Topology-Aware Deployment of Scientific Applications in Cloud Computing

Pei Fan, Zhenbang Chen, Ji Wang
National Laboratory for Parallel & Distributed Processing,
National University of Defense Technology
Changsha, 410073, P.R.China
{peifan, zbchen}@nudt.edu.cn, jiwang@ios.ac.cn

Zibin Zheng, Michael R. Lyu
Dept. of Computer Science & Engineering
The Chinese University of Hong Kong
Hong Kong, China
{zbzheng, lyu}@cse.cuhk.edu.hk

***Abstract*—Nowadays, more and more scientific applications are moving to cloud computing. The optimal deployment of scientific applications is critical for providing good services to users. Scientific applications are usually topology-aware applications. Therefore, considering the topology of a scientific application during the development will benefit the performance of the application. However, it is challenging to automatically discover and make use of the communication pattern of a scientific application while deploying the application on cloud. To attack this challenge, in this paper, we propose a framework to discover the communication topology of a scientific application by pre-execution and multi-scale graph clustering, based on which the deployment can be optimized. Comprehensive experiments are conducted by employing a well-known MPI benchmark and comparing the performance of our method with those of other methods. The experimental results show the effectiveness of our topology-aware deployment method.**

Keywords-Topology-aware; communication topology; scientific applications; deployment; cloud computing;

I. INTRODUCTION

Scientific computing involves the usages of mathematical models and numerical solution techniques to solve scientific, social scientific and engineering problems [1]. Scientific computing usually needs huge computing resources to carry out large scale scientific experiments. In addition, the data transportation in scientific experiments requires a high bandwidth. Recently, cloud computing has been under a growing spotlight as a possible solution for providing a flexible, on-demand computing infrastructure for scientific applications [2]. Compared with other computing platforms, cloud computing is deemed as the next generation of IT platforms and promising to be a cheaper alternative to supercomputers and specialized clusters, a much more reliable platform than grids, and much more scalable platform than the largest common clusters or resource pools [3][4]. However, the nature of distributing and latency/bandwidth diversity of cloud nodes makes deploying and executing scientific applications over cloud a challenging problem.

There are three kinds of methods for deploying applications on cloud: Random, Ranking and Clustering-based. A random method selects cloud nodes randomly. A ranking method will rank available cloud nodes based on their QoS (Quality of Service) values and select the best ones. Ranking methods are usually used for computation-intensive applications, but not appropriate for communication-intensive applications (*e.g.*, Message Passing Interface MPI programs) [5]. The reason is a ranking method cannot consider the communication performance between cloud nodes. For deploying communication-intensive applications, clustering based methods [5] [6] are proposed. The basic idea of a clustering-based method is to cluster the cloud nodes that have a good communication performance together to deploy an application.

Scientific applications can usually be decomposed into interdependent components, connected according to a specific topology, and capable of exploiting different types of computational resources: this is what we call topology-aware applications [7]. However, current deployment methods rarely consider the communication topology information of deployed applications. Thus, in the general case, for a clustering-based method, an application may continuously communicate back and forth between clusters, with a significant impact on performance. Therefore, we need to consider topology information when deploying scientific applications. A few approaches try to use the topology information to improve the performance of systems (*e.g.*, in [7]). These approaches usually need users to describe a topology for a deployed application. However, this requirement is not practical in cloud computing, since the scientific application may not be developed by the user, and even the sources may be not available. In this paper, we propose a topology-aware framework to automatically discover topology information and use the topology information during deployment.

The main contributions of this paper are three-folds: first, we propose an automatic topology detection method, which uses pre-execution and multi-scale clustering to discover the topology of a scientific application; second, based on topology information, we propose a deployment method that can improve the performance of a scientific application; third, for the validation of our method, large scale real-world experiments are conducted to compare our method with other methods.

The rest of this paper is organized as follows: Section II introduces motivation and system architecture; Section III presents our topology-aware deployment method; Section IV

describes experiments; Section V discusses the related work and Section VI concludes the paper.

II. Motivation and Architecture

A. Motivation

Here we describe a topology-aware application that we will use for testing our topology-aware deployment method.

A scientific application usually needs the collaborations of computing nodes, and there are a lot of communications between these nodes. An example of a communication information graph of a MPI application is shown in Fig. 1. The left numbers are the node numbers and the right is the communication graph. White arrows in Fig. 1 represent the messages exchanged between nodes. The application shown in Fig. 1 has been parallelized in a manner that combines 8 nodes to conduct this MPI application, and Fig. 1 shows:

- The communications in the first 4 nodes are frequent, and the same situation also happens in the last 4 nodes. However, the communications between these two groups are obviously less.
- Based on the communication information, we can partition the first 4 nodes into a same communication topology structure and the rest nodes into another structure.

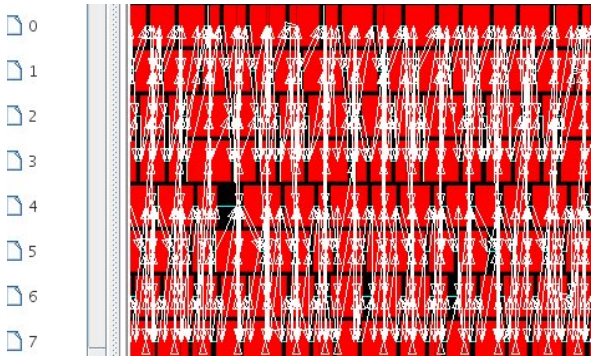


Figure 1. The communication graph of a MPI application

In order to deploy a scientific application on cloud, we can use clustering methods to select nodes [5] [6], since a clustering method can reflect the relations between nodes and partition similarly nodes (low latency between nodes) into a same cluster. However, the result of a clustering method would be very poor in the following scenarios.

- Choose nodes from multi-clusters: the nodes in a same topology structure may be from different clusters if the cloud service provider selects the nodes across multi-clusters. For example, in Fig. 1 the nodes (0-3rd) should be selected from one cluster (topology structure). However, in an across clusters scenario, the 1st and 2nd nodes may be selected from one cluster, but the rest nodes from another cluster, which may lead to a poor performance.
- Overload: all selected nodes may be in a same cluster when using a ranking or clustering method. Under this situation, if users deploy several applications on these nodes, overload will happen.

In order to address the aforementioned problems, we propose a topology-aware framework to deploy scientific applications on cloud based on communication topology. Our method can take into account not only the communication performance between nodes, but also the communication topology of a scientific application. The details of this framework will be introduced in the following

B. Topology-aware Nodes Selection Framework

Fig. 2 shows the architecture of our proposed topology-aware method for deploying scientific applications on cloud. The workflow of our framework is as follows:

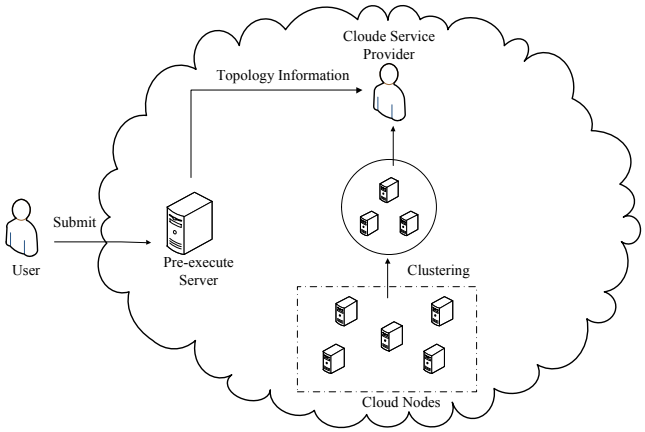


Figure 2. Topology-aware deployment framework

- A cloud user submits an application to the cloud environment. This application will be sent to the pre-execution server. The pre-execution server takes charge of discovering and analyzing the communication topology of the application. To ensure of the effectiveness of pre-execution phase, we employ a method that reduces the problem size of the application when pre-executing the application. The reason is the problem size does not influence the communication topology [8]. After pre-execution, the communication information will be recorded, based on which the topology can be extracted. In our experiments (Section IV) using MPI programs, we develop a slog-2 logfile [9] analysis tool that can discover the communication topology of a MPI program based on the MPI slog2sdk (SLOG-2 software development kit) [9], which can record the message exchanges of a MPI program when running.
- Each cloud node runs a monitor program, which takes charge of monitoring the computing and communication performances of the cloud node. To precisely measure the computing and communication performances of the cloud node, we use the average value during a period as the value of each performance. According to the communication performance, cloud nodes will be partitioned into different clusters via clustering analysis.

- Based on the communication topology of a scientific application, the cloud service provider can map the topology of the cloud node clusters with the topology of the application, and select nodes from appropriate clusters.

III. DEPLOY METHOD

This section presents our topology-aware method for deploying scientific applications in cloud, which is explained in two steps. First, we will introduce how to use a multi-scale clustering algorithm to discover the communication topology of a scientific application. Next, we will use a spectral clustering method to partition cloud nodes into different clusters and present how to select cloud nodes from the generated clusters with respect to the topology information.

A. Logical Topology Discovery

The communicate pattern of a scientific application can be modeled by an undirected (weighted) graph, and it is assumed that two adjacent nodes in the graph have some communications. An undirected graph is usually represented as an adjacency matrix each entry of which represents the communication frequency between the node pair. For example, the following is an adjacency matrix of a scientific application that is deployed on 4 nodes.

	A	*B*	*C*	*D*
A	0	3	1	0
B	3	0	0	1
C	1	0	0	3
D	0	1	3	0

(1)

From (1), we can observe that there have a lot of communications in the node pair (*A*, *B*) or (*C*, *D*), and less communications in (*A*, *C*) and (*B*, *D*). In this paper, we formulate the communication topology discovery problem as a graph clustering problem: we want to find the structure of adjacency, in which nodes are joined together in a tightly knit structure (which means that the nodes within a same structure have more communications between each other). And, there are only looser connections between structures.

Usually, a graph clustering algorithm partitions a set of nodes into *k* groups, where *k* is an input to the algorithm. Therefore, we should know the value of *k* before using a graph clustering algorithm to discover topology. However, this assumption is not practical for cloud computing, since a cloud user or provider may not be the developer of the applications to be deployed. To attack this challenge, we use a hierarchical clustering algorithm [10]. A hierarchical clustering algorithm does not assume any particular number of clusters. Instead a desired number of clusters can be obtained by "cutting" the *dendrogram* at a proper level [11]. An example of dendrogram is shown in Fig. 3. The results of a hierarchical clustering algorithm are often improved with refinement algorithms, which iteratively reassign nodes to different clusters [12]. In this subsection we use a multi-scale refinement algorithm [13] [14] to discover a communication topology. The details of graph clustering and clustering criteria will be introduced in the following.

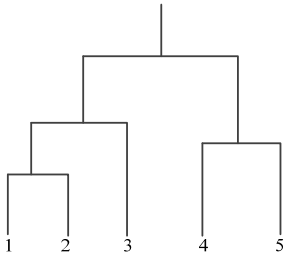


Figure 3. Dendrogram graph

An undirected graph *G* is defined as (*V, E*), where *V* is the node set and *E* is the edge set. The weights of edges are defined by a total function $f: V \times V \to N$. For an undirected graph, $f(u, v)=f(v, u)$, where $u, v \in V$. The degree of a node *v*, denoted by $deg(v)$, is defined as the total weight of its edges, *i.e.*, $\sum_{u \in V} f(v,u)$. The degree of a nodes set $deg(C)$ is defined as $\sum_{u \in C} \deg(u)$; and the weight of two node sets, $f(V_1, V_2)$ is defined as $\sum_{u \in V_1, v \in V_2} f(u,v)$. A merging operation assigns to each cluster pair (*C*, *D*) a real number called merging priority, and thereby determines the order in which an algorithm merges cluster pairs. In this paper, we use *Weight Density* as a merge prioritization to merge cluster pairs. The Weight Density of a cluster pair is defined as

$$\frac{f(C,D)}{\deg(C)\deg(D)} \tag{2}$$

Informally, we denote a subgraph as a graph cluster if it has many internal edges and few edges to the remaining. This can be formalized by defining a measure for the coupling between subgraphs, such that a smaller coupling indicates a better clustering.

Modularity is a quality measure for graph clustering. Newman [15] proposes a modularity measure of the coupling for *k* disjoint sets of nodes, which is defined in (3)

$$Q(V_1, \cdots, V_k) = \sum_{i,\ j=1}^{k} \left(\frac{cut(V_i, V_j)}{|E|} - \frac{\deg(V_i)^2}{\deg(V)^2} \right) \tag{3}$$

In (3), $|E|$ is the number of edges, $cut(V_i, V_j)$ is the sum of the weights of the cut wedges, the first term is the fraction of all edges that are within V_i, and the second term is the expected value of this quantity [16]. It can be easily verified that merging two clusters *C* and *D* increases the modularity by the following equation:

$$\Delta Q_{C,D} := \frac{2f(C,D)}{f(V,V)} - \frac{2\deg(C)\deg(D)}{\deg(V)^2} \tag{4}$$

In addition, moving a node *v* from its current cluster *C* to another cluster *D* increases the modularity, which explained by (5).

$$\Delta Q_{v \to D} := \frac{2f(v,D) - 2f(v, C-v)}{f(V,V)} - \frac{2\deg(v)\deg(D) - 2\deg(v)\deg(C-v)}{\deg(V)^2} \tag{5}$$

Our multi-scale algorithm for discovering a logical topology has two stages: first, use a hierarchical algorithm to partition nodes into clusters; second, use a refinement algorithm to refine the clusters. The algorithm is shown in Fig. 4.

Input: Adjacency matrix *M*, Edge set *E*, Node set *N*
Output: Topology structure that includes *k* groups

```
1   bMerge = true
2   While bMerge
3       bMerge = false
4       Use Equation (2) to calculate weight densities;
5       E=Sort(E) //based on weight densities
6       For each e ∈ E do
7           If e.weight density < atedges/atparis Break
8           If e.startnode or e.endnode merged Continue
9           n=Merge e.startnode and e.endnode;
10          bMerge = true
11          N=N-{e.startnode}-{e.endnode}
12          N=N+{n}
13      End
14      Calculate new adjacency matrix M, Edge set E;
15  End
16  l=level of dendrogram;
17  for l from l_max-1 to 1 do
18      Repeat
19          (v, D) ← best node move;
20          If ΔQ_{v→D} > 0 then
21              Move node v to the cluster D;
22          End
23      Until ΔQ_{v→D} ≤ 0
24  End
```

Figure 4. Multi-scale graph clustering algorithm

- Step 1 (4-5): Calculate weight density via (2), and then descend edge rankings based on weight densities.
- Step 2 (line 6-12): If the start node and end node of an edge have been not merged, the Algorithm will merge these two nodes when the weight density of the edge is greater than *atedges*/*atparis*, where *atedges* is the sum of the edge weights of the graph, and *atparis* is the square of the sum of the node weights. Then calculate the new adjacency matrix *M* and edge *E*, until no more clusters can be merge.
- Step 3 (line 16-19): *l* is the level of dendrogram (*c.f.* Fig. 3), and equal to the loop count in Step 2. In line 16, the algorithm uses (5) to calculate the modularity of moving a node to another cluster, and selects the best node move (*v*, *D*). Here the best node move is a move with the largest modularity increase $\Delta Q_{v\to D}$ (*c.f.* (5)).
- Step 4 (line 20-22): Move *v* to *D*, if (*v*, *D*) is the best move and $\Delta Q_{v\to D} > 0$. The process will be repeated until $\Delta Q_{v\to D} \le 0$.

The multi-scale algorithm can generate the topology structure of a scientific application. In the next section, we will explain how to discover the physical topology of the nodes in cloud, and then map a logical topology to a physical topology.

B. Physical Topology discover

Topology-aware deployment requires the information of two aspects: the logical topology (or communication topology) and the physical topology (or cloud node topology). In the before subsection, we describe the method to obtain a logical topology. This subsection introduces the method of obtaining the physical topology of cloud nodes.

If we restrict the communications in applications only between neighbor nodes, we can have a better utilization of the available bandwidth. Therefore, we want to have the physical topology of cloud nodes, and the nodes that are close to each other will be in a same topology structure. The nodes of different clusters will have a higher latency. Thus, if we denote the latency relations of the nodes in cloud as an adjacency matrix, the physical topology discovery problem can also be formulated as a graph clustering problem. In our previous works [5] [6], we propose a spectral clustering-based method to discover the topology of cloud nodes. In this paper, we use the discovery method in [6] to get the topology of cloud nodes.

After getting a physical topology, cloud nodes are partitioned to different clusters. Then, we can select the nodes for deployment based on the logic topology information of an application. As shown in Fig. 5, the procedure of selecting nodes based on topology structures is a mapping operation, which maps a communication topology to proper clusters of a physical topology. For example, in Figure 5, this application has two topology structures each of which includes 4 nodes. The cloud nodes are partitioned into 3 clusters. When selecting nodes, we rank these three clusters and select the first two clusters, and then based on the topology structure we select 4 nodes from each cluster.

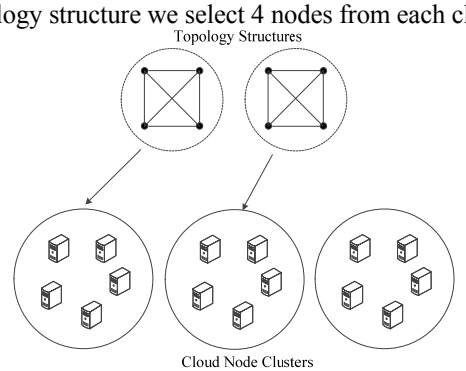


Figure 5. Select nodes based on topology structures

We use a greedy algorithm [6] to rank the generated clusters. After mapping and greedy ranking, the required cloud nodes of deploying an application can be selected. In the next section, we will present the experiments to justify our method.

IV. EXPERIMENTS

In this section, we evaluate our topology aware deployment method by some real-world experiments and give a comprehensive performance comparison with other

methods. We first describe our experiment setup along with the benchmark, followed by the evaluation results.

A. Experiment Setup and Benchmark

We carried out our experiments on PlanetLab [17], which is a global overlay network for developing and accessing broad coverage network services. Our experimental environment consists of 100 distributed nodes that serve as cloud nodes. Our framework is implemented with JDK 1.6. In Section II-B, we introduce that there is a montior program running on every cloud node for evaluating the communication and computing performances. To obtain the accurate values of communication performance, our framework measures the response time between two nodes periodically, and uses the average respone time during a period as the value of performance. The computing power of a cloud node pair is difficult to measure, since cloud nodes are usually heterogeneous. For measuring computing power, we run a benchmark (*e.g.*, calculating PI) on each cloud node periodically, and use the average execution time of two nodes as the value of computing power. Our framework ran about 53 days, and we conducted above 3851 times to measure computing power and above 3052 times for communication performance.

Our experiment includes two parts: first, in the case of selecting nodes across multiple clusters, we compare the performance of our topology-aware method against others; second, with respect to the load performance, we deploy multiple applications, and compare the load performance of our method with those of others. Our experiments use a MPI benchmark called NPB (NAS Parallel Bechmarks) [18]. NPB is a widely used MPI Benchmark, which consists of programs designed to help evaluate the performance of supercomputers. The benchmark is dervied from Computational Fluid Dynamics (CFD) applications.

B. Performance Comparison

To justify the effectiveness, we compare our topology-aware method with the clustering based methods [5][6] that only use clustering analysis to select nodes for an application without considering the topology information.

We use the following two metrics in this experiment.

- Makespan: The makespan of a job is defined as the duration between sending out a job and receiving the correct result.
- Throughput**:** The throughput of a job is defined as the total million operations per second rate (Mop/s) over the number of processes.

Table 1. The Structure Numbers of the Programs in NPB

	4	8
CG	2	2
MG	2	2
SP	2	-
BT	2	-

We first use a pre-execution (*c.f.* Section II-B) and a logical topology discoverer (*c.f.* Section III-A) to get the topology structures of the programs in NPB. Table 1 shows the topology structure numbers of these programs. The first line of Table 1 displays the numbers of the nodes used for deployment. Some benchmarks (*e.g.*, CG and MG) can only run on a power-of-2 number of cloud nodes. The rest (SP and BT) can only run on a square number of cloud nodes. Therefore, SP and BT cannot be deployed on 8 nodes. The entities in Table 1 are the numbers of topology structures (*e.g.*, the number of topology structures is 2 when CG is deployed on 4 nodes). In our experiment we partitioned cloud nodes into 3 clusters.

In each experiment, we select a small set of nodes randomly from 100 nodes (*e.g.*, as shown in Table 2, we select 7 or 8 nodes randomly from 100 nodes). Usually, selecting nodes is across multi-clusters. In order to obtain precise results, all benchmarks were run 10 times, and we use the average result. In Tables 2 and 3, Topology means our topology-aware method, and Untopology is the method introduced in [6] that does not consider the communication topology of a scientific application. These results in Table 2 and 3 show that: for most of the programs, our topology-aware method performs better than Untopology method (less execution time and high throughput). The reason is our method deploys applications with respect to their communication topologies.

Fig. 6 shows the detail results of run CG.8 ten times. In most cases, the topology aware method has a better performance. On the contrary, the Untopology method may have a very poor performance in some cases (*e.g.*, the execute time of CG.8 is about 1500s in the 4th execution).

Table 2. Makespan of Different Method (s)

		Topology	Untopology
CG.4	7	164.9	268.9
	8	110.0	222.1
MG.4	7	202.0	248.2
	8	130.7	189.1
BT.4	7	81.2	95.5
	8	72.9	83.1
SP.4	7	125.1	130.2
	8	94.5	100.6
CG.8	14	304.6	461.5
	15	195.9	289.1
MG.8	14	136.6	171.4
	15	121.9	170.7

Table 3. Throughput of Different Method (Mop/s)

		Topology	Untopology
CG.4	7	14.18	7.38
	8	18.62	13.14
MG.4	7	23.43	17.88
	8	35.73	30.39
BT.4	7	3.07	2.47
	8	4.18	3.77
SP.4	7	0.92	0.86
	8	1.06	1.04
CG.8	14	5.56	4.75
	15	9.27	6.37
MG.8	14	35.2	24.4
	15	34.4	24.2

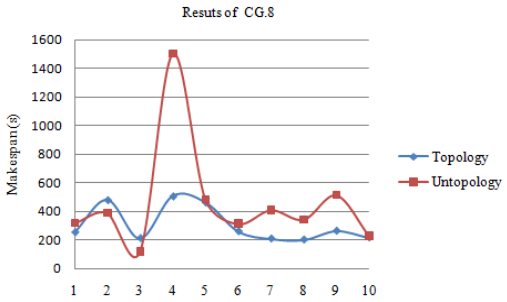
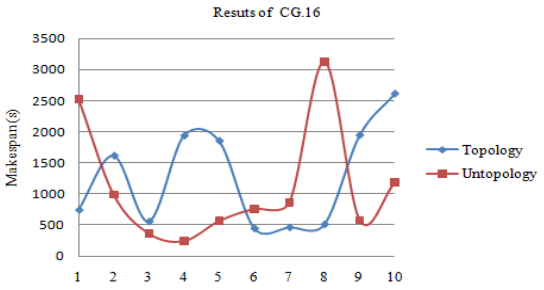
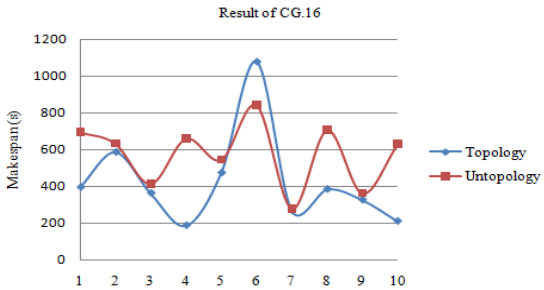


Figure 6. Results of CG.8

We also deploy the programs in NPB on 16 nodes. The benchmark in this experiment is CG.16, and the numbers of topology structures are 2 and 4. We randomly select 30 nodes and then partition these nodes into 4 clusters. Fig. 7(a) shows the detailed results of executing ten times when the number of topology structures equals to 4. From Fig. 7(a), we can observe that the effectiveness of our topology-aware method is not obvious (the average makespan of Topology method is 1272.7s, and that of Untopology method is 1118s). The reason is that the number of topology structures is bigger, and the selected nodes are distributed in the clusters that have poor performance. For example, in this experiment, because the number of topology structures is 4, the nodes are selected from 4 clusters. However, the 3rd or 4th cluster has a poor performance (descending cluster ranking). For justifying this reason, we set the number of topology structures to be 2 which can be obtained by merging (1st, 2nd) and (3rd, 4th) topology structures. Fig. 7(b) shows the results after changing the number of topology structures. We can observe that in most cases Topology method is better than Untopology method (the average makespan of Topology method is 426.7s, and that of Untopology method is 575.9s).



(a). The number of topology structures is 4



(b) The number of topology structures is 2
Figure 7. Results of CG.16

C. *Load Experiment*

In this subsection, we analyze the load performance when deploying multiple applications on cloud nodes. As introduced before, Topology method deploys an application based on the communication topology (*c.f.* Fig. 5). In this experiment we use Topology method to deploy multi-applications on 2 clusters, and use Untopolgy method to deploy same applications on 1 cluster (Untopology method cannot consider the communication topology of an application and only uses the best cluster). The deployment processes of these two methods are shown in Fig. 8.

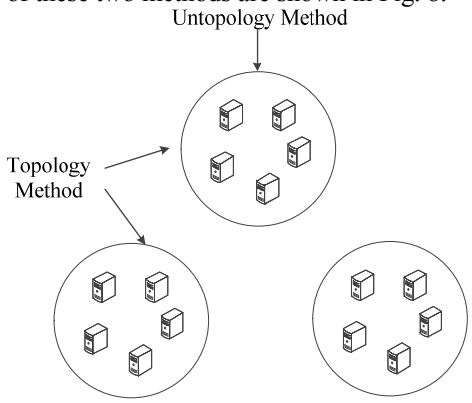
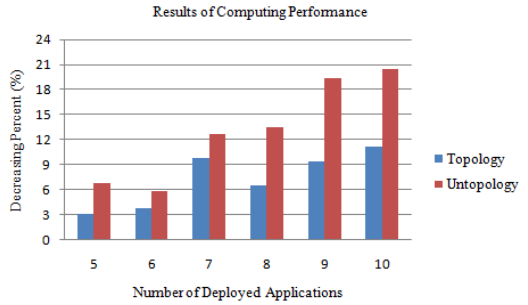


Figure 8. Process of deployment

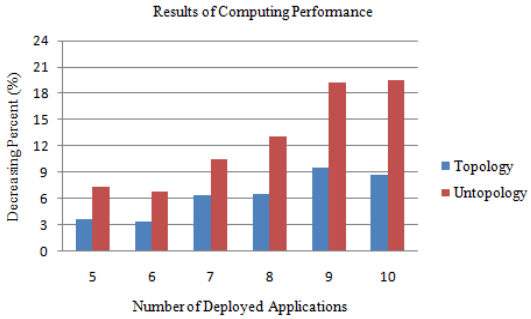
In order to obtain precise results, we partitioned all nodes into 3 clusters and 4 clusters for two methods, respectively. The benchmarks used in this subsection are CG.4, CG.8 and CG.16. The number of topology structures of all benchmarks is 2, which means we will deploy these applications on 2 clusters by using topology-ware method. We change the number of deployed applications from 5 to 10 with a step value 1. The metrics used in this experiment are the values of decreasing percents of communication performance and computing performance. Running more scientific applications needs more cloud resources, such as bandwidth and CPU. Therefore, the communication performance and the computing performance will be decreased. Since Topology method deploys applications in multiple clusters, we use the average decreased percent as the performance decreasing of using Topology method. Fig. 9 shows the results of decreasing percents of computing and communication performances.

Fig. 9(a) displays the results when partitioning all nodes into 4 clusters (*c.f.* Section III.B), and Fig. 9(b) shows the results of partitioning nodes into 3 clusters. These results show that:

- In all cases, Topology method obtains a lower value of decreasing percent of computing performance. The reason is these applications deployed on multiple clusters when using Topology method. This procedure can be viewed as a load balancing procedure.
- With the increasing number of deployed applications, the computing performance is decreased gradually. The reason is more resources are consumed after deploying more applications.



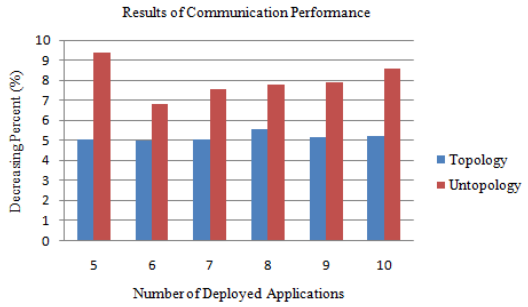
(a) Cluster number is 4



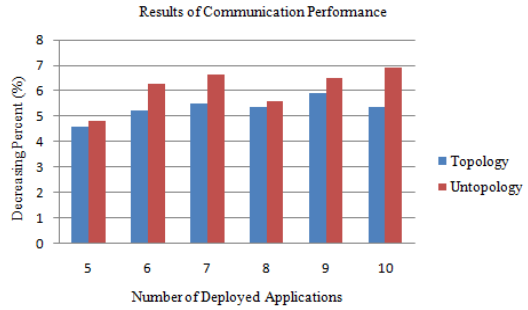
(b) Cluster number is 3

Figure 9. Decreased percent of computing performance

Fig. 10 shows the results of decreasing percents of communication performance, and the results are similar to those of the computing performance experiment.



(a)Cluster number is 4



(b) Cluster number is 3

Figure 10. Decreased percent of communication performance

V. Related Work and Discussion

For deploying applications or services on cloud, a number of approaches have been proposed. Zheng *et al.* [19] [20] use component ranking method for fault-tolerant cloud applications. Kang *et al.* [21] propose a user experience-based mechanism to redeploy cloud services. P4P [22] used to control traffic between applications and network providers. These approaches are ranking-based methods and usually used for computing-intensive applications. Scientific applications usually have a lot of communications between the involved nodes. There are existing literatures for improving the communication performance of scientific applications. Qin *et al.* [23] propose a communication-aware load balancing method for improving the performance of communication-intensive applications by increasing the effective utilization of the networks in cluster environments. Jimenez *et al.* [24] present some sharing policies of information loading in communication-intensive applications. In [25], a common deployment model for grid systems is proposed. We propose [5] [6] a framework that considers the node relations and uses clustering analysis to deploy communication-intensive applications. Compared with our work in this paper, the existing work does not consider the communication topologies of scientific applications in cloud computing, and a poor performance or overload may occur in some scenarios when using these methods.

Collective operations are critical in MPI applications. To improve the performance of MPI applications, a number of collective algorithms have been proposed. In [26] [27], some topology-aware collective communication algorithms are presented for large-scale clusters. Traff [28] uses a topology mechanism to implement MPI. Hoefler *et al.* [29] propose a new scalable process topology interface for MPI 2.2. These works focuses on how to implement MPI library or collective operations. Different from previous work, our work focuses on how to provide an optimal deployment for scientific applications.

There exist literatures for running or deploying applications with respect to topology information. In [30], a generic application description model is proposed for automatic deployment of the applications on computational Grids. Bar *et al.* [7] design a topology-aware grid middleware to schedule the topology-aware applications in grid. Coti *et al.* [31] propose a topology-aware approach to deploying MPI applications in Grid. In [32], an API for topology-aware task mapping is introduced. All of these approaches need users or developers to describe the communication patterns of scientific applications, and then map or schedule tasks on nodes. However, it is not practical for cloud users to provide communication patterns. Compared with the existing methods, we use pre-execution and clustering analysis to get topology information automatically.

VI. Conclusion and Future Work

In this paper, we propose an automatic topology-aware deployment method for the scientific applications in cloud. By taking the advantage of pre-execution and multi-scale clustering algorithms, our approach does not need cloud users to provide the communication patterns of applications. After obtaining topology information, an application will be deployed on cloud optimally. Extensive experiments are

carried out, and the experimental results show that our method outperforms the existing un-topology methods.

In a cloud environment, scientific applications and cloud nodes have a special topology. Currently, we have not considered the grain of topology structures. How to get a fine-grained topology structure needs more investigations in our future work. In addition, user experiences are important for deployment. Our next step also includes the study of a user collaboration based method for deployment.

ACKNOWLEDGMENT

This research is support by the National Basic Research Program (973) of china under the Grant No.2011CB302603, and the National Natural Science Foundation of China under the Grant No.61100078, SRFDP 20114307120015, and the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 415311).

REFERENCES

[1] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Proc. 10th Int'l Symp. on Pervasive Sstems, Algorithms, and Network* (*ISPAN'09*), 2009, pp. 4–16.

[2] H. Christina, M. Gaurang, F. Tim, D. Ewa, K. Kate, B. Bruce and G. John, "On the use of cloud computing for scienific workfolws", in *Proc. 4th Int'l Conf. on eScience* (*eScience'08*), 2008, pp.640-645

[3] R. Buyya, D. Abramson, and J. Giddy, "An economy driven resource management architecture for global computational power grids," in *Proc. 6th Int'l Conf.Parallel and Distributed Processing Techniques and Applications* (*PDPTA'00*), 2000,pp.1

[4] I. R. Ian T. Foster, Yong Zhao and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. 4th Workshop on Grid Computing Environments*, 2008, pp. 1–10.

[5] P. Fan, J. Wang, Z. Zheng, M. R. Lyu, "Toward optimal deployment of communication-intensive cloud applications". *In Proc. 4th Int'l Conf. On Cloud Computing* (*CLOUD' 11*), 2011, pp. 460-467

[6] P. Fan, J. Wang, Z. Chen, Z. Zheng, M.R.Lyu, "A spectral clustering-based optimal deployment method for scientific applications in cloud", *Int'l Journal of Web and Grid Services*. Accept.

[7] P. Bar, C. Coti, D. Groen, T. Herault, V. Kravtsov and M.T. Swain, "Running parallel applications with topology-Aware grid Middleware", in *Proc. 5th Int'l Conf. on e-Science* (*e-Science'09*), 2009, pp.292-299

[8] Ananth. G, Anshul. G, George. K and Vipin. K, "Introduction to parallel computing (2nd ed)," Addison-Wesley, 2002.

[9] A. Chan, W. Gropp, and E. Lusk, "An efficient format for nearly constant-time access to arbitrary time intervals in large trace files," *Scientific Programming,* vol. 16, no. 2-3, pp. 155–165, 2008.

[10] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv*., vol. 31, no.3, pp. 264–323, 1999.

[11] D. Jiang, C. Tang and A. Zhang, "Cluster analysis for gene expression data: A Survey", *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no.11, pp.1370-1386. 2004

[12] G. Karypis, E.-H. Han, and V. Kumar, "Multilevel refinement for hierarchical clustering," TR-99-020, *Department of Computer Science, University of Minnesota, Minneapolis, Tech. Rep*., 1999.

[13] A. Noack and R. Rotta, "Multi-level algorithms for modularity clustering", in *Proc. 8th Int'l Symp. On Experimental Algorithms* (*SEA'09*), 2009, pp.257-268.

[14] R. Hadany and D. Harel, "A multi-scale algorithm for drawing graphs nicely," *Discrete Applied Mathematics*, vol. 113, no. 1, pp. 3–21, 2001.

[15] M. E. J. Newman, "Analysis of weighted networks," *Phys. Rev. E*, vol. 70, no. 5, p. 056131, Nov. 2004.

[16] A. Noack, "Energy models for graph clustering", *Journal of Grpah Algorithms applications*, vol. 11, no. 2, pp. 453-480, 2007.

[17] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay tested for broad-coverage services," *SIGCOMM Comput. Commun. Rev,*. Vol. 33, No. 3, pp. 3-12. 2003.

[18] J.M- Alonso, T. Mercero, E. Ogando, "Performance of an Infiniband cluster running MPI applications". *Technical Reprot EHU-KAT-IK-03-07*, University of the Basque Country. 2007

[19] Z. Zheng, Y. Zhang, and M.R. Lyu, "Cloudrank: a qos-driven component ranking framework for cloud computing,"in *Proc. 29th Int'l Symp. Reliable Distributed Systems* (*SRDS'10*), 2010, pp. 184-193.

[20] Z. Zheng, T.C. Zhou, M.R. Lyu and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Transaction on Service Computing*, Accept.

[21] Y. Kang, Y. Zhou, Z. Zheng and M.R. Lyu, "A user experience-based cloud service redeployment mechanism", in *Proc. 4th Int'l Conf. On Cloud Computing* (*CLOUD'11*), 2011, pp. 227-234.

[22] H. Xie, Y.R Yang, A. Krishnamurthy, Y. Liu and A. Silberschatz, "P4P: provider portal for applications," in *Proc. 24th ACM SIGCOMM Conf. on Data Communication* (*SIGCOMM'08*), 2008, pp. 315-362.

[23] X. Qin, H. Jiang, A. Manzanares, X. Ruan and S. Yin, "Communication-aware load balancing for parallel applications on clusters", *IEEE Transactions on Computers*, Vol. 59 No.1, pp. 42-52.

[24] J.B. Jimenez, D. Caromel, M. Leyton and J.M Piquer. "Load information sharing policies in communication-intensive parallel applicatioins", in Priol, Thierry; Vanneschi, Marco (Eds), *From Grid to Service and Pervasive Computing,* Springer. pp.111-121.

[25] M. Coppola, M. Danelutto, S. Lacour, C. Perez, T. Priol, N. Tonellotto, and C. Zoccolo. "Towards a common deployment model for grid systems", In S. Gorlatch and M. Danelutto, editors, *CoreGRID Workshop on Integrated research in Grid Computing* (*CoreGRID'05*), pp. 31-40

[26] R. Kumar, A.R. Mamidala and D.K. Panda, "Scaling alltoall collective on multi-core systems", in *Proc. 22nd Int'l Conf. Parallel & Distributed Processing Symposium* (*IPDSP'08*), 2008, pp.1-8

[27] K. C. Kandalla, H.Subramoni A. Vishnu and D.K. Panda, "Designing a pology-aware collective communication algorithms for large scale infiniBand clusters: case sudeise with scatter and gather", in *Proc. 24nd Int'l Conf. Parallel & Distributed Processing Symposium* (*IPDSP'*10), 2010, pp.1-8

[28] J.L Traff, "Implementing the MPI procss togology mechanism", in *Proc 15th Int'l Conf. On Super Computing* (*SC'02*), 2002, pp.1-14

[29] T. Hoefler, R. Rabenseifner, H. Ritzdorf, B.R. de Supinski, R. Thakur and J. L Traff, "The scalable precess topology interface of MPI 2.2", *Concurrency and Computaion:Pracice and Experience*, vol. 23, no. 4, 2011, pp.293-310.

[30] S. Lacour, C. Perez and T. Priol, "Generic application description model: toward automatic deploymen of applications on computational grids", in *Proc. 6th Int'l Conf. on Grid Computing* (*GRID'05*), 2005, pp.284-287.

[31] C. Coti, T. Herault and F. Cappello, "MPI Applications on Grid: A topology aware approach", in *Proc. 15th European Conf. on Parallel and Distributed Computing* (*EuroPar'09*), 2009, pp. 466-477.

[32] A. Bhatele, E.J. Bohm and L.V. Kale, "Topology aware task mapping techniques: an api and case study", in *Proc. 14th Int'l Symp. On Principles and Practice of Parallel Progamning* (*PPOPP'09*), 2009, pp.301-302