmissions occur in high-speed channels.

FIGURE 5: Total time consumed by data movements without fixed-position datasets.

In the 2nd experiment, 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 simulation results are shown in Figures 6 and 7. The average reduction percentages of data movement frequency are, respectively, 24.9% and 8.6% compared with the random strategy and Yuan's strategy. And,

FIGURE 6: Total data movement amount with 10% fixed-position datasets.

FIGURE 7: Total time consumed by data movements with 10% fixed-position datasets.

on the other hand, the average reduction percentages of transfer time consumption are 28.5% and 15% compared with the random strategy and Yuan's strategy. Therefore, we can believe, with 10% fixed-position datasets, that our proposed method is effective for decreasing data movements and their time consumptions.

6. Conclusions and Future Work

In this paper, we proposed a data placement algorithm for data-intensive applications in cloud system. Compared with previous work, first, both the datasets and the cloud platform are abstracted into tree structures. The contributions of this modeling are as follows. High data dependencies have been

retained in the inner group, while the high-speed network groups have been found in a hierarchical manner. And then, a heuristic data allocation method has been issued, and it can make frequent data movements occur in high-bandwidth network environment, so as to achieve the goal of reducing global transmission time. Simulations results indicate that our data placement strategy can effectively reduce data movement amount and time consumption during execution.

In the current work, more placement factors will be taken into consideration, such as the computation capacity of each server and load balance. Furthermore, replication of frequently used data is an effective solution to achieve good performance in terms of system reliability and response time; therefore, this will be another focus of our future research.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (61272509), the National Natural Science Foundation of China (61402332), the Natural Science Foundation of Tianjin University of Science and Technology (20130124), and the key technologies R & D program of Tianjin no. 15ZCZDZX00200.

References

- [1] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the Montage example," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '08)*, Austin, Tex, USA, November 2008.
- [2] T. Kosar and M. Livny, "A framework for reliable and efficient data placement in distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 65, no. 10, pp. 1146–1157, 2005.
- [3] T. Kosar and M. Livny, "Stork: making data placement a first class citizen in the grid," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS '04)*, pp. 342–349, Tokyo, Japan, March 2004.
- [4] H. Liu and D. Orban, "GridBatch: cloud computing for large-scale data-intensive batch applications," in *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, pp. 295–305, Lyon, France, May 2008.
- [5] E. Deelman and A. Chervenak, "Data management challenges of data-intensive scientific workflows," in *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, pp. 687–692, Lyon, France, May 2008.
- [6] A. Weiss, "Computing in the clouds," *netWorker*, vol. 11, no. 4, pp. 16–25, 2007.
- [7] Google App Engine, <http://code.google.com/appengine/>.
- [8] Amazon Elastic Computing Cloud, <http://aws.amazon.com/ec2/>.
- [9] Hadoop, <http://hadoop.apache.org/>.
- [10] J. Xie, S. Yin, X. Ruan et al., "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, April 2010.

- [11] A. Chervenak, E. Deelman, M. Livny et al., "Data placement for scientific applications in distributed environments," in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, pp. 267–274, IEEE, Austin, Tex, USA, September 2007.
- [12] G. Singh, K. Vahi, A. Ramakrishnan et al., "Optimizing workflow data footprint," *Scientific Programming*, vol. 15, no. 4, pp. 249–268, 2007.
- [13] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *Journal of Supercomputing*, vol. 63, no. 1, pp. 256–293, 2013.
- [14] A. Talukder, M. Kirley, and R. Buyya, "Multiobjective differential evolution for scheduling workflow applications on global grids," *Concurrency & Computation Practice & Experience*, vol. 21, no. 13, pp. 1742–1756, 2009.
- [15] X. Liu, Z. Ni, Z. Wu, D. Yuan, J. Chen, and Y. Yang, "A novel general framework for automatic and cost-effective handling of recoverable temporal violations in scientific workflow systems," *Journal of Systems and Software*, vol. 84, no. 3, pp. 492–509, 2011.
- [16] T. Ghafarian and B. Javadi, "Cloud-aware data intensive workflow scheduling on volunteer computing systems," *Future Generation Computer Systems*, vol. 51, pp. 87–97, 2014.
- [17] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010.
- [18] E.-D. Zhao, Y.-Q. Qi, X.-X. Xiang, and Y. Chen, "A data placement strategy based on genetic algorithm for scientific workflows," in *Proceedings of the 8th International Conference on Computational Intelligence and Security (CIS '12)*, pp. 146–149, Guangzhou, China, November 2012.
- [19] Q. Zhao and C. Xiong, "An improved data layout algorithm based on data correlation clustering in cloud," in *Proceedings of the International Symposium on Information Technology Convergence*, October 2014.
- [20] Q. Zhao, C. Xiong, X. Zhao, C. Yu, and J. Xiao, "A data placement strategy for data-intensive scientific workflows in cloud," in *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '15)*, pp. 928–934, IEEE, Shenzhen, China, May 2015.