Efficient Process Mapping in Geo-Distributed Cloud Data Centers

Amelie Chi Zhou*
Shenzhen University

Yifan Gong
TuSimple

Bingsheng He
National University of Singapore

Jidong Zhai
Tsinghua University

ABSTRACT

Recently, various applications including data analytics and machine learning have been developed for geo-distributed cloud data centers. For those applications, the ways to map parallel processes to physical nodes (i.e., "process mapping") could significantly impact the performance of the applications because of non-uniform communication cost in such geo-distributed environments. While process mapping has been widely studied in grid/cluster environments, few of the existing studies have considered the problem in geo-distributed cloud environments. In this paper, we propose a novel model to formulate the geo-distributed process mapping problem and develop a new method to efficiently find the near optimal solution. Our algorithm considers both the network communication performance of geo-distributed data centers as well as the communication matrix of the target application. Evaluation results with real experiments on Amazon EC2 and simulations demonstrate that our proposal achieves significant performance improvement (50% on average) compared to the state-of-the-art algorithms.

KEYWORDS

Process Mapping; Geo-distributed Data Centers; Cloud Computing

ACM Reference format:
Amelie Chi Zhou, Yifan Gong, Bingsheng He, and Jidong Zhai. 2017. Efficient Process Mapping in Geo-Distributed Cloud Data Centers. In *Proceedings of SC17, Denver, CO, USA, November 12–17, 2017,* 12 pages.
DOI: 10.1145/3126908.3126913

1 INTRODUCTION

Many big data analysis applications involve analyzing a large volume of data generated in a geo-distributed manner. Many data-intensive applications, such as social networks, involve large sets of data spread in multiple geographically distributed (geo-distributed) cloud data centers [32]. For example, Facebook receives terabytes of text, image and video data everyday from users around the world. In order to provide reliable and low-latency services to the users, Facebook has built four geo-distributed DCs to maintain and manage those data. With increasing data volumes generated and stored across geo-distributed data centers, how to distribute computation to take advantage of data locality has become an emerging research topic, rather than aggregating all data required for computation to a single data center. Also, many data owners prohibit moving (raw) data out of data centers due to various reasons. On the one hand, the cost of moving data across geographic regions is usually large due to the huge data volume and the scarcity of cross-data center network bandwidth. On the other hand, regulations and laws such as data privacy do not allow raw data to be moved around geographically [16]. Thus, for those applications, one important problem is how to effectively and efficiently "move computation to data" in order to improve the performance of data analysis and better utilize the network bandwidth resources.

Currently, most cloud providers offer geo-distributed cloud services to serve users that need to run their applications in multiple geographic locations. For example, Amazon EC2 currently has 11 service regions geographically distributed around the world, Windows Azure operates from 20 regions and Google Compute Engine (GCE) provides services from four geo-distributed regions. Recently, various applications including data analytics [27, 47], machine learning [9] and high performance computing [39] have been developed for geo-distributed cloud data centers. All those applications adopt a "move computation to data" paradigm to take advantage of data locality and also satisfy other constraints like data privacy. Among those geo-distributed applications, the "move computation to data" problem for parallel and distributed data analysis programs is equivalent to a geo-distributed *process mapping problem* [25]. A process mapping problem decides how to map parallel processes to processors (or nodes) such that the application communication topology efficiently utilizes the physical links. Different process mapping decisions could lead to significant difference in the application performances. For example, effective process mapping solutions have shown significant performance improvement in the grid/cluster environments [10, 30]. Those algorithms have to be revisited in the geo-distributed cloud data centers.

The geo-distributed process mapping problem is non-trivial due to several reasons, which are rooted at the requirements of data-intensive applications in the geo-distributed environment. First, the network bandwidth in the geo-distributed cloud environment is highly heterogeneous. For example, the network bandwidth within a single geographic region is usually much faster than the bandwidth across regions (see Table 1). As we observe from real cloud environments, the cross-region network performance (including both bandwidth and latency) is often highly related to the geographic

*This work is partly done when Amelie Chi Zhou was working as a Postdoc in Inria Rennes, France.

SC17, Denver, CO, USA

DOI: 10.1145/3126908.3126913

distance of the regions. Those features are very different from the grid/cluster environments. Second, the high cross-region data movement cost and regulations and laws such as data privacy regulations add another dimension of constraints on the process mapping problem. Various approaches have been proposed to optimize the process mapping problem for arbitrary process topologies and arbitrary network topologies [25]. Some approaches require users to provide the topology information of the target machines [35]. However, few studies have discussed the process mapping problem in geo-distributed cloud environment with the consideration of unique network performance features and optimization constraints and new optimization approaches should be proposed.

In this paper, we study the geo-distributed process mapping problem, in consideration of the heterogeneous network bandwidths and data movement constraints. Our goal is to find the optimal mapping function in a timely manner so that the performance of the geo-distributed application can be maximized. Specifically, we build a model to mathematically formulate the geo-distributed process mapping problem as a constraint optimization problem. We further propose an efficient algorithm to guide the design of performance optimization, which fully considers the special features of the geo-distributed environment.

We make the following contributions.

- We propose a novel network model and mathematically formulate the geo-distributed process mapping problem with the awareness of heterogeneous network bandwidth and latency, as well as data movement constraints. The model can be used as an essential building block in many geo-distributed applications (e.g., [9, 27, 39, 47]).
- Based on the communication matrix of the application, we propose a geo-distributed greedy method to efficiently find near optimal solutions for the problem.
- We implement our optimization method on Amazon EC2 across four regions. As there are no public benchmarks for geo-distributed systems, we start with HPC applications and machine learning algorithms as micro benchmarks. Particularly, we evaluate our method on a number of HPC applications, including LU, BT and SP in NPB [5] benchmarks and two machine learning applications, including K-means clustering [29] and DNN [4]. Experimental results indicate that our proposed method can obtain 50% of performance improvement on average (up to 90%) compared to the state-of-the-art algorithms [12, 26].

The rest of this paper is organized as follows. Section 2 gives the background of process mapping and geo-distributed applications. We present the problem definition in Section 3 and our geo-distributed optimization method for the problem in Section 4. We present our evaluation results in Section 5 and summarize the related work in Section 6. Finally, we conclude this paper in Section 7.

2 BACKGROUND

In this section, we briefly introduce the preliminary and the related work on geo-distributed data centers and process mapping problems.

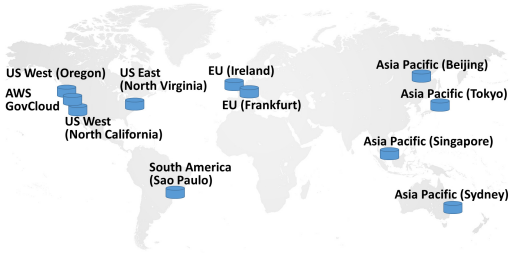


Figure 1: Geographic distributions of Amazon EC2 regions.

Instance type	US East	Singapore	Cross-region
m1.small	15	22	5.4
m1.medium	80	78	6.3
m1.large	84	82	6.3
m1.xlarge	102	103	6.4
c3.8xlarge	148	204	6.6

Table 1: Average network bandwidths (MB/sec) of five instance types within US East, Singapore, and between the two regions.

	US West	Ireland	Singapore
Bandwidth	21	19	6.6
Latency	0.16	0.17	0.35
Distance	Short	Medium	Long

Table 2: Average network bandwidths (MB/sec) and latencies (ms) of c3.8xlarge instances between US East and three other Amazon EC2 regions.

2.1 Geo-distributed Cloud

Many of existing public clouds, such as Amazon EC2, Windows Azure and Google Cloud Engine, provide multiple geographically-isolated data centers distributed around the world to guarantee low-latency and high availability service requirements. Figure 1 shows the geographic distribution of the 11 cloud data centers provided by Amazon EC2 (as of Nov 2015 [28]).

Since network performance is a key factor to the performance of process mapping, we have empirically studied the network performance features of data centers in the geo-distributed cloud environment. We have observed two special features across the data centers (i.e., regions) in Amazon EC2.

Observation 1: the network bandwidth across multiple regions is much smaller than that inside a single region. Table 1 shows that the network bandwidth within a single cloud region can be over ten times higher than that between two geo-distributed regions in Amazon EC2. This observation holds for different types of instances (i.e., virtual machines).

Observation 2: the cross-region network performance is highly related to the geographic distance of the regions. Table 2 shows that the network bandwidth between two regions with relatively short distance (i.e., US East and US West) can be three times higher than that between two regions with longer distance (i.e., US East and Singapore). We have similar observation on the cross-region network latency as shown in the table.

To demonstrate the generality of our observations, we also perform the measurement on Windows Azure regions and observe similar results as shown in Table 3.

	East US	West Europe	Japan East
Bandwidth	62	2.9	1.3
Latency	0.82	42	77
Distance	Intra-Region	Medium	Long

Table 3: Average network bandwidths (MB/sec) and latencies (ms) of Standard D2 instances within East US region and between East US and two other Window Azure regions.

These two observations clearly show the heterogeneity in geo-distributed cloud network performance. Because of such network performance heterogeneity, the process mapping problem has to be carefully revisited for efficiency. In this paper, we mainly focus our study on Amazon EC2, and leave the study on other cloud providers such as Windows Azure as future work.

2.2 Process Mapping Problem

Assigning a set of processes to machines such that the process communication efficiently utilizes the physical links in the network is called process mapping [25]. In the traditional process mapping problem, the communication pattern is represented by utilizing a weighted directed graph $G = V_G, E_G$.

- V_G is the set of processes, $V_G = \{v_i | v_i \in G\}$, where v_i is the vertex in the graph.
- E_G is the set of edges which represents the communications between any two vertexes. We further define the weights in E_G: W_G is the set of communication volumes between v_i and v_j (defined as $w_{i,j}$, $v_i, v_j \in V_G$). That is, $W_G = \{w_{i,j} | v_i, v_j \in V_G\}$. The weight is zero if no such communication occurs.

Besides the definition of the communication pattern graph, the (physical) interconnection performance is modeled by a weighted directed graph $T = M_T, D_T$.

- M_T is a set of machine nodes or processors, $M_T = \{m_i | m_i \in T\}$, where m_i is the vertex in the graph.
- D_T is a set of network performance (e.g., latency or bandwidth) between any two vertices. $D_T = \{d_{i,j} | m_i, m_j \in M_T\}$, where $d_{i,j}$ is the network performance between m_i and m_j.

The definition of graph mapping from G to T is denoted as a mapping graph $H'_{GT} =< V'_{GT}, E'_{GT} >$.

- V'_{GT} is a set of processes after being mapped to the set of machine nodes or processors, $V'_{GT} = \{v'_{ik} | v_i \in V_G, m_k \in M_T\}$, where v'_{ik} is the vertex of the mapping graph.
- E'_{GT} is the set of edges after being mapped. Accordingly, we define the weights W'_{GT} to be the set of communication costs after mapping. We have $W'_{GT} = \{fw_{i,j}, d_{k,l} | v'_{ik}, v'_{jl} \in V'_{GT}\}$, where fw, d is the mapping function which calculates the communication cost when the communication pattern graph edge is mapped to the interconnection network. It can be different equations for different applications.

The optimization goal of graph mapping is to minimize the real communication cost via intelligent mapping of the communication pattern graph to the interconnection network graph.

$$minimize \qquad COSTH'_{GT} \tag{1}$$

Name	size	Description
N	1	Number of processes
M	1	Number of sites
C_G	$N \times N$	Communication pattern matrix
A_G	$N \times N$	Number of messages between different processes
L_T	$M \times M$	Inter/intra-site latency
B_T	$M \times M$	Inter/intra-site bandwidth
PC	$M \times 2$	The physical coordinates of each site
I	$M \times 1$	Number of physical nodes in each site
C	$N \times 1$	Constraint vector
P	$N \times 1$	The process mapping result

Table 4: Notations in the paper

$$\text{where} \quad COSTH'_{GT} = {}_{c \in W'_{GT}}\, c = {}_{v'_{ik}, v'_{jl} \in V'_{GT}}\, fw_{i,j}, d_{k,l} \tag{2}$$

3 PROBLEM FORMULATION

We study the process mapping problem in geo-distributed cloud environments. In this section, we first point out the special design considerations brought by the geo-distributed cloud environment to the process mapping problem. Second, we mathematically formulate the problem as a constrained optimization problem. Also note that, our model is sufficiently general to be used as an essential building block in many emerging geo-distributed applications (e.g., [9, 27, 39, 47]). Table 4 summarizes the key notations in this paper.

3.1 Model Overview

Consider executing an application with N processes in M different sites (regions), where M is much smaller than N. We assume that the application is executed on instances of the same type, which is common in practice.

Compared to the traditional process mapping problem [25], we consider a more general model for geo-distributed applications. Particularly, we have three major considerations in our design, including network performance heterogeneity, data movement constraints and computation-communication patterns.

Network performance heterogeneity. As observed in real cloud environments (Section 2), the network performance in geo-distributed clouds is highly heterogeneous: 1) the intra-site and inter-site network bandwidths and latencies are highly different; 2) the inter-site network bandwidths and latencies among different pairs of sites are also highly different. We need to consider the heterogeneity of the geo-distributed network during our design.

Compared to the traditional process mapping problem, the all-link interconnection network graph T could not be directly obtained due to the high calibration overhead and network heterogeneity. In order to efficiently calculate the point to point performance with low measurement overhead, we propose to utilize the inter/intra-site performance model to replace interconnection network graph. We define L_T and B_T as two $M \times M$ matrices representing the latency and bandwidth between different sites, respectively. The element $L_T k', l'$ and $B_T k', l'$ are the latency and bandwidth between site k' and l'. The elements in the diagonal represent the intra-site latency and bandwidth, while the other elements are the inter-site latency and bandwidth. Because of the network heterogeneity, both of the two matrices are asymmetric. Based on the above definitions

of inter/intra-site latency and bandwidth, we can model the geo-distributed environment as a heterogeneous network architecture and reduce the measurement overhead from ON^2 to OM^2.

We further adopt the α-β model [45] to calculate the communication cost. In the α-β model, network performance is represented with two parameters: the latency (α) and the bandwidth (β). The transfer time for sending the data of n bytes is estimated to be $\alpha + \frac{n}{\beta}$. This model can be used to estimate the transfer time for a message of arbitrary sizes. While more sophisticated models such as LogP [17] and LogGP [2] exist, they involve more parameters and thus have higher calibration cost.

In order to fit for the α-β model, we define the communication pattern matrix C_G, where each element $C_G i, j$ indicates the volume of communication between process i and j. With our calibrated communication pattern matrix, the α-β model is lightweight and is sufficient for the needs of our problem. We define the count matrix A_G, where each element $A_G i, j$ represents the number of message sending from process i to process j. Based on these definitions, when the process i is mapped to site k' and process j is mapped to site l', the communication cost can be calculated as

$$\begin{aligned} fw_{i,j}, d_{k',l'} &= \sum_{p=1}^{A_G i,j} L_T k', l' + \frac{c_p i, j}{B_T k', l'} \\ &= \sum_{p=1}^{A_G i,j} L_T k', l' + \frac{\sum_{p=1}^{A_G i,j} c_p i, j}{B_T k', l'} \\ &= A_G i, j \times L_T k', l' + \frac{C_G i, j}{B_T k', l'} \end{aligned} \tag{3}$$

where $c_p i, j$ is the volume of communication between process i and j at the p-th message sending. As $C_G i, j$ is the total volume of communication between process i and j and $A_G i, j$ is the total number of communication between the two processes, we can indicate that $\sum_{p=1}^{A_G i,j} c_p i, j = C_G i, j$.

Data movement constraints. In the geo-distributed cloud environment, data movements can be constrained for different reasons, including the prohibitively large data movement cost and regulations and laws such as data privacy regulations. For example, many European countries have strict laws around data residency and many of the U.S. cloud providers start to open data centers in Europe. Due to such limitations on data movement, some processes may be constrained to execute on certain data centers and such processes could not be arbitrarily mapped. This case has not been considered in traditional process mapping, where all the data are inside the same region and there is no data movement limitation. We propose to use a constraint vector and formulate the geo-distributed process mapping problem as a constraint optimization problem. For simplicity, we define a constraint vector C with size $N \times 1$. It represents which process should be mapped to which site. In this paper, we only consider the data movement constraint on individual sites and leave the extension to multiple site constraints in our future work.

3.2 Problem Definition

We formulate the geo-distributed process mapping problem as a constrained optimization problem. Our goal is to minimize the cost function of applications while meeting data movement constraints.

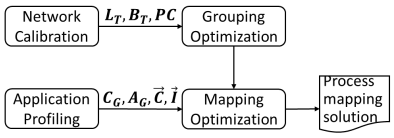


Figure 2: Overview of the optimization algorithm

We utilize P, which is an N-dimensional vector, to represent the mapping graph function H'_{GT}. The i-th element of P represents the site number that process i has been mapped to. I is an M-dimensional vector, where the i-th element represents the number of physical nodes in site i. C is an N-dimensional vector, where the i-th element represents the site number that process i should be mapped to, because of data movement constraints. If the value of C_i ($i = 0, 1, ...N$) is 0, it means that there is no constraint on process i. We further define a function $countm, n$, which counts the number of elements in n with value equaling to m. Based on the above definitions, the problem can be formulated as follows:

$$minimize \quad CostP \tag{4}$$

$$subject\ to \quad P - C \circ C = 0$$

$$and \quad count\, j, P \leq I_j \quad j = 1, 2, ...M \tag{5}$$

The two constraints need to be further discussed. The first one is the data movement constraint. The component wise multiplication indicates that for each process i, either $P_i = C_i$ or $C_i = 0$. The second one means that the number of processes mapped to site j should be equal to I_j, which represents the number of physical nodes in site j.

4 PROCESS MAPPING ALGORITHM

In this section, we propose an efficient method to solve the geo-distributed process mapping problem (formulated in Formula 4).

4.1 Algorithm Overview

Figure 2 gives an overview of the proposed optimization method. The mapping optimization algorithm takes the network calibration and application profiling results as input. Given the network calibration, we use grouping optimization to cluster nearby sites into large sites to reduce the optimization overhead.

The design rationale behind our proposal lies in the following major aspects. First, with network calibration and application profiling modules, we are able to automate the optimization procedure. Thus, users do not need to provide any information on the network or applications. Second, the efficiency and effectiveness are both important to the mapping optimization component. Since the solution space of geo-distributed process mapping problem is ON^M, it is infeasible to find the optimal solution in a timely manner. Our goal is to find a high-quality solution with a reasonably low overhead.

4.2 Algorithm Details

Application Profiling. The communication pattern matrix C_G and count matrix A_G are key structures for representing the traffic requirement of an application. Our implementation has considered two major aspects. First, the profiling method should be general for different types of applications. Second, the profiling method should be lightweight to avoid excessive runtime overhead.

There have been many different approaches for profiling communication patterns [42, 51]. In our implementation, we estimate the two matrices of applications using CYPRESS [51]. CYPRESS combines static program analysis with dynamic runtime trace compression. It extracts program structure at compile time, and obtains critical loop/branch control structures, which enable the runtime compression module to easily identify similar communication patterns. We run CYPRESS in an offline manner to obtain the communication pattern matrices.

CYPRESS well suites our requirement in two respects. First, the previous study [51] has already demonstrated the accuracy of CYPRESS. The performance improvement obtained by our proposal also validates that the accuracy of CYPRESS is sufficient for our application scenario. Second, the offline execution does not impose runtime overhead.

Network Calibration. In our algorithm, we need to obtain the parameter L_T and B_T. In order to calibrate the network performance of a site pair from Site k' to Site l', we select one instance in Site k' and one instance in Site l'. We then use the function Pingpong_Send_Recv in a benchmark called SKaMPI [41] to send and receive messages and measure the elapsed time. The latency L_T is the elapsed time of sending a one-byte message and the bandwidth B_T is calculated from the elapsed time of sending 8 MB data. In our experiment, when the message size is larger than 8 MB, the results are stable. Existing study [21] has shown that, using this pair-wise network calibration between instances can give effective performance optimizations in the cloud (without the explicit information of network topologies).

We keep measuring the inter-site and intra-site latency and bandwidth for several days. With those calibrated data, we calculate the average network latency and bandwidth to obtain the communication time using Formula 3. We find that the network performance of inter-site and intra-site is rather stable, generally with small variation (smaller than 5%). Although the variation of intra-site performance is relatively larger (as also observed in the previous study [22]), it only contributes a small fraction to the overall performance since the intra-site network performance is high.

The calibration overhead of our method is low. For example, we assume that there are 4 sites and 128 physical nodes per site, and the calibration overhead for each pair of node is one minute. The calibration overhead of the traditional approach [21] which measures the network performance of all node pairs would take over 180 days, while the overhead of our approach is only 12 minutes.

Mapping Optimization. Process mapping problems for arbitrary topologies and arbitrary network have been proved to be NP-hard [18]. For our problem with N processes and M sites, the solution space can be up to ON^M, which becomes very huge as N and M increase. The previous studies [18] have shown that there is no approximate algorithm that can effectively solve the problem.

We propose a novel approach with full consideration of the geo-distributed heterogeneous network architecture. The basic idea is as follows. First, we need to map the processes with data movement constraints to specified physical nodes. After that, we consider a mapping graph function for the sites, instead of the processes as in the previous studies [12, 26]. Given an order of the sites, we utilize a heuristic approach to generate a good mapping solution. Then we search for all the possible orders of the sites and generate a good mapping solution for each order. We select the solution with the best optimization results as our final mapping solution.

Grouping Optimization. If the number of sites M becomes large, the optimization overhead can increase dramatically ($OM!$). We modify our algorithm to improve the optimization efficiency in case of large number of sites. Motivated by the second observation in Section 2, we first group the nearby sites according to their physical distance and then treat those groups as large sites. Second, we utilize our algorithm on the new groups and recursively apply the proposed algorithm inside each group. In our algorithm, we group the sites by utilizing the K-means clustering method [29]. We use the physical coordinates PC and the Euclidean distance. The i-th element PC_i is a two-dimensional vector, whose first and second values are the latitude and longitude of the i-th site, respectively. The physical coordinates PC can be easily obtained from cloud providers' information. We select Forgy method [24] to determine the κ initial means. Based on our design, we can limit the number of groups in our approach and largely reduce the optimization overhead.

Algorithm 1 Overview of our process mapping algorithm

Input: L_T, B_T, C_G, A_G, PC, I and C as defined in Table 1, κ
Output: P

1: Utilizing K-means clustering algorithm to divide the M sites into κ groups;
2: Given an order of all the groups θ from $1, 2, ..., \kappa$ to $S_1, S_2, ..., S_\kappa$;
3: Set all the processes and sites as *unselected*;
4: $P^\theta_k = C_k$ and set process k as *selected* ($\forall C_k > 0$);
5: $I_j = I_j - count\, j, C$ ($\forall j = 1, 2, ...M$);
6: Set the site j as *selected* if $I_j = 0$ ($\forall j = 1, 2, ...M$);
7: **for** i=1,2,...,κ **do**
8: **for** j=1,2,...,N_i (N_i is the number of sites in group S_i) **do**
9: Select the *unselected* process $t_{0,j}$ with the heaviest communication quantity;
10: Select the *unselected* site $m_{S_i,j}$ in group S_i with the largest number of available nodes;
11: /* Map process $t_{0,j}$ to site $m_{S_i,j}$ */
 $P^\theta_{t_{0,j}} = m_{S_i,j}$ and set process $t_{0,j}$ as *selected*;
12: **for** r=1,2,..., $I_{m_{S_i,j}}$-1 **do**
13: Select the *unselected* process $t_{r,j}$ so that $t_{r,j}$ has the heaviest communication quantity with the *selected* processes in $m_{S_i,j}$;
14: /* Map process $t_{r,j}$ to site $m_{S_i,j}$ */
 $P^\theta_{t_{r,j}} = m_{S_i,j}$ and set process $t_{r,j}$ as *selected*;
15: Set the site $m_{S_i,j}$ as *selected*;
16: Calculate the cost function $COSTP^\theta$
17: Compare all the orders of the groups $\Phi = \{\theta | \theta : 1, 2, ..., \kappa \mapsto S_1, S_2, ..., S_\kappa\}$ and select the one θ' with the minimal cost as the final result $P = P^{\theta'}$;
18: **return** P;

4.3 Algorithm Design

Algorithm 1 shows the procedure of our geo-distributed process mapping approach. We first use K-means clustering algorithm to divide the M sites into κ groups (line 1).

For a given order of the groups, we map the processes with constraints to the required sites (lines 2-6) and then map the rest of the processes using a heuristic algorithm (lines 7-18). We briefly explain the heuristic algorithm. The algorithm starts at the unselected process t_0 with the heaviest communication quantity, and chooses

the site m_0 with the largest number of physical nodes. Next, the algorithm maps t_0's heaviest neighboring process in the same site until the site is full. The algorithm iteratively repeats the same step and the mapping process finishes when all processes have their mappings to the sites. We calculate the cost for each mapping solution P^θ (line 19) obtained by the heuristic algorithm and select the one with the minimum cost as the final solution (line 20).

Complexity analysis. Given an order of the groups θ, in order to select the k-th process, the communication quantity of the rest processes needs to be calculated, which requires $N-k-1$ times of calculations. The total calculation time is thus $N+N-1+...+1 = \frac{1+N\times N}{2} = ON^2$. As the number of all possible orders is $\kappa!$, the overall complexity of the algorithm is $O\kappa! \times N^2$. When we select a small value of κ (usually less than 5), the complexity of our algorithm is the same as the greedy algorithm [26] (ON^2) and lower than the MPIPP algorithm [12] (ON^3). The greedy algorithm and the MPIPP algorithm can be considered as the state-of-the-art algorithms for process mapping in the grid/cluster environments. In Section 5, we compare the overhead and performance improvement of our algorithm with those state-of-the-art approaches and analyze the scalability of our algorithm.

5 EVALUATIONS

We conduct four major sets of experiments to evaluate the effectiveness and efficiency of our proposed algorithm. Specifically, we study the optimization overhead of our approach in Section 5.2. We compare our algorithm with the state-of-the-art approaches on real cloud environment (Amazon EC2) in Section 5.3 to demonstrate the effectiveness of our algorithm. Further, we compare the algorithms using simulations to study the scalability in Section 5.4.

5.1 Experimental Setup

We adopt two complementary experimental settings, namely real clouds and simulations, to evaluate the compared algorithms. For real clouds, we perform the experiments on the Amazon EC2 clouds. For simulations, we use ns-2[1] to simulate a cluster. The simulations enable us to have full control of the network and to study the efficiency and effectiveness of our approach in different scales of geo-distributed environments.

Experiments with Amazon EC2. On Amazon EC2, we deploy our proposed algorithm across four geographic regions, namely US east, US west, Singapore and Ireland. We set up a cluster of 16 instances of m4.xlarge type from the same availability zone in each region. In our experiments, one process is attached to one instance. That means, the number of processes is 64 in total. We run each of the applications on Amazon EC2 for 100 times and calculate the average execution results for evaluations. The error bars represent the standard error of the results.

Simulations. As for simulations, we use the real traces of network performance in different regions calibrated in March 2016. We simulate the geo-distributed environment in different scales in numbers of instances, ranging from 64, 128, 256, ..., 4096 to 8192. We fix the number of regions to be four and the machines are evenly distributed in each region. With our grouping optimization, when the number of sites gets large, they are grouped into a reasonably small number of sites and thus the site dimension does not affect the scalability significantly. We also utilize the Monte Carlo method [11] to obtain the distribution of the solutions and compare the performance improvement of our algorithm with the "optimal" solution obtained using a large number of runs in Monte Carlo simulations.

Applications. Geo-distributed cloud data centers are suitable for many data analytics, machine learning and HPC applications, where their data are generated and stored geo-distributedly. As there is no public benchmark for geo-distributed systems, we start with HPC applications and machine learning algorithms as micro benchmarks. They cover different computation-communication behaviors.

We first apply our model to NAS Parallel Benchmarks (NPB [5]) kernels version 2.4. In order to measure the performance improvement, we select three pseudo applications, including BT (Block Tri-diagonal solver), SP (Scalar Penta-diagonal solver) and LU (Lower-upper Gauss-Seidel solver). The default problem size is CLASS C. We run each of the applications 100 times in a back-to-back manner to extend them to large scale computing.

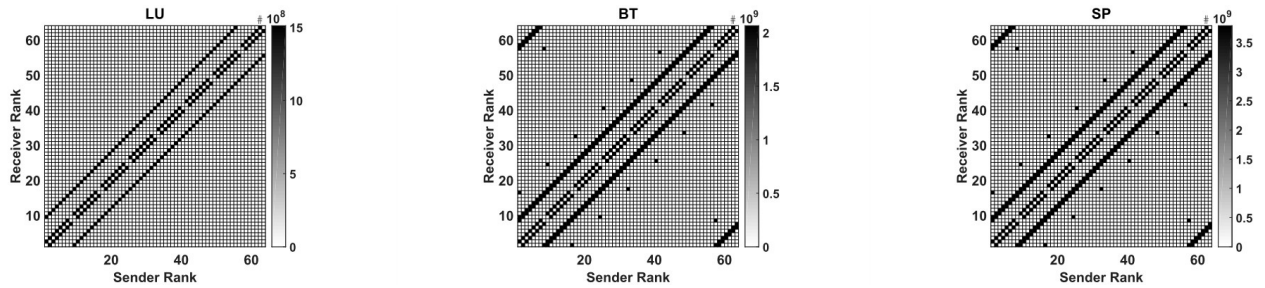
To further evaluate the impact of our geo-distributed algorithms, we have implemented two machine learning applications namely K-means clustering and deep neutral network (DNN). K-means clustering [29] is a popular unsupervised learning algorithm, aiming at partitioning n observations into k clusters, where each observation belongs to the cluster with the nearest mean. The algorithm is widely used in various topics, including computer vision, astronomy and agriculture. We select the parallel K-means clustering algorithm [29] to evaluate our proposed algorithm. DNN [4] is a commonly used algorithm for supervised learning, which is utilized to estimate or approximate functions that can depend on a large number of inputs. It is an artificial neural network with multiple hidden layers of units between the input and output layers. We utilize a parallel stochastic parallel gradient descent algorithm [52] to minimize the cost function.

Figure 3 shows the communication pattern matrices for the five applications from application profiling on 64 processes. We have three major observations. First, LU, BT and SP have near diagonal matrices. It indicates the existence of communication locality in these applications. For example, the process 1 only communicates with processes 2 and 8 for LU. There are only two types of message sizes, namely 43KB and 83KB. Second, for DNN, the total amount of message passing is small. It indicates that DNN is a computation-intensive application. Third, for K-means clustering, the communication pattern is complex, which requires careful design of the mapping function to obtain good performance improvement.

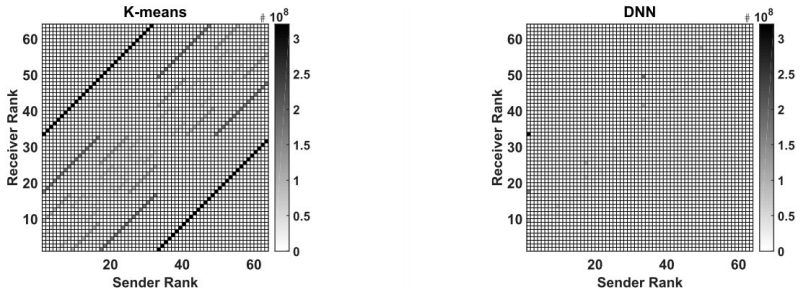
We use constraint ratio to simulate the data movement constraint. The constraint ratio represents the ratio of processes that have fixed mapping. When the ratio is 0, it means there is no constraint in the application. When the ratio is 1, it means the mapping function has been determined by the application and there is no optimization space. We set the constraint ratio to 0.2 by default for all the applications. Given a constraint ratio, we randomly choose the constrained processes and their mapped sites.

Comparisons. We assess the impact of the performance improvement by comparing the following four approaches.

[1] http://www.isi.edu/nsnam/ns/



(a) NPB applications



(b) Machine learning applications

Figure 3: Communication pattern matrices. The color scale indicates communication volume in Bytes (darker means heavier traffic).

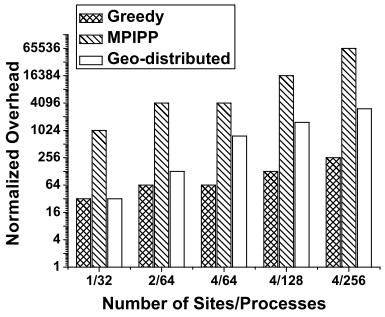


Figure 4: Overhead of compared algorithms under different scales (normalized to Baseline)

- **Baseline.** This approach simulates the scenario of running directly in the geo-distributed data centers without any optimization. We use the random mapping algorithm, which maps each vertex in the communication pattern graph to a vertex in the physical node graph randomly.
- **Greedy.** Hoefler et al. [26] propose a heuristic greedy algorithm for heterogeneous network architectures. We denote this approach as Greedy. To the best of our knowledge, this approach is the state-of-the-art process mapping algorithm.
- **MPIPP.** Chen et al. [12] propose an iterative algorithm to find the optimized mapping for arbitrary message passing applications. They modify the heuristic k-way graph partitioning algorithm [33] and achieve better performance improvement. We denote their approach as MPIPP.
- **Geo-distributed.** This approach is our proposed algorithm, which fully considers the network heterogeneity problem and data movement constraints.

MPIPP and Greedy are chosen as representative methods for process mapping problem in symmetric and asymmetric environments, respectively. Benefiting for its large searching space, MPIPP can outperform Greedy for some applications in the asymmetric environment.

5.2 Optimization Overhead

Figure 4 shows the optimization overhead of running the compared algorithms under different scales (in the format of "#sites/#processes"). All results are normalized to that of Baseline. In our experiment, we find that the overhead results are quite stable, and thus omit the error bars in the figure. The absolute overhead of Geo-distributed is less than 1 minute with four sites and 64 processes, which contributes to less than 1% of the total elapsed time of all applications.

We have four major observations on comparing the optimization overhead of different approaches. First, the overheads of Greedy and MPIPP can only be affected by the number of processes. As the number of processes increases, the overheads of Greedy is almost linearly increasing compared to Baseline and the overhead of MPIPP increases much faster than Greedy. Second, the overhead of Geo-distributed can be impacted by both the number of processes and the number of sites. Compared with the Baseline, the increasing speed is linear to the number of processes and is super-linear to the number of sites. This motivates our grouping algorithm to limit the solution space in a relatively small scale. Third, the overhead of MPIPP is much larger than the overhead of Greedy and Geo-distributed.

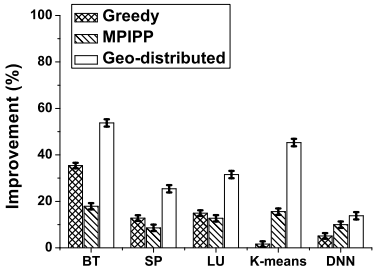


Figure 5: Overall performance improvement for the five applications on Amazon EC2 (normalized to the average of Baseline)

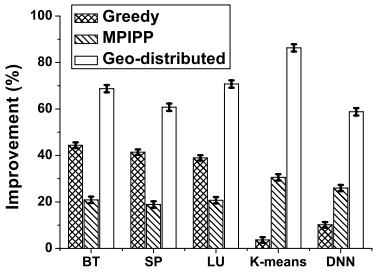


Figure 6: Overall communication performance improvement comparison for the five applications on simulation (normalized to the average of Baseline)

Lastly, when the number of sites is one, Geo-distributed is actually equivalent to Greedy and thus has the same overhead as Greedy. As the number of sites and processes increases, the overhead of Geo-distributed becomes larger than the overhead of Greedy. But when the number of sites is small, the overhead of the two approaches are comparable.

5.3 Results with Amazon EC2

For a fair comparison, all the results include the optimization overhead of running each approach. The overhead is generally smaller than 1% of the total execution time for Greedy and Geo-distributed.

Overall Performance Improvement. Figure 5 shows the overall performance improvement over Baseline for BT, SP, LU, K-means and DNN on Amazon EC2. Overall, Geo-distributed consistently outperforms other comparison approaches for all applications running on Amazon EC2. By fully considering the network performance heterogeneity in geo-distributed data centers, Geo-distributed effectively improves the communication performance.

We have four major observations on comparing the performance improvement. First, Greedy outperforms MPIPP with 20–25% performance improvement for BT, SP and LU. The reason is that communication pattern matrices of the three applications are diagonal and the heuristic greedy approach is very suitable for applications with good communication locality. Second, for k-means clustering, compared with Baseline, the performance improvement of Greedy is very small (less than 5%). It indicates that Greedy is not efficient for applications with complex communication pattern matrices. Third, MPIPP has similar performance improvement (10-20%) for all the applications, because MPIPP does not consider the special communication pattern matrices. Fourth, the performance improvement of DNN is small with all the three approaches due to the ratio of communication in DNN is small. Still, our approach is able to outperform other approaches.

5.4 Results with Simulations

Communication performance improvement. In the simulation, we focus on the communication time of each applications and ignore the computation and I/O time. This allows us to have a detailed understanding and a direct illustration of the real contribution of our proposed network performance aware process mapping approach.

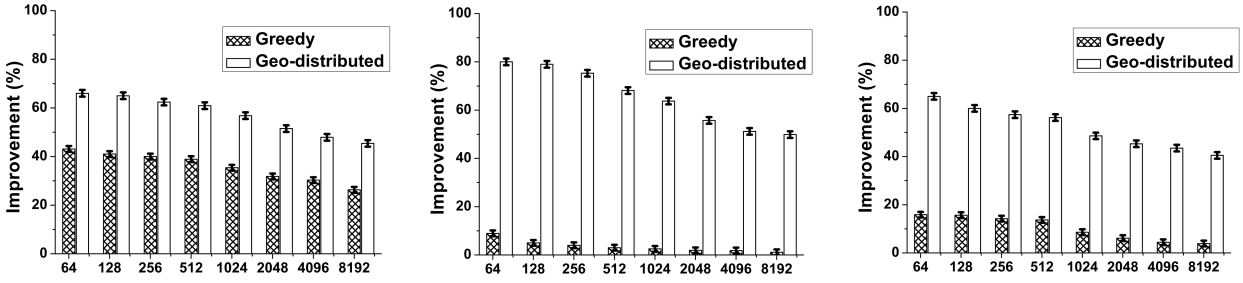
Figure 6 shows the performance improvement of the communication part for the five applications on simulation (normalized to that of Baseline). We utilize the same setting as those on Amazon EC2 and have two major observations. First, Geo-distributed can outperform Baseline by over 60% for all the applications. MPIPP can outperform Baseline by 20–30%. Greedy performs well in BT, SP and LU with more than 40% performance improvement. But the performance improvements of K-means and DNN are less than 10%. Second, compared with the performance improvement on Amazon EC2, the performance improvement is larger in simulations. The results are reasonable because we omit the computation and I/O time in our simulation, for which parts our algorithm has no contribution in their performance improvement.

Performance improvement in different scales. We simulate different scales of geo-distributed clusters from 64 to 8192 machines. When the number of processes is larger than 1000, MPIPP is very inefficient for its large runtime overhead and the performance improvement could not compensate the large overhead of executing the algorithm. So we only compare Greedy and Geo-distributed with Baseline in large scale clusters.

Figure 7 shows the performance improvement of LU, K-means and DNN in different scales. We obtain similar performance improvement of BT and SP with LU. We make the following observations. First, as the number of processes increases, the performance improvement starts to decrease for both Greedy and Geo-distributed. The reason is that the searching space of the optimization problem is increasing dramatically (O(N!)) and it becomes more and more difficult to obtain the optimal solution. Second, even when the number of processes reaches over 8000, Geo-distributed can still outperform Baseline with more than 50% performance improvement for all the applications. It indicates that our proposed algorithm can fit well for the large scale environment. Third, for K-means and DNN, Greedy has little performance improvement (less than 10%) compared with Baseline. But for LU, the performance improvement of Greedy is more than 30%. It shows that Greedy can only work efficiently for some special applications.

Data movement constraint study. We vary the constraint ratio of our problem and study the performance improvement.

Figure 8 shows the performance improvement of Geo-distributed with different ratios of constraints over the Greedy approach. We have observed similar results of improvement over other approaches.



(a) LU (b) K-means (c) DNN

Figure 7: Performance improvement in different scales (in numbers of machines)

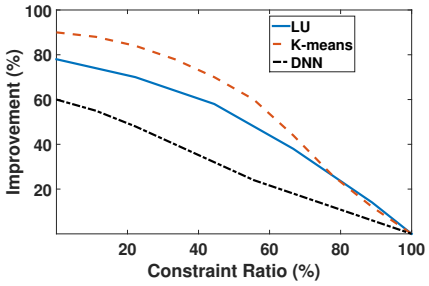


Figure 8: Performance improvement in different constraint ratios for data movement (over Greedy)

Thus, our approach achieves better performance compared to the other approaches for all different ratios. We have two major observations in this figure. First, the performance improvement functions of LU and K-means are concave. It indicates that when the ratio of constraints is small, the performance improvement decreases slowly with the increase of data movement constraint ratio. Thus, our algorithm works well for real-world applications with partial data movement constraints. For example, in case of different privacy levels, only data from sites with high privacy levels are constrained to their own sites, while data with low privacy levels are allowed to be mapped to those highly private sites. Second, for DNN, the performance improvement decreases almost linearly with the increase of constraint ratio.

Monte Carlo Results. We utilize Monte Carlo method [11] to obtain the distribution of communication time for different applications. For each measurement, we randomly map the processes to the physical nodes and simulate the communication time. We repeat the experiments for 10 million times so that the best result from Monte Carlo simulations keeps rather stable. The simulation setting is the same with that on Amazon EC2.

Figure 9 shows the simulation results for LU, K-means and DNN. We have two major observations. First, Geo-distributed is near optimal for all the applications. For LU, when randomly selecting a mapping function, the probability that the performance of such mapping is better than Geo-distributed is less than 1%. For K-means and DNN, the probability is even less than 0.1%. Second, Greedy outperforms MPIPP in LU, but is worse than MPIPP in the other two applications. For K-means and DNN, the performance improvement of Greedy is similar to the random mapping algorithm, which can be predicted as the performance with 50% probability.

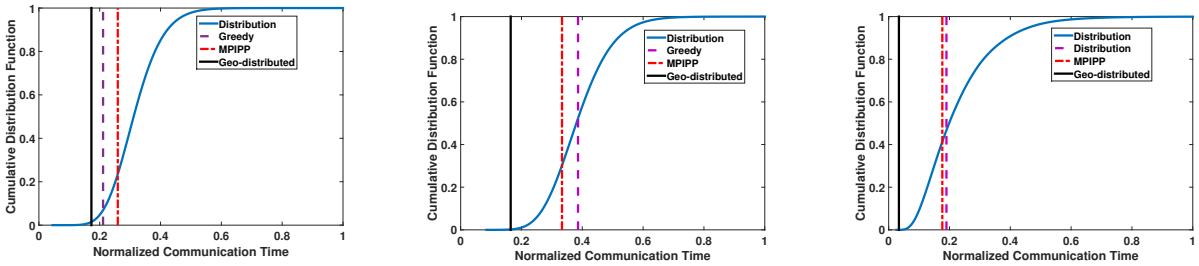
We further utilize Monte Carlo method to randomly map the processes to physical nodes for K times and then select the mapping function with the minimal communication time. We vary the parameter K and the evaluation results are shown in Figure 10. We observe that the decreasing speed of the minimum execution time of theses applications is nearly $\log(K)$. It indicates that the randomly mapping method is very inefficient for its low decreasing speed. The deep point of each application is the normalized execution time of Geo-distributed. Our proposed approach can obtain similar performance improvement as the best result of Monte Carlo method (the optimal solution) with much smaller value of K (only about 10^4).

6 RELATED WORK

In the following, we briefly review the recent and related work on geo-distributed cloud systems and the general process mapping problem. We find that achieving effective and efficient process mapping in geo-distributed clouds is still an open problem.

Scheduling problems in geo-distributed environments. Over the last few decades, many scheduling algorithms have been proposed for scientific applications running in the grid environment. Those studies mainly focused on the dynamicity, unpredictability and heterogeneity features of the grid, in order to provide either adaptive task scheduling [10, 43] and data management [13, 31, 40], or guarantee QoS requirements [1]. Scheduling algorithms such as Apache Oozie [3] and PIKACHU [20] have also been proposed for MapReduce-like applications executing in local clusters or a single data center. Those schedulers mainly focused on the features of MapReduce model, in order to improve load balancing or data locality. However, none of those studies has considered the unique features of network heterogeneities in geo-distributed clouds.

Recently, there are some studies focusing on various optimization problems in geo-distributed clouds. Kielmann et al. [30] proposed a library of collective communication operations for wide area systems, to reduce the traffic on the slow wide area links. Some big data frameworks are operated on geo-distributed clusters [6, 14] due to the geo-distribution of their input data. Corbett et al. [15] proposed a scalable geo-distributed database named Google Spanner. Rabkin et al. [38] presented JetStream, a stream processing system for OLAP data cubes, to support geo-distributed data analytics. Pu et al. [37]



(a) LU (b) K-means (c) DNN

Figure 9: CDF for the normalized communication time

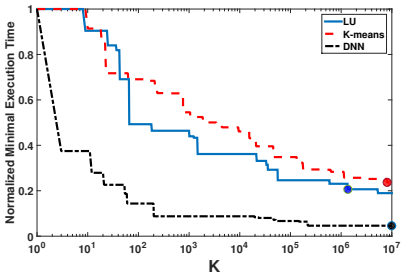


Figure 10: Normalized Minimal Execution Time

proposed a low-latency geo-distributed data analytics system called Iridium. They also consider the heterogeneous Internet bandwidths to optimize data and task placement specially for MapReduce and Spark. However, none of them has considered both the heterogeneity problem and the data movement constraints in cloud-based machine learning applications and/or application kernels (K-means, DNN, and LU for simple regression).

Besides data analytics, there have also been studies on other data-intensive applications such as scientific workflows [19, 44] and batch-based big data services [36]. Similar to those studies, this study also advocates the paradigm of "move computation to data" to the process mapping problem. Thai et al. [44] presented a framework for optimal deployment of geo-distributed workflow engines on Amazon EC2. Diaz-Granados et al. [19] proposed a framework to schedule scientific workflows onto federated multi-cloud environment with customized scheduling policies. The resource provisioning problem in geo-distributed cloud systems has been discussed in a number of existing studies [23, 36, 48–50]. Some of them target at optimizing the energy efficiency of batch jobs [36] and workflows [49] in geo-distributed data centers, utilizing the electricity price differences in different geographic locations. Cost-aware scheduling strategies are proposed for the geo-distributed cloud environments in order to minimize cross-region data transfers [50] and achieve cost-efficient task placement [23]. However, those related studies either consider a very different problem to process mapping or do not consider the network performance heterogeneity of geo-distributed data centers.

Process mapping problem. Many previous studies have been proposed to investigate the process mapping problem [25]. Lee et al. [33] proposed a heuristic k-way graph partitioning algorithm as the basis to solve the graph mapping problem. Chen et al. [12] improved those results and further proposed an iterative approach to obtain the optimal mapping solution. However, their strategies have low efficiency with prohibitively large calibration overhead. Bokhari [7] utilized graph isomorphism to solve the process mapping problem. But they did not consider the edges that are not mapped, which can reduce the performance improvement. Lee et al. [34] further proposed a two-stage optimization algorithm, which considers all the edges in the graph. Bollinger et al. [8] proposed an accurate model to solve the process mapping problem with a simulated annealing method. However, all of those studies focus on optimizing process mapping with arbitrary topologies on parallel computers, which fail to utilize the unique network performance features of geo-distributed data centers.

Träff et al. [46] considered the strictly hierarchical networks and proposed an implementation for such process mapping problems. Hoefler et al. [26] proposed a greedy heuristic algorithm for heterogeneous network architectures. Basically, the task with the largest data volume to transfer is mapped to the machines with the highest total bandwidth of all its associated links. Still, they do not consider the data movement constraints in geo-distributed environments. Particularly, while the approach proposed by Hoefler et al. [26] is designed for process mapping on general heterogeneous environments, our study takes advantage of the unique features of geo-distributed clouds, and develops efficient algorithms for finding good solutions. Therefore, existing approaches cannot be directly applicable or only offer an inefficient solution to our problem, as also demonstrated in the experiments. To the best of our knowledge, this paper is the first attempt to solve the process mapping problem in the environment of geo-distributed data centers.

7 CONCLUSIONS

Geo-distributed cloud data centers have become emerging for many distributed applications such as data analytics, machine learning and HPC. We believe that this trend will continue as increasing data volumes with different privacy requirements are being generated and stored across geo-distributed data centers. The process mapping problem is very important and challenging for the performance of applications running on such geo-distributed environments.

In our paper, we observe the unique properties of solving the process mapping problem in geo-distributed cloud data centers, including network performance heterogeneity, data movement constraints and computation-communication patterns of applications. Particularly, we formulate the process mapping problem in geo-distributed data centers as a constraint optimization problem, and develop a model to guide the design of performance optimization. The model can be used for current or future geo-distributed applications. We further propose an efficient greedy method to find near optimal solutions with low overhead. We implement our model on Amazon EC2 and evaluate its efficiency with NPB benchmarks and machine learning algorithms. Our experimental results show that 1) the process mapping optimization is critical for the performance of applications running on geo-distributed environments and 2) our approach can improve the performance over the state-of-the-art algorithm [26] by 50% on average (up to 90%). As future work, we plan to first extend this study onto different clouds such as Windows Azure and later consider the problem in the more complicated geo-distributed environment with multiple cloud providers.

ACKNOWLEDGMENTS

We would like to thank Prof. Amy Apon for shepherding the paper and anonymous reviewers from SC17. This work is supported by a MoE AcRF Tier 1 grant (T1 251RES1610) in Singapore, a collaborative grant from Microsoft Research Asia and NSFC Project 61628204 in China. Jidong Zhai's research is partly supported by National Key Research and Development Program of China 2016YFB0200100 and NSFC Project 61472201. Yifan Gong's research is partly supported by the HPC Group of TuSimple. Amelie Chi Zhou's work is partly supported by grant NSFC-Guangdong U1301252. Amelie Chi Zhou, Bingsheng He and Jidong Zhai are corresponding authors of the paper.

REFERENCES

[1] D. Abramson, J. Giddy, and L. Kotler. 2000. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?. In *IPDPS '00*. 520–.

[2] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. 1995. LogGP: Incorporating Long Messages into the LogP Model&Mdash;One Step Closer Towards a Realistic Model for Parallel Computation. In *SPAA '95*. 95–105.

[3] Apache. 2011. Apache Oozie. http://oozie.apache.org/. (2011).

[4] Ebru Arisoy, Tara N Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. 2012. Deep neural network language models. In *WLM '12*. 20–28.

[5] The NAS Parallel Benchmarks. *http://www.nas.nasa.gov/publications/npb.html*.

[6] Philip A Bernstein, Nathan Goodman, Eugene Wong, Christopher L Reeve, and James B Rothnie Jr. 1981. Query processing in a system for distributed databases (SDD-1). *ACM TODS* (1981).

[7] Shahid H Bokhari. 1981. On the mapping problem. *IEEE TOC* (1981).

[8] S Wayne Bollinger and Scott F Midkiff. 1991. Heuristic technique for processor and link assignment in multicomputers. *IEEE TOC* (1991).

[9] Ignacio Cano, Markus Weimer, Dhruv Mahajan, Carlo Curino, and Giovanni Matteo Fumarola. 2016. Towards Geo-Distributed Machine Learning. *CoRR* abs/1603.09035 (2016).

[10] Henri Casanova, Dmitrii Zagorodnov, Francine Berman, and Arnaud Legrand. 2000. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *HCW '00*. 349–.

[11] ED Cashwell and CJ Everett. 1959. Monte carlo method. *New York* (1959).

[12] Hu Chen, Wenguang Chen, Jian Huang, Bob Robert, and Harold Kuhn. 2006. MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. In *ICS'06*. 353–360.

[13] Ann Chervenak, Robert Schuler, Carl Kesselman, Scott Koranda, and Brian Moe. 2008. Wide Area Data Replication for Scientific Collaborations. *Int. J. High Perform. Comput. Netw.* 5, 3 (Oct. 2008), 124–134.

[14] Wesley W. Chu and Paul Hurley. 1982. Optimal query processing for distributed database systems. *IEEE TOC* (1982).

[15] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, and others. 2013. Spanner: Googles globally distributed database. *ACM TOCS* (2013).

[16] Court of Justice of the European Union . 2015. The court of justice declares that the commissions us safe harbour decision is invalid. https://curia.europa.eu/jcms/upload/docs/application/pdf/2015-10/cp150117en.pdf. (2015).

[17] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. 1993. LogP: Towards a Realistic Model of Parallel Computation. In *PPOPP '93*. 1–12.

[18] Josep Díaz, Jordi Petit, and Maria Serna. 2002. A survey of graph layout problems. *ACM Computing Surveys (CSUR)* (2002).

[19] J. Diaz-Montes, M. Diaz-Granados, M. Zou, S. Tao, and M. Parashar. 2017. Supporting Data-intensive Workflows in Software-defined Federated Multi-Clouds. *IEEE TCC* PP, 99 (2017), 1–1.

[20] Rohan Gandhi, Di Xie, and Y. Charlie Hu. 2013. PIKACHU: How to Rebalance Load in Optimizing Mapreduce on Heterogeneous Clusters. In *USENIX ATC'13*. 61–66.

[21] Yifan Gong, Bingsheng He, and Dan Li. 2014. Finding constant from change: Revisiting network performance aware optimizations on iaas clouds. In *SC'14*. 982–993.

[22] Yifan Gong, Bingsheng He, and Amelie Chi Zhou. 2015. Monetary cost optimizations for mpi-based hpc applications on amazon clouds: Checkpoints and replicated execution. In *SC'15*. Article 32, 12 pages.

[23] Lin Gu, Deze Zeng, Peng Li, and Song Guo. 2014. Cost Minimization for Big Data Processing in Geo-Distributed Data Centers. *IEEE TETC* 2, 3 (2014), 314–323.

[24] Greg Hamerly and Charles Elkan. 2002. Alternatives to the k-means algorithm that find better clusterings. In *CIKM '02*. 600–607.

[25] Torsten Hoefler, Emmanuel Jeannot, and Guillaume Mercier. 2014. An overview of topology mapping algorithms and techniques in high-performance computing. *High-Performance Computing on Complex Environments* (2014).

[26] Torsten Hoefler and Marc Snir. 2011. Generic topology mapping strategies for large-scale parallel architectures. In *ICS'11*.

[27] Chien-Chun Hung, Leana Golubchik, and Minlan Yu. 2015. Scheduling Jobs Across Geo-distributed Datacenters. In *SoCC '15*. 111–124.

[28] AWS Global Infrastructure. *https://aws.amazon.com/about-aws/global-infrastructure/*. accessed on Dec 2015.

[29] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. 2002. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE TPAMI* 24, 7 (2002), 881–892.

[30] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. 1999. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *PPoPP '99*. 131–140.

[31] Tevfik Kosar and Miron Livny. 2004. Stork: Making Data Placement a First Class Citizen in the Grid. In *ICDCS '04*. 342–349.

[32] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a Social Network or a News Media?. In *WWW '10*. 591–600.

[33] Cheol H Lee, Myunghwan Kim, Chan Park, and others. 1990. An efficient k-way graph partitioning algorithm for task allocation in parallel computing systems. In *ISCI '90*. 748–751.

[34] Soo-Young Lee and JK Aggarwal. 1987. A mapping strategy for parallel processing. *IEEE TOC* (1987).

[35] Avneesh Pant and Hassan Jafri. 2004. Communicating efficiently on cluster based grids with MPICH-VMI. In *Cluster Computing, 2004 IEEE International Conference on*. IEEE, 23–33.

[36] Marco Polverini, Antonio Cianfrani, Shaolei Ren, and Athanasios V. Vasilakos. 2014. Thermal-Aware Scheduling of Batch Jobs in Geographically Distributed Data Centers. *IEEE TCC* 2, 1 (2014), 71–84.

[37] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data analytics. In *SIGCOMM'15*. 421–434.

[38] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S Pai, and Michael J Freedman. 2014. Aggregation and degradation in JetStream: Streaming analytics in the wide area. In *NSDI'14*. 275–288.

[39] Aboozar Rajabi, Hamid Reza Faragardi, and Thomas Nolte. 2014. *An Efficient Scheduling of HPC Applications on Geographically Distributed Cloud Data Centers*. 155–167.

[40] Kavitha Ranganathan and Ian Foster. 2002. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *HPDC '02*. 352–.

[41] Ralf Reussner, Peter S, Lutz Prechelt, and Matthias Muller. 1998. SKaMPI: A detailed, accurate MPI benchmark. In *Recent advances in Parallel Virtual Machine and Message Passing Interface*. 52–59.

[42] Marcin Skowron, Mathias Theunis, Stefan Rank, and Anna Borowiec. 2011. Effect of affective profile on communication patterns and affective expressions in interactions with a dialog system. In *Affective Computing and Intelligent*

Interaction. 347–356.
[43] N Spring and Rich Wolski. 1998. Application level scheduling: Gene sequence library comparison. In *ICS'98*, Vol. 1.
[44] Long Thai, Adam Barker, Blesson Varghese, Ozgur Akgun, and Ian Miguel. 2014. Optimal deployment of geographically distributed workflow engines on the Cloud. In *CloudCom'14*. 811–816.
[45] Rajeev Thakur and Rolf Rabenseifner. 2005. Optimization of Collective communication operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (Feb. 2005), 49–66.
[46] Jesper Larsson Träff. 2002. Implementing the MPI process topology mechanism. In *SC'02*.
[47] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. 2016. CLARINET: WAN-Aware Optimization for Analytics Queries. In *OSDI'16*. 435–450.
[48] Xudong Xiang, Chuang Lin, Fu Chen, and Xin Chen. 2014. Greening Geo-distributed Data Centers by Joint Optimization of Request Routing and Virtual Machine Scheduling. In *UCC '14*. 1–10.
[49] Xiaolong Xu, Wanchun Dou, Xuyun Zhang, and Jinjun Chen. 2015. EnReal: An Energy-Aware Resource Allocation Method for Scientific Workflow Executions in Cloud Environment. *IEEE TCC* 1 (2015), 1–1.
[50] Lingyan Yin, Jizhou Sun, Laiping Zhao, Chenzhou Cui, Jian Xiao, and Ce Yu. 2015. Joint Scheduling of Data and Computation in Geo-Distributed Cloud Systems. In *CCGrid '15*. 657–666.
[51] Jidong Zhai, Jianfei Hu, Xiongchao Tang, Xiaosong Ma, and Wenguang Chen. 2014. Cypress: combining static and dynamic analysis for top-down communication trace compression. In *SC'14*. 143–153.
[52] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. 2010. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23*. Curran Associates, Inc., 2595–2603.

A ARTIFACT DESCRIPTION: EFFICIENT PROCESS MAPPING IN GEO-DISTRIBUTED CLOUD DATA CENTERS

A.1 Abstract

This description contains the information needed to launch some experiments of the SC17 paper submission "Efficient Process Mapping in Geo-Distributed Cloud Data Centers".

A.2 Description

A.2.1 Check-list (artifact meta information).

- **Algorithm:** A process mapping algorithm including application profiling, network calibration, grouping optimization and mapping optimization components.
- **Program:** C++ code.
- **Compilation:** C++ compiler. We used gcc version 5.4.0.
- **Data sets:** Publicly available benchmarks as indicated in the main paper.
- **Hardware:** For running our algorithm and the compared methods, we used a machine with 24GB DRAM and a 6-core Intel Xeon CPU. However, we do not have specific requirement on the hardware.
- **Output:** Process mapping solutions.
- **Experiment workflow:** Git clone source code, download datasets, and run scripts.
- **Publicly available?:** Yes.

A.2.2 How software can be obtained (if available). A URL will be released.

A.2.3 Hardware dependencies. None.

A.2.4 Software dependencies. Gcc compiler version 5.4.0 and CYPRESS system.

A.2.5 Datasets. The BT (Block Tri-diagonal solver), SP (Scalar Penta-diagonal solver) and LU (Lower-upper Gauss-Seidel solver) applications from the NAS Parallel Benchmark kernels version 2.4 (http://www.nas.nasa.gov/publications/npb.html) are chosen as HPC applications. The default problem size is CLASS C. We use publicly available nbody dataset (http://db.cs.washington.edu/projects/nuage/benchmark/astro-nbody/) for the K-means application. For the DNN application, we use the CIFAR-10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html) to train Resnet.

A.3 Installation

First, download and install the dependency software. Second, download the source code from the URL and use the makefile to build the executable.

A.4 Evaluation and expected result

After running an experiment, the results are available in the folder *results*. The results contain the process mapping solution to the tested application. Run the script in *scripts/simulate.sh* to obtain the simulated average and standard error of the performance results.