Finding Constant From Change: Revisiting Network Performance Aware Optimizations on IaaS Clouds

Yifan Gong
NEWRI
Interdisciplinary Graduate School
Nanyang Technological University, Singapore

Bingsheng He
School of Computer Engineering
Nanyang Technological University, Singapore

Dan Li
Tsinghua University, China

***Abstract*—Network performance aware optimizations have long been an effective approach to optimizing distributed applications on traditional network environments. However, the assumptions of network topology or direct use of several measurements of pair-wise network performance for optimizations are no longer valid on IaaS clouds. Virtualization hides network topology from users, and direct use of network performance measurements may not represent long-term performance.**

To enable existing network performance aware optimizations on IaaS clouds, we propose to decouple constant component from dynamic network performance while minimizing the difference by a mathematical method called RPCA (Robust Principal Component Analysis). We use the constant component to guide network performance aware optimizations and demonstrate the efficiency of our approach by adopting network aware optimizations for collective communications of MPI and generic topology mapping as well as two real-world applications, N-body and conjugate gradient (CG). Our experiments on Amazon EC2 and simulations demonstrate significant performance improvement on guiding the optimizations.

Keywords—Cloud Computing, Network Performance Aware Optimization, RPCA

I. Introduction

Infrastructure-as-a-service (IaaS) clouds have emerged as a popular computing infrastructure for many distributed applications. For example, many scientific and data-intensive applications have been deployed in Amazon EC2, Windows Azure and Google Compute Engine, including life sciences [26], [29], physics [31], [32], [28], [40], big data processing [17], [8] and others listed in Amazon case studies [1]. Compared with the traditional cluster and grid computing environments, cloud computing offers on-demand virtual machines in the *pay-as-you-go* manner. Every one with a credit card can buy the computational resources and storage from public cloud providers. Due to the pay-as-you-go nature, performance optimizations are important in not only improving the productivity but also reducing the total ownership cost. Network performance is often a key issue for the overall performance of distributed applications. Although there have been many research studies on designing novel network bandwidth allocations (e.g., [43], [2], [33]) or data center networks [16] for IaaS clouds, little attention has been paid to how applications can adapt their optimizations to IaaS clouds. Therefore, this paper revisits the network performance aware optimizations on IaaS clouds.

Network performance aware optimizations have long been an effective approach to optimize distributed applications on traditional network environments (e.g., local clusters and grids [39], [24], [21], [38], [3]). Those optimizations have the assumptions of the a-priori knowledge of network topology or direct use of several measurements of network performance. Essentially, those assumptions rely on estimating or measuring the all-link network performance in a cluster [19], [3]. Given the all-link performance, communication links are carefully selected for minimizing the network transfer time of the application. For example, one could select the best performing links for constructing the communication tree in an MPI collective operation [3].

When revisiting the network performance aware optimizations on IaaS clouds, we start with studying the network performance of a virtual cluster (a set of virtual machines). Data centers consisting of tens of thousands of commodity servers are the underlying infrastructure for IaaS clouds. Previous studies have studied the impact of virtualization [41] and network interference [4] in IaaS clouds. Machine pairs can have very different network performance as shown in the previous studies [14], [2]. That means, link selection continues to be important in virtual clusters, and network performance aware optimizations are still important to improve the application performance, especially for the communication-intensive applications. A natural question is whether and how we can apply existing network performance aware optimizations on virtual clusters of IaaS clouds.

Unfortunately, we find that the assumptions of existing network performance aware optimizations are no longer valid on IaaS clouds. The topology information is unavailable or inaccurate in virtual clusters. Virtualization hides the network hardware and topology from users, without exposing the actual configurations of the underlying hardware. Moreover, due to the cloud system dynamics such as virtual machine consolidation [37], flexible resource management [25] and dynamic network flow scheduling [4], the static topology information is no longer sufficient for representing the network performance. Some recent studies [9], [10] make optimization decisions based on only a few *ad-hoc* measurements on the end-to-end performance. However, such direct use of measurements is inherently affected by dynamic network and is inaccurate to reflect the long-term performance.

To enable existing network performance aware optimizations on IaaS clouds, we propose to decouple the constant component from the dynamic network performance while minimizing the difference between the network performance

SC14, November 16-21, 2014, New Orleans, LA, USA
978-1-4799-5500-8/14/$31.00 © 2014 IEEE

and the constant component. In our work, we treat the constant component as the component in the network performance that lasts for a long period until we observe some significant changes in the network performance. The difference can also be considered as *error*, since we use the constant component to guide network performance aware optimizations. It is a non-trivial task to find the constant component from dynamic network performance.

Interestingly, this problem can be cast into a common problem in the computer vision, named RPCA (Robust Principal Component Analysis) [6]. RPCA is to solve the following problem: for a data matrix, RPCA is used to identify a low-rank component and a sparse component with minimized norm, subject to that the sum of the two components are equal to the data matrix. There are many important applications with the data that can naturally be modeled as a low-rank plus a sparse component [6]. Specifically, we develop a novel approach based on RPCA with special design and optimizations for practical use on IaaS clouds, and leverage the theoretical properties of RPCA to find the constant component from the dynamic network performance. We model each row of the data matrix to be one snap-shot of all-link performance for the cloud at a certain point of time, and apply RPCA on that data matrix to obtain the constant component and error as the low-rank and sparse components, respectively.

This seemingly simple design of decoupling the constant component from network performance enables existing or new network performance aware optimizations in virtual clusters. Based on the constant component, conventional network performance optimizations become valid, i.e., we can select the best performing links with the minimized errors. On the other hand, with the error component, we are able to determine the effectiveness of network performance aware optimizations in virtual clusters, e.g., if the error is too large, the network of the IaaS cloud is too dynamic and network performance aware optimizations are useless.

We conduct our experiments with two complementary approaches: one is with the calibration on Amazon EC2 and the other is with a simulator based on ns-2. The first experiment is to assess our approach in the public cloud, and the latter one is for full control of the network traffic on a large-scale cluster. We assess the impact of network performance aware optimizations on two kinds of basic applications including collective communications of MPI (Message Passing Interface) [39] and the generic topology mapping strategy [19] as well as two real-world applications, N-body and conjugate gradient (CG).

Our experiments show that our RPCA-based approach can determine the degree of network dynamics for virtual clusters in the cloud. We find that the current network of Amazon EC2 is *relatively* stable, and network performance aware optimizations are still important on Amazon EC2. Moreover, our RPCA-based approach effectively guides the network performance aware optimizations. On Amazon EC2, the proposed approach significantly improves the performance, reducing the average elapsed time of broadcast and scatter of MPI and topology mapping by 20–40% and 8–20% over the baseline approach and the approach based on direct use of the network measurements. For N-body and CG, the average improvement can reach 25% and 31% over the baseline, respectively. In the simulation on ns-2, we compare with the topology-aware algorithm [21], [38] and find that our approach obtains 25–40% performance improvement.

The rest of the paper is organized as follows. We introduce the preliminary and related work on cloud networks, RPCA and two examples of network performance aware optimizations in Section II. We present the problem definition in Section III, and the RPCA-based approach in Section IV. In Section V, we show our experimental results. Finally, we conclude this paper in Section VI.

II. Preliminary and Related Work

In this section, we briefly introduce the preliminary and the related work that are closely related to our study.

A. Cloud Network

Previous studies (e.g., [2], [41], [8]) have shown significant variability between network performance of different machines in data centers. The network performance variability negatively impacts application performance, and also makes traditional network performance aware optimizations (e.g., [19], [3]) infeasible. To avoid re-inventing all those network performance aware optimizations, this paper develops a new approach to capture the long-term network performance in cloud, and allow existing/new optimizations applicable to cloud.

Researchers have developed mechanisms on network bandwidth allocation in order to obtain predictable performance for the user. ElasticSwitch [33], SecondNet [15], Oktopus [2] and TIVC [43] aim at reserving network bandwidth between each pair of VMs to offer guaranteed network bandwidth allocations. *Those studies are mainly from the cloud provider's perspective, whereas this paper optimizes the network performance for virtual clusters created by users, mainly from users' perspective.* Therefore, we do not have the information on the underlying hardware or topology, or the runtime dynamics about network transfers and virtualization details.

Network topology inference techniques have been investigated in the traditional environments [23], [36] and cloud environment [12]. We refer readers to a survey [7] for more details on classic techniques for network topology discovery and inference. The information given by basic diagnostic tools like *traceroute* is incomplete in the virtualized cloud.

B. Robust Principal Component Analysis

PCA is arguably the most widely used statistical tool for data analysis and dimensionality reduction. However, the accuracy of PCA is prone to noise or gross errors in the input data. Robust Principal Component Analysis (RPCA) [6] was proposed to improve the robustness of PCA under noisy or error measurements. The basic idea is to recover a low-rank matrix from a series of corrupted measurements and to minimize the noise component that is assumed to be sparse but unknown. Suppose A is a data matrix, D is a low-rank matrix and E is a sparse matrix. RPCA is to solve the following optimization problem, where $\|E\|_0$ is the zero norm of E.

$$\begin{array}{ll} minimize & rank(D) + \lambda \|E\|_0 \\ subject\ to & A = D + E \end{array}$$

RPCA has been widely used in computer vision. It can be used to solve many important applications like video surveillance, face recognition and latent semantic indexing [6]. In a video surveillance application, we need to identify the activities that occur in the background. Particularly, we consider each video frame as a row of the data matrix A. RPCA is used to divide A into two components: D representing the information of the background and E including the moving objects which may appear accidentally in each video frame. D is a low-rank matrix because the background is stable across the video frames of a reasonably long period. In the scenario of network performance in the IaaS cloud, we expect to find the constant component that can represent the long-term performance in the dynamic network environment. Our problem has the analogy to RPCA in computer vision.

There are many approaches that have been proposed to solve this optimization problem (e.g., [5], [20]). We choose the approach by Ji et al. [20] (their implementation [35]), which is a polynomial-time algorithm with strong performance guarantees on the error.

C. Network Performance Aware Optimizations

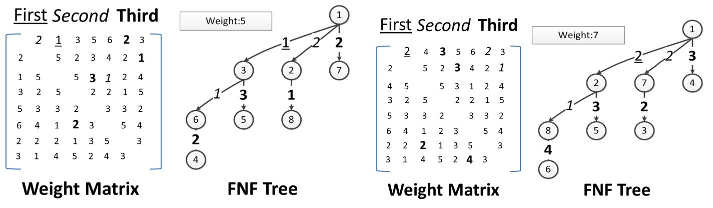
Network performance optimizations have been a hot research topic in the cluster/grid environments (e.g., [19], [3]) and cloud environments (e.g., [9], [10]). Many of them rely on the network topology, and some of them assume the a-priori knowledge of all-link (or pair-wise) network performance in the (virtual) machines under study. However, without understanding and capturing the long-term network performance of virtual clusters, the existing studies in the cloud [9], [10] simply adopt the methods in cluster/grid environments according to several *ad-hoc* measurements/calibrations.

We use two basic applications – MPI collective operations [24], [21], [38] and topology mapping [3] as examples to demonstrate network aware optimizations with the knowledge of pair-wise network performance.

MPI collective operations. Network performance optimization is very important for the overall performance of collective communications of MPI [24], [21], [38]. This study focuses on the four basic collective operations: broadcast, reduce, gather and scatter. Binomial tree is one of the basic communication tree structures for collective operations.

Due to the unevenness of pair-wise network bandwidth among the virtual machines [14], [2], we need to carefully choose the link in constructing the communication tree in MPI collective operations. It causes significant performance loss if the links are wrongly chosen.

Given all pair-wise network performance, the traditional approach can build an efficient communication tree without process migration so that the communication links can better utilize the network bandwidth. In this case, we assume that each process has been successfully allocated to the right machine. Given the all-link network performance for a set of machines, Banikazemi et al. developed a near-optimal greedy algorithm named Fastest-Node First (FNF) to construct a binomial tree [3]. The basic idea is described as below. We assume that each machine has only one process, the root of the communication tree is fixed and the extension to multiple processes per machine is straightforward. The algorithm works in multiple iterations. In each iteration, it maintains two sets of machines, S and U, to represent the machines that have been selected in the tree structure and have not been selected, respectively. Initially, S consists of the root for the binomial tree and U consists of all the machines under study except the root. In each iteration, for each machine s in S, we pick a machine in U having the best network performance with m, according to the pair-wise network performance. Then, the resultant machine (denoted as r) is removed from U immediately and is added to S after this iteration. That creates an edge in the tree from s to r (meaning that s and r are the sender and receiver, respectively). When considering a machine in S, the selection order is according to the order added into S. The algorithm stops when U becomes empty.



(a) A running example of FNF (b) Revised example of FNF

Fig. 1. An example of weight matrix as all-link network performance and FNF tree structure

A running example is given in Figure 1(a). On the left, it is the weight matrix for pair-wise network performance. A smaller weight indicates a better network performance. We assume that Machine 1 is the root. In the first iteration, Machine 1 is the sender and Machine 3 is chosen as the receiver for its smallest weight. Now, S consists of machines 1 and 3. In the next iteration, we choose Machine 2 to receive message from Machine 1 and Machine 6 to receive message from Machine 3. On the right of Figure 1(a) shows the resultant tree structure by the FNF algorithm. The total weight of the longest path is five.

Topology Mapping. Assigning a set of tasks to machines such that the task communication efficiently utilizes the physical links in the network is called topology mapping [19]. In this paper, we use the Greedy Heuristic Algorithm approach [19]. 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. Therefore, all-link network performance is important for guiding this mapping.

The algorithm is described as follows. Again, we assume that each machine has only one process, and the extension to multiple processes per machine is straightforward. The algorithm assumes two inputs: (1) a task graph G: a vertex representing a task and the edge represents data transfer between two tasks (its weight represents the data volume for the communication); (2) a machine graph H: a vertex representing a machine and the edge represents the connectivity between two machines (its weight represents the network bandwidth of the two machines).

Given the two directed graphs G and H, the algorithm determines the mapping between G and H. Now let the weight of a vertex v in a graph (either G or H) be the sum of the weights of all edges associated with v. The algorithm starts at the heaviest vertex v_0 in H, chooses the heaviest vertex s_0 in G and maps v_0 to s_0. Next, the algorithm maps v's heaviest neighboring vertices in H to the neighboring vertices in G with the heaviest connections. The mapping process finishes when all the vertices in H have their mappings to distinct vertices in G.

III. Problem Definition

We consider the scenario where users apply network performance aware optimizations to their applications running in the virtual cluster, built on the IaaS cloud. The unique cloud network performance features motivate us to understand the cloud network performance of virtual clusters and to investigate how to improve the application performance. This section gives the definitions on our problem.

Network performance. We can measure the performance of each link, and the all-link network performance for a set of N virtual machines (or instances). We denote this set of virtual machines to be a *virtual cluster*.

We adopt the α-β model [39] to model the network performance for each link. In the α-β model, each link is represented with two parameters: the latency (α) and the bandwidth (β) between the two machines. 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.

We define the pair-wise network performance with a matrix (namely *performance matrix*). Since the network performance varies, we extend the performance matrix with the time dimension. In particular, given two virtual machines i and j, the network performance parameters of the link from i to j at time t_0 is denoted as $\alpha_{ij}(t_0)$ and $\beta_{ij}(t_0)$. We represent the network performance of the virtual cluster with two $N \times N$ performance matrices: $L(t_0) = (\alpha_{ij}(t_0))$ and $B(t_0) = (\beta_{ij}(t_0))$ $(1 \leq i, j \leq N)$. In those matrices, we require all the pair-wise performance information between any pair of the instances in the virtual cluster.

We have two major motivations for developing the performance matrices. First, the performance matrices offer a general performance model which can be used for network performance aware optimizations of different applications. Second, individual link performance is significant to the network performance aware optimizations, because they rely on the comparison of the long-term network performance of all links. For example, in Figure 1(a), if we change the weight of the link (Machine 1, Machine 3) to be four, the tree structure will be largely changed. The new structure is shown in Figure 1(b) and the total weight of the longest path reaches seven (instead of five in the original case). This demonstrates the importance of individual link performance.

Temporal performance matrix. Performance matrix can only reflect a snapshot of network performance at a certain point of time. It does not necessarily represent the long-term network performance. We have performed a detailed study on the long-term network performance of a virtual machine pair in Amazon EC2 and made interesting observations. First, while the network performance from consecutive measurements forms a clear band, it is almost unpredictable at a single point. This is the combined effect from the constant and volatility components of the network performance. Second, we did observe significant network performance changes on Amazon EC2. Thus, our approach should be designed to handle this change. Due to the space limitation, we present the details of our observations in Appendix A of our technical report [13].

In order to capture the long-term network performance, we can perform a series of measurements (i.e., performance matrices). Each measurement on all-link network performance in the virtual cluster is one row. We arrange the n rows according to their measurement time and denote the matrix as N_A.

Formally, we define temporal performance matrix (*TP-matrix*) on $[T_0, T_1]$ $(T_0 < T_1)$ as follows, where $T_0 \leq t_i \leq T_1$, $t_i \leq t_{i+1}$, $P_{A_{t_i}}$ is the performance matrix of the virtual cluster at t_i. We layout each $P_{A_{t_i}}$ into a vector of N^2 dimensions by the row order. Thus, the *TP-matrix* is a $n \times N^2$ matrix.

$$N_A[T_0, T_1] = \begin{bmatrix} P_{A_{t_0}} \\ P_{A_{t_1}} \\ \vdots \\ P_{A_{t_{n-1}}} \end{bmatrix}$$

Problem definition. Consider the scenario of applying a network performance optimization to a network communication operation (e.g., a collective operation in an MPI application) in the virtual cluster from T_0 to T_1. Moreover, we consider an *offline* scenario that we have the a-priori knowledge of network performance at each moment from T_0 to T_1. If we run the network communication operation at time t_i $(T_0 \leq t_i \leq T_1)$, we can effectively apply the network performance optimization according to the pair-wise network performance $P_{A_{t_i}}$. However, this offline approach is impossible at practice, since we do not have the a-priori knowledge on the exact network performance in the future.

Without the a-priori knowledge on the network performance, we want to find a constant component from the time-varying network performance so that the difference between the network performance and constant component is minimized. The basic idea is, although we cannot predict the exact network performance of each link, the constant component represents the long-term performance of each link. According to the link performance in the constant component, we can perform the network performance aware optimizations (e.g., a link tends to have better performance if it has a better performance in the constant component). By minimizing the difference, we can obtain the best total performance of running at all t_i that we can achieve with the constant component.

Similar to the format of temporal performance matrix, we define two matrices:

- Temporal constant matrix (*TC-matrix*), where each row gives the estimated pair-wise network performance in the constant component. All rows are the same, and therefore the rank of the matrix is one. An example TC-matrix N_D is shown below, where all $P_{D_{t_i}}$ are the same.

$$N_D[T_0, T_1] = \begin{bmatrix} P_{D_{t_0}} \\ P_{D_{t_1}} \\ \vdots \\ P_{D_{t_{n-1}}} \end{bmatrix}$$

- Temporal error matrix (*TE-matrix*), where each row gives the estimated pair-wise network performance error. An example TE-matrix N_E is shown below.

$$N_E[T_0, T_1] = \begin{bmatrix} P_{E_{t_0}} \\ P_{E_{t_1}} \\ \vdots \\ P_{E_{t_{n-1}}} \end{bmatrix}$$

Let us formulate our problem in an offline setting. Given the TP-matrix N_A on [T_0, T_1], we define TC-matrix N_D and TE-matrix N_E accordingly. Our problem is formulated as below.

$$\begin{aligned} minimize \quad & \|N_E\|_0 \\ subject\ to \quad & N_A = N_D + N_E, \\ and \quad & rank(N_D) = 1. \end{aligned}$$

We note that, our current problem definition is still impractical, since it requires the knowledge of entire N_A. However, interestingly, this optimization problem can be cast into the RPCA problem (as we introduced in Section II). In the next section, we present an RPCA-based mechanism with a few calibrations on the virtual cluster, and use them to calculate the constant component for the network performance during the calibrations. By then, we need to perform re-calibrations and re-run the approach.

IV. RPCA-BASED APPROACH

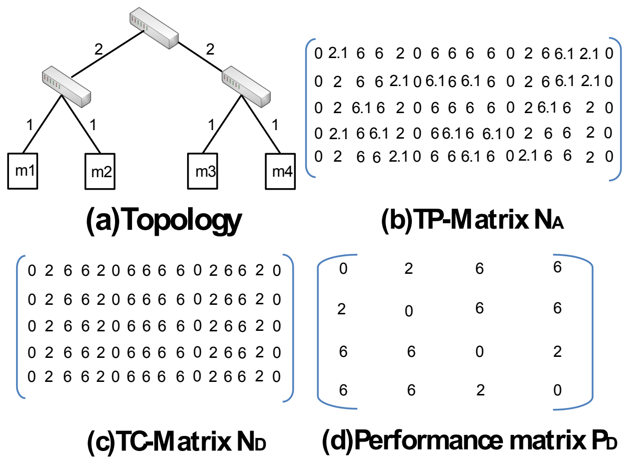
In this section, we present the details on our RPCA-based approach.

A. Approach Design

Recall that we have given an offline definition on finding the most effective constant component in Section III. We develop an adaptive approach to incorporate RPCA to approximate the solution to our problem. We have the following two major considerations in our design.

First, we need to detect the significant changes in the network performance of the virtual cluster in an IaaS cloud. When we say "significant", we mean that the change is so large that it affects the constant component. However, an IaaS cloud user does not have the underlying workloads in the cloud, and thus cannot accurately determine the significant changes in the network performance. On the other hand, one may propose to periodically measure the pair-wise network performance of the virtual cluster. That approach can be prohibitively costly, resulting in high overhead. Thus, we use a simple and lightweight approach on comparing the expected performance of the network communication operation with the real performance. If the real performance is significantly different from the expected performance, we decide that significant changes in the network performance of the virtual cluster occur.

Second, during the period that the network performance does not change significantly (the network performance does change dynamically though), we can safely use one constant component in a part of the period to estimate the constant component of the entire period. Basically, we perform a few measurements on the link-wise network performance on the virtual cluster. Using our RPCA-based approach, we obtain



$$\begin{pmatrix} 0 & 2.1 & 6 & 6 & 2 & 0 & 6 & 6 & 6 & 6 & 0 & 2 & 6 & 6.1 & 2.1 & 0 \\ 0 & 2 & 6 & 6 & 2.1 & 0 & 6.1 & 6 & 6.1 & 6 & 0 & 2 & 6 & 6.1 & 2.1 & 0 \\ 0 & 2 & 6.1 & 6 & 2 & 0 & 6 & 6 & 6 & 6 & 0 & 2 & 6.1 & 6 & 2 & 0 \\ 0 & 2.1 & 6 & 6.1 & 2 & 0 & 6 & 6.1 & 6 & 6.1 & 0 & 2 & 6 & 6 & 2 & 0 \\ 0 & 2 & 6 & 6 & 2.1 & 0 & 6 & 6 & 6.1 & 6 & 0 & 2.1 & 6 & 6 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 2 & 6 & 6 & 2 & 0 & 6 & 6 & 6 & 6 & 0 & 2 & 6 & 6 & 2 & 0 \\ 0 & 2 & 6 & 6 & 2 & 0 & 6 & 6 & 6 & 6 & 0 & 2 & 6 & 6 & 2 & 0 \\ 0 & 2 & 6 & 6 & 2 & 0 & 6 & 6 & 6 & 6 & 0 & 2 & 6 & 6 & 2 & 0 \\ 0 & 2 & 6 & 6 & 2 & 0 & 6 & 6 & 6 & 6 & 0 & 2 & 6 & 6 & 2 & 0 \\ 0 & 2 & 6 & 6 & 2 & 0 & 6 & 6 & 6 & 6 & 0 & 2 & 6 & 6 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 2 & 6 & 6 \\ 2 & 0 & 6 & 6 \\ 6 & 6 & 0 & 2 \\ 6 & 6 & 2 & 0 \end{pmatrix}$$

Fig. 2. An example of calculating $N_A = N_D + N_E$ with RPCA

the constant component in an offline manner. We then use that constant component for network performance aware optimizations until significant changes in the network performance of the virtual cluster occur.

Algorithm 1 shows the procedure of our RPCA-based approach on guiding the optimization for a network communication operation on a virtual cluster C. On the cloud, we first calibrate a series of performance matrices forming a TP-matrix N_A. N_A is the input data matrix A for RPCA. Next, we run the RPCA approach by Ji et al. [20], and get N_D and N_E. We choose Ji et al's approach for its efficiency and effectiveness over other approaches [20]. First, N_D is the same as matrix D in RPCA with rank one. All the rows in N_D represent the same comparison among different pair-wise performance, which are used as inputs to many network performance aware optimizations. Second, N_E is the matrix E in RPCA. It represents the performance error. We can calculate the norm of N_E to determine the effectiveness of optimizations. The process of calculating RPCA is very efficient, and the overhead can be ignored in the experiments.

In the following, we describe the details of these two aspects.

Guiding performance optimization with N_D. With N_D, we can apply traditional network performance aware optimization to the network communication operation, for example, applying the FNF algorithm [3] to construct the binomial tree. Note, the network communication operation can run on a virtual cluster C', where $C' \subseteq C$, and we can simply use the pair-wise performance belonging to C'. In most of the cases, our model can work well in a long term without any maintenance. But after a long period, we still need to recalibrate the matrix and rebuild the model. During this period, we use the same $P_{D_{t_i}}$ in N_D for many times until there is a significant change in the network performance (e.g., the virtual machine is migrated to another rack). We detect the changes on the network performance and re-calibrate the matrix (Lines 4–9). More details about approach maintenance will be described in the next sub-section.

Figure 2 illustrates an example of calculating $N_A = N_D + N_E$ with RPCA. Figure 2(a) shows a simplified topology of a virtual cluster of four machines. The number labeled on the edge between two machines represents the network performance of the link. We next perform five calibrations and form a TP-matrix (Figure 2(b)). Each row in the TP-matrix represents a performance matrix obtained from one calibration. Then, we run RPCA on the TP-matrix, and obtain a rank-one matrix N_D (Figure 2(c)). From N_D, we can obtain a performance matrix in Figure 2(d), which can be used for optimizations.

Algorithm 1 Overview of our RPCA-based approach

```
1: Given a virtual cluster $C$, calibrate the TP-matrix, and let the matrix be $N_A$;
2: Run the RPCA approach by Ji et al. [20], and get $N_D$ and $N_E$.
3: Given $N_D$, we apply some network performance aware optimization algorithms to the network communication operation $\mathfrak{A}$ running on a virtual cluster $C'$, where $C' \subseteq C$;
4: Measure the network performance of $\mathfrak{A}$ and let it be $\mathfrak{t}$;
5: Let the expected performance of $\mathfrak{A}$ be $\mathfrak{t}'$;
6: if $\frac{|\mathfrak{t}-\mathfrak{t}'|}{\mathfrak{t}'} \geq threshold$ then
7:     Go to Line 1; /* update maintenance*/
8: else
9:     Go to Line 3; /* use the same $N_D$ for later optimizations*/
```

Determining the effectiveness of optimizations. We aim to study the relationship between the performance error and the effectiveness of optimizations. When we view the measurement performance N_A as the most effective optimization solution from off-line, the performance error is the difference between the most effective solution and our performance aware optimizations solution which is based on N_D. Thus, the effectiveness of network performance aware optimizations on IaaS clouds is highly correlated with the performance error, N_E. Thus, we define the relative norm of error matrix N_E, $Norm(N_E) = \frac{\|N_E\|_0}{\|N_A\|_0} (0 \leq Norm(N_E) \leq 1)$ to measure the effectiveness of network performance aware optimizations in the cloud.

B. Implementation Details

The previous sections have presented that we can use RPCA to capture the long-term performance of the network in virtual cluster. However, there are some implementation issues that are worth discussions.

Model calibration. We need to obtain the parameter $\alpha_{ij}(t)$ and $\beta_{ij}(t)$ in a given interval $[T_0, T_1]$ and use this data to predict the network performance in the future. In order to calibrate the network performance of an instance pair from Instance i to Instance j, we use the function Pingpong_Send_Recv in a benchmark called SKaMPI [34] to send and receive messages and to measure the elapsed time. The latency $\alpha_{ij}(t)$ is the elapsed time of sending a one-byte message and the bandwidth $\beta_{ij}(t)$ is calculated from the elapsed time of sending 8MB data. In our experiment, when the message size is larger than 8 MB, the results are stable.

We have to calibrate each cell of the performance matrix. There have been some network coordinate algorithms (e.g., [11], [30]) to obtain the all-link network performance with a smaller number of cell measurements. Those approaches are not applicable to data center networks, because the triangle condition is not satisfied [4], [22].

The overhead of calibrating the performance one by one pair is too high if N is large. In order to reduce the overhead, at each step we choose $\frac{N}{2}$ instances to send messages and the other $\frac{N}{2}$ to receive. In this way, we could obtain $N/2$ pairs at a time, and the time consumption is $2 \times N$. While it significantly reduces the calibration overhead, the concurrent message transfers may cause interference with each other. Fortunately, the data center is usually large enough in the scale of tens of thousands of servers, and the interference of the virtual cluster (in the scale of hundreds of virtual machines) should be small. We study the impact of different scale in Section V.

The number of rows in the TP-matrix is another key tuning parameter in RPCA. We call this parameter *time step*. If the time step is too large (i.e., the TP-matrix consists of too many rows), the results may be more accurate, but the overhead is too high. In contrast, if the time step is too small, the result obtained from RPCA may not fully reflect the long-term performance. We experimentally evaluate the impact of different time steps in Section V.

Update maintenance. We use the real performance as feedbacks for the update maintenance. As Lines 4–9 in Algorithm 1, we monitor the performance of the network communication operation, and then compare the expected performance estimated from history. To support different data sizes in the performance estimation, we use α-β model [39] to estimate the network performance, with the input of N_D. If we find the difference is more than $threshold$, we could conjecture that the network performance has significant changes. Thus, we need to re-calibrate the TP-matrix and re-run the approach. The parameter $threshold$ is a key parameter for update maintenance. We experimentally evaluate its impact in Section V.

V. Evaluations

This section presents our experimental results on evaluating the proposed model and applications. Overall, there are three groups of experiments. First, we study the overhead of calibrating the temporal performance matrix (Section V-B). Second, we study the performance impact of our RPCA-based approach in comparison with other baseline and heuristics approach on real cloud environments (Section V-D). Our experiments are running on Amazon EC2. Third, we study the impact of our approach in a large-scale cluster with simulations (Section V-E).

A. Experimental Setup

We use two complementary evaluation approaches including real experiments and simulations. The real experiments were performed on Amazon EC2 in August 2013, with the focus on assessing the practical performance impact of our proposed approach. As for simulations, we use ns-2[1] to simulate a cluster. With the network simulations from ns-2, we are able to define different background traffics, and study the impact of network interference. Moreover, we can simulate different network topologies, and compare the topology aware optimizations with our approach.

Let us present more details about the experimental setup for each approach.

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

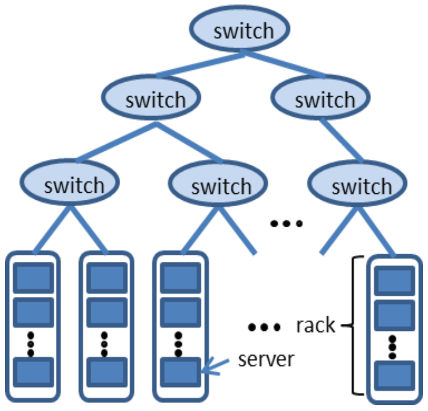


Fig. 3. Tree-structured network topology

Experiments with Amazon EC2. On Amazon EC2, we consider different scales of virtual cluster in the real cloud environment. In particular, we consider two virtual clusters: one with 64 medium instances and the other with 196 medium instances. By default, we report the results for 196 medium instances.

For each virtual cluster size, the real experiment takes around one week, with one experimental run every 30 minutes. In each run, we run the following experiments one by one: calibration, MPI and topology mapping applications. For each application, we run the compared algorithms one by one. The calibration is to generate the trace for network performance of virtual cluster. We replay the trace to validate our findings on Amazon EC2 for more details. The trace essentially forms a TP-matrix of our experimental period under study. Given the network performance measurements at a point of time, we use α-β model [39] to estimate the performance of one application. We use the trace to study the impact of optimizations and tunings.

Simulations. Recall that many data centers adopt the tree-structured network design [22], [4]. As a start, we use the tree-structured topology to interconnect servers, as illustrated on Figure 3. Machines are first grouped into racks, and then racks are connected with higher-level switches. Specifically, we simulate a cluster of 1024 machines. There are totally 32 racks and each rack contains 32 servers. There are two level of switches. In the first level, there are 32 switches (one for each rack). In the second level, only one switch link with the 32 switches in the first level. The bandwidth in the same rack is 1Gb/s and the bandwidth between different racks is 10Gb/s. To simulate the shared environment like IaaS clouds, we make some of the machines keep on sending messages to some others. We call it background traffic. We first choose the links and then vary two parameters to control the background traffic: message size and the distribution of waiting time between sending the message. For each link, we assume the waiting time satisfies poisson distribution and the expected value is λ. We vary these two parameters to study the impact of errors in our RPCA-based approach (N_E). Applications are run in the simulator, independently with the background traffic. We measure their performance in the similar way as they run on the real cloud environment.

Applications. We apply the proposed RPCA-based approach to two kinds of basic applications – MPI collective operations and topology mapping. As we introduce in Section II, both applications have network performance aware optimizations according to pair-wise network performance in the virtual cluster. In MPI, we study the basic collective operations including broadcast, reduce, scatter and gather. We obtain similar results of reduce and gather as broadcast and scatter, respectively. This is not surprising, because reduce and gather are the dual operations of broadcast and scatter, respectively. Thus, we present the results for broadcast and scatter only. We report the results for the following default setting, unless otherwise specified. The message size for broadcast and scatter is 8MB. The impact of message sizes is investigated in Figure 9(c). The root process is randomly chosen from the virtual cluster. One MPI process runs on each instance. In topology mapping, we create the task graph by randomly generating the weight between 5MB to 10MB. Each task is mapped to one instance.

To further evaluate the impact of our network performance aware algorithms, we have implemented two real-world applications namely N-body and conjugate gradient (CG). N-Body is an astronomy model, aiming at simulating the movement, position and other attributes of bodies with gravitational forces exerted on one another. The parameters of N-Body include the number of steps for the simulation (#Step) and the number of bodies. We increase the message sizes to assess the impact of the number of bodies. CG [18] is a commonly used algorithm for the numerical solution of particular systems of linear equations. The conjugate gradient method is an iterative method, with the core operation of sparse matrix vector multiplication (SpMV). CG converges as more iterations are conducted, and we set the convergence condition: $\|r\| \leq 10^{-5} \times g_0$ (r is the residual norm and g_0 is the initial gradient). In both applications, we implement the all-to-all communication with a gather followed by a broadcast, which is also used in MPICH2 [27]. This simple implementation is sufficient for us to investigate the impact of network performance aware optimizations on real-world distributed applications. During the execution period of both applications, we observed little changes in the network performance, and the temporal performance matrix is calibrated once for one execution of each application.

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

- **Baseline.** This simulates the scenario of running directly in the cloud environment, essentially without network performance aware optimizations. In MPI, the binomial tree algorithm is used in MPI_Bcast and MPI_Scatter. We use the implementations from MPICH2. In topology mapping, we use the ring mapping algorithm, which maps each vertex in the task graph to a vertex in the machine graph one by one like a ring.
- **Topology.** In the ns-2 simulation, we use the topology information to optimize applications [39], [19], [21], [38]. This approach is denoted as "Topology-aware". In the experiments on Amazon EC2, we do not include the comparison with this approach, because topology is not available in Amazon EC2.
- **RPCA.** We denote the proposed approach to be "RP-

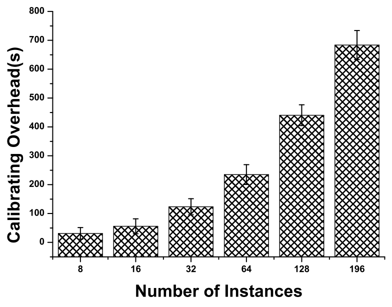


Fig. 4. Overhead of calibrating temporal performance matrix

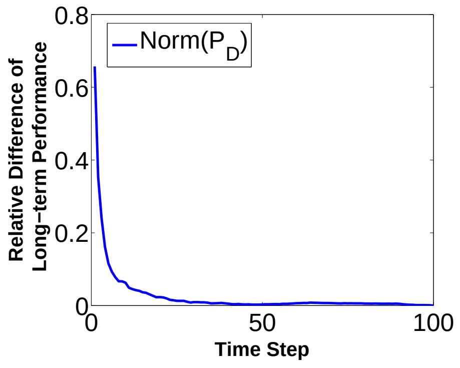


Fig. 5. The relative difference of long-term performance with different time steps

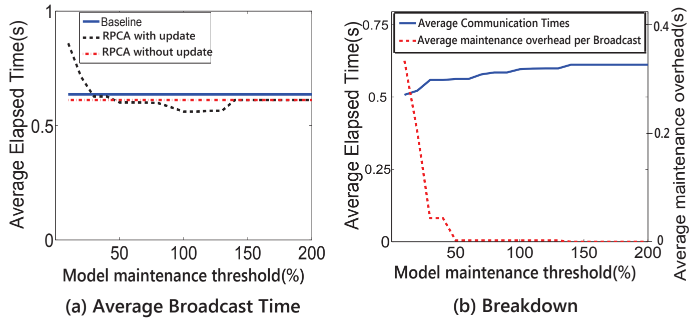


Fig. 6. The performance comparison with different maintenance *threshold*

CA". The network performance aware optimizations are guided by the long-term part captured by our RPCA approach. The traditional network performance optimizations (FNF for MPI and greedy heuristic algorithm for topology mapping) are used. We set time step =10 and *threshold* = 100% for calibration and update maintenance.

- **Heuristics.** We capture the TP-matrix and use the average value of each column to optimize the applications. We denote this approach to be "Heuristics". "Heuristics" represents the direct use of a few measurements of the network performance.

The choice on "Heuristics" is worth further discussions. First, we also use other approaches, for example, minimal value or exponential weighted average. For those approaches, we obtain similar results to the Heuristics approach. Overall, those heuristics based approaches only consider the link separately, whereas RPCA considers the relationship among all the links. Thus, RPCA is able to obtain more information from the same measurement. Second, one may consider performing network performance aware optimizations according to the distribution for each link. However, in order to get the meaningful distribution, excessive measurements are required and the overhead is unacceptably high in practice.

B. Calibrating Overhead

Figure 4 shows the overhead for different numbers of instances for calibrating a single temporal performance matrix when time step is ten. As the number of instances increases, the overhead of calibrating temporal performance matrix is almost linear to the number of instances. When the number is 64, the overhead is less than 4 minutes and when it reaches to 196 instances, the overhead is only about 10 minutes. Given this calibration overhead, our RPCA approach is more useful and acceptable.

Moreover, we calculate the runtime cost of running RPCA analysis. The execution time for running RPCA once is less than 1 minute in the experiments with 196 instances.

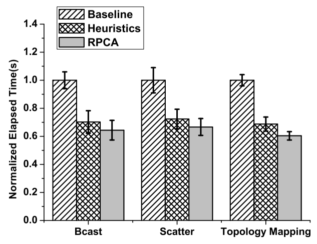
C. Parameter Study

For each experiment, we vary one parameter while keeping other parameters fixed to their default settings (time step =10 and *threshold* = 100% for calibration and update maintenance). We have studied all the applications, and focus on broadcast for a detailed result study.

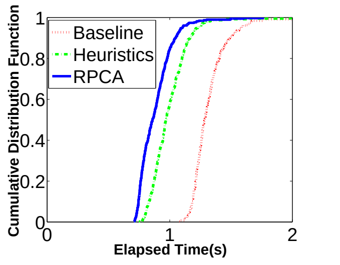
Time Step. We define a function to calculate the accuracy in different time step. With a specified time step, we can calculate the performance matrix P_D as the predicted long-term performance, which essentially is a row of the TC-matrix N_D. On the other hand, we use the whole TP-matrix to obtain the accurate value of the oracle long-term performance P'_D. We define the relative difference of long-term performance to be the accuracy of our prediction, i.e., $Norm(P_D) = \frac{\|P_D - P'_D\|_0}{\|P'_D\|_0}$. When the difference is zero, it means the value is 100% accurate.

Figure 5 shows the relative difference of long-term performance with different time steps in the calibration. As the time step increases, the relative difference becomes smaller. A larger time step means larger overhead. Thus, there is a trade-off between the overhead and accuracy. We select the maximum time step when the relative difference is within 10% among different time steps. In this experiment, we find that the suitable time step is ten, and we use this setting in all the experiments on Amazon EC2.

Update maintenance ***threshold*****.** Figure 6(a) shows the broadcast performance when the *threshold* varies. Figure 6(b) shows the breakdown of the average response time: the average Bcast time for communication time only and the average update maintenance overhead. In this experiment, we find when the percentage is less than 20%, the maintenance is very frequent which leads to large overhead and the average elapsed time will be so huge. In contrast, when the percentage is more than about 150%, there will be no re-measurement of TP-matrix N_A and no change for TC-matrix N_D. When the percentage is about 100%, it almost achieves the best



(a) Overall performance



(b) CDF for the elapsed time of broadcast

Fig. 7. Overall performance comparison for the three applications on Amazon EC2 (all normalized to the average of Baseline)

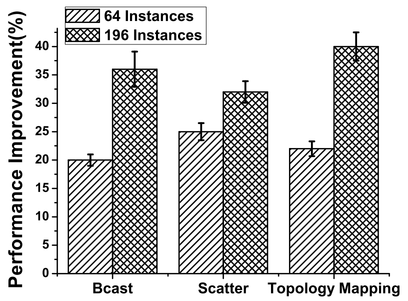


Fig. 8. Overall performance comparison with baseline for the different number of instances on Amazon EC2

performance. The overhead is not so large and the performance improvement is comparable to the best. We have run the experiments for one week on Amazon EC2, and conducted three calibrations in total (day 0, day 2 and day 5). The re-calibration is not so often (less than once for a day) in our experiment. The overhead for one calibration is only 10 minutes for 196 instances, as shown in Figure 4. This overhead is usually ignorable for long-running HPC applications.

D. Results with Amazon EC2

We present the following results related to Amazon EC2. First, we present the overall performance comparison on running real experiments on Amazon EC2. Second, we present some detailed studies with replaying the trace of network performance measurements from Amazon EC2.

1) Results on basic applications: Figure 7(a) shows the average performance comparison of the broadcast, scatter and topology mapping on 196 medium instances of Amazon EC2. The performance is normalized to Baseline. We repeat our experiments for more than 100 times and show the average results. Figure 7(b) shows the CDF for the execution time of broadcast in this experiment.

Overall, RPCA consistently outperforms other comparison approaches for all applications running on Amazon EC2. By carefully selecting the links with the best performance according to the constant component, RPCA improves the effectiveness of network performance aware optimizations.

We have two major observations. First, both Heuristics and RPCA significantly outperform Baseline, with the performance improvement 32–40%. It indicates the importance of the network performance awareness in the cloud environment. With the knowledge of long-term performance captured by RPCA, those optimizations can select the suitable links for a better performance. Second, on Amazon EC2, we analyze the trace and find that the network performance error is relatively small ($N_E = 0.1$). Still, RPCA is 8–10% better than Heuristics. We also observe that as N_E becomes higher, RPCA can outperform Heuristics even more (details are presented in Section V-D3).

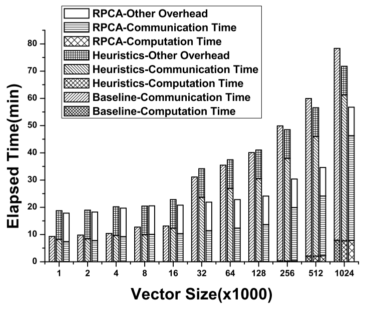
Figure 8 shows the performance improvement of RPCA over Baseline for different numbers of instances in Amazon EC2. The improvement on 196 instances is much higher than that on 64 instances. That is because, when the virtual cluster is large, its virtual machines may be more likely to be located in different racks in Amazon data center. We also observed that the improvement is relatively larger for larger message sizes. This is consistent with our previous observations that the contribution of the update maintenance becomes smaller to the overall execution time. We observe similar results when comparing RPCA with other comparisons.

2) Results on real-world applications: We study the breakdown of the execution time in the real-world applications. In particular, we divide the entire application execution time into two parts: computation and communication. For our proposed algorithms, we also present the initialization cost including calibration and RPCA calculation (denoted as “Other Overheads”). We also note that RPCA calculation contributes to less than 2% of the total overhead.

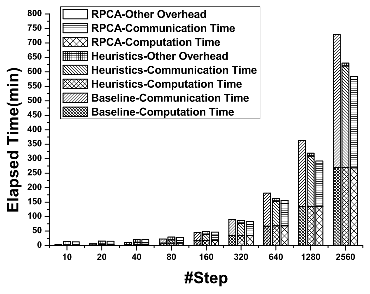
Figure 9(a) shows the comparison studies for CG. In this experiment, we vary the vector size from 1000 to 1024000. We make two observations. First, the CG performance is network-bounded, with communication time contributing over 90% to the total execution time in MPICH2. Second, when the vector size is small, our algorithm is slower than MPICH2-based CG, due to the calibrating and calculating overheads. As the vector size increases, more iterations are required for convergence, and the network performance aware optimization reduces the network communication time. The performance gain compensates the overhead, with 31% and 14% performance improvement over baseline and Heuristics. Figures 9(b) and 9(c) show the performance comparison for N-body. We firstly fix the message size as 1M bytes and vary #Step from 10 to 2560. Then we fix #Step to be 2560 and vary the message size from 1K to 1M bytes. As the message size and #Step increase, the computation and communication play a more important role and the overhead becomes insignificant. Our network performance aware algorithms reduce the network communication time by 36%, and the total execution time by 25% over the baseline approach. The performance improvement over Heuristics approach is around 10%.

3) Detailed Performance Comparison: The network environment of Amazon EC2 is dynamic. For repeatable experiments on studying different settings, we use the method of replaying the trace from the calibrations on network performance of a virtual cluster in Amazon EC2, and estimate the application performance given the pair-wise network performance measurement in the trace.

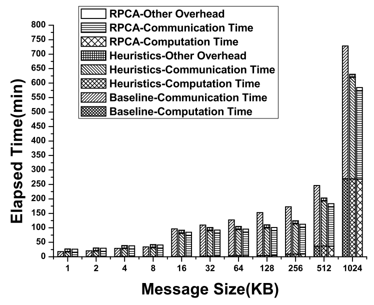
In the following, we first study the accuracy of our trace-replay approach by comparing the performance distributions obtained from our trace-replay approach and from measurements. Next, we study the impact of N_E by introducing noises



(a) CG

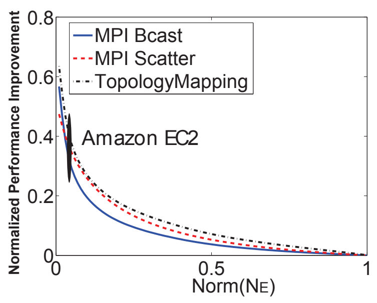


(b) N-body(Message Size=1MB)

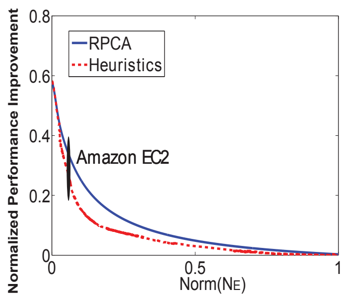


(c) N-body(#Step=2560)

Fig. 9. Performance comparison of real-world applications on 196 medium instances: N-Body and CG.

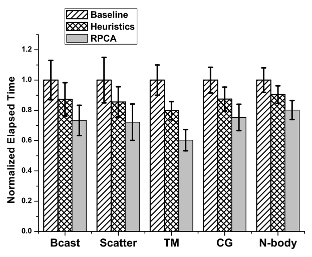


(a) Different applications(RPCA)

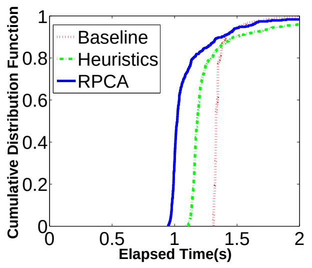


(b) Different optimizations (broadcast)

Fig. 10. The impact of N_E in the virtual cluster



(a) Overall performance



(b) CDF for the elapsed time of broadcast

Fig. 11. Overall performance comparison with $Norm(N_E)$=0.2

to the network performance and have a detailed study on a case when $Norm(N_E)$=0.2.

Accuracy of performance estimations. We compare the estimated performance distribution and the real measurements. With the performance estimation, our performance estimation from the trace-replay approach is close to the real measurements in the cloud. The average difference is only 18% and 9% for baseline and RPCA, respectively. Due to the space limitation, we study the accuracy of our trace-replay approach in Appendix B of our technical report [13].

Impact of N_E. We study the impact of N_E of the virtual cluster. To study more scenarios for N_E, we randomly assign noises to the trace so that N_E is generated. For each time of adding noise, we change the network performance by 1% (increase or decrease, subject to the predefined N_E and current N_E). Then, we run our RPCA-based approach. If the updated N_E reaches the predefined value, we stop. Otherwise, we repeat the process.

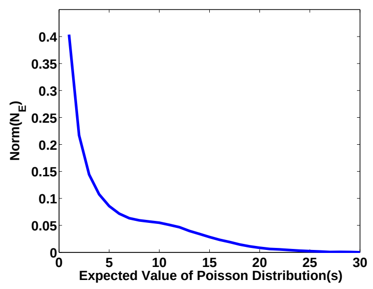
Figure 10(a) shows the *expected* performance improvement of our RPCA approach in broadcast, scatter and topology mapping when $Norm(N_E)$ is varied. The $Norm(N_E)$ significant impacts the effectiveness of the network optimization on the three applications, compared with Baseline. As $Norm(N_E)$ increases, the performance improvement decreases. When it is less than 0.1, the improvement can reach to more than 40%. But when it is more than 0.2, the improvement is less than 20%. On real environment of Amazon EC2, the network is relatively stable ($Norm(N_E)$ is around 0.1), which is highlighted in Figure 10(a).

Figure 10(b) shows the performance improvement of comparing RPCA with Heuristics in broadcast when $Norm(N_E)$ is varied. Overall, both of the approaches can obtain good performance improvement. Our RPCA approach has better efficiency of optimization. When $Norm(N_E)$ is small, which means the network is stable, the network interference plays a less important role. When $Norm(N_E)$ is too large, the network is so dynamic that the network performance aware optimizations have little impact on the performance. When $Norm(N_E)$ is about 0.2, RPCA can obtain about 20% more improvement than Heuristics.

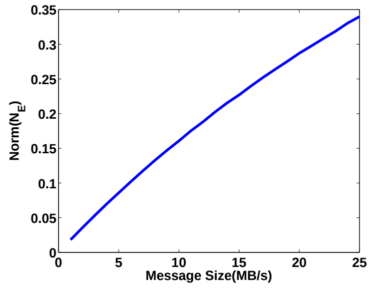
A detailed study. We have a detailed study on a case when $Norm(N_E)$=0.2, which is more dynamic than real Amazon EC2 environments. Figure 11(a) shows the average performance comparison, and Figure 11(b) shows the CDF for the execution time of broadcast in this experiment. The performance improvement of Heuristics and RPCA over Baseline on Amazon EC2 is consistent with our predictions in Figure 10(a) and 10(b). When $Norm(N_E)$ is about 0.2, the improvement decreases for both of the two approaches. However, our RPCA approach can obtain better performance improvement than Heuristics. In this case, RPCA outperforms Baseline by 20–28%, and outperforms Heuristics by 12–20%.

E. Simulations on Large-scale Cluster

We report our simulation results on the simulated cluster of 1024 machines. In Figure 12(a), we fix the message size in the background as 100MB and vary the expected value λ from 1 to 30 seconds. The expected value λ represents the frequency of the network interference. The figure shows that, as λ increases,

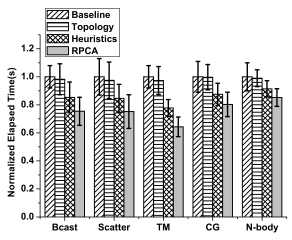


(a) Different expected values(λ) of Poisson Distribution

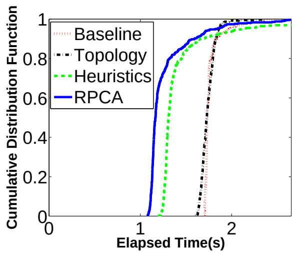


(b) Different message sizes

Fig. 12. The impact of network interference on $Norm(N_E)$



(a) Overall performance comparison



(b) CDF for the elapsed time of broadcast

Fig. 13. Performance Comparison in ns-2-based simulations

$Norm(N_E)$ largely reduces and the network becomes more stable. In Figure 12(b), we fix λ as 5s and vary the message size from 10MB to 500 MB. The figure shows that there is an almost linear relationship between the message size in the background and $Norm(N_E)$. From these two experiments, we observe that $Norm(N_E)$ clearly has positive correlations with the background traffic in the simulated cluster. It can explain why we perform an experiment on Amazon EC2 for many times in order to obtain meaningful results. Also, the performance improvement results on Amazon EC2 are similar to our simulations when they have the same $Norm(N_E)$. It represents the interference of network performance satisfies some stable distribution.

Figure 13(a) shows the overall performance comparison of broadcast, scatter and topology mapping in the simulated cluster. We use the same execution settings for each application as those in Amazon EC2. Machines are randomly selected from the simulated cluster. We simulate the background traffic so that $Norm(N_E) = 0.1$. We obtain similar comparison results against Baseline and Heuristics approaches, and focus on the comparison with the topology-aware algorithm. We find that the topology-aware and baseline performs very similar, which indicates that the topology-aware optimization is not effective in such a dynamic environment. RPCA is 25%-40% better than Baseline/Topology-aware approaches and 10%-15% better than Heuristics. Figure 13(b) shows the CDF of the elapsed time of broadcast. The results are similar to those on Amazon EC2.

We also vary different N_E to study its impact, and observe similar results as Amazon EC2. For example, as $Norm(N_E)$ increases, the performance improvement of RPCA over other approaches increases and the traditional optimizations relying on the topology and direct use of network measurements are no longer effective in such dynamic environments. When $Norm(N_E)$ is too high (e.g., higher than 0.5), the improvement of network performance aware optimizations becomes marginal. Compared with traditional optimizations, our RPCA-based approach accurately predicts this trend with the estimation on $Norm(N_E)$.

VI. Conclusions

This paper revisits network performance aware optimizations of distributed applications running on virtual clusters in IaaS cloud. We find that the important assumptions of existing network performance aware optimizations are no longer valid, because the network topology information is not available, and direct use of a few network performance measurements cannot capture the long-term performance. In this paper, we propose a novel approach based on RPCA (a well-known problem in computer vision) to find the constant component from dynamic network performance while minimizing the difference between the constant component and network performance. Moreover, the RPCA-based approach adapts to significant changes in the network performance. Based on our approach, we are able to determine the effectiveness of network optimizations on the applications running in the virtual cluster. In the experiments, we find that Amazon EC2 has relatively stable network performance ($N_E = 0.1$), and network performance aware optimizations are still important on Amazon EC2. Moreover, our RPCA-based approach is more robust than other approaches for different degree of network dynamics, and always outperforms the other comparison approaches. On Amazon EC2, our RPCA-based approach achieves 20–40% and 8–20% over the baseline approach and the approach based on direct use of the ad-hoc measurements, respectively. In the simulation on ns-2, we compare with the topology-aware algorithm and find that our approach obtains 25–40% performance improvement.

As for future work, we plan to investigate the economic impacts [42] of our approach, and evaluate our approach with more complicated workloads such as scientific workflows [44].

VII. Acknowledgments

The authors would like to thank our reviews and shepherd, Dieter Kranzlmueller, Ewa Deelman and Satoshi Sekiguchi, for their valuable comments. We acknowledge the support from the Singapore National Research Foundation under its Environmental & Water Technologies Strategic Research Programme and administered by the Environment & Water Industry Programme Office (EWI) of the PUB, under project 1002-IRIS-09. Yifan is supported by the scholarship from NEWRI, IGS(Interdisciplinary Graduate School). Bingsheng is in part supported by a MoE AcRF Tier 1 (2014-T1-001-145) of Singapore. Dan's work is partially supported by the National Key Basic Research Program of China (973 program) under Grant 2014CB347800 and the National Natural Science Foundation of China under Grant No.61170291.

References

[1] Amazon Case Studies. *http://aws.amazon.com/hpc-applications/.*

[2] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *SIGCOMM '2011.*

[3] M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *ICPP '98*.

[4] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[5] J.-F. Cai, E. J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM J. on Optimization, 2010*.

[6] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *J. ACM, 2011*.

[7] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: recent developments. *Statistical Science, 2004*.

[8] R. Chen, M. Yang, X. Weng, B. Choi, B. He, and X. Li. Improving large graph processing on partitioned graphs in the cloud. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 3:1–3:13, New York, NY, USA, 2012. ACM.

[9] M. Chowdhury and I. Stoica. Coflow: An application layer abstraction for cluster networking. Technical report, EECS Department, University of California, Berkeley, 2012.

[10] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *SIGCOMM '11*.

[11] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. *SIGCOMM '04*.

[12] D. B. N. Frejnik, S. Goel, O. Kao, and D. Warneke. Evaluation of network topology inference in opaque compute clouds through end-to-end measurements. In *IEEE CLOUD, 2011*.

[13] Y. Gong, B. He, and D. Li. Finding constant from change: Revisiting network performance aware optimizations on iaas clouds. Technical Report 2014-TR-105, Nanyang Technological University, Singapore, http://pdcc.ntu.edu.sg/xtra/tr/2014-TR-105-RPCA.pdf, 2014.

[14] Y. Gong, B. He, and J. Zhong. Network performance aware MPI collective communication operations in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 99:1, 2013.

[15] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference, 2010*.

[16] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: a scalable and fault-tolerant network structure for data centers. *SIGCOMM, 2008*.

[17] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou. Comet: batched stream processing for data intensive distributed computing. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 63–74. ACM, 2010.

[18] M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS, 1952.

[19] T. Hoefler and M. Snir. Generic topology mapping strategies for large-scale parallel architectures. In *ICS*, 2011.

[20] S. Ji and J. Ye. An accelerated gradient method for trace norm minimization. In *ICML '09*.

[21] K. C. Kandalla, H. Subramoni, A. Vishnu, and D. K. Panda. Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with scatter and gather. In *10th Workshop on Communication Architecture for Clusters, 2010*.

[22] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 202–208. ACM, 2009.

[23] N. T. Karonis, B. Toonen, and I. Foster. Mpich-g2: a grid-enabled implementation of the message passing interface. *J. Parallel Distrib. Comput, 2003*.

[24] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. Magpie: Mpi's collective communication operations for clustered wide area systems. In *PPoPP*, 1999.

[25] H. Liu and B. He. Reciprocal resource fairness: Towards cooperative multiple resource fair sharing in iaas clouds. In *SC*, 2014.

[26] W. Lu, J. Jackson, and R. Barga. Azureblast: a case study of developing science applications on the cloud. In *1st Workshop on Scientific Cloud Computing*, 2010.

[27] MPICH2. *http://www.mcs.anl.gov/research/projects/mpich2/*.

[28] J. C. Mudge, P. Chandrasekhar, G. S. Heinson, and S. Thiel. Evolving inversion methods in geophysics with cloud computing - a case study of an escience collaboration. In *ESCIENCE*, 2011.

[29] A. Nagavaram, G. Agrawal, M. A. Freitas, K. H. Telu, G. Mehta, R. G. Mayani, and E. Deelman. A cloud-based dynamic workflow for mass spectrometry data analysis. In *ESCIENCE*, 2011.

[30] T. S. E. Ng and H. Zhang. Towards global network positioning. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, pages 25–29, New York, NY, USA, 2001. ACM.

[31] S. Nunez, B. Bethwaite, J. Brenes, G. Barrantes, J. Castro, E. Malavassi, and D. Abramson. Ng-tephra: A massively parallel, nimrod/g-enabled volcanic simulation in the grid and the cloud. In *ESCIENCE*, 2010.

[32] K. A. Ocaña, D. de Oliveira, J. Dias, E. Ogasawara, and M. Mattoso. Optimizing phylogenetic analysis using scihmm cloud-based scientific workflow. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 62–69. IEEE, 2011.

[33] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos. Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 351–362. ACM, 2013.

[34] R. Reussner, P. S, L. Prechelt, and M. Muller. Skampi: A detailed, accurate mpi benchmark. In *In Vassuk Alexandrov and Jack Dongarra, editors, Recent advances in Parallel Virtual Machine and Message Passing Interface*, 1998.

[35] RPCA Accelerated Proximal Gradient(APG) Method . *http://perception.csl.illinois.edu/matrix-rank/sample_code.html*.

[36] M.-F. Shih and A. Hero. Hierarchical inference of unicast network topologies based on end-to-end measurements. *Trans. Sig. Proc, 2007*.

[37] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *HotPower*, 2008.

[38] H. Subramoni, K. Kandalla, J. Vienne, S. Sur, B. Barth, K. Tomko, R. McLay, K. Schulz, and D. K. Panda. Design and evaluation of network topology-/speed-aware broadcast algorithms for infiniband clusters. In *IEEE Cluster*, 2011.

[39] R. Thakur and R. Rabenseifner. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 2005.

[40] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman. Experiences using cloud computing for a scientific workflow application. In *ScienceCloud*, 2011.

[41] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM*, 2010.

[42] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou. Distributed systems meet economics: pricing in the cloud. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 6–6. USENIX Association, 2010.

[43] D. Xie, N. Ding, Y. C. Hu, and R. Kompella. The only constant is change: incorporating time-varying network reservations in data centers. *SIGCOMM, 2012*.

[44] A. Zhou and B. He. Transformation-based monetary cost optimizations for workflows in the cloud. *Cloud Computing, IEEE Transactions on*, 2(1):85–98, Jan 2014.