Topology-Aware Deployment of Scientific Applications in Cloud Computing

Pei Fan*

China HuaYi Broadcasting Corporation,
Fuzhou, 350001, P.R.China
E-mail: peifan@nudt.edu.cn
*Corresponding author

Zhenbang Chen and Ji Wang

Science & Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology,
Changsha, 410073, P.R.China
E-mail: {zbchen, wj}@nudt.edu.cn

Zibin Zheng[1,2], Michael R. Lyu[1,2]

Shenzhen Research Institute[1],
Dept. of Computer Science & Engineering[2],
The Chinese University of Hong Kong,
Hong Kong, China
E-mail: {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. In addition, we present a set of efficient collective operations for cloud based on the common interconnect topology. 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;

Reference to this paper should be made as follows: AAAA., BBB., CCC., DDD. and EEE. (xxxx) 'Topology-Aware Deployment of Scientific Applications in cloud computing', *Int. J. Web and Grid Services*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Pei Fan received his Ph.D. degree from the school of Computer Science at National University of Defense Technology in China. His main research interests focus on Cloud Computing and service-oriented computing.

Zhenbang Chen is an Assistant Professor at the National Laboratory for Parallel and Distributed Processing in China. His research interests include formal methods for component-based system and Service-Oriented Computing, new network computing techniques and software verification.

Ji Wang is a Professor at the Science and Technology on Parallel and Distributed Processing Laboratory in China. He is also the Deputy Director of the National Laboratory for Parallel and Distributed Processing in China. His research interests include high confidence software development, software engineering and distributed computing. He served as in program committees for many conferences including COMPSAC, APSEC, ATVA, EMSOFT, HASE, QSIC, ICTAC, ICFEM, ISoLA, and SCAM. He is on the editorial board of the Journal of Systems and Software. He has authored and co-authored more than 80 research papers in journals, conferences and workshops.

Zibin Zheng is an associate research fellow at Shenzhen Research Institute, The Chinese University of Hong Kong. He received his Ph.D. degree from the Chinese University of Hong Kong in 2011. He received ACM SIGSOFT Distinguished Paper Award at ICSE'2010, Best Student Paper Award at 2010 ICWS'2010, First Runner-up Award at 2010 IEEE Hong Kong Postgraduate Research Paper Competition, adn IBM Ph.D. Fellowship Award 2010-2011. He served as program committee member for many conferences including CLOUD, SCC, etc. His research interests include service computing, cloud computing and software reliability engineering.

Michael R. Lyu is an IEEE Fellow and an AAAS Fellow, for his contributions to software reliability engineering and software fault tolerance. He is also a Croucher Senior Research Fellow. Michael R. Lyu received the Ph.D. degree in computer science from the University of California, Los Angeles, in 1988. He is currently a Professor in the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, China. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, mobile networks, Web technologies, multimedia information processing, and E-commerce systems. He has published over 270 refereed journal and conference papers in these areas. He served as in program committees for many conferences including HASE, ICECCS, ISIT, FTCS, DSN, ICDSN, EUROMICRO, APSEC, PRDC, PSAM, ICCCN, ISESE, and WI.

1 Introduction

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. Nowadays, a lot of scientific applications are deployed in grid systems because they have a high performance and massive storage. Traditional grids are based on batch-queue mechanisms that execute programs according to a pre-established scheduling policy, and they offer scalability by increasing the number of utilized computing nodes (Evoy et al, 2011). However, building a grid system is extremely expensive and it is not available for scientists all over the world to use. Recently, cloud computing has been under a growing spotlight as a possible solution for providing a flexible, on-demand computing infrastructure for scientific applications (Hoffa et al, 2008; Gunarathne et al, 2011). 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 (Buyya, Abramson and Giddy, 2000; Foster, Zhao and Lu, 2008). However, the nature of distributing and latency/bandwidth diversity of cloud nodes makes deploying and executing the scientific applications over clouds 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) (AAA et al, 2011). The reason is a ranking method cannot consider the communication performance between cloud nodes. For deploying communication-intensive applications, clustering based methods (AAA et al, 2011, 2012) 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 (Bar et al, 2009). 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 (Bar et al, 2009)). 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.

Collective operations are often the critical factor determining the ultimate performance of parallel scientific applications. Therefore, how to optimize the collective operations is a challenge for researchers. Exist approaches usually optimize collective operations via implement collective operations algorithm (Almási et al, 2005) or MPI library (Hoefler et al, 2011). However, implement collective operations algorithm or MPI library is difficult to a cloud user, since cloud user may not be the developer of a scientific application. In order to optimize collective operations in cloud environment, we analysis the common interconnect topology of collective operations, and deploy scientific applications based on these common interconnect topology.

This paper is an extension of our conference paper (AAA et al, 2012). Compared with the work in (AAA et al, 2012), there are following extensions:

- We analyze the common interconnect topologies of collective operations, and propose a topology-aware method to optimize collective operations. Instead of improving the collective operations algorithms directly, our topology-aware method optimizes collective operations based on their common interconnect topologies.
- For determining the appropriate value of logical topology structures, we analyze the relation between a logical topology and its physical topology structures, and present appropriate logical topology structures via experiments.
- More extensive real-world experiments have been conducted.

The rest of this paper is organized as follows: Section 2 discusses the related work; Section 3 introduces the motivation and the system architecture; Section 4 presents our topology-aware deployment method; Section 5 describes the experiments and Section 6 concludes the paper.

2 Related work

Recently, scientific applications in clouds have attracted great interests, since clouds provide an alternative to clusters, grids and supercomputers for scientists with a lower cost (Petruch, Stantchev and Tamm, 2011). For deploying applications or services on distributed systems or clouds, a number of approaches have been proposed. BOINC (Anderson, 2004) is a volunteer computing framework, which uses a random method to select the nodes for deployment. Although it is easy to choose nodes randomly, the performance is often poor. RIDGE (Budati et al, 2007) is a reliability-aware system that uses the prior performance and behaviour of a node to make more effective scheduling decisions. Sonnek, Chandra and Weissman (2007) propose the schedule algorithms that employ the estimated reliability ratings of nodes for task allocations. QoS (Quality of Service) can be employed for describing the non-functional information of cloud nodes (Sun et al, 2011; Zheng, Zhang and Lyu, 2010). CloudRank

(Zheng, Zhang and Lyu, 2010) is a QoS-driven component ranking framework for cloud computing. Zheng et al (2010) propose a component ranking method for fault-tolerant cloud applications. Kang et al (2011) propose a user experience-based mechanism to redeploy cloud services. 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. However, ranking methods merely consider node performance, without the relationship between nodes (*e.g.*, the communication between cloud nodes), which is important for communication-intensive scientific applications. The sizes of data have grown exponentially in the past, and data has been distributed widely in a grid or cloud environment (Taniar et al, 2008). In order to improve the performance of data-intensive applications, Yuan et al (2010) propose a matrix based k-means clustering strategy for the data placement in scientific cloud workflows. SSS (Nakada, Ogawa and Kudoh, 2012) is a MapReduce based stream processing system, which is capable of processing the streams of large scale static data. Sailfish (Rao et al, 2012) is a new Map-Reduce framework for large scale data processing. Different from the preceding existing work, our work focuses on the optimal deployment of the scientific applications on cloud platforms and the communication performance. However, for data-intensive applications, we believe our method is also feasible.

There are also existing the literatures for improving the communication performance of the applications in cloud. Hoffa et al (2008) indicate that the communication performance is important for the scientific applications in cloud computing. Ostermann et al (2009) analyze the performance of the EC2 cloud computing services for scientific computing. Koehler et al (2010) present a service-oriented infrastructure that integrates grid computing technologies with a Cloud infrastructure to support the scheduling of dynamic scientific workflows. P4P (Xie et al, 2008) uses application layer traffic optimization to control the traffic between applications and network providers. Bessai et al (2012) propose three bi-criteria complementary approaches to tackle the allocation and scheduling workflow problems in Cloud environments. Chronos (Kapoor et al, 2012) is an architecture to reduce data center application latency especially at the tail. Based on the cross service information and user locations, Kang, Zheng and Lyu (2012) propose a latency-aware co-deployment mechanism for cloud-based services. We propose (AAA et al, 2011, 2012) a framework that considers the node relations and uses clustering analysis to deploy communication-intensive applications. In an Infrastructure-as-a-Service (IaaS) Cloud, resources or services are provided to users in the form of virtual machines (VMs). Therefore, several schedule algorithms that based on virtual machines are proposed. Chen, Zhang and Li (2011) propose a static method for tasks placement and scheduling based on virtual machines. Li et al (2012) present a resource optimization mechanism in heterogeneous IaaS federated multi-cloud systems, which enables preemptable task scheduling. Compared with our work in this paper, the existing work does not consider the communication topologies of the scientific applications in cloud computing, and a poor performance or overload may occur in some scenarios when using these methods.

There are also some existing literatures for running or deploying applications with respect to topology information. In (Lacour, Pérez and Priol, 2005), a generic application description model is proposed for the automatic deployment of the applications on computational grids. Bar et al (2009) design a topology-aware grid middleware to schedule the topology-aware applications in grids. Coti, Hérault and Cappello (2009) propose a topology-aware approach to deploying MPI applications in Grid. In (Bhatele, Bohm and Kaleé, 2009), an API for topology-aware task mapping is introduced. Chan, Viswanathan and Wood (2012) present a placement algorithm that exploits the Euclidean triangular inequality property of network topologies. 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.

For many parallel programs, such as MPI program, collective operations are critical. The speed of MPI collectives is, needless to say, often the critical factor determining the ultimate performance of the applications. To improve the performance, a number of collective algorithms have been proposed. In (Kumar, Mamidala and Panda, 2008; Kandalla et al, 2010), some topology-aware collective communication algorithms are presented for large-scale clusters. Träff (2002) uses a topology mechanism to implement MPI. Hoefler et al (2011) propose a new scalable process topology interface for MPI 2.2. Faraj and Yuan (2005) present a system that produces efficient MPI collective communication routines. Almási et al (2005) implement a number of optimized collective operations on BlueGene/L systems. These approaches focus on how to implement the underlying library or collective operations. Different from previous work, our work focuses on how to provide an optimal deployment for collective operations. We have analyzed the common interconnect topologies of collective operations, and use the topology information to optimize collective operations.

3 Motivation and architecture

3.1 Motivation

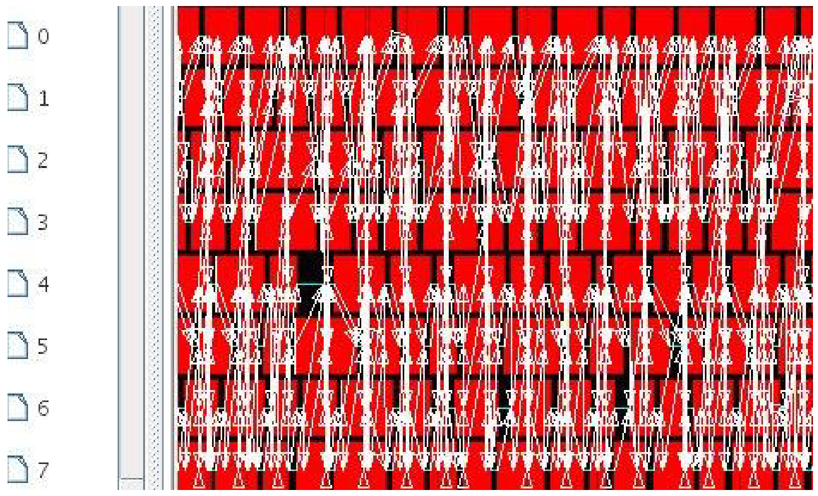
Here we describe a topology-aware application that we will use for testing our topology-aware deployment method.

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 Figure 1. The left numbers are the node numbers (this application deployed on 8 nodes) and the right is the communication graph (x-axis represent the time). White arrows in Figure 1 represent the messages exchanged between nodes. The application shown in Figure 1 has been parallelized in a manner that combines 8 nodes to conduct this MPI application, and Figure 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 an MPI application



In order to deploy a scientific application on cloud, we can use clustering methods to select nodes, 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 Figure 1 the nodes (0-3rd) should be selected from one cluster. 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.

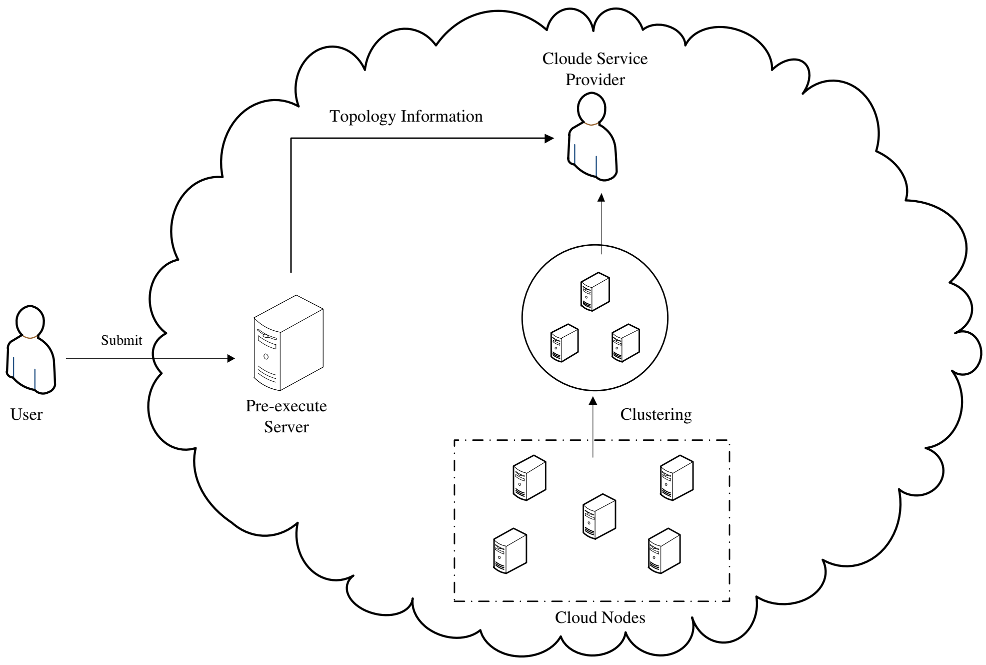
- Overload: Taniar and Leung (2003) indicates that the overload has been one of the major problems in a distributed and parallel environment. All selected nodes may be in a same cluster when using a ranking or clustering method. Under this situation, if users deploy several applications on these nodes, 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 section.

3.2 Topology-aware node selection framework

Figure 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 (Ananth et al, 2003). After pre-execution, the communication information will be recorded, based on which the topology can be extracted. In our experiments (Section 5) using MPI programs, we develop a slog-2 logfile (Chan, Gropp and Lusk, 2008) analysis tool that can discover the communication topology of a MPI program based on the MPI slog2sdk (SLOG-2 software development kit) (Chan, Gropp and Lusk, 2008), which can record the message exchanges of a MPI program when running. More details will be introduced in Section 4.1.

- 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. More details of node partition and selection will be introduced in Section 4.2.

4 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.

4.1 Logical topology discovery

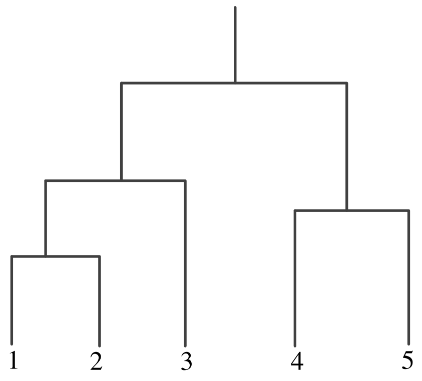
The communicate pattern of a scientific application can be modeled by an undirected (weighted) graph, and it is assumed that two adjacent nodes in the graph have some communications. An undirected graph is usually represented as an adjacency matrix each entry of which represents the communication frequency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

$$\begin{array}{c c} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{pmatrix} 0 & 3 & 1 & 0 \\ 3 & 0 & 0 & 1 \\ 1 & 0 & 0 & 3 \\ 0 & 1 & 3 & 0 \end{pmatrix} \end{array} \tag{1}$$

From Equation 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, which mean the nodes in different structure have less communications.

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 (Jain, Murty and Flynn, 1999). 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 (Jiang, Tang and Zhang, 2004). An example of dendrogram is shown in Figure 3. The results of a hierarchical clustering algorithm are often improved with refinement algorithms, which iteratively reassign nodes to different clusters (Karypis, Han and Kumar, 1999). In this subsection we use a multi-scale refinement algorithm (Noack and Rotta, 2009; Hadany and Harel, 2001) 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 $f : V \times V \rightarrow N$. 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 clusters pairs. In this paper, we use $WeightDensity$ 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 (Newman, 2004) proposes a modularity measure of the coupling for k disjoint sets of nodes, which is defined in Equation 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 Equation 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 (Noack, 2007). It can be easily verified that merging two clusters C and D increases the modularity by the following equation:

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

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

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

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 Algorithm 1.

- Step 1 (4-5): Calculate weight density via Equation (2), and then descend edge rankings based on weight densities.
- Step 2 (line 6-18): 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 20-23): l is the level of dendrogram (c.f. Figure 3), and equal to the loop count in Step 2. in line 23, the algorithm uses Equation 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 (c.f. Equation 5).
- Step 4 (line:24-27): Move v to D, if (v, D) is the best move and $\Delta Q_{v\to D} > 0$. The process will be repeated until $\Delta Q_{v\to D} \leq 0$.

The step 2 of Algorithm 1 means merge two nodes when $\Delta Q_{C,D} > 0$ (Equation 4), the proof show in the following.

Proof: Two nodes merged when the weight density of the edge is greater than $atedges/atparis$ in step 2. Hence

$$\frac{f(C,D)}{deg(C)deg(D)} > \frac{atedges}{atparis}$$

Where, $atedges = f(V,V)$ that is the sum of the weights of all edges, $atparis = deg(V)^2$ that is the square of the degree of all nodes. Hence

$$\frac{f(C,D)}{deg(C)deg(D)} > \frac{f(V,V)}{deg(V)^2}$$

Algorithm 1: Multi-scale graph clustering algorithm

```
Input: Adjacency matrix $M$, Edge set $E$, Node set $N$
Output: Topology structure that includes $k$ groups
bMerge=$true$;
while $bMerge$ do
    bMerge=$false$;
    Use Equation 2 to calculate weight densities;
    $E = Sort(E)$ //based on weight density;
    foreach $e \in E$ do
        if $e.weight\ density < atedges/atparis$ then
            Break
        end
        if $e.startnode\ or\ e.endnode\ merged$ then
            Continue
        end
        $n =$ Merge $e.startnode$ and $e.endnode$;
        bMerge=$true$;
        $N = N - \{e.startnode\} - \{e.endnode\}$;
        $N = N + \{n\}$;
    end
    Calculate new adjacency matrix $M$, Edge set $E$;
end
$l$=level of dendrogram;
for $l$ from $l_{max} - 1$ to $1$ do
    repeat
        $(v, D) \leftarrow$ best node move;
        if $\Delta Q_{v \leftarrow D} > 0$ then
            Move node $v$ to the cluster $D$
        end
    until $\Delta Q_{v \rightarrow D} \leq 0$;
end
```

Since all items are positive number. Hence

$$\frac{f(C,D)}{f(V,V)} > \frac{deg(C)deg(D)}{deg(V)^2}$$

Hence

$$\frac{2f(C,D)}{f(V,V)} - \frac{2deg(C)deg(D)}{deg(V)^2} > 0$$

Which means merge two nodes when $\Delta Q_{C,D} > 0$. □

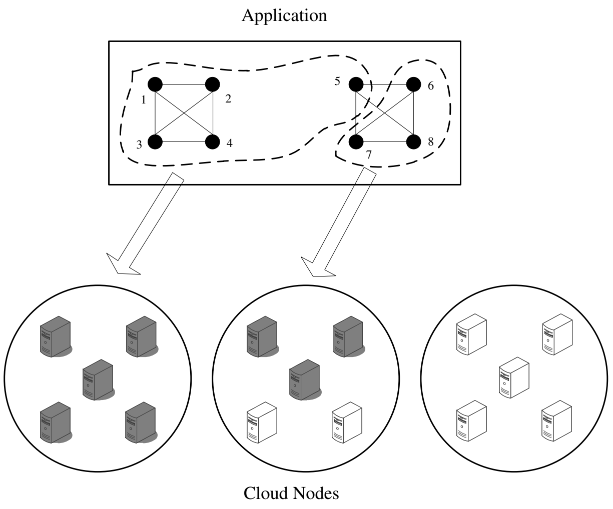
The multi-scale algorithm can generate the topology structure of a scientific application automatically. In the next section, we will explain how to discover the physical topology of the nodes in cloud, and then map a logical topology to a physical topology.

4.2 Physical topology discovery

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 AAA et al (2011, 2012), we propose a spectral clustering-based method to discover the topology of cloud nodes. In this paper, we use the discovery method in AAA et al (2012) to get the topology of cloud nodes.

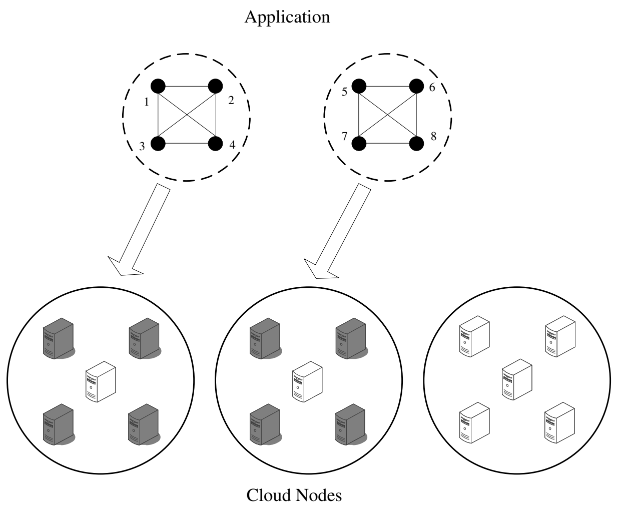
Figure 4 Map tasks without consider topology



After getting a physical topology, cloud nodes are partitioned to different clusters. Then, we can map the task to these nodes for deployment. In the map phase, we may obtain poor performance if we do not consider the communication topology. We use a simple example in Figure 4 to demonstrate the problem. In Figure 4, the application have 8 tasks and should be deployed on 15 cloud nodes. We can observe the communications between these tasks can be partitioned into 2 topology structures each of which includes 4 tasks, and cloud nodes are partitioned in 3 clusters each of which has 5 nodes. In this scenario, nodes should be selected from different nodes clusters, since the number of nodes in each cluster is smaller than the number of tasks. If we use the map method that does not consider the communication topology (*e.g.*, the method introduced in AAA et al (2012)), it will select all the nodes from the first cluster for deploying the 1st-5th tasks, and 3 nodes from the second cluster for deploying the 6th-8th tasks. However, in this scenario, there have a lot of communications between the first cluster and second cluster, since the 5th task deployed on the first cluster has a lot of communication with the 6th-8th tasks. Therefore, mapping tasks to cloud nodes without considering topology information may lead to a poor performance.

In order to address the problem, we select the nodes for deployment based on the logic topology information of an application. As shown in Figure 5, we first rank these three clusters based on the performance of these clusters, and select the first two clusters. Then, we select 4 nodes from the first cluster for deploy 1st-4th tasks, and select 4 nodes from the second cluster for deploy 5th-8th tasks.

Figure 5 Select nodes based on topology structures

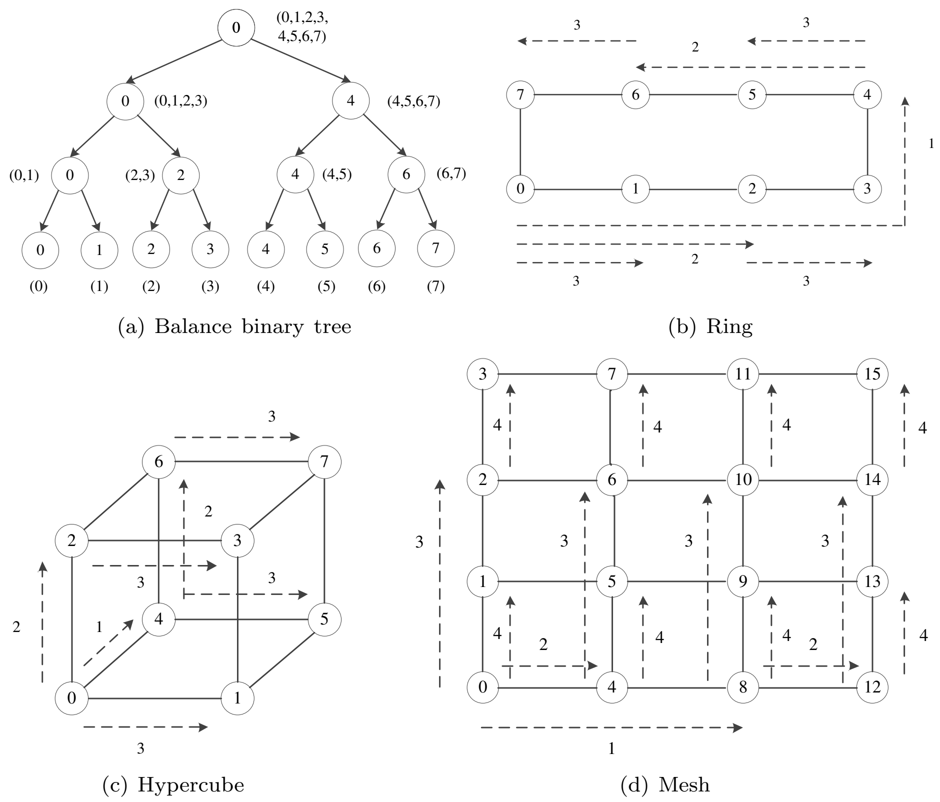


We use a greedy algorithm (AAA et al, 2012) to rank the generated clusters. After mapping and greedy ranking, the required cloud nodes of deploying an application can be selected.

4.3 Collective operations

Collective communication is an important and frequently used component of parallel scientific applications. A study conducted at the Stuttgart High-Performance Computing Center shows that 45% of the overall time on their Cray T3E is spent on MPI routines (Coti, Hérault and Cappello, 2009; Rabenseifner, 2004). Collective operations have been studied and optimized during the last decades. A hierarchical broadcast algorithm is presented in (Cappello et al, 2001). Furthermore, topology information can be used to organize point-to-point communications within collective operations. MPICH-G2 (Karonis et al, 2000) uses topology-discovery features to implement a topology-aware hierarchical algorithm. These approaches are focused on implementing and optimizing MPI library and collective operations. However, cloud users may not be the developer of the applications, and a cloud platform usually has a special environments (*e.g.*, OpenMPI). Thus, it is difficult for users or providers to optimize collective operations. To attack this challenge, we analyze the common topologies of collective operations, and use topology information and hierarchical models to optimize collective operations.

Figure 6 MPI_Scatter operations on different topologies



(a) Balance binary tree (b) Ring

(c) Hypercube (d) Mesh

We introduce a variety of topologies usually used for collective operations before describing our optimization method. Figure 6 shows four topologies (balance binary tree, ring, hypercube and mesh) used to implement Scatter operations. Figure 6(a) displays the detail communication steps for a Scatter operation on eight-node tree. In Figure 6(a), the source node (node 0) contains all the messages. The messages are identified by the labels of the destination nodes. In the first communication step, the source transfers half of the messages to the 4th node. In subsequent steps, each node that has some data transfers half of it to a related node (*e.g.*, 0 send to the 2nd node, and the 4th send to the 6th node) that has not yet to receive any data. There is a total of $logN$ (N is the number of nodes) communication steps. The communication steps of the Scatter operations on ring and hypercube topologies are similar to those of balance binary tree. Figure 6(d) shows a Scatter operation on 16-nodes mesh, which also have $logN$ communication steps. From Figure 6, we can observe that the communication of MPI_Scatter can be organized as a hierarchical model, where messages are transmitted form the top level to the lowest level, and the nodes can be partitioned into different topological groups based on the communications. For example, in Figure 6(a), we can partition 8 nodes into two topological groups (left subtree of root node and right subtree of root node) each of which has 4 nodes. Therefore, instead of implementing special collective operation algorithms, we can use the hierarchical model and the topology information for optimization.

In order to use the topology information for collective operations, we have the following rules for mapping nodes to the structures and the corresponding communicators: a) each topological structure is assigned with a set of nodes which are close to each other; b) for all the groups, a master is defined (*e.g.*, rank 0 is the master of all groups), and a sub-master is defined within some groups (*e.g.*, rank 4 is a sub-master of a group); c) the nodes within some topology structures have special ranks, *e.g.*, the nodes mapped in a given topology structure have consecutive in Scatter operations, because the process with rank 0 sends messages to the processes with rank 1 and 2, and the process with rank 2 sends message to the process with rank 3. Next, we use three collective operations as examples to show our optimizations.

MPI_Scatter. We start a Scatter operation at the process of rank 0 (root node) and conduct the operation in a hierarchical way: a root node (rank 0) is defined for topology structures, and a sub-root is defined in each topology structure. Messages are sent from root node to sub-root nodes. Each sub-root node then sends the messages to its “sub-structure” nodes, until the nodes at the lowest-level are reached.

MPI_Bcast. Broadcast operations are quite similar to that of scatter operations. The communication patterns of one-to-all Broadcast and Scatter are identical. Only differences are the size and the content of messages. Therefore, we can conduct a broadcast operation in the same way of scatter operations.

MPI_Alltoall: Alltoall is a generalization of one-to-all broadcast, and all the nodes involved simultaneously initiate a broadcast. We can use a hypercube to conduct Alltoall operations. In hypercube mode, communications take place along a different dimension of the 8-node hypercube in each step. In every step, pairs of nodes exchange their data and double the size of the messages to be transmitted in the next step by concatenating the received messages with their current data. For example, in Figure 6(c), data is simultaneously exchanged in these four node pairs {(0,1),(2,3),(4,5),(6,7)}, and then communications are happened in {(0,2),(1,3),(4,6),(5,7)}. At the last step, each node pair in {(0,4),(1,5),(2,6),(3,7)} exchange data.

4.4 Grain of logical topology structures

In Section 4.1 and 4.2, we obtain the logical and physical topology structures. We define the number of logical topology structures as k_{log}, and k_{phy} is the number of physical structures. Instead of inputting k_{log} before running, multi-scale algorithm can obtain an adequate k_{log} automatically. However, the value of k_{log} may not be proper in some scenarios when mapping a logical topology to a physical topology, since the value of k_{phy} will greatly influence the result of mapping. For example, after obtaining logical and physical topologies, $k_{log} = 4$ and $k_{phy} = 4$. Thus, the four logical topology structures are mapped to the four physical topology structures, which means we select the nodes from 4 physical topology structures for deploying scientific applications. However, the 3rd or 4th cluster has a poor performance (descending cluster ranking). Thus, the nodes that have poor performance may be selected. To address this problem, we analyze the relation between k_{log} and k_{phy}, and there are two scenarios of mapping a logical topology to a physical topology:

- $k_{log} > k_{phy}$: Intuitively, the value of k_{log} should be reduced to adapt to the value of k_{phy}.
- $k_{log} \leq k_{phy}$: This scenario is more complex than $k_{log} > k_{phy}$. In this scenario, we need to determine whether to reduce the value of k_{log} to adapt the physical topology structures to avoid the problem mentioned before.

In this paper, we focus on the second scenario, *i.e.*, $k_{log} \leq k_{phy}$. In this scenario, we determine whether to reduce the value of k_{log} with respect to the value of k_{log}/k_{phy}: if k_{log}/k_{phy} is indeed less than a special value, the topology-aware method will be better than other methods. For determining an appropriate value of k_{log}/k_{phy}, we conduct a series of experiments that compare our topology-aware method with other methods in different values of k_{log}/k_{phy}. Based on these experiments, we obtain the appropriate value of k_{log}. More details of experiment results will be introduced in Section 5.2.

5 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.

5.1 Experiment setup and benchmark

We carried out our experiments on PlanetLab (Chun et al, 2003), 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 2.2, we introduce that there is a monitor 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 133 days, and we conducted above 9576 times to measure computing power and above 7660 times for communication performance.

Our experiment includes four parts: first, in the case of selecting nodes across multiple clusters, we compare the performance of our topology-aware method against others; second, we conduct a series of experiments for determine the appropriate value of k_{log}/k_{phy}. third, we deploy multiple collective operations applications for justify our topology-aware method is also effective to deploy collective operations. fourth, 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 two MPI benchmark: NPB (NAS Parallel Bechmarks) and IMB (Intel MPI benchmark) (Miguel, Mercero and Ogando, 2007). NPB were dervied from Computational Fluid Dynamics (CFD) applications. NPB are a small set of benchmark programs (e.g., CG, MG, BT et al) designed to compare the performance of parallel computers and are widely recognized as a standard indicator of computer performance. The Intel MPI benchmark (Version 3.2) perform a set of MPI performance measurements for point-to-point and global communication operations for a range of message sizes. The generated benchmark data fully characterizes the performance of a distributed system, including node performance, network latency, and throughput.

5.2 Performance Comparison

To justify the effectiveness, we compare our topology-aware method with the clustering based methods (AAA et al, 2011, 2012) 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.

We first use a pre-execution (c.f. Section 3.2) and logical topology discoverer (c.f. Section 4.1) method 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. The problem size and the number of cloud nodes must be assigned when compile the NPB programs. In NPB, Some NPB programs (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. In our experiments, the problem size of CG and MG is Class A, the BT and SP is Class B. 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) that obtained by the logical topology discovery algorithm (c.f. Section 4.1). In our experiment we partitioned cloud nodes into 3 clusters (which means k_{phy}=3).

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

Table 1 The Number of Topology Structures of NPB Programs

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

method introduced in (AAA et al, 2012) 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.

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

Figure 7 shows the detail results of run CG.8 and MG.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). The reason is the nodes in some communication topology structures may be selected from different clusters. Therefore, the communications between nodes have poor performance.

5.3 Grain of logical topology structures

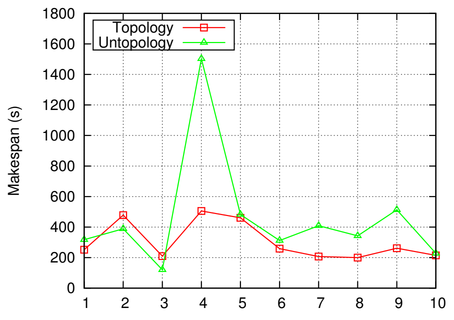
In this section, we will conduct a series of experiments to determine the grain of logical topology structures. The first thing we should do is to demonstrate the necessity of determining the grain of logical topology structures.

We deploy the programs in NPB on 16 nodes. The benchmark in this experiment is CG.16, and the numbers of topology structures is 4 ($k_{log} = 4$). We randomly select 30 nodes from all nodes and then partition these nodes into 4 clusters (which mean the physical topology structures is 4). Figure 8(a) shows the detailed results of executing ten times when the number of logical topology structures equals to 4. From Figure 8(a) we can observer that the

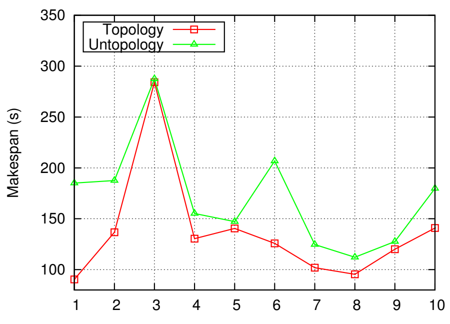
Table 3 Throughout of Different Method (Mop/s)

		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 7 Details of ten results



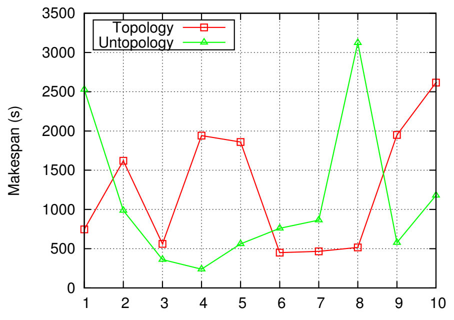
(a) Details of CG.9



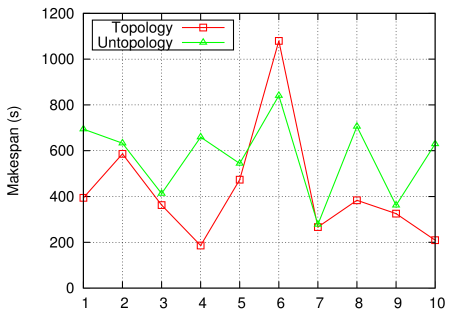
(b) Details of MG.9

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 logical 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. Figure 8(b) shows the results after changing the number of topology structures. We can observer 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). The reason is the nodes will be selected from the 1st and 2nd clusters for deployment when the logical topology structures are merged to 2, and the nodes selected from the 1st and 2nd clusters have better performance than the 3rd and 4th clusters. Therefore, Topology method will be better than Untopology method.

Figure 8 The results of CG.16 in different number of topology structures

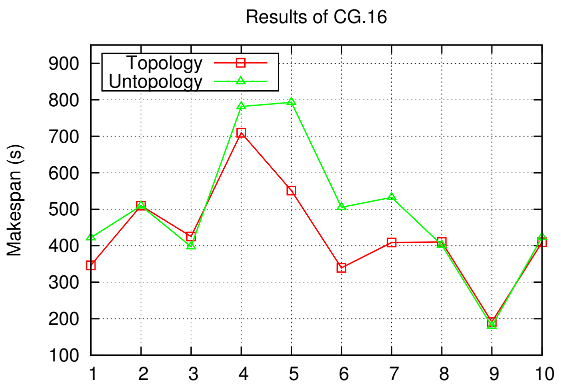


(a) The number of topology structure is 4



(b) The number of topology structure is 2

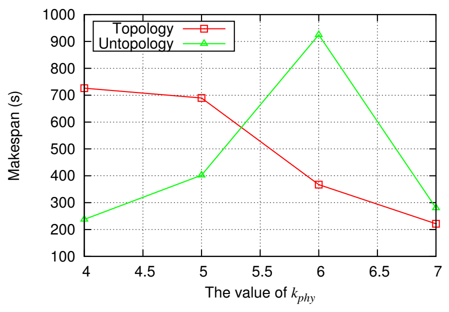
Figure 9 The result of $k_{log} > k_{phy}$



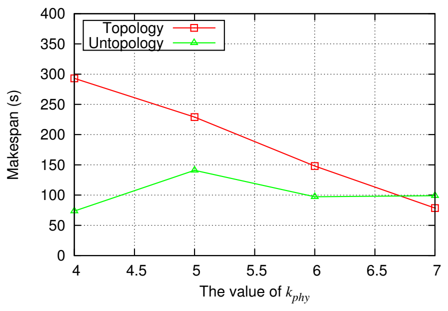
As introduced Section 4.4, there are two scenarios when mapping a logical topology to a physical topology. Firstly, the logical topology structures need to be reduced when $k_{log} > k_{phy}$. In this scenario, we randomly select 30 nodes from all nodes and set $k_{log} = 4$ and $k_{phy} = 3$, we deploy CG.16 on 16 nodes and run 10 times. We reduce the value of k_{log} to 2 by merging (1st, 2nd) and (3rd, 4th) topology structures. Figure 9 shows the detail results of run CG.16 10 times. In most cases, topology-aware method is better than Untopology method (the average makespan of Topology method is 430.1s, and that of Untopology method is 494.7s). This experiment shows our topology aware method can obtain better performance after decreasing the value of k_{log} to adapt k_{phy} when $k_{log} > k_{phy}$.

It is difficult to determine k_{log} when $k_{log} \leq k_{phy}$. As introduced before, we make the decision with respect to the value of k_{log}/k_{phy}. To study the impact of k_{log}/k_{phy}, we vary the values of k_{phy} from 4 to 7 with a step value of 1, and we set k_{log} are 4 and 2 in this experiment, respectively. The MPI programs used in this experiment are CG.16 and MG.16. All benchmarks were run 3 times, and we use the average result. Firstly, we set $k_{log} = 4$, and Figure 10 show the details of experiment results.

Figure 10 Impact of k_{log}/k_{phy} when $k_{log} = 4$



(a) The result of CG.16

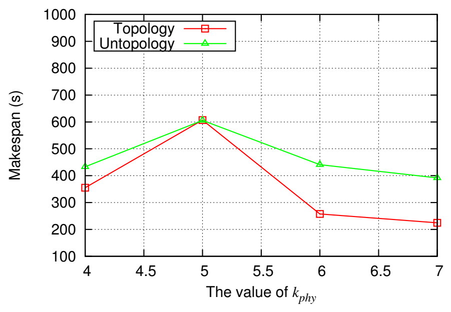


(b) The result of MG.16

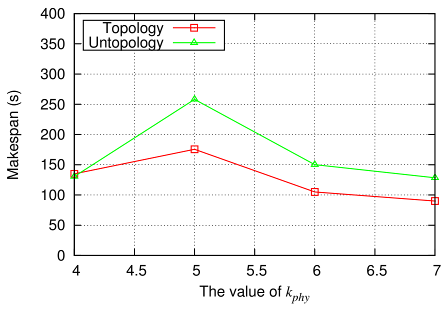
Figure 10(a) and Figure 10(b) show the CG.16 and MG.16 results of makespan. Figure 10 shows the makespan of topology-ware method decreases when the value of k_{phy} increased (means k_{log}/k_{phy} decreased) from 4 to 7. In addition, we can observe that the performance of Topology method is worse than Untopology method when $k_{log}/k_{phy} \geq 0.8$ (k_{phy} in [4-5]), and better when $k_{log}/k_{phy} \leq 0.67$ (k_{phy} in [6-7]). This is because when k_{log} is fine grain (*e.g.*, $k_{log} = 4$) and k_{phy} has a small value, it will map a logical topology to the clusters that have bad performance.

We conduct another experiment to study the impact of k_{log}/k_{phy}. In this experiment, we reduce k_{log} to 2 that is obtained by merging (1st, 2nd) and (3rd, 4th) topology structures. Figure 11(a) and Figure 11(b) show the results of CG.16 and MG.16. Figure 11 shows that, in most cases ($k_{log}/k_{phy} \leq 0.5$), topology-aware method has a better performance than Untopology method. The reason is logical topology structures be mapped to the first two physical topolgy structures that have better performance. From Figure 10 and 11, we can observe that the performance of topology-ware method is better than Untopology method when $k_{log}/k_{phy} < 0.6$. Therefore, we can reduce the value of k_{log} to make $k_{log}/k_{phy} < 0.6$ for obtaining a better performance when $k_{log} \leq k_{phy}$.

Figure 11 Impact of k_{log}/k_{phy} when $k_{log} = 2$



(a) The result of CG.16



(b) The result of MG.16

5.4 Collective operation experiments

In Section 4.3, we introduce to use topology information and hierarchical models to optimize collective operations. To justify the effectiveness of this method, we compare it with the Untopology method in (AAA et al, 2012). We use IMB benchmark in this experiment. IMB benchmark contains serials of benchmarks, and we choose **Scatter**, **Reduce**, **Alltoall**, **Bcast** and **Barrier** collective programs. We randomly select 14 and 15 nodes from all nodes, respectively. These nodes are partitioned in to 3 clusters. We deploy these collective operations applications on 8 nodes based on the topology information and the hierarchical model that introduced in Section 4.3. Each IMB benchmark should be run with some message lengths, except **Barrier**. In our experiments, the message lengths are varied from 2, 4, . . ., 1024 bytes (2^1-2^{10}). In order to obtain precise results, all benchmarks were run 5 times, and we use the average result.

Figure 12 and 13 show the results of running collective benchmarks. From the figures, we have the following observations:

- In Scatter and Reduce benchmarks, Topology method obtains the best communication performance, *i.e.*, has the least latency time under all the different message sizes. Same result is gotten also in Alltoall, Bcast and Barrier (Table 4).
- With the increasing of message size, the latency also increases, since nodes need more time to transfer messages.

Figure 12 Results of collective operations when nodes numbers is 15

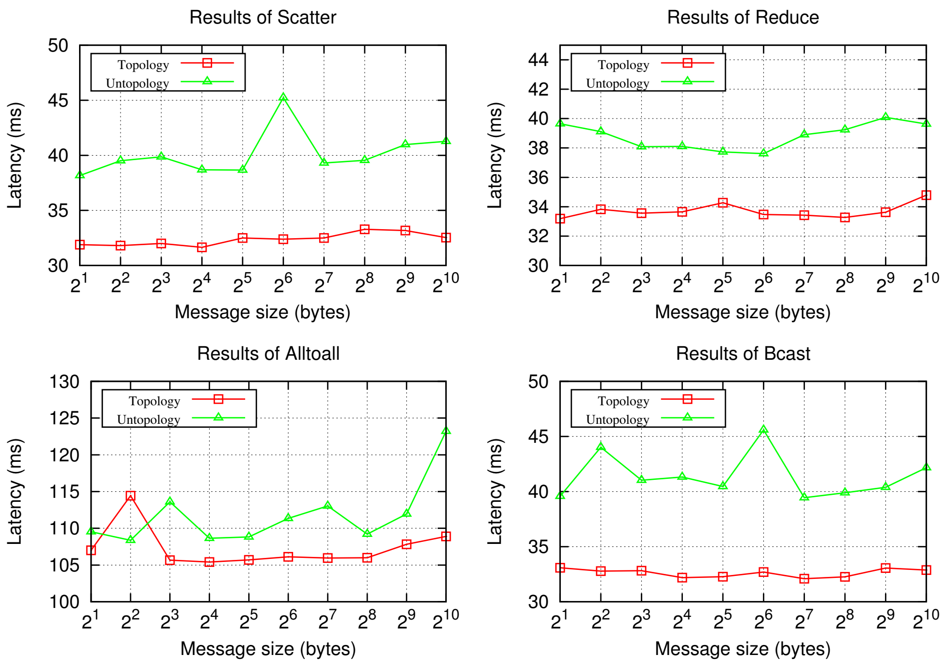


Figure 13 Results of collective operations when nodes number is 14

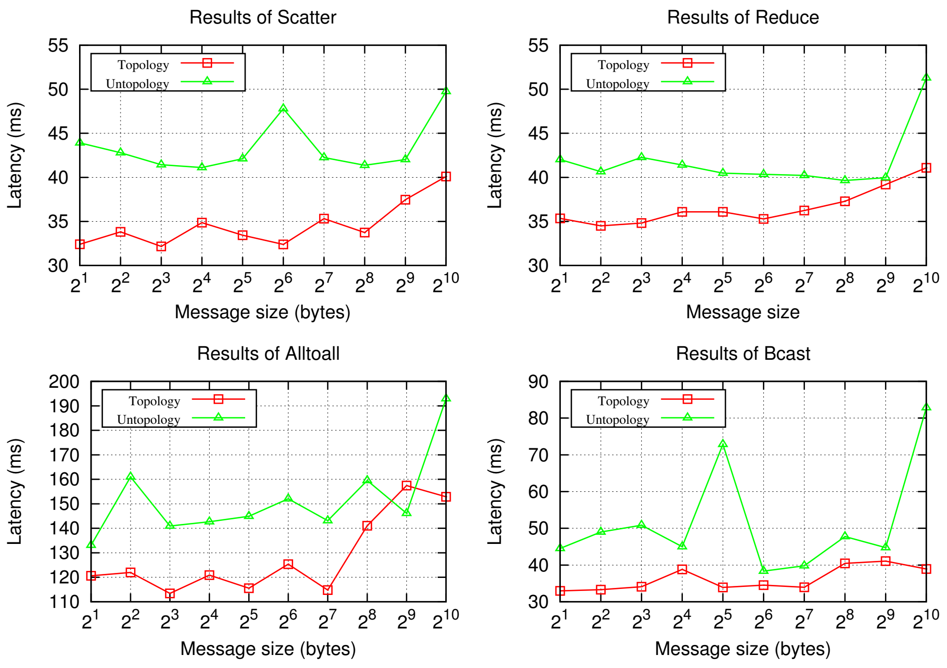


Table 4 Result of Barrier (ms)

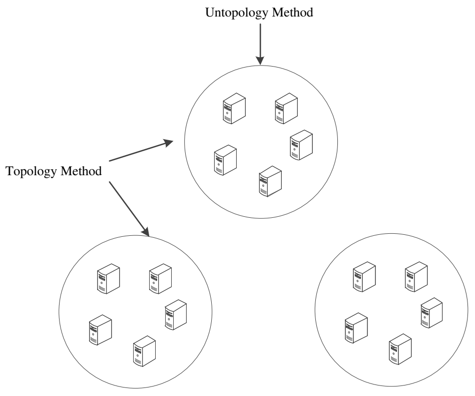
	15		14	
	Topology	Untopology	Topology	Untopology
1	123.9	149.7	135.8	315.4
2	115.1	145.3	140.9	141.2
3	133.8	140.3	128.9	163.8
4	140.3	128.9	144.8	199.5
5	130.4	205.8	142.6	151.7
Average	128.7	154.0	138.6	194.3

Instead of implementing collective operation algorithms, our optimization method analyzes the common topologies of collective operations, and uses the topology information and the hierarchical model to optimize collective operations. The experimental results show the effectiveness of our topology-aware deployment method.

5.5 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.* Figure 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 Figure 14.

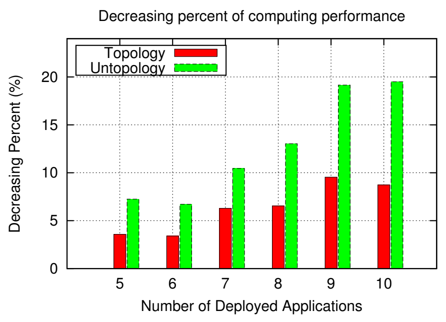
Figure 14 Process of deployment



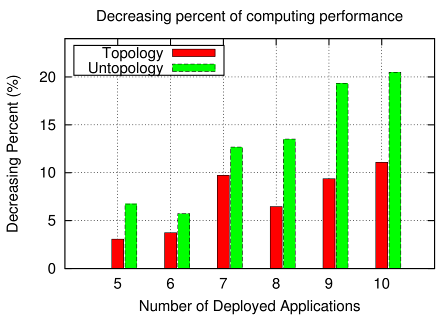
In order to obtain precise results, we partitioned all nodes into 3 clusters and 4 clusters for two methods, respectively. The benchmarks used in this subsection are CG.4, CG.8 and CG.16. The number of topology structures of all benchmarks is 2, which means we will deploy these applications on 2 clusters 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. Figure 15 and 16 shows the results of decreasing percents of computing and communication performances.

Figure 15(a) displays the results when partitioning all nodes into 4 clusters (*c.f.* Section 4.2), and Figure 15(b) shows the results of partitioning nodes into 3 clusters. These results show that:

Figure 15 Decreased percent of computing performance



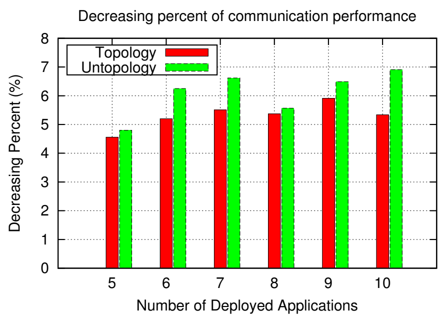
(a) Cluster number is 3



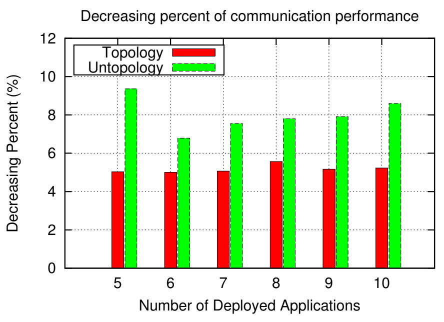
(b) Cluster number is 4

- 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.

Figure 16 Decreased percent of communication performance



(a) Cluster number is 3



(b) Cluster number is 4

Figure 16 shows the results of decreasing percents of communication performance, and the results are similar to those of the computing performance experiment.

6 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 user experiences when deploy application in cloud. Our next step includes the study of a user collaboration based method for deployment. In addition, online optimization is import schedule tasks on IaaS cloud system. Our furture work also includes the study of online schedule technology.

Acknowledgements

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, and the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 415311).

References

Ananth, G., Anshul, G., George, K. and Vipin, K. Introduction to parallel computing (2nd ed), Addison-Wesley, Boston.

Anderson, D.P. (2004) 'Boinc: A system for public- resource computing and storage', in *GRID 2004: Proceedings of 5th International Workshop On Grid Computing*, Pittsburgh, USA, pp.4–10.

Almási, G., Heidelberger, P., Archer, C., Martorell, X., Erway, C.C., Moreira, J., Steinmacher-Burow, B. and Zheng,Y. (2005) 'Optimization of MPI collective communication on BlueGene/L systems', in ICS 2005:Proceeding of the 19th International Conference on Supercomputing, Massachusetts, USA, pp.253–262.

Buyya, R., Abramson, D. and Giddy, J. (2000) 'An Economy Driven Resource Management Architecture for Global Computational Power Grids', in PDPTA 2000:Proceeding of 6th International Conference Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, pp.1–9.

Bhatele, A., Bohm, E.J. and Kalé, L.V. (2009) 'Topology aware task mapping techniques: an api and case study', in *PPOPP 2009: Proceeding of 14th International Symposium On Principles and Practice of Parallel Progamning*, Raleigh, North Carolina, pp.301–302.

Bar, P., Coti, C., Groen, D., Hérault, T. and Swain, M.T. (2009) 'Running Parallel Applications with Topology-Aware Grid Middleware', in *eScience 2009: Proceeding of 5th International Conference on e-Science*, Oxford, UK pp.292–299.

Budati, K., Sonnek, J.D., Chandra, A. and Weissman, J.B. (2007) 'Ridge: combining reliability and performance in open grid platforms', in *HPDC 2007: Proceedings of 3rd International Symposium On High Performance Computing and Communications*, Monterey, USA, pp.55–64.

Bessai, K., Youcef, S., Oulamara, A., Godart, C. and Nurcan, S. (2012) 'Bi-criteria workflow tasks allocation and scheduling in Cloud computing environments'. in *CLOUD 2012: Proceedings of 5th International Conference on Cloud Computing*, Honolulu, USA, pp.638–645.

Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M.and Bowman, M. (2003) 'Planetlab: an overlay testbed for broad- coverage services', *ACM SIGCOMM-Computer Communication Review*, Vol. 33 No. 3, pp.3–12.

Cappello, F., Fraigniaud, P., Mans, B. and Rosenberg, A.L. (2001) 'HiHCoHP: Toward a realistic communication model for hierarchical hyperclusters of heterogeneous processors', in *IPDPS 2001: Proceeding of 15th International Symposium on Parallel and Distributed Processing*, San Francisco, USA,pp.42.

Chan, A., Gropp, W and Lusk, E. (2008) '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.

Coti. C., Hérault, T. and Cappello, F. (2009) 'MPI Applications on Grid: A Topology Aware Approach', in *EuroPar 2009: Proceeding of 15th European Conference on Parallel and Distributed Computing*, Delft, The Netherlands,pp.466–477.

Chang, F., Viswanathan, R., and Wood, T.L. (2012) 'Placement in clouds for application-level latency requirements' in *CLOUD 2012: Proceeding of 5th Internation Conference on Cloud Computing*, Honolulu, USA, pp.327–335.

Chen, X.J., Zhang, J. and Li, J.H. (2011) 'A method for task placement and scheduling based on virtural machines', *KSII Transactions on Internet and Information System*, Vol. 5, No. 9, pp.1544-1572.

Evoy, M., Giacomo, V., Schulze, B. and Garcia, E. L. M. (2011) 'Performance and deployment evaluation of a parallel application on a private Cloud', *Concurrency and Computation: Practice and Experience*, Vol. 23, No. 17, pp.2048–2062.

AAA., BBB., CCC., DDD. and EEE. (2012) 'XXXXX',*XXXX*, XXX, XXX, pp.XXX–XXX.

AAA, BBB, CCC. and DDD. (2011) 'XXXXXXX', in *XXXXXX*, XXX, XXX, pp.XXX–XXX.

AAA, BBB, CCC, DDD and EEE. (2012) 'XXXX',*XXXXX*, Vol. 8, No. 1, pp.XXXX–XXX.

Faraj, A. and Yuan, X. (2005) 'Automatic generation and tuning of MPI collective communication rountines', in *SC 2005: Proceeding of 19th International Conference on Supercomputing*, Seattle, USA, pp.393–402.

Foster, I.R.I.T., Zhao, Y. and Lu, S. (2008) 'Cloud computing and grid computing 360-degree compared', in *GCE 2008: Proceeding of 4th International Workshop on Grid Computing Environments*, Austin, USA, pp.1–10.

Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L. and Woodall, T.S. (2004) 'Open MPI:Goals, Concept, and Design of a Next Generation MPI Implementation', in *PVM/MPI 2004: Proceeding of 11th European PVM/MPI User's Grop Meeting*, Venice, Italy, pp.97–104.

Gunarathne, T., Wu, T. L., Choi, J. Y., Bae, S. H. and Qiu, J. (2011) 'Cloud computing paradigms for pleasingly parallel biomedical applications', in *Concurrency and Computation: Practice and Experience*, Vol. 23, No. 17, pp.2338–2354.

Hadany. R. and Harel. D. (2001) 'A multi-scale algorithm for drawing graphs nicely',*Discrete Applied Mathematics*, Vol. 113, No. 1, pp.3–21.

Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B. and Good, J. (2008) ‘On the use of cloud computing for scientific workflows’, in *eScience 2008: Proceeding of 4th International Conference on eScience*, Indiana, USA, pp.640–645.

Hoefler, T., Rabenseifner, R., Ritzdorf. H., Supinski, B.R., Thakur, R. and Träff, L. (2011) ‘The scalable process topology interface of MPI 2.2’, in Concurrency and Computation:Practice and Experience, Vol. 23, No. 4, pp.293–310.

Jain, A.K., Murty, A.K. and Flynn, A.K. (1999) ‘Data clustering: a review’, *ACM Computing Surveys*, Vol. 31, No. 3, pp.264–323.

Jiang, D., Tang., C. and Zhang. A. (2004) ‘Cluster analysis for gene expression data: A survey’, *IEEE Transaction on Knowledge and Data Engineering*, Vol. 16, No. 11, pp.1370–1386.

Karypis, G., Han, E. and Kumar. V. (1999)‘Multilevel refinement for hierarchical clustering.’, Technical report, TR-99-020, Department of Computer Science, University of Minnesota, Minneapolis, 1999.

Kumar, R., Mamidala, A.R. and Panda, D.K. (2008) ‘Scaling alltoall collective on multi-core system’, in *IPDPS 2008: Proceeding of 22nd International Symposium on Parallel and Distributed Processing*, Miami, USA, pp.1–8.

Koehler, M., Ruckenbauer, M., Janciak, I., Benkner, S., Lischka, H. and Gansterer, W.N. (2010) ‘A grid services cloud for molecular modelling workflows’, *International Journal of Web and Grid Services*, Vol. 6 No. 2, pp.176–195

Karonis, N.T., Supinski, B.R., Foster, I.T., Gropp. M., Lusk, E.L. and Bresnahan, J. (2000) ‘Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance’, in *IPDPS 2000: Proceeding of 14th International Symposium on Parallel and Distributed Processing*, Cancun, Mexico, pp.377-382.

Kandalla, K.C., Subramoni, H., Vishnu, A. and Panda, D.K. (2010) ‘Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with scatter and gather’, in *IPDPS Workshop 2010: Proceeding of the 22nd International Symposium on Parallel and Distributed Processing*, Atlanta, USA, pp.1–8.

Kapoor, R., Porter, G., Tewari, M., Voelker, G. M. and Vahdat, A. (2012). ‘Chronos: predictable low latency for data center applications’. in *SoCC 2012: Proceeding of the 3rd ACM Symposium on Cloud Computing*, San Jose, USA, pp.1–14.

Kang, Y., Zheng, Z. and Lyu, M. R. (2012). ‘A latency-aware co-deployment mechanism for cloud-based services’. in *CLOUD 2012: Proceeding of the 5th IEEE International Conference on Cloud Computing*, Honolulu, USA, pp.630–637.

Kang, Y., Zhou, Y., Zheng, Z. and Lyu, M.R. (2011) ‘A user experience-based cloud service redeployment mechanism’, in *Cloud 2011: Proceeding of 4th International Conference on Cloud Computing*, Washington, D.C., USA, pp.227–234.

Lacour, S., Pérez, C. and Priol, T. (2005) ‘Generic application description model: toward automatic deployment of application on computional grids’, in *GRID 2005: Proceeding of 6th International Conference on Grid Computing*, Seattle, USA, pp.284–287.

Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X. and Gu, Z. (2012) ‘Online optimization for scheduling preemptable tasks on IaaS cloud systems’, *Journal of Parallel Distributed Computing*, Vol. 72, No. 5, pp.666–677.

Miguel-Alonso, J., Mercero, T. and Ogando, E. (2007) ‘Performance of an infiniband cluster running mpi applications’, Technical report, EHU-KAT-IK-03-07.

Newman, M.E.J. (2004) ‘Analysis of weighted networks.’, *Physical Review E*, Vol. 70, No. 5, pp.056131.

Noack, A. (2007) 'Energy models for graph clustering', *Journal of Graph Algorithms and Applications*, Vol. 11, No. 2, pp.453–480.

Nakada, H., Ogawa, H. and Kudoh, T. (2012). 'Stream processing with bigdata: Sss-mapreduce'. in *CloudCom 2012: Proceeding of 4th International Conference on Cloud Computing Technology and Science*, Taipei, TaiWan, pp. 618–621.

Noack, A. and Rotta, R. (2009) 'Multi-level algorithms for modularity clustering', in *SEA 2009: Proceeding of 6th International Symposium on Experimental Algorithms*, Berlin, Germany, pp.257–268.

Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T. and Epema, D.H.J. (2009) 'A performance analysis of ec2 cloud computing services for scientifc computing', in *CloudComp 2009: Proceeding of 1st International Conference on Cloud Computing*, Munich, Germany, pp.115–131.

Petruch, K., Stantchev, V. and Tamm, G. (2011) 'A survey on IT-governance aspects of cloud computing', *International Journal of Web and Grid Services*, Vol. 7, No. 3, pp.268–303.

Rabenseifner, R. (2004) 'Optimization of Collective Reduction Operations', in *ICCS 2004: Proceeing of 4th International Conference on Computational Science*, Krakow, Poland, pp.1–9.

Rao, S., Ramakrishnan, R., Silberstein, A., Ovsiannikov, M. and Reeves, D. (2012) 'Sailfish: A framework for large scale data processing'. in *SOCC 2012: Proceedings of the 3rd ACM Symposium on Cloud Computing*, Santa Clara, USA, pp.1–14.

Sonnek, J.D., Chandra, A. and Weissman, J.B. (2007) 'Adaptive reputation-based scheduling on unreliable distributed infrastructures', *IEEE Transactions on Parallel Distribute System*, Vol. 18, No. 11, pp.1551–1564.

Sun, Q., Wang, S., Zou, H. and Yang, F. (2011) 'QSSA: A QoS-aware Service Selection Approach', in *International Journal of Web and Grid Services*, Vol. 7, No. 2. pp.147–169.

Träff, J.L. (2002) 'Implementing the MPI process topology mechanism', in *SC 2002: Proceeding of 16th International Conference on Supercomputing*, Baltimore, USA, pp.1–14.

Taniar, D. and Leung, C.H.C. (2003) 'The impact of load balancing to object-oriented query execution scheduling in parallel machine environment ', in *Information Sciences*, Vol. 157. pp.33–71.

Taniar, D., Leung, C.H.C., Rahayu, W. and Goel, S. (2008) 'High Performance Parallel Database Processing and Grid Databases', John Wiley and Sons, Hoboken, NJ.

Xie, H., Yang, YR., Krishnamurthy, A,, Liu, Y. and Silberschatz, A. (2008) 'P4P: provider portal for applications', in *SIGCOMM 2008: Proceeding of 24th ACM SIGCOMM Conference on Data Communication*, Seattle, USA, pp.17-22.

Yuan, D., Yang, Y., Liu, X. and Chen, J. (2010) 'A data placement strategy in scientific cloud workflows', in *Future Generation Computer Systems*, Vol. 26, No. 8, pp.1200–1214.

Zheng, Z., Zhang, Y. and Lyu, M.R. (2010) 'Cloudrank: A qos-driven component ranking framework for cloud computing', in *SRDS 2010: Proceeding of 29th International Symposium on Reliable Distributed Systems*, Delhi, India, pp.184–193.

Zheng, Z., Zhou, T.C., Lyu, M.R. and King, I. 'Component ranking for fault-tolerant cloud applications', in *IEEE Transactions on Service Computing*, Vol. 5, No. 4. pp.540–550.