Network Performance Aware Optimizations on IaaS Clouds

Yifan Gong, Bingsheng He, Dan Li

Abstract—Network performance aware optimizations have long been a hot research topic to optimize distributed applications on traditional network environments. However, those optimization techniques rely on a few measurements on pair-wise network performance, and such direct use of network measurements is no longer valid in Infrastructure-as-a-service (IaaS) clouds. First, the direct calibration is ineffective. Network performance measurements may not represent the long-term performance (informally the stable component inside network performance) because of virtualization and network performance interference in the cloud. Second, the direct calibration is inefficient because the measurement overhead of all pair-wise link performance in a cluster becomes prohibitively high as the number of instances increases. To effectively and efficiently utilize existing network performance aware optimizations on IaaS clouds, we propose to reduce the measurement overhead and decouple the constant component from the dynamic network performance while minimizing the difference between the network performance and the constant component. For effectiveness, we use the constant component to guide the network performance aware optimizations. For efficiency, we exploit a non-negative matrix factorization (NMF) method to reduce the calibration overhead. Furthermore, we observe a tradeoff between effectiveness and efficiency, and develop an adaptive approach to capture this tradeoff. We demonstrate effectiveness and efficiency of our approach by adopting network performance aware optimizations on two kinds of basic applications, collective communications of MPI and generic topology mapping, and two real-world applications, namely N-body and conjugate gradient (CG). Our experiments on Amazon EC2 and simulations demonstrate significant calibration overhead reduction and performance improvement on guiding network performance aware optimizations, when comparing our approach to other state-of-the-art approaches.

Index Terms—HPC, Cloud Computing, Network Performance-Aware Optimization, RPCA, Non-negative Matrix Factorization

1 INTRODUCTION

Network performance aware optimizations have long been a hot research topic to optimize distributed applications on traditional network environments (e.g., local clusters and grids [3], [23], [26], [38], [39]). The essential idea of those optimizations is to obtain the network performance metrics (mainly latency and bandwidth of all communication links in the cluster), and then to optimize overall performance by carefully selecting the links for communications. Those optimizations are based on 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 [3], [20]. 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]. Although there have been many research studies on designing novel network bandwidth allocations (e.g., [2], [34], [42]) or data center networks [18] for IaaS clouds, little attention has been paid to how applications can adapt their optimizations to IaaS clouds. Therefore, this paper revisits network performance aware optimizations on IaaS clouds.

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 [40] and network interference [4] in IaaS clouds. Machine pairs can have very different network performances as shown in the previous studies [2], [17]. That means link selection continues to be important on IaaS clouds and network performance aware optimizations are still important to improve the application performance, especially for communication-intensive applications. Natural questions are whether and how we can apply existing network performance aware optimizations on IaaS clouds.

In the study of network performance aware optimizations, we consider both effectiveness and efficiency. Effectiveness refers to the performance improvement for the network performance aware optimization and efficiency refers to the overhead of enabling the network performance aware optimization. Unfortunately, we find that adapting existing network performance aware optimizations is neither effective nor efficient on IaaS clouds. First, the topology information is unavailable or inaccurate on IaaS clouds. Virtualization hides the network hardware, topology and actual configurations of the underlying hardware from users. Second, due to the cloud system dynamics such as virtual machine consolidation [37] and dynamic network flow scheduling [4], the static topology information is no longer sufficient to represent 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. The effectiveness of such approaches on adapting existing network performance aware optimizations is questionable. Moreover, the all-link performance measurement leads to huge calibration time as the number of instances increases. It leads to low efficiency for utilizing network performance...
In this paper, we develop an effective and efficient approach for enabling network performance aware optimizations on IaaS clouds. We cast the effectiveness problem 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 the condition 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 while minimizing the difference between the network performance and the constant component. 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. We model each row of the data matrix to be one snapshot 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 on IaaS clouds. Based on the constant component, conventional network performance optimizations become valid, i.e., we can select the best performing links with minimized errors. On the other hand, with the error component, we are able to determine the effectiveness of network performance aware optimizations, e.g., if the error is too large, the network of the IaaS cloud is too dynamic and optimizations are useless.

For further improve the efficiency of the network performance aware optimizations, we formulate the efficiency problem as an NCS (Network Coordinates Systems) problem and exploit non-negative matrix factorization (denoted as NMF [29]) method to reduce the measurement overhead. NCS has been proposed to model the network as a geometric space. The geometric distance between two nodes can predict the network performance, e.g., for the sum of the two components are equal to the data matrix. We can use the approximated RPCA to obtain the constant component and error as the constant component with minimized errors.

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 the condition 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 while minimizing the difference between the network performance and the constant component. 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. We model each row of the data matrix to be one snapshot 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.

For further improve the efficiency of the network performance aware optimizations, we formulate the efficiency problem as an NCS (Network Coordinates Systems) problem and exploit non-negative matrix factorization (denoted as NMF [29]) method to reduce the measurement overhead. NCS has been proposed to model the network as a geometric space. The geometric distance between two nodes can predict the network performance (latency or bandwidth) and explicit measurements are not required anymore. As an efficient solution to NCS, the basic idea of NMF is to approximate a large matrix, whose elements represent pairwise performance, by the product of two smaller matrices.

However, there is a trade-off between effectiveness and efficiency. The problem is that when we utilize NMF to reduce the calibration overhead and improve the efficiency, we only measure parts of the network performances, which result in inaccuracy of the performance matrix. When we use the approximated matrix to guide the performance optimization, the performance improvement is reduced and the effectiveness can be affected. We further propose a model to balance the effectiveness and efficiency and minimize the total elapsed time.

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 [20] as well as two real-world applications, N-body and conjugate gradient (CG).

Our experiments show that our proposed approach can determine the degree of network dynamics 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 proposed RPCA approach can efficiently reduce the measurement overhead. On Amazon EC2 and simulations, the proposed approach significantly reduces the measurement overhead by 84%. Our RPCA approach effectively guides the network performance aware optimizations and 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 to 25% and 31% over the baseline, respectively. In the simulation with up to 16384 servers, the proposed approach can obtain the performance improvement of the three applications by 10–15%, 15%–20% over baseline and 8%–12%, 10%–15% over state-of-the-art approaches. Moreover, our model can balance the efficiency and effectiveness for the performance aware optimizations.

In summary, we make the following contributions in this paper.

- We propose to consider both effectiveness and efficiency for network performance aware optimizations on IaaS clouds.
- We propose a novel RPCA-based approach and an NMF-based method to improve the effectiveness of network performance aware optimizations, and to reduce the calibrating overhead, respectively. Furthermore, we develop an adaptive approach to capture the tradeoff and obtain the optimal solution.
- We study the efficiency of our approach in different scale of virtual cloud clusters on Amazon EC2 and situations.

The rest of the paper is organized as follows. We introduce the background on cloud networks, NCS and NMF, RPCA and two examples of network performance optimizations in Section 2. We present the problem definition in Section 3, and our RPCA approach in Section 4. In Section 5, we show our experimental results. Finally, we conclude this paper in Section 6.

2 Preliminary and Related Work

2.1 Cloud Network

Due to the significant scale, the network environment in the cloud differs from traditional clusters. The current cloud practice is to use the commodity switch-based tree structure to interconnect the servers [18], as illustrated on Figure 1.

Machines are first grouped into rack, and then racks are connected with higher-level switches. Many data centers adopt the tree-structured network design [4], [24]. The key problem of the tree topology is the network bandwidth of any machine pairs is not uniform, depending on how the switch connecting the two machines. For example, the intra-rack bandwidth is much higher than the cross-rack bandwidth.

Previous studies (e.g., [2], [8], [40]) have shown significant variability of network performance between different machines in data centers. The network performance variability negatively impacts application performance, and also makes traditional network
performance aware optimizations (e.g., [3], [20]) infeasible. To avoid re-inventing all those network performance aware optimizations, this paper develops a new approach to capture the long-term network performance, which represents the stable component of network performance in cloud, and makes existing/new optimizations applicable to cloud.

Network topology inference techniques have been investigated in the traditional environments [25], [36] and cloud environments [15]. We refer readers to a survey [7] for more details on classic techniques for network topology discovery and inferences. The information given by basic diagnostic tools like traceroute is incomplete in the virtualized cloud.

2.2 Network Coordinates Systems and Non-negative Matrix Fraction

Network Coordinates Systems [12] have been proposed to allow hosts to estimate performance without direct calibration and thus, reduce the consumption of network resources. The basic idea is to model the network as a geometric space. The position of any node is characterized by a coordinate in this space. The geometric distance between two nodes can predict the network performance (latency or bandwidth) and explicit measurements are no longer required.

Many proposed approaches [11], [32], [33] are based on the network coordinates estimated by Euclidean distances. Such approaches require network performance to satisfy symmetric and triangle inequality. However, triangle inequality is not satisfied in the cloud environment [27], [44]. In order to get out of the limitation, Non-negative Matrix Fraction (denoted as NMF [29]) method has been proposed. Suppose \( \hat{M} \) is a sparse matrix from direct measurement, \( X \) and \( Y \) are low-rank matrices. NMF is to solve the following optimization problem, where \( \| \|_0 \) is the zero norm.

\[
\begin{align*}
\text{minimize} & \quad \|M - \hat{M}\|_0 \\
\text{subject to} & \quad M = XY^T
\end{align*}
\]

NMF is based on the assumption that two nearby hosts have similar distances to all the other hosts in the network. In this case, their corresponding rows in the distance matrix will be nearly identical. More generally, there may be many rows in the distance matrix that are equal or nearly equal to linear combinations of other rows. Applying this approach in network performance measurement is straightforward.

2.3 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.

\[
\begin{align*}
\text{minimize} & \quad \text{rank}(D) + \lambda \|E\|_0 \\
\text{subject to} & \quad A = D + E
\end{align*}
\]

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 is analogous to RPCA in computer vision.

There are many approaches that have been proposed to solve this optimization problem (e.g., [5], [22]). We choose the approach by Ji et al. [22] (their implementation [35]), which is a polynomial-time algorithm with strong performance guarantees on the error.

2.4 Network Performance Aware Optimizations

Network performance aware optimizations have been a hot research topic in the cluster/grid environments (e.g., [3], [20]) and cloud environments (e.g., [9], [10]). Network performance aware optimizations rely on the measured network performance metrics (e.g., latency and bandwidth) and utilize the information to optimize the execution time or network throughput of the applications. Many studies 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. However, without understanding and capturing the long-term network performance, 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 [23], [26], [38] and topology mapping [3] as examples to demonstrate network performance aware optimizations with the knowledge of pair-wise network performance. We select the two applications for their sensitive to the communication link selection.

**MPI collective operations.** MPI is a de facto standard for distributed and parallel programs running on computer clusters or supercomputers. It supports both point-to-point and collective communication. Many large-scale scientific applications are MPI-based applications, and recently have been migrated to heterogeneous clusters [16] and public clouds [14], [30], [43].

Network performance optimization is very important for the overall performance of collective communications of MPI [23], [26], [38]. This study focuses on the four basic collective operations: broadcast, reduce, gather and scatter.

We introduce the details of broadcast as an example. Broadcast is one of the standard collective communication operations. During a broadcast, one process sends the same data to all the other
The algorithm works in multiple iterations. In each iteration, the first four processes send the data to the rest four processes. The communication pattern of broadcast is shown in Figure 2(a). In this example, process 0 is the root process, and it has the initial copy of data. All of the other processes receive the copy of data. In MPI Broadcast, binomial tree, which is shown in Figure 2(b) is one of the basic communication tree structures. In this example, process 0 (root) sends the copy of data to process 1 in the first step. After this step, both process 0 and process 1 have the copy of data. In the second step, process 0 sends the data to process 2 while process 1 sends the data to process 3. In the final step, the first four processes send the data to the rest four processes and the broadcast has been completed.

However, due to the unevenness of pair-wise network bandwidth among the virtual machines, we need to carefully choose the links to construct 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 effective 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, Bankazemi 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 of 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 \( U \) which has the best network performance with \( s \), 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 running example is given in Figure 3(a). On the left side, 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. The right side of Figure 3(a) shows the resultant tree structure by the FNF algorithm. The total weight of the longest path is five.

We also consider a generalization for other communication patterns besides the four basis collective operators. In our experiments, we implement two real-world applications, N-body and conjugate gradient (CG). They have more complex communication patterns. For example, there is an all-to-all communication pattern in N-body. More details will be presented in Section 5.1.

**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 [20]. In this paper, we use the Greedy Heuristic Algorithm approach [20]. 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. There are some other topology mapping techniques [21], [28], which aim at optimizing for latency communication time. However, we only utilize the most commonly used approach for bandwidth sensitive operations as an example. Our approach can be easily extended to utilize on such kinds of techniques. In fact, our approach is proposed to efficiently and effectively utilize performance aware optimizations, which is independent to either bandwidth-optimized or latency-optimized algorithms.

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 \( G \).

**2.5 Terminology**

In our work, 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 network performance aware optimization.

**Virtual cluster.** We can measure the performance of each link, and the all-link network performance for a set of virtual machines...
Network performance. We adopt the α-β model [39] to model the network performance of each link. In the α-β model, each link is represented by 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 messages of arbitrary sizes.

Performance matrix. We define the pair-wise network performance with a matrix (namely performance matrix). In particular, given two virtual machines $i$ and $j$, the network performance parameters of the link from $i$ to $j$ is denoted as $\alpha_{ij}$ and $\beta_{ij}$. We represent the network performance of the virtual cluster with two $N \times N$ performance matrices: $L = (\alpha_{ij})$ and $B = (\beta_{ij})$ ($1 \leq i, j \leq N$). 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 to develop the performance matrices. Firstly, the performance matrices offer a general performance model which can be used for network performance aware optimizations in different applications. Secondly, individual link performance is crucial to the network performance aware optimizations, which rely on the comparison of the long-term network performance of all links. For example, in Figure 3(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 3(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.

3 Problem Formulation

Since the network performance varies, we extend the performance matrix (defined in Section 2.5) with a time dimension. The network performance parameters of the link from virtual machine $i$ to $j$ at time $t_0$ is denoted as $\alpha_{ij}(t_0)$ and $\beta_{ij}(t_0)$. And the network performance at time $t_0$ of the virtual cluster can be represented as: $L(t_0) = (\alpha_{ij}(t_0))$ and $B(t_0) = (\beta_{ij}(t_0))$ ($1 \leq i, j \leq N$). Based on these definitions, we can extend the concept of performance matrix.

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 the supplementary file.

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 an $n \times N^2$ matrix.

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

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 should find a constant component from the time-varying network performance, which represents the long-term network performance. Ideally, the difference between the network performance and the 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 (i.e., 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. We define TC-matrix to represent the constant component of network. As each row is the constant part of a snapshot of network performance, any pair of rows should be linearly dependent and therefore the rank of the matrix is one. An example of TC-matrix $N_D$ is shown below, where all $P_{D_{t_i}}$ ($0 \leq i < n$) are linearly dependent.

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

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

$$N_E[T_0, T_1] = \begin{bmatrix} P_{E_{t_0}} \\ P_{E_{t_1}} \\ \vdots \\ P_{E_{t_{n-1}}} \end{bmatrix}$$
Calibrating TP-matrix $N_A$. Our first problem is how to efficiently and accurately obtain the TP-matrix $N_A$ on $[T_0, T_1]$. Considering the performance matrix $P_{A_i}$ at time $t_i$, where $T_0 \leq t_i \leq T_1$, $t_i \leq t_{i+1}$. Suppose $\bar{P}_{A_i}$ is a sparse matrix from direct measurement at time $t_i$ and the $N$ instances belong to at most $d$ racks and the instances in the same rack have similar network performance. Then if we select one instance in each rack as the basis coordinates, we can prove that the performance matrix $P_{A_i}$, whose scale is $N \times N$ is at most $d$ ranks [33]. It can be represented by the multiplication of two $N \times d$ matrices. Then our problem can be formulated as below.

$$\begin{align*}
\text{minimize} & \quad \|P_{A_i} - \bar{P}_{A_i}\|_0 \\
\text{subject to} & \quad P_{A_i} = XY^T
\end{align*}$$

where $X$ and $Y$ are two $N \times d$ matrices. By solving this problem, we can measure a part of the performance matrix $P_{A_i}$ at $t_i$ and recover the real performance matrix $P_{A_i}$ with at most $d$ rank. Then we can get the performance matrix $P_{A_i}$ at each time $t_i$, and rebuild the TP-matrix $N_A$. Assume that we select $\xi$ percentage ($0 \leq \xi \leq 1$) pairs of links from the whole pair to calibrate the performance matrix at each time $t_i$ and the total calibrating overhead for the whole TP-matrix is $T_o$, we can reduce the calibrating time to $\xi \times T_o$.

Applying RPCA to network performance. After obtaining the TP-matrix $N_A$, we need to find the constant component to guide the design of performance aware optimizations. Given the TP-matrix $N_A$ on $[T_0, T_1]$, we define TC-matrix $N_D$ and TE-matrix $N_E$ accordingly. This problem can be formulated as below.

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

By solving this problem, we can decouple the constant component from the dynamic network performance and then utilize the constant component to optimize network performance. We treat that constant component as lasting for a long period until we observe some significant changes in the network performance. By then, we need to perform re-calibrations and re-run the approach. Assume that the execution time of the application is $T_e$ and the network performance aware optimization can reduce the execution time to $\psi$ percentage ($0 \leq \psi \leq 1$), the execution time can be reduced to $\psi \times T_e$.

Problem Formulation. Based on these definitions, we can calculate the total elapsed time. We assume to run the application for $K$ times and the total elapsed time is equal to the sum of the calibrating overhead and the execution time of the application for $K$ times. Our goal is to minimize the total elapsed time and the problem can be formulated as:

$$\begin{align*}
\text{minimize} & \quad T(\xi, \psi) \\
\text{where} & \quad T(\xi, \psi) = \xi \times T_o + K \times \psi \times T_e
\end{align*}$$

4 Optimization Approach

4.1 Problem Analysis

There is a trade-off between calibration overhead reduction $\xi$ and performance improvement $\psi$. We first define the relationship

$$f(\xi) = \frac{T_0}{K \times T_e}$$

where $T_0 = T_o + K \times f(\xi) \times T_e$ (2)

We sample some points of the relationship function and utilize a regression method to obtain $f(\xi)$ [13]. Figure 4(a) shows the relationship function $\psi = f(\xi)$ on Amazon EC2. We find that $f(\xi)$ is convex ($f''(\xi) \geq 0$) so that the optimization problem is convex, because $T' \left(\xi\right) = K \times f'(\xi) \times T_e \geq 0$. And the optimal solution can be obtained when $T'(\xi) = 0$, which can be calculated as

$$f' \left(\xi\right) = -\frac{T_o}{K \times T_e}$$

The derived function of $f(a)$ is shown in Figure 4(b). We have three major findings.

- When $T_o \gg T_e$, which means the calibration overhead is much larger than the execution time of the application, $f' \left(\xi\right) \to -\infty$ and $\xi \to 0$. It means we should not use performance aware optimizations because of the large calibration time.
- When $T_o \ll T_e$, which means the calibration overhead is much smaller than the execution time of the application, $f' \left(\xi\right) \to 0$ and $\xi \to 1$. Thus, we can directly use the performance matrix since the calibration overhead is ignorable.
- In other cases, there is a trade-off between the calibration overhead reduction and performance improvement. And we can obtain the optimal solution of our problem by setting the variable $\xi$ as $\xi'$ where $f' \left(\xi'\right) = -\frac{T_o}{K \times T_e}$.

4.2 Approach Design

We develop an adaptive approach to incorporate NMF and 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 pairwise 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 proposed RPCA approach, we obtain 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.

In the following section, we present our approach in addressing the efficiency and effectiveness. Particularly, we introduce NMF to improve the efficiency and reduce the overhead of calibrating TP-matrix. For effectiveness, we utilize an RPCA approach proposed by Ji et al. [22] to guide performance optimization and determine the effectiveness of optimization. Furthermore, we use the real performance as feedbacks for update maintenance.

4.3 Algorithm Details

Algorithm 1 shows the procedure of our optimization approach on a virtual cluster $C$. On the cloud, we first decide the variable $\xi$ and utilize NMF [29] to efficiently 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. [22], and get $N_D$ and $N_E$. We choose Ji et al’s approach for its low overhead and high performance over other approaches [22]. 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.

Algorithm 1 Overview of our RPCA approach

1: Given a virtual cluster $C$, decide the optimal value $\xi$ and utilize NMF [29] to efficiently calibrate the TP-matrix, and let the matrix be $N_A$;
2: Run the RPCA approach by Ji et al. [22], and get $N_D$ and $N_E$;
3: Given $N_D$, we apply some network performance aware optimization algorithms to the network communication operation $A$ running on a virtual cluster $C'$, where $C' \subseteq C$;
4: Measure the network performance of $A$ and let it be $t$;
5: Let the expected performance of $A$ be $t'$;
6: if $\frac{t}{t'} \geq \text{threshold}$ then
7: Go to Line 1; /* update maintenance*/
8: else
9: Go to Line 3; /* use the same $N_D$ for later optimizations*/

In the following, we describe the details of the algorithm.

4.3.1 NMF Approach and Model Calibration

In order to utilize performance aware optimization techniques, we first have to calibrate each cell of the performance matrix. But the large overhead of calibrating the performance one by one pair leads to low efficiency. The total calibration time is $O(N^2)$. To reduce the overhead, at each step we choose $N/2$ instances to send messages and the other $N/2$ to receive. In this way, we could obtain $N/2$ pairs at a time, and the time consumption is $2 \times N$.

However, the simple optimization is not enough for large scale environment. When the number of processes $N$ is large, the time consumption $(2 \times N)$ is still unacceptable. Fortunately, we find that in the cloud environment, the nearby instances have similar network performance to all the other instances. In this case, these rows (the nearby instances) in the performance matrix will be nearly identical and there can be many rows in the performance matrix that are equal to linear combinations of other rows. The observations show that we do not need to calibrate all-link network performance. Instead, we can measure some links from network and calculate the network performance for the rest part. We exploit a method called NMF [29] to efficiently reduce the measurement time while keeping our network performance aware optimizations effectively. Based on this method, we can make a trade-off between calibration time and performance improvement.

Concretely, in calibrating each performance matrix, we randomly select $p$ instances from $N$ instances. For each selected instance, we randomly select $q$ instances to exchange message. From the calibration, we can obtain an $N \times N$ sparse matrix with $p \times q$ elements. We use this matrix as the input of NMF algorithm and recover the whole $N \times N$ performance matrix. Based on this method, we can reduce the calibration time to $O(q)$ and at each step, there are only $p$ pairs in communication. Based on the previous definitions in Section 3, the variable $\xi$ can be calculated as $\xi = \frac{E_{\text{row}}^{\text{max}}}{N^2}$ $(0 \leq \xi \leq 1)$ to measure how large to reduce the calibration time. For example, $\xi = 10\%$ represents that we select $10\%$ pairs of links from the whole pairs to calibrate the network performance. The optimal value of $\xi$ can be obtained by Formula 3.

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 5.

4.3.2 RPCA Approach

We efficiently obtain the TP-matrix $N_A$ from the model calibration and effectively utilize $N_A$ as the input data matrix $A$ for RPCA. After running the RPCA approach by Ji et al. [22], we get $N_D$ and $N_E$. We exploit $N_D$ to guide performance optimization and $N_E$ to determine the effectiveness of optimizations.

Guiding performance optimization with $N_D$. With $N_D$, we can apply traditional network performance aware optimization to the network communication operations, for example, applying the FNF algorithm [3] to construct the binomial tree. Note, the network communication operations 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 $P_{Di}$ 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 5 illustrates an example of calculating $N_A = N_D + N_E$ with RPCA. Figure 5(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 5(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 5(c)). From \( N_D \), we can obtain a performance matrix in Figure 5(d), which can be used for 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 optimal optimization solution from offline, the performance error is the difference between the optimal 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 \( N_E \), \( \text{Norm}(N_E) = \frac{\| N_E - N_D \|_F}{\text{Norm}(N_D)} (0 \leq \text{Norm}(N_E) \leq 1) \) to measure the effectiveness of network performance aware optimizations in the cloud.

**Utilizing Performance Aware Optimization.** We use the same example in Figure 3(a) to describe how to use performance aware optimizations to our applications. First, we obtain the TC-matrix (the weight matrix in Figure 3(a)) from RPCA. Second, we utilize FNF tree to build a binomial tree as shown in Figure 3(a). In the final step, we build a new Communicator. Communicator is an MPI object which connects a group of processes in an MPI session. The size of the communicator is the number of processes in it, and each process is given an identifier (rank) 0, 1, ..., N-1, where N is the size of the communicator. The rank id in the new Communicator is reordered by the FNF tree algorithm. and the MPI collective operators are executed in the new Communicator.

**4.3.3 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 \( \alpha-\beta \) 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 5.

The information of VM placement is hidden from the IaaS cloud and is unavailable to cloud users. Moreover, VM placement-aware optimization is transparent and complementary to our approach. Nevertheless, we have considered the VM migration problem in our approach design. If many VM migrations occur, network condition will change significantly. For this reason, we design to use the real performance as feedbacks and then utilize update maintenance techniques to adapt to such network changes.

## 5 Evaluations

### 5.1 Experimental Setup

We use two complementary evaluation approaches including simulations and real experiments. The real experiments were performed on Amazon EC2 from 2014 to 2016, 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 large scale virtual cluster environment and study the relationship between calibration overhead and performance. Finally, we simulate different network topologies, and compare the topology aware optimizations with our approach.

**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. We automatically determine the optimal value of \( \zeta \). 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 \( \alpha-\beta \) 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 [4], [24]. As a start, we use the tree-structured topology to interconnect servers, as illustrated on Figure 1. Machines are first grouped into racks, and then racks are connected with higher-level switches.

First, we simulate a small cluster of 1024 machines, which we denote as SIM-S. 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 links with the 32 switches in the first level. The bandwidth within 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 and call it background traffic. We first choose the links and then vary two parameters to control for 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 \( \lambda \). We vary these two parameters to study the impact of errors in our RPCA 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.

Second, we extend the scale of simulated virtual cluster to 4096 machines (denoted as SIM-M). There are 64 racks and each rack contains 64 servers. Other settings are the same with SIM-S. We control \( N_E \) to 0.1 and utilize SIM-M to evaluate the effectiveness of our proposed approach.

\(^1\) http://www.isi.edu/nsnam/ns/
Third, we simulate a large scale virtual cluster with 16384 machines (denoted as SIM-L). In SIM-L, the number of racks is 128 and there are 128 servers in each rack. Other settings are the same with SIM-S and SIM-M. We control $N_E$ to 0.1 and study the performance improvement of our proposed approach from SIM-L.

Besides tree-structured network, we further simulate a three-dimensional torus interconnect network, which is widely used for many large clusters [1], [16], with 1024 machines (denoted as SIM-torus). Other settings are the same with SIM-S, SIM-M and SIM-L. We control $N_E$ to 0.1 and study the performance improvement of our proposed approach from SIM-torus.

**Comparisons.** We assess the impact of the network performance optimizations by comparing the following five 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 [20], [23], [38], [39]. 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.

- **Heuristics.** We capture the TP-matrix with NMF 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.

- **RPCA.** We denote our proposed approach to be “RPCA”. We utilize NMF to calibrate the TP-matrix. 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.

- **RPCA without NMF.** We capture the whole TP-matrix without utilizing NMF. We still use the long-term part captured by RPCA to guide the network performance aware optimizations. Ideally, this approach is the most effective for network performance aware optimizations without considering the overhead of calibrating TP-matrix.

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.

**Applications.** We apply the proposed RPCA approach to two kinds of basic applications – MPI collective operations and topology mapping. As we introduce in Section 2, both applications have network performance aware optimizations according to pairwise 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. 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 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. We repeat each of the applications for 1000 times and calculate the total execution time.

To further evaluate the impact of our network performance aware algorithms, we implement two real-world applications namely N-body and conjugate gradient (CG). N-Body is an astronomy model, aimed at simulating the movement, position and other attributes of bodies with gravitational forces exerted on one another. We utilize Barnes-Hut tree algorithm [41] for N-body. The communication pattern is all-to-all in each step. Our approach is applicable to some other particle mesh algorithms, because these algorithms are proposed to optimize the computation and the communication patterns are similar. The parameters of N-Body include the number of steps for the simulation (#Step) and the number of bodies. CG [19] is a commonly used algorithm for the numerical solutions 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 [31]. 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 change in the network performance, and the temporal performance matrix is calibrated once for one execution of each application. For each application, we first run Baseline, immediately followed by Heuristics, RPCA without NMF and our RPCA approach.

5.2 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). Due to the space limitation, we summarize our findings here, and present the details on parameter study in Appendix C of the supplementary file.

**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 $P_D^\prime$. We define the relative difference of long-term performance to be the accuracy of our prediction i.e., $\text{Norm}(P_D) = \frac{||P_D - P_D^\prime||_F}{||P_D^\prime||_F}$. When the difference is zero, it means the value is 100% accurate.

In our experiment, we find that 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 on broadcast, the suitable time step is ten. We obtain the similar results for other applications.
The selection of optimal value $T$ improves in different scale of virtual clusters. We set the optimal value of $T$ to 64, 256, 1024, 4096, and 16384. The evaluation results show that the overhead of calibration is up to 4 and 16 hours. However, when the number of instances is larger than 1024, the optimal value of $T$ is less than 0.1 and the calibration overhead can be largely reduced.

Figure 7(b) shows the normalized elapsed time for Baseline, RPCA without NMF and RPCA. When the number of instances is less than 256, RPCA without NMF and RPCA have similar performance. But when the number of instances is larger than 512, RPCA can outperform RPCA without NMF, because of the huge calibration overhead. When the number of instances is larger than 2048, RPCA without NMF is even slower than Baseline.

### 5.5 Effectiveness Results with Amazon EC2

#### 5.5.1 Results on basic applications

Figure 8(a) shows the average performance comparison of the broadcast, scatter and topology mapping on 196 medium instances of Amazon EC2. The execution time $T_e$ of each application is around one to two hour. The optimal value of $\xi$ is about 0.25. The performance is normalized to Baseline. We repeat our experiments for more than 100 times and show the average results. Figure 8(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 three major observations. First, Heuristics, RPCA without NMF and RPCA significantly outperform Baseline, with performance improvement of 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, compared with RPCA without NMF, RPCA obtains 6% performance improvement. It indicates that the automatic selection of variable $\xi$ (RPCA) is better than the fixed value, where $\xi=1$ (RPCA without NMF). Third, 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 5.5.3).

Figure 9 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 are more likely to be located in different racks in Amazon data center. We also find that the improvement is
relatively larger for larger message sizes. We observe similar results when comparing RPCA with other comparisons.

### 5.5.2 Results on real-world applications

We study the breakdown of the execution time in the real-world application. 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”).

Figure 10(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 10(b) and 10(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 increases, 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 Baseline. The performance improvement over Heuristics is around 10%.

### 5.5.3 Detailed Performance Comparison

The network environments of Amazon EC2 are 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 $SP$ for the efficiency and effectiveness. Then we study the impact of $N_E$ by introducing noises 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 the supplementary file.

#### Impact of message sizes

We further study the impact of message sizes and observe that as the message size becomes larger, the update maintenance overhead plays a less important role. It is almost ignorable for those message sizes under study. Due to the space limitation, we present the details of our observations in Appendix C of the supplementary file.

#### 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 approach. If the updated $N_E$ reaches the predefined value, we stop. Otherwise, we repeat the process.

Figure 11(a) shows the expected performance improvement of our RPCA approach in broadcast, scatter and topology mapping when $Norm(N_E)$ varies. The $Norm(N_E)$ significantly 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 11(a).

Figure 11(b) shows the performance improvement of comparing RPCA with Heuristics in broadcast when $Norm(N_E)$ varies. 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 12(a) shows the average performance comparison, and Figure 12(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 11(a) and 11(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...
Fig. 10. Performance comparison of real-world applications on 196 medium instances: N-Body and CG.

Fig. 12. Overall performance comparison with $\text{Norm}(N_E)=0.2$

Fig. 13. Detailed performance of Heuristics and RPCA with different strategies

Heuristics by 12–20%.

Impact of significant changes. We report the results for a trace that lasts for three days. The performance of one virtual machine pair in the three-day trace was studied in Appendix A of the supplementary file. Thus, we are able to study both relatively stable network performance (first two days) and significant changes (from day 2 to day 3) from the real cloud environment.

Figure 13 compares the detailed performance of broadcast in the trace of three days. We show the results of Heuristics and RPCA with and without maintenance. The performance difference between RPCA with and without maintenance becomes larger around the end of day 2, where a significant change occurs on the network performance. Our maintenance approach successfully captures the significant change and performs maintenance on the model, which better follows the performance trend of the network performance.

5.6 Results with Simulations

We first report our simulation results on SIM-S. We can control the network traffic and study the efficiency and effectiveness of our approach in the controlled environment. Next, we show the simulation results on SIM-M and SIM-L, which are utilized to study the performance of RPCA in medium and large scale virtual clusters. Finally, we introduce the performance improvement of different approaches on SIM-torus.

5.6.1 Simulation results on SIM-S

Network Interference Study. In Figure 14(a), we fix the message size in the background as 100MB and vary the expected value $\lambda$ from 1 to 30 seconds. The expected value $\lambda$ represents the frequency of the network interference. The figure shows that, as $\lambda$ increases, $\text{Norm}(N_E)$ largely reduces and the network becomes more stable. In Figure 14(b), we fix $\lambda$ 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 $\text{Norm}(N_E)$. From these two experiments, we observe that $\text{Norm}(N_E)$ 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 $\text{Norm}(N_E)$. It represents the interference of network performance satisfies a stable distribution.
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2016.2609387, IEEE Transactions on Computers

Comparison with Topology-aware Algorithm. Figure 15(a) shows the overall performance comparison of broadcast, scatter, topology mapping, CG and N-body in SIM-S. We use the same execution setting for each application as those in Amazon EC2. Machines are randomly selected from the simulated cluster. We simulate the background traffic so that \( \text{Norm}(N_E) = 0.1 \). The execution time \( T_e \) of each application is around three hours and the optimal value \( \xi \) is about 0.16. 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 perform very similar, which indicates that the topology-aware optimization is not effective in such a dynamic environment. RPCA is 20%-35% better than Baseline/Topology-aware approaches and 8%-12% better than Heuristics. Figure 15(b) shows the CDF of the elapsed time of broadcast. The results are similar to those on Amazon EC2.

Impact of \( N_E \). We use different \( N_E \) to study its impact, and observe similar results as Amazon EC2. For example, as \( \text{Norm}(N_E) \) 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 \( \text{Norm}(N_E) \) is too high (e.g., higher than 0.5), the improvement of network performance aware optimizations becomes marginal. Compared with traditional optimizations, our RPCA approach accurately predicts this trend with the estimation on \( \text{Norm}(N_E) \).

Impact of subscription ratios. We study the impact of the subscription ratio of the fat-tree topology in our study. The subscription ratio is defined to be the upload bandwidth divided by the download bandwidth of the fat-tree topology. Specifically, we vary subscription ratio of the fat-tree topology on SIM-S. In this experiments, we control the background traffic so that \( \text{Norm}(N_E) = 0.1 \). We observe that the performance improvements of RPCA over other approaches are similar as the subscription ratio varies from 1:3 to 3:1. The reason is that although the subscription ratio impacts the performance for all the approaches, the performance improvement is mainly determined by the network traffic dynamics. Due to the space limitation, we show the evaluation results in Appendix C of the supplementary file.

5.6.3 Simulation results on SIM-torus
In SIM-torus, we utilize the same execution settings for each application as those in SIM-S. The execution time \( T_e \) of each application is around three hours and the optimal value \( \xi \) is about 0.16. Figure 17 shows the overall performance of SIM-torus. We observe that (1) RPCA without NMF is still much slower than Baseline because of the large overhead in calibration. (2) Topology-aware approaches could not work so effective in the dynamic network environment. (3) The performance improvement for both Heuristics and RPCA in SIM-torus is smaller than that in SIM-S, because in SIM-M and SIM-L, the calibrating overhead is very large. We have to further reduce the calibrating time (lower value of \( \xi \)) in order to improve the efficiency, which decreases the effectiveness. (4) RPCA without NMF is much slower than Baseline, which indicates then importance of NMF in improving the efficiency.

6 Conclusions
This paper revisits network performance aware optimizations of distributed applications running on virtual clusters in IaaS cloud. In this paper, we propose a novel approach based on NMF (Non-negative Matrix Fraction) and RPCA (a well-known problem in computer vision) to efficiently reduce the calibration overhead and effectively find the constant component from dynamic network performance while minimizing the difference between the constant component and network performance. Based on our approach, we are able to determine the efficiency and effectiveness of network optimizations on applications running in the virtual cluster. We can make a trade-off between efficiency and effectiveness. 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 proposed approach is more robust than other approaches for different degrees of network dynamics, and always outperforms other comparison approaches in both real experiments and simulations.
REFERENCES


