A General Scalable and Accurate Decentralized Level Monitoring Method for Large-scale Dynamic Service Provision in Hybrid Clouds[1]

Yongquan Fu[a,*], Yijie Wang[a], Ernst Biersack[b]

[a]*National Key Laboratory for Parallel and Distributed Processing, College of Computer Science, National University of Defense Technology, Hunan province, China, 410073.*
[b]*Networking and Security Department, EURECOM, France.*

Abstract

Hybrid cloud computing combines private clouds with geographically-distributed resources from public clouds, desktop grids or in-house gateways to provide the most flexibility of each kind of cloud platforms. Service provisioning for wide-area applications such as cloud backup or cloud network games is sensitive to wide-area network metric such as round trip time, bandwidth, loss rates. In order to optimize the quality of the service provision in hybrid clouds, it is highly valuable for the hybrid clouds to collect detailed network metric between participating nodes of the hybrid clouds. However, since nodes can be large-scale and dynamic, the network metric may be diverse for different cloud services, it is challenging to increase the generality, scalability, accuracy and the robustness of the measurement process. We propose a novel distributed level monitoring method HPM (Hierarchical Performance Measurement) satisfying these requirements. For each kind of network metric, HPM represents the degree of pairwise closeness with discrete level values inspired by the hierarchical clustering tree. HPM maps probed metric to discrete levels based on an existing distributed K-means clustering method that helps maximize the similarity of the network metric in the same level, which therefore optimizes the matching between pairwise levels and the real-world pairwise proximity. Furthermore, HPM computes the pairwise levels with decentralized coordinates for scalability. Each node independently maintains its low-dimensional coordinate based on a novel decentralized implementation of the Maximum Margin Matrix Factorization method that optimizes

[1]To appear in Future Generation Computer Systems

the mapping between the network metric and the level values. Simulation results for the RTT, bandwidth, loss and hop metric confirm that HPM converges fast, is robust to parameter settings, scales well with increasing levels or system size, and adapts well to diverse metric. A prototyping deployment on the PlanetLab platform shows that HPM not only converges fast, but also incurs modest maintenance bandwidth costs. Finally, applying HPM to optimize the service provision of hybrid clouds shows how HPM can achieve close to optimal solutions.

Keywords:
decentralized algorithm, application health monitoring, hierarchical decomposition, K-means clustering, decentralized matrix factorization

1. Introduction

Service provisioning in hybrid clouds that combines geographical-distributed and heterogeneous platforms such as private and public clouds, clusters, grids, desktop grids, or in-house gateways, can maximize the benefits each kind of platforms. For example, high performance computing (HPC) applications can be better completed by combining the internal capacity of the private clusters and elastic resources of the public clouds; cloud backup (e.g. Wuala [1]) services can store users' data on nearby in-house gateways for fast response and remote distributed desktop grids for redundancy; network gaming (e.g Halo [2]) services can scale to millions of clients by combining processing capacity of the private clouds, elastic resources of public clouds and low-latency desktop grids. Several hybrid clouds such as Aneka [3, 4], MOON [5], NaDa [6], Elastic Cluster [7] have already attracted tremendous attention from both academic and industry fields.

Unfortunately, service provisioning with wide-area distributed nodes comes with costs, since the quality of many cloud services such as high performance computing (HPC), cloud backup (e.g. Wuala [1]), network gaming (e.g Halo [2]) is sensitive to end to end network metric. For instance, the HPC scientific applications need minimum pairwise delays when synchronizing states of different nodes [4]; the file backup service is affected by low bandwidth or high packet losses; game players' experiences may be impaired by high RTTs or packet losses.

*Corresponding author. Phone: +8613875828390.
Email addresses: yongquanf@nudt.edu.cn (Yongquan Fu), wangyijie@nudt.edu.cn (Yijie Wang), erbi@eurecom.fr (Ernst Biersack)

As a result, in order to improve the quality of the service provision in hybrid clouds, the hybrid cloud platforms need to monitor the network conditions between participating nodes. However, the number of participating nodes could grow to thousands or millions, which implies that directly measuring pairwise network conditions does not scale well; worse still, the participating nodes may also join or leave the hybrid cloud dynamically because of failures, maintenance or decentralization. Therefore, a scalable and decentralized measurement method is valuable for performance optimization in hybrid clouds.

1.1. Related Work

Existing work on measuring network conditions for large-scale and dynamic nodes can be categorized into the **absolute-value measurements** and the **relative-value measurements**:

- The absolute-value measurement provides detailed end to end network metric, such as the RTT, bandwidth that can satisfy diverse performance-optimization requirements. However, measuring the precise absolute values is costly, since covering all-pair routing paths for $O(N)$ sized systems requires $O(N^2)$ measurements. As a result, most absolute-value measurements use mathematical models to predict pairwise network metric.

- The relative-value measurement only provides degrees of proximity between nodes, which is less powerful than the absolute-value measurements, but can also fulfil many performance-optimization needs. For example, they allow to select proximity nodes for matchmaking in network games in terms of RTTs, or losses, or select the backup servers based on the proximity of the bandwidth values or the loss rates. Moreover, since the measurements only need to infer the relative proximity relations, the measurement bandwidth cost can be reduced.

1.1.1. Absolute-Value Measurement

Many absolute-value measurements predict end to end network metric in order to improve the scalability of the measurements. Existing prediction methods can be categorized into network coordinate based methods, topology based methods and network tomography based methods.

First, the network coordinate based methods embed nodes into low-dimensional coordinate space and predict end-to-end absolute-value metric based on point-to-point coordinate distances. Each node maintains its own coordinate using a fixed number of neighbors, the overall bandwidth cost of the method is $O(N)$. However,

most network coordinate methods are specifically suitable for the RTT metric, such as GNP [8], Vivaldi [9], Htrae [10], NetICE9 [11], DMF [12], Phoenix [13], which limits the generality of the measurement process. Recently, researchers have extended network coordinates to predict the bandwidth or hop metric. The Non-Metric [14] method can predict both delays and bandwidth with the min-plus metric. Beaumont et al. [15] and Douceur et al. [16] predict end-to-end bandwidth based on the constraints of the upload and download capacity of decentralized nodes. Xing et al. [17] have proposed an Ultra metric based bandwidth estimation scheme. However, these extended network coordinates for bandwidth or loss rates are less accurate than those that predict the RTT metric.

Second, the topology based methods predict a virtual topology for participating nodes and estimate end-to-end metric using topology distances. The topology expresses flexible proximity information between nodes. iPlane [18] and iPlane Nano [19] create an approximated Internet topology based on extensive Traceroute measurements from distributed vantage points. The Sequoia methods [20, 21] construct multiple trees to predict the delays and bandwidth between decentralized nodes. Unfortunately, due to the dynamics of the network metric and the participating nodes, maintaining the topology incurs high bandwidth cost.

Third, some of the network tomography based methods predict end-to-end absolute-value metric using algebra based models. The algebra models assume a linear relationship between path links and path metric such as the delay and loss metric. As a result, the models are no longer useful when such linear assumption is violated, e.g., the bandwidth metric. Chen et al. [22] propose to select a basis set of routing paths to monitor and estimate pairwise delay and loss of overlay nodes with linear systems. But it is not clear whether such an approach applies to other performance metric. Coates et al. [23] estimate end-to-end path metric based on the diffusion wavelets and nonlinear estimation that tolerate incomplete path measurements. Qazi and Moors [24] further show that the overlay monitoring quality can significantly degrade when some parts of the routing topology are missing or incorrect. Besides, the routing topology is usually assumed to be known for establishing the algebra models, which may be challenging for hybrid clouds involving distributed and dynamic nodes.

1.1.2. Relative-value Measurement

The relative-value measurement methods directly compute pairwise proximity based on mathematical models for scalability. Existing methods can be categorized into divisive clustering methods, hierarchical clustering methods and the coordinate based methods.

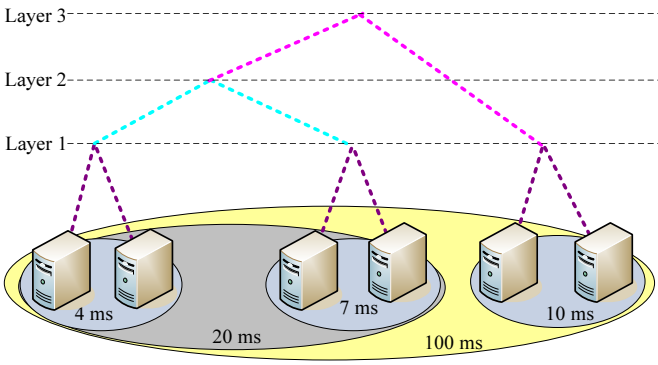


Figure 1: A simple hierarchical clustering example. There are three levels in the logical tree. The level number decreases one per layer from the top layer to the bottom layer.

First, the divisive clustering based methods group nearby nodes into the same cluster. The intra-cluster nodes are closer than inter-cluster ones. Unfortunately, nodes in the same cluster are assumed to be equally close to each other and no proximity information is available for inter-cluster nodes. Beaumont et al. [25] aggregate distributed resources into proximity clusters of nodes based on approximately solving the problem of bin covering under distance constraints. SOLARE [26] constructs utility-optimized proximity clusters of nodes in the P2P structured overlay. Malik et al. [27] iteratively divide nodes into proximity clusters based on static proximity threshold values.

Second, the hierarchical clustering based methods constructs a logical tree to represent multilevel proximity. The hierarchical clustering method [28] recursively groups nearby nodes into multiple **levels** to produce a logical tree structure. Tiers [29] organizes distributed nodes as a logical tree where cluster-head nodes recursively re-cluster themselves in a bottom-up manner. Sequoia [20] creates logical trees where distributed physical nodes are added into the tree as the leaf vertices and virtual nodes are added to connect all leaves as a tree. Wieser and Böszöényi [30] recursively aggregate P2P nodes based on the pairwise hops on the overlays, and nodes with the highest identifiers are recursively selected as the cluster heads at each level.

Figure 1 plots an example of the hierarchical clustering tree for six nodes based on the RTT metric. The level value of two nodes at the bottom level of the tree is the layer number of their **closest common ancestor** in the tree that is depicted by the horizonal dashed lines in the figure. Higher levels correspond to larger RTTs.

Although the hierarchical clustering tree provides intuitive and powerful proximity information, it also has two drawbacks:

- It is unsuitable for dynamic nodes, since we have to frequently update the logical topology when some nodes leave or join the tree or the pairwise network metric changes, which increases the maintenance costs.

- It cannot guarantee the matching between the topology structure and the ground-truth metric, since most trees are created based on local heuristics. Several network metric such as the available bandwidth [31, 32, 33], routing hops are asymmetric, i.e., the bandwidth or hop count from one direction differs from that in the reverse direction. For example, the asymmetry of the network metric is missing on the topology structure.

Third, the coordinate based methods embed nodes into proximity space and represent the pairwise proximity based on the coordinate distances. Unfortunately, most existing coordinate based approaches typically focus on one kind of network metric. Netvigator [34] creates relative coordinates using the vectors of RTTs from edge nodes to landmarks and some 'milestone' routers found based on traceroute measurements. Netvigator's similarity is computed as clusters of coordinates. CRP [35] constructs coordinates of end hosts as the frequency of being forwarded to different CDN edge servers. The pairwise similarity of CRP is based on the cosine similarity of two coordinates. Shen and Hwang [36] computes the vectors of RTTs from edge nodes to landmarks as coordinates and uses the space-filling curves to represent the pairwise proximity of nodes. Liao et al. [37] propose a general and distributed method to map absolute-value measurements to binary performance classes, i.e., good or bad, which is perhaps the most related work with us. However, there are three differences between [37] and our study:

- *The level mapping process is different*. Liao et al. [37] only performs binary classification, while we explicitly map network metric to a tunable number of levels. Since the network metric values are usually skewed, we propose a distributed K-means clustering based level mapping process that maximizes the similarity of network metric values in each level.

- *The coordinate structure is different.* Liao et al. [37] represent the coordinate structure with matrix factorization and take the sign of the coordinate distances as the binary performance classes. We represent the coordinates

with matrix factorization and thresholds that optimally map the coordinate distances to discrete levels.

- *The coordinate movement is different*. Liao et al. [37] use the stochastic gradient descent method to update coordinates with fixed movement step. We propose a distributed conjugate gradient method to adjust the coordinates with optimal movement steps.

1.2. Our Approach

Our objective is to provide fine-grained relative-value measurements for optimizing the service provision of hybrid clouds. To that end, inspired by the layered proximity of the hierarchical clustering tree, we propose a novel network metering metric, i.e., the **level** number, to quantify the pairwise proximity. The level number is represented by integer numbers, where higher level numbers mean worse network metric. The level number is computed for each direction of a pair of nodes. Therefore, the asymmetry of network metric can be preserved by defining suitable level mapping procedures, which improves the granularity of proximity by the hierarchical clustering tree.

We propose a novel distributed level measurement method called **Hierarchical Performance Measurement** (HPM) that can measure the levels for any kind of network metric for large-scale and dynamic nodes of hybrid clouds. For each kind of network metric, HPM first maps measured network metric to level values that preserve the proximity of network metric values and then estimates pairwise levels based on decentralized coordinates for scalability.

First, in order to provide accurate proximity information, we propose a distributed K-means clustering based level mapping method to group the most similar network metric to the same level. Therefore, the level values are able to match the pairwise proximity between nodes. The number of levels can be tuned to show detailed proximity of nodes. Besides, we also recommend how to set the number of levels based on the distributions of network metric. Since we separately compute level values for unidirectional measurements, the asymmetry of network metric values can be preserved.

Second, since directly measuring all-pair level values does not scale well, we propose a novel distributed implementation of a well-known collaborative filtering method called Maximum Margin Matrix Factorization (MMMF) [38, 39]. The key idea of MMMF is to learn adaptive thresholds to optimally map coordinate distances to level values. For level completion, each node independently maps the coordinate distances from itself to other nodes to discrete levels using its own

thresholds that are part of the coordinate structure. We propose a decentralized implementation of MMMF method to adapt the large-scale and dynamic nodes. Each node maintains its coordinate based on the distributed implementation of the conjugate gradient method. At each step, the coordinate is adjusted with an optimal step size towards the conjugate gradient direction.

In summary, we make the following contributions:

- We propose a novel level based network metering method that represents the hierarchical proximity for any kind of network metric. The method is general for different network metric such as RTTs, bandwidth, hop counts or loss rates.
- We propose a scalable and accurate decentralized level measurement procedure HPM that estimates pairwise levels in a fully decentralized manner.
- We extensively evaluate the efficiency and efficacy of HPM with simulations and the PlanetLab-deployment experiments.
- We apply the level metric to optimize the service provision of hybrid clouds and show its superiority against state-of-art methods for estimating pairwise network metric.

The rest of the paper is organized as follows. Section 2 defines the level concept and presents the level-estimation problem. Section 3 introduces the data sets used in this paper. Section 4 presents key ideas of HPM. Section 5 shows how to map network metric to optimal levels. Section 6 presents the level estimation method. Section 7 presents the simulation results on the real-world data sets. Section 8 presents the implementation of HPM and the performance on the PlanetLab. Section 9 shows several application examples of HPM. Section 10 concludes the paper.

2. Problem Definition

2.1. Level

As discussed in Section 1, the concept of the level is inspired by the hierarchical clustering tree. Each level value is an integer for ease of representation. Higher level values imply worse network performance.

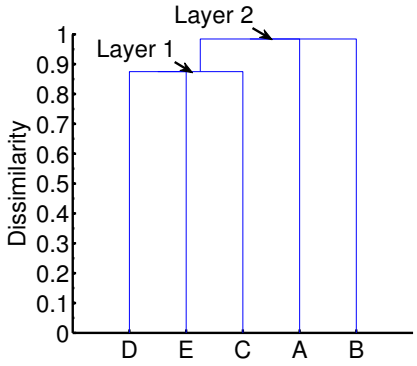


Figure 2: A hierarchical clustering example for the bandwidth matrix. Since the hierarchical clustering requires the dissimilarity metric, but the bandwidth is a similarity metric, i.e., higher bandwidth corresponds to higher proximity, we transform the bandwidth to dissimilarity value as $\widehat{\mathbf{D}}_{ij} = 1 - (\mathbf{D}_{ij}/\mathbf{MaxBW})$, where $\mathbf{MaxBW}$ denotes the maximal pairwise bandwidth, i.e., 8 Mbps.

We illustrate the level values with the hierarchical clustering. Consider five geo-distributed machines indexed by A, B, C, D and E that are located in different networks. The pairwise bandwidth matrix $\mathbf{D}$ (Mbps) is given as:

$$\mathbf{D} = \begin{bmatrix} 0 & 0.125 & 0.125 & 0.125 & 0.125 \\ 0.125 & 0 & 0.125 & 0.125 & 0.125 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 8 & 8 & 0 \end{bmatrix} \tag{1}$$

where the i-th row vector corresponds to the bandwidth from the i-th node to other nodes.

We use the standard hierarchical clustering method to create the logical tree for these five nodes. Figure 2 plots the result. We set the level number between two nodes as the layer number of their closest common ancestor in the tree. For example, node C, D and E are grouped in a cluster, and their pairwise level value is 1; since node A and B have low bandwidth to all other nodes, their level values to other nodes are 2. In fact, the pairwise levels by the hierarchy clustering are symmetric, since there is only one closest common ancestor for any two leaf nodes.

However, the symmetry of the hierarchical clustering tree distorts the structure of the bandwidth matrix, since the bandwidth matrix $\mathbf{D}$ is extremely asymmetric. On the other hand, the pairwise level numbers can preserve the asymmetry, since we can separately compute the level number for each unidirectional network metric value. As a result, the level value is more general than the hierarchical clustering tree.

2.2. Measurement Goal and Challenges

Given a set of large-scale and distributed nodes provisioned by hybrid clouds. These nodes may dynamically join or leave cloud services. Assume that these nodes are able to probe any kind of network metric to each other using active measurement tools. However, due to the limited capacity and fixed access bandwidth constraints, each node can concurrently probe a small number of nodes.

Our goal is to compute pairwise levels for these large-scale and dynamic nodes. There are two main challenges for the level measurement process:

- *How to map metric values into optimal levels*. An ideal level mapping process should maximize the similarity of network metric values that are mapped into the same level, and separate those dissimilar ones into different levels.

- *How to scale the level measurements*. The measurement procedure should incur modest computation overhead and bandwidth overhead with increasing number of nodes or levels.

3. Data Sets

We use four representative network metric for our evaluation: the RTT metric, the available bandwidth metric, the end-to-end routing hop metric and the end-to-end loss rate metric. We then choose four publicly available data sets for the experiments:

- **RTT**, the pairwise RTT metric between 169 PlanetLab machines from the pairwise Ping project [40];

- **Bandwidth**, the pairwise available bandwidth matric between 360 PlanetLab machines from the S^3 project [41];

- **Hop**, the pairwise routing hop metric between 188 PlanetLab machines by the iPlane project [18];

- **Loss**, the loss rate metric between 146 DNS servers by the Queen project [42].

Some of the data sets are asymmetric, which are illustrated by the **asymmetry ratio**:

$$r_{asy} = \frac{\min\left(D_{ij}, D_{ji}\right)}{\max\left(D_{ij}, D_{ji}\right)} \tag{2}$$

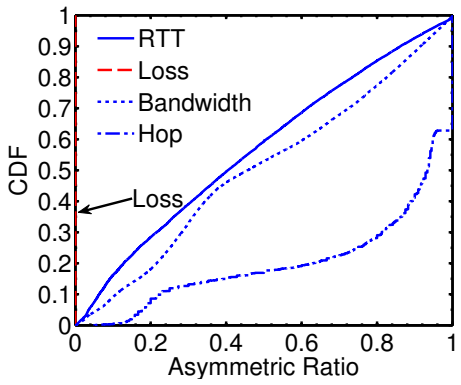


Figure 3: The asymmetry ratio of all data sets.

for $D_{ij} \geq 0, D_{ji} \geq 0$ but $D_{ij} \neq D_{ji}$. The asymmetry ratio r_{asy} is always ≤ 1. A data set is symmetric if and only if $r_{asy} = 1$ for every pair of nodes. We set $r_{asy} = 0$ when D_{ij} or D_{ji} is zero. Figure 3 plots the results. We can see that all data sets contain a fraction of asymmetric metric values. The loss rate data set has nearly 100% asymmetry for all node pairs.

4. Our Design

We design and implement a scalable decentralized level estimation method called **Hierarchical Performance Measurements** (**HPM**). Assume that a set of decentralized nodes run the HPM method. For each kind of network metric, each node estimates level values to other nodes based on decentralized coordinates that is computed in a fully decentralized manner: Each node periodically probes the network metric to a small number of sampled nodes (called neighbors), then computes the optimal levels for these probed metric, and finally incrementally updates its coordinate based on these sampled level values and the coordinates of its neighbors.

In order to accurately preserve the proximity of nodes into the pairwise levels, HPM proposes two complement techniques:

- *Distributed K-means clustering based level mapping*. Given a network metric, each node maps the absolute-value measurements into discrete levels based on an existing distributed K-means clustering method [43] that maximizes the similarity of the network metric values in the same level.

- *Distributed maximum margin matrix factorization based level estimation*. We complete unobserved level values based on a novel distributed implementation of the Maximum Margin Matrix Factorization method, in order to optimize the mapping of coordinate distances to discrete levels. The coordinate includes low-dimensional vectors and thresholds. The products of vectors are treated as the coordinate distances; the thresholds are used to map coordinate distances to discrete levels. Each node incrementally adjusts its coordinate based on a distributed conjugate gradient method that has low computational costs and fast convergence speed.

HPM has the following advantages:

- **Applies to diverse metric**. The level estimation process assumes that the pairwise level matrix has low effective ranks. Such an assumption holds for a wide range of network metric. Since many end hosts share partial routing paths to some target nodes [20], the network metric values from end hosts to the targets are correlated with each other. The pairwise level values between these end hosts to the targets are also correlated with each other. Therefore, the pairwise level matrix can be well approximated by a low-rank matrix.

- **Scales with increasing number of hosts**. The measurement process is fully distributed.

- **Adapts to system churn**. The estimation accuracy does not change significantly when hosts join or leave the system.

- **Adapts to the temporal variations of network metric**. The estimation keeps up-to-date levels using periodical coordinate updates.

5. Mapping Network metric Into Discrete Levels

In this section, we introduce how to optimally map a network metric into a discrete level in a distributed manner.

5.1. Problem Formulation

As discussed in the introduction section, the level mapping process should maximize the similarity of network metric values in the same level, so that the level results stably represent latent proximity between nodes. To that end, since the network metric values can be arranged into points in a line according to their magnitudes, we should perform clustering on the network metric values.

Clustering network metric values means we need to cluster points in a one-dimensional space. Let the L_1 distance between two network metric values d_x, d_y be the absolute difference of d_x and d_y, i.e.,

$$|d_x - d_y| \tag{3}$$

The clustering objective is to optimally separate points into dense regions in the line. We choose the well-known K-means clustering method [28] to cluster network metric values to L groups:

$$\underset{I=\{I_l, l\in[1,L]\}}{\arg\min} \sum_{l=1}^{L} \sum_{\mathbf{D}_{ij}\in I_l} \left(\mathbf{D}_{ij} - \mu_l\right)^2 \tag{4}$$

where L denotes the number of levels, $\mathbf{D}_{ij}$ denotes the network metric value from node i to node j, I_l represents the set of network metric values in the l-th cluster, I denotes the whole set of network metric values, and μ_l represents the cluster centroid of the l-th cluster:

$$\mu_l = \frac{\sum_{D_{ij}\in I_l} D_{ij}}{|I_l|} \tag{5}$$

As a result, the objective of the level mapping is transformed to solve Eq (4) for large-scale and dynamic nodes.

5.2. Outlier Detection

When the network metric values that do not belong to typical clusters, i.e., outliers, are adopted for the K-means clustering process, the cluster quality can be severely impaired. For robust clustering, we need to remove the effects of these outliers.

Intuitively, in a one-dimensional space, the outlier points are far away from other points. Therefore, if a network metric value is not in dense clusters, this value can be regarded as an outlier. We propose a hierarchical clustering based outlier detection method. For each node A, it performs the following steps:

1. In order to construct accurate clusters, node A collects network metric values measured by its neighbors. For each neighbor, node A randomly samples at most n_{outlier} network metric values without replacement.
2. Node A constructs a hierarchical clustering tree for the network metric values. The distance between two network metrics is calculated by Eq (3). Let the maximum layer number in the hierarchical clustering tree be $L_{\text{cluster}}^{\max}$.

3. In the hierarchical clustering tree, each network metric value that has layer number $L_{\text{cluster}}^{\max}$ with every other network metric values is regarded as an outlier. In other words, network metric values in singleton clusters are treated as outliers.

After detecting outliers from the samples, the remained network metric values that are not outliers are used for obtaining cluster centroids in Section 5.3.

Removing these network metric values may be inappropriate, since the cluster centroids may evolve due to the dynamics of network metric or nodes, therefore the set of outliers may also change dynamically. As a result, after a network metric value is regarded as an outlier, this value is only skipped for one round of the K-means clustering process.

For a node A with $|S_A|$ neighbors, constructing the hierarchical cluster tree needs $O((n_{\text{outlier}} \times |S_A| + n_A)^2)$ computing complexity in the worst case, where n_{outlier} denotes the maximum number of samples from each neighbor, n_A denotes the number of network metric values measured by node A.

5.3. Obtaining Cluster Centroids

For K-means clustering, each network metric value is mapped to the nearest clustering centroid. Therefore, the key problem is to obtain the optimal clustering centroids $\vec{\mu} = \{\mu_1, \ldots, \mu_L\}$.

We use an existing distributed K-means clustering method [43] to optimize Eq (4). The basic idea is to let each node learn the optimal cluster centroids via gossip communications. When a node A joins the system, it randomly samples a number of online nodes as neighbors. Node A then initializes its global cluster centroids $\vec{\mu}_A$ as a random vector. Then each node updates its global cluster centroids with its neighbors via rounds of gossip communications. The global cluster centroids of different nodes gradually converge to identical positions.

In order to adapt the dynamics of network metric values, each node A periodically performs the following operations:

1. *Measurement based local-centroid update*. Node A periodically performs measurements to its neighbors and then computes its local cluster centroids $\vec{w}_A$ based on the current global cluster centroids $\vec{\mu}_A$: For each network metric value d_x, node A selects the centroid $\mu_A(j)$ in $\vec{\mu_A}$ that is nearest to d_x, i.e.,

$$j = \arg\min_{i \in [1,L]} |\mu_A(i) - d_x| \tag{6}$$

Then, for each cluster $i \in [1, L]$, node A computes the local centroid w_A^i as the average value of all network metric values assigned to each cluster, i.e.,

$$w_A^i = \frac{1}{|D_i|} \sum_{d_l \in D_i} d_l \tag{7}$$

where D_i denotes the set of network metric values in the i-th cluster. Let the size of the i-th cluster be $n_A^i = |D_i|$.

2. *Gossip based global-centroid update*. Let the vector of the local cluster centroids be $\vec{w}_A = \{w_A^1, \ldots, w_A^L\}$. Let the vector of the cluster sizes be $\vec{n}_A = \{n_A^1, \ldots, n_A^L\}$. Node A periodically pushes its local centroids $\vec{w}_A$ and the size of each cluster $\vec{n}_A$ to its neighbors and pulls back the local centroids and size of each cluster of neighbors. Then, each node A aggregates its j-th global centroid $\mu_A(j)$ as:

$$\mu_A(j) = \frac{\sum_{i \in S_A} n_i^j w_i^j}{\sum_{i \in S_A} n_i^j} \tag{8}$$

for $j \in [1, L]$, where S_A includes neighbors and node A itself, n_i^j denotes the size of the j-th cluster of node i in S_A, w_i^j denotes the j-th centroid of node i.

5.4. Mapping Network Metric to Levels

Having obtained the clustering centroids, we next assign level values for network metric values in each cluster. The cluster centroids are sorted in an ascending (or descending for bandwidth) order. Then each node A stores the sorted centroids into a list L_{centroid}. Each network metric value is mapped to the nearest centroid. The level number is calculated as the index of that mapped centroid in the list. Formally, given a network metric sample d_x measured by node A. Node A then maps d_x to the nearest centroid $\mu_A(l)$ in the list L_{centroid}:

$$l = \arg\min_{i \in [1,L]} |d_x - \mu_A(i)| \tag{9}$$

Figure 4 illustrates the results after the level mapping process. We can see that points of a cluster are mapped to an identical level number. Therefore, the similarity of network metric values in a level is maximized.

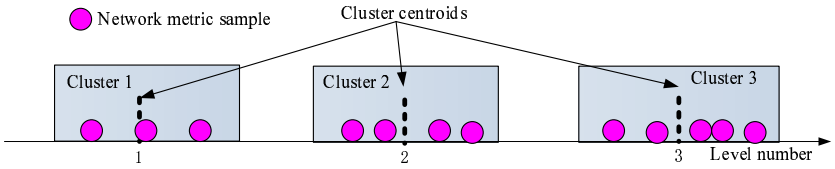


Figure 4: Map network metric values to levels.

5.5. Determining the Number of Levels

The number of levels is up to the applications' needs and can be configured to a wide range of integer numbers. However, all nodes should have the same number of levels for interoperability.

We also propose an offline heuristic to recommend the number of levels L that tries to find the optimal clustering structure for a kind of network metric. Assume that a node A collects the measurement results from some of online nodes. Then the node A computes the optimal number L of clusters that minimizes Eq (4), i.e., the K-means clustering error function. Finally, node A distributes the optimal number L of clusters to other nodes.

5.6. Example

We next use an example to illustrate the process of the level mapping. Let the number L of clusters be 2. Assume that each node has two neighbors. Using the measurements to neighbors, we construct an **incomplete bandwidth matrix** $\mathbf{D1}$ as follows:

$$\mathbf{D1} = \begin{bmatrix} 0 & 0 & 0.125 & 0 & 0.125 \\ 0.125 & 0 & 0.125 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 8 & 0 \end{bmatrix} \tag{10}$$

based on the bandwidth matrix $\mathbf{D}$ in Eq (1).

We first perform the outlier detection. Node *E*, i.e., the node corresponding to the fifth row vector of $\mathbf{D1}$, constructs a hierarchical cluster tree based on network metric values $(1, 8, 0.125, 0.125, 1, 1)$ that include the measurements from its neighbors A and D. From the hierarchical tree, we found that only the sample 8Mbps is in a singleton cluster. As a result, the value 8Mbps is regarded as an outlier.

We next calculate the K-means clustering centroids. We run the distributed K-means clustering until all nodes reach identical centroids, which are 0.125 Mbps and 1 Mbps.

Finally, we compute the level value for each measured bandwidth in $\mathbf{D1}$. For example, 0.125 Mbps should be mapped to level 1, 1 Mbps and 8 Mbps should

be mapped to level 2. As a result, the pairwise level matrix Y for the bandwidth matrix $\mathbf{D}$ and the level matrix $Y1$ for the bandwidth matrix $\mathbf{D1}$ are as follows:

$$Y = \begin{bmatrix} 0 & 2 & 2 & 2 & 2 \\ 2 & 0 & 2 & 2 & 2 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \; Y1 = \begin{bmatrix} 0 & 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{11}$$

The level matrices Y and $Y1$ preserve most observed asymmetries in the bandwidth matrix, except the one between D and E because of a limited number of levels.

6. Decentralized Level Estimation

Having shown how to map measured network metric values to discrete levels, we next introduce how to estimate level numbers with decentralized coordinates for unobserved network metric values. In order to accurately predict pairwise levels, we model the optimization objective of finding the coordinates based on the well-known Maximum Margin Matrix Factorization (MMMF) [38, 39]. We implement a decentralized MMMF algorithm based on a novel distributed conjugate gradient method that converges within a few tens of rounds and keeps to be stably accurate after convergence. Table 1 summarizes key notations in HPM.

6.1. Problem Formulation

For large-scale and dynamic nodes in hybrid clouds, we prefer to estimate pairwise levels in a distributed manner. Besides, in order to preserve the asymmetry of level values, we should estimate unidirectional level values for a pair of users.

6.1.1. Estimating the Level

We define the coordinate distances based on the matrix factorization, in order to be general enough to adapt different kinds of network metric: The pairwise coordinate distance is represented by the linear combination of two low-rank matrices:

$$\widehat{X} = U \times V \tag{12}$$

where U is a $N \times d$ matrix and V is a $d \times N$ matrix and d denotes the rank of the matrices U, V, $d \ll N$.

Suppose that there are a total of L levels. Since a pairwise level Y_{ij} is an ordinal value, i.e., $Y_{ij} \in \{1, 2, \ldots, L\}$, but the coordinate distance $\widehat{X}_{ij}$ is a real-valued number, we have to classify the continuous coordinate distance $\widehat{X}_{ij}$ into the ground-truth level Y_{ij}.

Notation	Meaning
N	the number of nodes
$\mathbf{D}$	performance measurement matrix
L	number of layers
d	coordinate dimension
I_l	l-th cluster of the performance measurements
I	whole set of clusters I_l
μ_l	averaged performance value of the l-th cluster
S	set of neighbors of a node
Y	layer matrix
$\widehat{X}_{ij}$	coordinate distance from node i to node j
$\widehat{\theta}$	thresholds for layer mappings
$h\left(z\right)$	soft-margin loss function
T	a function quantifying the layer estimation errors
$\left(\widehat{u_i}, \widehat{v_i}, \widehat{\theta_i}\right)$	coordinate of each node, $\widehat{X}_{ij} = \widehat{u_i} \times \widehat{v_j}$
α	the regularization constant
Ω	the set of observed layer values
Δ	steepest direction
Λ	conjugate direction
β_i	Polak-Ribière scalar
α_i	movement step
$\widehat{Y}$	an estimated layer matrix
l_p	number of UDP packets per measurement
C_M	maximal number of neighbors
T_g	inter-gossip interval

Table 1: Key notations in HPM.

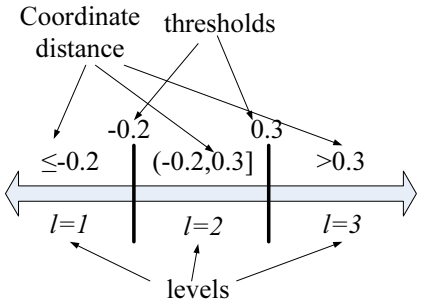


Figure 5: Map coordinate distances to level values.

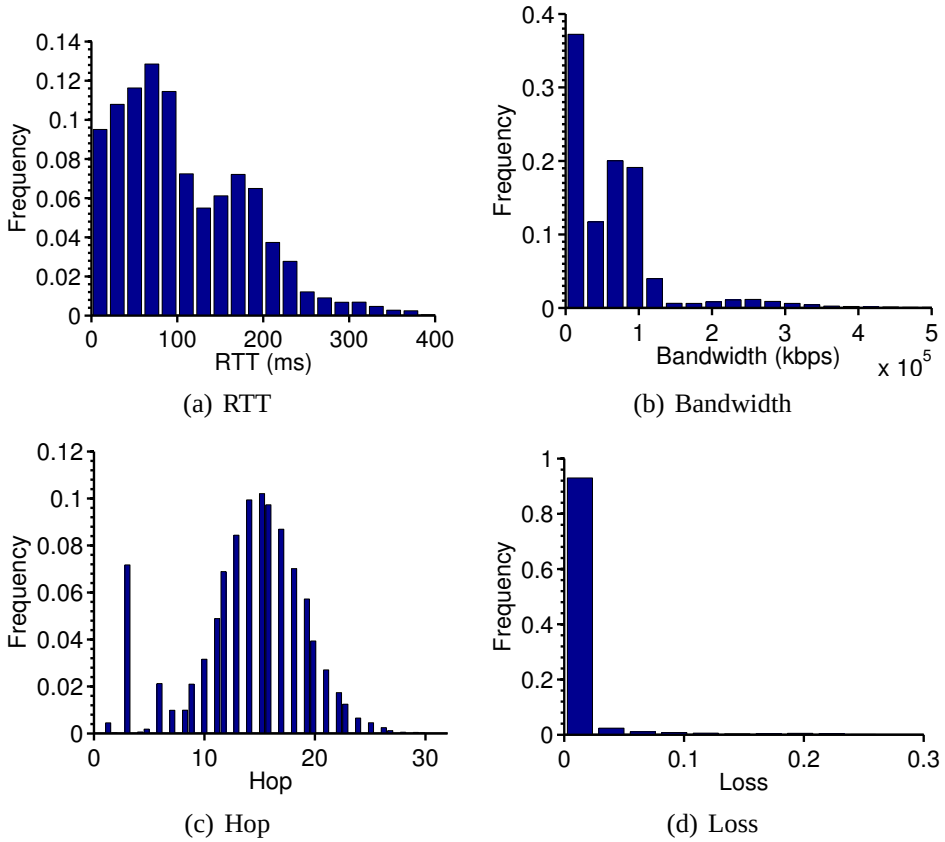
For robust classification against coordinate errors, we use thresholds to map coordinate distances to level values. Each node maintains adaptive thresholds and use thresholds as separation points of mapping continuous coordinate distances to ordinal level values. For each node i, it maintains $L-1$ real-valued thresholds $\vec{\theta_i} = \left(\theta_{i1}, \ldots, \theta_{i(L-1)}\right)$. The thresholds separate the whole range of real values into L disjoint intervals: $\left(-\infty, \theta_{i1}\right], \left(\theta_{i1}, \theta_{i2}\right], \ldots, \left(\theta_{i(L-1)}, +\infty\right)$. Then, node i maps each coordinate distance to the interval that contains the distance. Finally, node i compute the level value as the index of that mapped interval. For example, let the number of levels L be 3 and let two real-valued numbers -0.2, 0.3 be two thresholds. Figure 5 plots such an example of how to compute the levels for a specific node.

Setting proper thresholds is fundamental for accurately computing the levels. Uniformly distributed thresholds do not preserve the proximity of network metric values, since most kinds of network metric are distributed non-uniformly, as shown in Figure 6. Therefore, we need to adaptively set the thresholds according to the distributions of network metric values. To that end, we treat the thresholds $\vec{\theta_i}$ of a node i as new dimensions of the coordinate, which will be computed in a fully decentralized manner.

6.1.2. Coordinate Structure

The coordinate includes low-rank vectors and the thresholds. Let $\vec{u_i}$ denote i-th row vector of U, $\vec{v_i}$ denote the i-th column vector of V and $\vec{\theta_i}$ denote the i-th row vector of the threshold matrix $\theta = \left(\vec{\theta_1}, \ldots, \vec{\theta_N}\right)$. We set the **coordinate** of node i to be the vector $\vec{x}_i = \left(\vec{u_i}, \vec{v_i}, \vec{\theta_i}\right)$.

The **coordinate distance** $\widehat{X}_{ij}$ between node i and j is computed by the inner



(a) RTT (b) Bandwidth

(c) Hop (d) Loss

Figure 6: Distributions of popular network metric based on publicly available data sets in Section 3.

product of $\vec{u_i}$ and $\vec{v_j}$:

$$\widehat{X}_{ij} = \sum_{m=1}^{d} u_{im} v_{mj} \tag{13}$$

The coordinate distance can be symmetric or asymmetric, since $\widehat{X}_{ij}$ may differ from $\widehat{X}_{ji}$.

For level estimation, each node i independently maps the coordinate distances from itself to other nodes using node i's thresholds $\vec{\theta_i}$. Suppose that we need to estimate the level value from node i to node j. We first compute the coordinate distance $\widehat{X}_{ij}$ from node i to node j. Then, we map $\widehat{X}_{ij}$ into a level value using node i's threshold vector $\vec{\theta_i}$. Algorithm 1 summarizes the steps. Step 3 computes the coordinate distance. Step 4 initializes the level number. Steps $5 \rightarrow 9$ iterate

through the threshold values $\vec{\theta_i}$ to find a numerical interval that contains $\widehat{X}_{ij}$. Steps $6 \rightarrow 8$ detects whether the current numerical interval contains $\widehat{X}_{ij}$.

Algorithm 1: level Completion from node i to node j.

{Input: node i's coordinate $\vec{u_i}, \vec{v_i}, \vec{\theta_i}$, node j's coordinate $\vec{u_j}, \vec{v_j}, \vec{\theta_j}$}
{Output: estimated level value l}
$\widehat{X}_{ij} = \vec{u_i} \times \vec{v_j}$
$l = 1$
for $k = 1 : (L-1)$ **do**
  if $\widehat{X}_{ij} \geq \theta_{ik}$ **then**
    $l \leftarrow l + 1$
  end if
end for

6.1.3. Level Estimation Error

Having defined how to map coordinate distances to discrete levels, we next define the optimization objective for computing the coordinates, which measures the difference between the ground-truth level values and the estimated level values.

Assume that we have measured the level value Y_{ij} ($Y_{ij} \in [1, L]$) from node i to node j based on Section 5. Then in order to estimate the ground-truth level values by Algorithm 1, node i should map the coordinate distance $\widehat{X}_{ij}$ to the numerical interval $\left(\theta_{i(Y_{ij}-1)}, \theta_{iY_{ij}}\right]$, where $\theta_{i0} = -\infty$, $\theta_{iL} = +\infty$. Otherwise, the level estimation process incurs an error, where the coordinate distance $\widehat{X}_{ij}$ is either mapped to the left intervals of the threshold $\theta_{i(Y_{ij}-1)}$, i.e., $\hat{X}_{ij} - \theta_{i(Y_{ij}-1)} < 0$ or to the right intervals of the threshold $\theta_{iY_{ij}}$, i.e., $\theta_{iY_{ij}} - \hat{X}_{ij} < 0$. We next measure the level-estimation error based on the **soft-margin loss function** $h(z) = \max(0, 1-z)$ for robustness against input noises:

$$f\left(Y_{ij}, \hat{X}_{ij}\right) = h\left(\widehat{X}_{ij} - \theta_{i(Y_{ij}-1)}\right) + h\left(\theta_{iY_{ij}} - \widehat{X}_{ij}\right) \tag{14}$$

$f\left(Y_{ij}, \hat{X}_{ij}\right)$ is a convex differential loss function that can be optimized efficiently by gradient based methods [38]. Therefore, when the estimated levels are incorrect, $f\left(Y_{ij}, \hat{X}_{ij}\right)$ is a positive value.

Besides, for robustness against threshold errors, we incorporate all thresholds

into Eq (14):

$$L\left(Y_{ij},\hat{X}_{ij}\right)=\sum_{r=1}^{Y_{ij}-1}h\left(\hat{X}_{ij}-\theta_{ir}\right)+\sum_{r=Y_{ij}}^{L-1}h\left(\theta_{ir}-\hat{X}_{ij}\right)$$
$$=\sum_{r=1}^{L-1}h\left(T_{ij}^{r}\left[r,Y_{ij}\right]\cdot\left(\theta_{ir}-\hat{X}_{ij}\right)\right) \tag{15}$$

where $T_{ij}^{r}\left[r,Y_{ij}\right]$ is defined as:

$$T_{ij}^{r}\left[r,Y_{ij}\right]=\begin{cases}+1 & r\geq Y_{ij}\\ -1 & r<Y_{ij}\end{cases} \tag{16}$$

Although Eq (15) defines the classification errors, finding optimal coordinates matching all observed levels is likely to fail (the overfitting phenomenon), since the level matrix Y is only approximately low-rank. In order to avoid such overfitting situations, combining the error function with an adequate regularization model is a common practice. Therefore, by incorporating the sum of the Frobenius norm of $\vec{u_i}$ and $\vec{v_i}$ as the regularization term for capacity control, we minimize the following regularized objective of Eq (15):

$$J_C=\sum_{i\neq j,i,j\in[1,N]}\sum_{r=1}^{L-1}h\left(T_{ij}^{r}\left[r,Y_{ij}\right]\cdot\left(\theta_{ir}-\left(\vec{u_i}\times\vec{v_j}\right)\right)\right)+$$
$$\frac{\alpha}{2}\left(\sum_{i=1}^{N}\left(\|\vec{u_i}\|_F^2+\|\vec{v_i}\|_F^2\right)\right) \tag{17}$$

where α denotes the regularization constant, and $\|\vec{z_{1:m}}\|_F^2=\sum_{i=1}^{m}z_i^2$ denotes the Frobenius norm.

6.2. Distributed Conjugate Gradient Optimization

We now introduce how to compute the coordinates for each node in a fully decentralized manner. Generally, assume that each node i periodically samples a number of online nodes S_i as neighbors. Node i probes several kinds of network metric values to and from its neighbors with cooperation. Then for each kind of network metric, node i maps the network metric values to discrete levels by Section 5. Each node then predicts unobserved pairwise level values to nodes that are not its neighbors.

The objective (17) needs the complete pairwise level matrix Y, which implies a centralized computation process. Since we prefer a fully distributed computation

process, we decompose the objective J_C into separable objectives J_D^i of each node i and its neighbors S_i:

$$J_D\left(\vec{x_i}\right) = \sum_{j \in S_i} \sum_{r=1}^{L-1} h\left(T_{ij}^r\left[r, Y_{ij}\right] \cdot \left(\theta_{ir} - \left(\vec{u_i} \times \vec{v_j}\right)\right)\right) + \frac{\alpha}{2}\left(\left(\|\vec{u_i}\|_F^2 + \|\vec{v_i}\|_F^2\right)\right) \tag{18}$$

Therefore, Eq (18) can be independently optimized by node i.

We choose the Polak-Ribière variant of the nonlinear conjugate gradient methods (PR-CG) to minimize the objective (18) due to its fast convergence and robustness [44]. For a nonlinear objective function, PR-CG iteratively updates the position of the solution vector $\vec{x}$ according to the conjugate direction of $\vec{x}$ until reaching a local minimum. There are two important parameters for PR-CG:

- **Conjugate direction** Λx. It is a conjugate version of the successive gradients obtained as the progress of the iterations.
- **Movement step** α. It determines how far we move in the conjugate direction. The movement step α is calculated through a line search method [44].

For example, let $\vec{x_i}$ be the concatenation of the coordinate components of i, i.e., $\vec{x_i} = \left[\vec{u_i}; \vec{v_i}; \vec{\theta_i}\right]$. Let $\vec{x_i}(0)$ be a random vector. In a round l ($l \geq 1$), for each node i, PR-CG updates node i's vector $\vec{x_i}(l-1)$ a small step α_i towards $\vec{x_i}(l-1)$'s conjugate direction $\Lambda x_i(l)$:

$$\vec{x_i}(l) \leftarrow \vec{x_i}(l-1) + \alpha_i \Lambda x_i(l) \tag{19}$$

We can see that PR-CG incurs low computation overhead that is comparable to gradient based methods.

Algorithm 2 presents the detailed steps of the distributed coordinate update procedure. Each node periodically updates its coordinate according to Eq (19). The procedure first constructs the vector $\vec{x_i}$ (line 3) and calculates the steepest direction Δ (line $4 \rightarrow 6$) that is the negative gradient of the vector $\vec{x_i}$. Then, line 7 computes the Polak-Ribière scalar β for updating the conjugate gradient. Line 8 updates the new conjugate direction Λ based on the steepest direction Δ and β times of the conjugate direction of last round. Line 9 determines the optimized step length α_i based on the line search method [44]. Then line 10 updates the vector based on the conjugate gradient vector Λ and the step length α_i. Lines

Algorithm 2: HPM Algorithm

1: {Input: node i's current coordinate $\vec{u_i}$, $\vec{v_i}$, $\vec{\theta_i}$, node i's steepest direction Δx_i, node i's conjugate direction Λx_i, the set of neighbors S, the level values Y from node i to its neighbors in S, the coordinates of neighbors in S.}

2: {Output: node i's updated coordinate $\vec{u_i}$, $\vec{v_i}$, $\vec{\theta_i}$, node i's updated steepest direction Δx_i, node i's updated conjugate direction Λx_i}

3: $\vec{x_i} \leftarrow \left[\vec{u_i}; \vec{v_i}; \vec{\theta_i}\right]$;

4:

$$\frac{\partial J_D}{\partial u_{ih}} = \alpha u_{ih} - \sum_{j \in S_i} \sum_{r=1}^{L-1} T_{ij}^r \left[r, Y_{ij}\right] \cdot h' \left(T_{ij}^r \left[r, Y_{ij}\right] \cdot \left(\theta_{ir} - \widehat{X}_{ij}\right)\right) v_{jh} \tag{20}$$

$$\frac{\partial J_D}{\partial v_{ih}} = \alpha v_{ih} - \sum_{j \in S_i} \sum_{r=1}^{L-1} T_{ji}^r \left[r, Y_{ji}\right] \cdot h' \left(T_{ji}^r \left[r, Y_{ji}\right] \cdot \left(\theta_{jr} - \widehat{X}_{ij}\right)\right) u_{jh} \tag{21}$$

$$\frac{\partial J_D}{\partial \theta_{ir}} = \sum_{j \in S_i} T_{ij}^r \left[r, Y_{ij}\right] \cdot h' \left(T_{ij}^r \left[r, Y_{ij}\right] \cdot \left(\theta_{ir} - \widehat{X}_{ij}\right)\right) \tag{22}$$

5:

$$\nabla_x J_D \left(\vec{x_i}\right) = \left[\frac{\partial J_D}{\partial u}; \frac{\partial J_D}{\partial v}; \frac{\partial J_D}{\partial \theta}\right] \tag{23}$$

6: $\Delta \leftarrow -\nabla_x J_D \left(\vec{x_i}\right)$;

7: $\beta \leftarrow \frac{\Delta^T (\Delta - \Delta x_i)}{\Delta x_i^T \Delta x_i}$;

8: $\Lambda \leftarrow \Delta + \beta \Lambda x_i$;

9: $\alpha_i \leftarrow \arg\min_{\alpha_i} J_D \left(\vec{x_i} + \alpha_i \Lambda\right)$;

10: $\vec{x_i} \leftarrow \vec{x_i} + \alpha_i \Lambda$;

11: $\Delta x_i \leftarrow \Delta$;

12: $\Lambda x_i \leftarrow \Lambda$;

13: $\vec{u_i} \leftarrow \vec{x_i}[1 : d]$;

14: $\vec{v_i} \leftarrow \vec{x_i}[(d+1) : 2d]$;

15: $\vec{\theta_i} \leftarrow \vec{x_i}[(2d+1) : (2d+L-1)]$;

$11 \rightarrow 12$ compute the new steepest direction Δx_i and the conjugate direction Λx_i. Finally, lines $13 \rightarrow 15$ update node i's coordinate.

We analyze the storage of each node from Algorithm 2. First, the length of the steepest direction and the conjugate direction both equals that of the vector x. Storing the node's coordinate, its steepest direction and its conjugate direction

Table 2: Default Parameter Configuration for HPM

Parameter	Value
number of levels L	10
regularization constant α	0.3
number of neighbors C_M	32
coordinate dimension d	5
neighbor choice	random
update round	120 rounds

requires $O(2d + L)$ space. Second, each node needs to store recent level samples to neighbors and the coordinates of neighbors. For $|S_i|$ neighbors, the storage becomes $O(|S_i| \times (2d + L))$. Therefore, the overall space overhead of Algorithm 2 is $O(|S_i| \times (2d + L))$.

7. Simulation

In this section, we evaluate HPM's performance with real-world data sets. We address four questions:

- Does HPM predict accurate hierarchical proximity that is consistent with the pairwise proximity of nodes?
- Is HPM sensitive to parameter settings?
- Is HPM robust to missing measurements or coordinate errors?
- Does HPM scale with increasing number of levels or system size?

We use the data sets in Section 3. We repeat the experiments in ten times and compute the average results and the corresponding standard deviations. HPM's default parameters are shown in Table 2.

7.1. Performance Comparison

We first test whether the level numbers predicted by HPM preserve the pairwise proximity of the data sets. For that purpose, we use a well-known metric

Cophenetic Correlation Coefficient (CCC) [45] to quantify the matching degree between the estimated pairwise levels $\hat{Y}$ and the pairwise proximity $\mathbf{D}$:

$$\mathrm{CCC} = \frac{\sum_{(i,j)\in\Omega}\left(\mathbf{D}_{ij}-\bar{\mathbf{D}}\right)\left(\hat{Y}_{ij}-\bar{Y}\right)}{\sqrt{\left[\sum_{(i,j)\in\Omega}\left(\mathbf{D}_{ij}-\bar{\mathbf{D}}\right)^2\right]\left[\sum_{(i,j)\in\Omega}\left(\hat{Y}_{ij}-\bar{Y}\right)^2\right]}} \tag{24}$$

where $\bar{\mathbf{D}} = \frac{1}{\sum_{(i,j)\in\Omega} 1}\sum_{(i,j)\in\Omega}\mathbf{D}_{ij}$ is the average value of the performance matrix $\mathbf{D}$, $\bar{Y} = \frac{1}{\sum_{(i,j)\in\Omega} 1}\sum_{(i,j)\in\Omega}\hat{Y}_{ij}$, is the average level value of the estimated level matrix $\hat{Y}$, and Ω is the set of performance measurements. The CCC takes values between -1 and +1. Higher CCC values mean that the pairwise levels match much closer with the pairwise proximity.

We compare HPM with six related methods:

- **Optimal**. We regard the pairwise levels computed by the *centralized K-means clustering method* as the **Optimal** pairwise levels, since the K-means clustering based results optimally preserve the similarity of the metric values mapped into each level number. Accordingly, HPM tries to estimate the pairwise levels computed by the K-means clustering method.

- **Hierarchical Clustering**. As discussed in the introduction section, the hierarchical clustering method can compute the logical tree that represents multilevel proximity between a set of nodes with respect to any distance metric. Accordingly, we compute the logical tree using the hierarchical clustering method, and treat the pairwise level of their closest common ancestor of two bottom leaf nodes as the estimated levels by the hierarchical clustering method.

- **NonMetric**. The NonMetric method [14] estimates the latency or the available bandwidth based on decentralized coordinates. The coordinates are updated based on the spring field simulation. However, the NonMetric method only estimates continuous distances. For ease of comparison, we compute the logical tree of the estimated distances based on (ii) the hierarchical clustering method.

- **LandmarkMDS**. The LandmarkMDS method [46] estimates the pairwise hops based on the Multidimensional Scaling (MDS) method. Similar to the

NonMetric method, we compute the logical tree of the estimated distances based on the hierarchical clustering method.

- **Vivaldi**. The Vivaldi method [9] estimates the RTTs based on decentralized coordinates that are calculated based on the spring field simulation. Similar to the NonMetric method, we compute the logical tree of the estimated distances based on the hierarchical clustering method.

- **Sequoia**. The Sequoia method [20] constructs the logical tree of participating nodes based on a tree embedding process.

We can see that all methods except Optimal and HPM do not specify the maximum number of allowed levels. For fair comparison, our evaluation consists of two parts depending on whether we limit the number of levels: First, we do not specify the maximum levels. We compute the logical trees for Hierarchical clustering, NonMetric, LandmarkMDS, Vivaldi and Sequoia. Then we compute the Cophenetic Correlation Coefficient values for these logical trees. Second, we limit the maximum levels for all methods. After we construct the logical trees for Hierarchical clustering, NonMetric, LandmarkMDS, Vivaldi and Sequoia, we compute the pairwise levels for these logical trees, then we re-scale the corresponding levels of the tree into the interval whose upper bound is the allowed maximum level. We finally compute the Cophenetic Correlation Coefficient values for the re-scaled levels.

Figure 7 shows the results for the case without limiting the maximum number of levels. The Optimal method has the highest matching degree with the ground-truth pairwise proximity, and HPM has similar accuracy as the Optimal approach. We can see that the Optimal and HPM methods are able to estimate the accurate levels that match the latent structure of the data sets.

However, Optimal and HPM methods have much lower CCC values on the loss data set than those on other data sets. This is because most pairwise loss rates are zeros, which makes the distributed clustering process prone to be trapped into bad local minimum.

On the other hand, the Hierarchical clustering, NonMetric, LandmarkMDS and Vivaldi methods have much smaller CCC values than the Optimal and HPM methods, which implies that the estimated logical trees do not well match the pairwise proximity of the data sets. Furthermore, Optimal and HPM have similar CCC values on all four network metric, but other methods have varying CCC values for different network metric, which means that Optimal and HPM have better generality with respect to different network metric than other methods. For

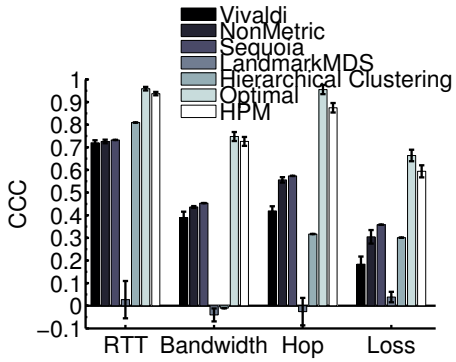


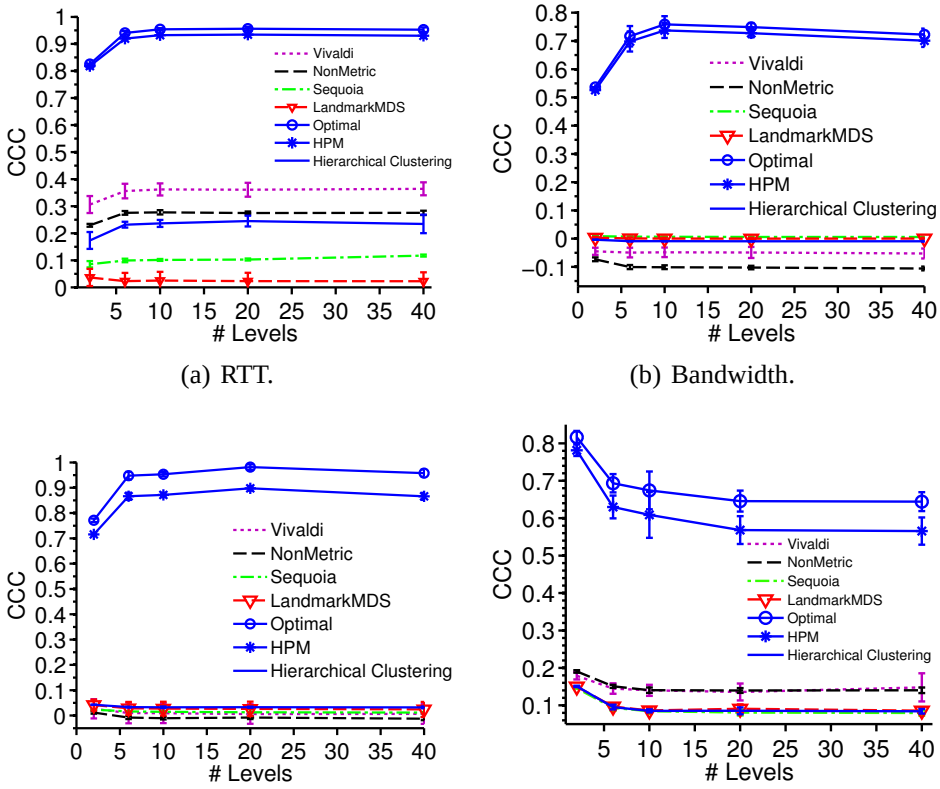
Figure 7: The average values and the corresponding standard deviations of the cophenetic correlation coefficients of different methods when we do not limit the maximum number of levels.

example, the Hierarchical clustering method has a high CCC value on the RTT metric, but has quite low CCC values on the other three metric.

Second, let all methods use the same number of levels, we compare the matching degree between the estimated levels and the pairwise proximity of the data sets. Figure 8 shows the Cophenetic Correlation Coefficients as we vary the maximum number of levels. Similar to Figure 7, the Optimal method has the highest matching degree. HPM has nearly the same accuracy as the Optimal approach. We can see that the Optimal and HPM methods are able to accurately predict the pairwise proximity of the data sets with respect to different number of levels. On the other hand, the Hierarchical clustering, NonMetric, LandmarkMDS, Vivaldi and Sequoia have much lower CCC values than the Optimal and the HPM method, less than 0.4 on average, which implies that the logical trees estimated by these four methods have a high degree of mismatch with respect to the ground-truth pairwise proximity.

7.2. Sensitivity Analysis

We analyze the effects of the parameter choice on the accuracy of HPM. Since HPM incrementally adjusts its coordinate position, we evaluate its convergence and robustness as a function of the number of rounds of coordinate updates increase. Assume that all nodes join the system at time zero, and each node updates its coordinate once per round. Let the default parameter configuration of HPM be defined in Table 2. We compare the differences between the estimated levels of the Optimal method and those of the HPM method based on the Normalized



(a) RTT. (b) Bandwidth.

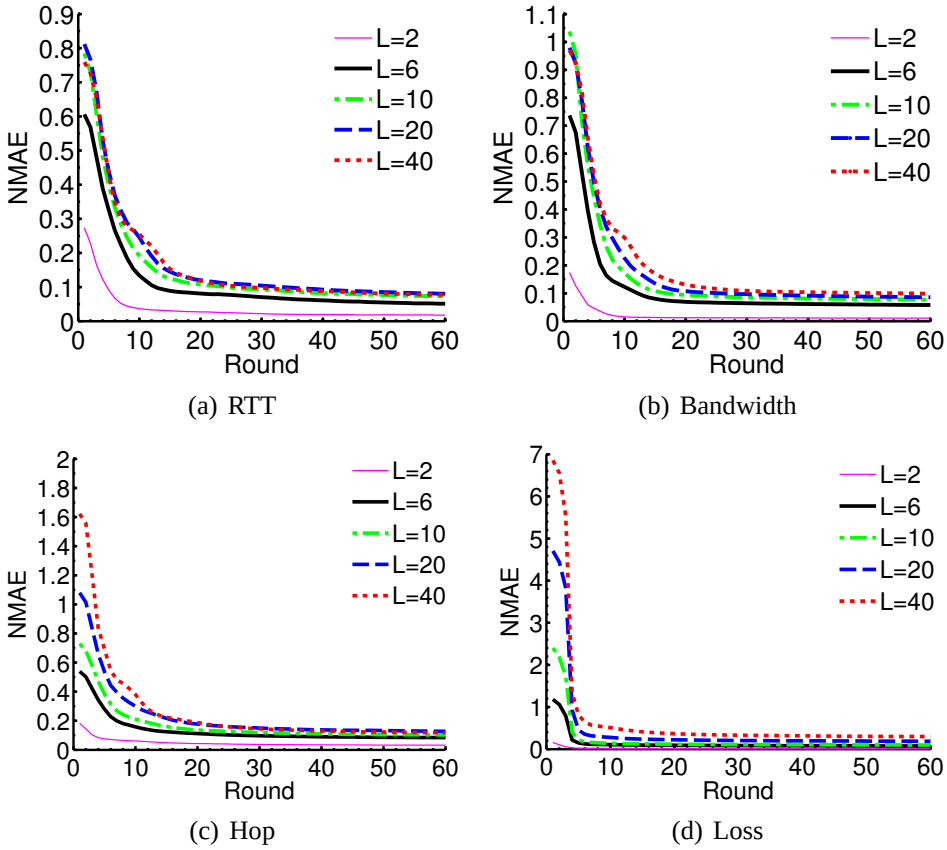
(c) Hop. (d) Loss.

Figure 8: The cophenetic correlation coefficients for various methods with increasing number of levels.

Mean Absolute Error (NMAE):

$$\text{NMAE} = \frac{\sum_{(i,j):Y_{ij}>0} \left| Y_{ij} - \hat{Y}_{ij} \right|}{\sum_{(i,j):Y_{ij}>0} Y_{ij}} \tag{25}$$

where Y_{ij} denotes the level value from i to j by the Optimal method, $\hat{Y}_{ij}$ denotes the estimated level value from i to j by HPM. the NMAE metric can adapt to various performance metric that have different level intervals. Smaller NMAE values correspond to higher prediction accuracy. We report the averaged results that are based on ten repeated simulations.



(a) RTT (b) Bandwidth

(c) Hop (d) Loss

Figure 9: The rate of convergence of HPM as we vary the number of levels.

7.2.1. Number of levels

We first test HPM's accuracy as a function of the number of levels. Figure 9 shows the convergence for different number of levels. HPM converges in about 20 rounds, then remains accurate afterwards. We see that HPM converges fast and that varying the number of levels does not affect the convergence speed of HPM. Therefore, HPM can achieve very good accuracy independent of the number of levels.

7.2.2. Size of Neighbors

We next evaluate the accuracy of HPM as a function of the number of neighbors. Figure 10 shows the results. We see that increasing the number of neighbors generally increases the accuracy of HPM. But the performance improvements become negligible when the number of neighbors exceeds 16. Therefore, a moderate

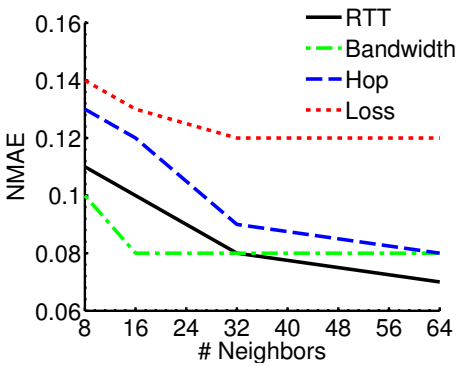


Figure 10: The effectiveness of the number of neighbors.

number of neighbors is enough for accurate level estimations.

7.2.3. Neighbor Selection Choice

We test whether the choices of neighbors affect HPM's accuracy. We use four kinds of neighbors:

- **Random**. We choose neighbors uniformly at random from the whole set of nodes.
- **Closest**. We choose neighbors that have lowest RTT, loss, hops or highest bandwidth.
- **Farthest**. We choose neighbors that have highest RTT, loss, hops or lowest bandwidth.
- **Hybrid**. We select half neighbors using the Closest based selection and the other half neighbors using the Farthest based selection.

Figure 11 shows that the Random based neighbor selection policy achieves the highest accuracy, compared to the other three policies. As a result, we can randomly choose neighbors for level estimations, which can be implemented easily.

7.2.4. Dimensionality

We next evaluate the effectiveness of the coordinate dimension on the accuracy of HPM. Figure 12 plots the simulation results with varying coordinate dimensions. The results show that low coordinate dimensions are enough for converging to accurate level predictions. Increasing the coordinate dimensions does not significantly increase the accuracy of level predictions.

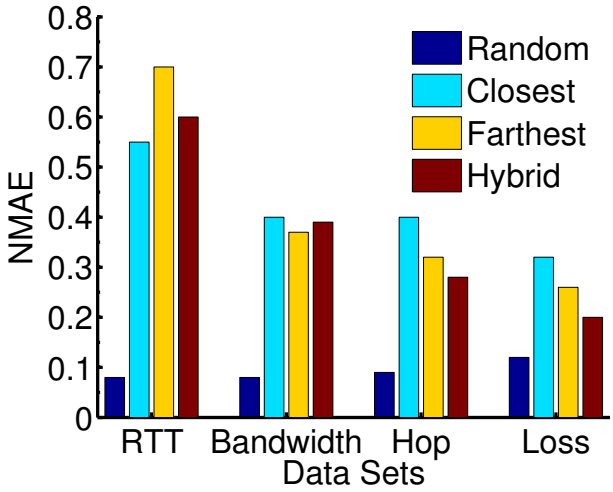


Figure 11: The effect of neighbor selection policies on rate of converge.

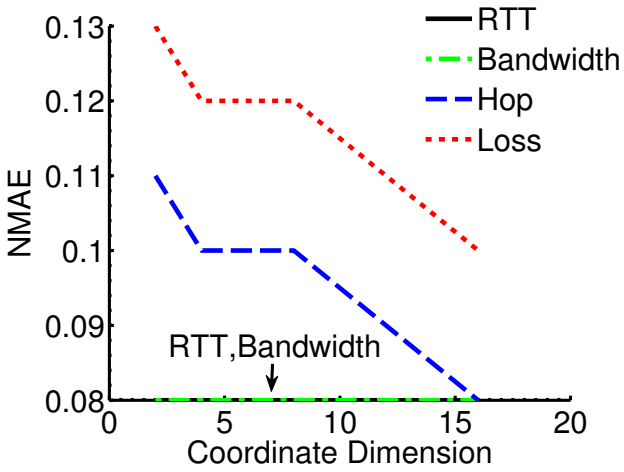


Figure 12: The effect of coordinate dimension on the rate of convergence. We vary the coordinate dimension ranging from 2 to 16 on all data sets.

7.2.5. Regularization constant α

We finally test whether the regularization parameter α in the loss function of Eq (18) affects HPM's accuracy. When varying α from 0.1 to 2, we did not see any significant effect on the accuracy. Therefore, HPM is quite robust against the choice of the regularization parameter α.

7.3. Robustness

We test the robustness of HPM in this section.

7.3.1. High Erroneous Nodes

We first test whether HPM is sensitive erroneous coordinates. We divide the overall set of nodes in the data sets into two equal halves, only half of the nodes join the system at time 0 and the other half nodes join the system after 40 rounds. As a result, the erroneous coordinates are injected into the system after 40 rounds. To quantify the stability of the coordinate of each node i, we calculate the **Coordinate Drift** using the l_1 norm defined as

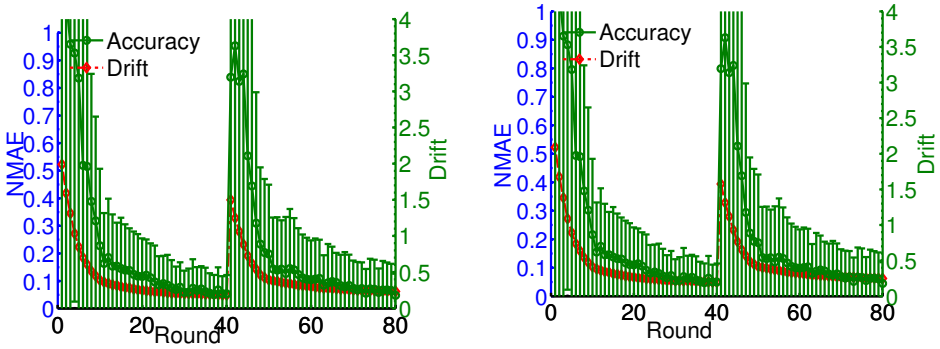
$$\sum\nolimits_{m=1}^{2d+L-1} \left| x_i\left(m\right) - \tilde{x_i}\left(m\right) \right| \tag{26}$$

where $\vec{x_i} = \left[\vec{u_i}; \vec{v_i}; \vec{\theta_i}\right]$ denotes the updated coordinate, and $\tilde{x_i}$ denotes the previous coordinate. We plot the accuracy of level estimations and coordinate drifts in Figure 13.

We can see that the first half nodes converges to stable coordinates within 20 rounds and keep steady until 40 rounds. Furthermore, the coordinate drifts decrease close to zeros after the coordinates are stabilized after 20 rounds. When the other half nodes join the system after 40 rounds, the overall coordinate errors increase sharply after 40 rounds, since the coordinates of newly-joined nodes are randomly initialized and incur high errors. Accordingly, the coordinate drifts also increase. However, the whole set of coordinates converge within the next 20 rounds to stable positions. The newly stabilized coordinates have the similar accuracy as those before 40 rounds and the coordinate drifts also decrease to the similar degrees as those before 40 rounds.

7.3.2. Missing Measurements

We next test HPM's performance when some level measurements to neighbors are unavailable due to node failures or routing disruptions. In each round, for each node, we choose uniformly at random a fraction of neighbors that do not respond



(a) RTT. (b) Bandwidth.

(c) Hop. (d) Loss.

Figure 13: The effect of high-error coordinates on the rate of convergence. The vertical line after 60 rounds indicates the joining event of new nodes.

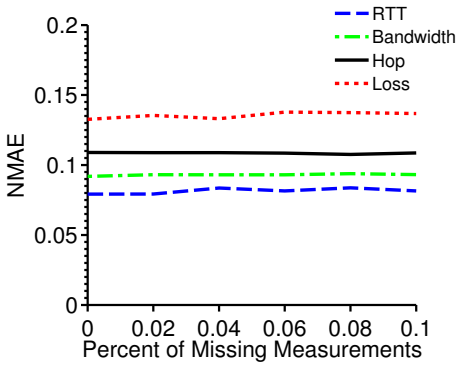


Figure 14: HPM's accuracy as a function of the percent of missing measurements to neighbors.

the level measurements. We than collect the final performance statistics of all nodes. Figure 14 plots the results. We can see that HPM is robust against missing measurements: the estimation accuracy degrades slightly with increasing percent of missing measurements. This is because the conjugate gradient optimization method is quite robust to incomplete information.

7.4. Scalability

In this section we show the scalability of HPM.

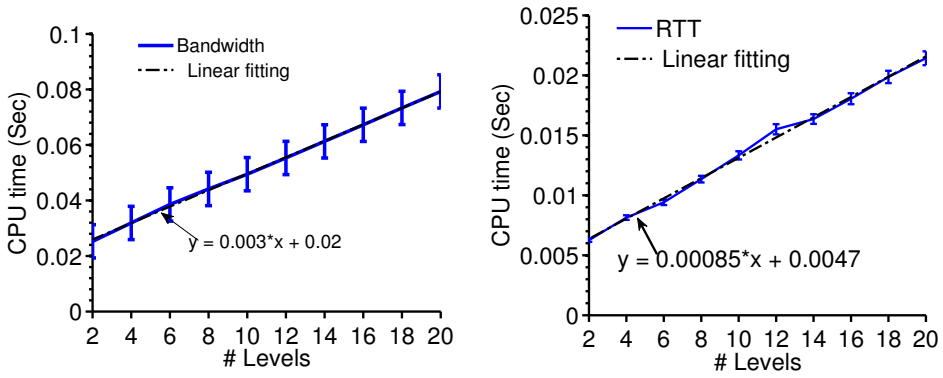
7.4.1. CPU Efficiency

We first test the efficiency of HPM as a function of the number of levels. We compute the CPU time of completing one round of distributed K-means clustering algorithm and that of finishing one round of coordinate update. From Figure 15 we can see that with increasing number of levels, the CPU time of the distributed K-means clustering and the coordinate update increase almost linearly. However, the slope of the fitted line is quite modest. As a result, HPM scales well with increasing number of levels.

Besides, we also evaluate the CPU efficiency of HPM in terms of the number of neighbors or the coordinate dimension. We found that the CPU time is quite stable as we increase the number of neighbors or the coordinate dimensions.

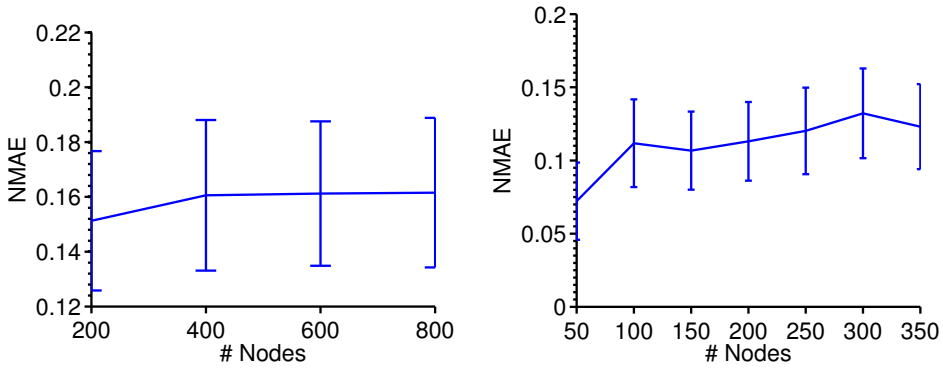
7.4.2. Accuracy With Increasing System Size

Our previous evaluation uses a fixed number of nodes. In this section, we test HPM's accuracy with increasing number of participating nodes. We choose the RTT and bandwidth metric as examples. Due to the limited size of the RTT data set, we use a larger RTT data set from the Meridian project [47].



(a) Distributed K-means clustering (b) Coordinate update

Figure 15: The CPU time as a function of the number of levels on the bandwidth data set.



(a) Meridian RTT data set (b) Bandwidth data set

Figure 16: HPM's accuracy as a function of the size of nodes.

From Figure 16, we can see that there are clearly phase changes with increasing number of nodes on the RTT and bandwidth data sets. The change points of different phases for RTT and bandwidth metric occur when the system size is much smaller than the overall size of the data sets. At the beginning, HPM incurs higher estimation errors with increasing number of nodes; but soon HPM keeps to be stably accurate. This is because smaller system size means narrower search spaces for the optimization problem, which implies that the solution is much closer to the global optimum [38].

8. PlanetLab Evaluation

8.1. Implementation

We have implemented HPM in Java. The main logic consists of approximately 4000 lines, including the **neighborhood management** component, the **performance measurement** component, the **layer mapping** component, the **layer estimation** component, as shown in Figure 17. Each node periodically triggers the performance measurement component to probe performance metric towards available neighbors, then calls the layer mapping component to calculate discrete levels with updated clustering centroids, and finally updates its coordinate based on the layer estimation component. Besides, each node can request the coordinates of any pairs of nodes based on the XML RPC interface and compute the corresponding two-direction levels based on the requested coordinates.

Our prototype measures the RTT and loss metric simultaneously. Each node A periodically triggers a measurement event, which sends a sequence of probe packets to a randomly selected neighbor B, and returns immediately in an asynchronous manner. Accordingly, the receiver B echoes the sender with an acknowledge packet as soon as it receives a measurement packet. Finally node A computes the RTT and loss metric as:

- *RTT*. The round trip time is calculated by averaging the period of each pair of a measurement packet and the corresponding acknowledge packet, $(\sum_{i=1}^{L_{success}} T_i)/L_{success}$, where $L_{success}$ is the number of successful pairs of measurement and acknowledge packets, T_i is the time period of the i-th pair of measurement and acknowledge packets.

- *Loss Rate*. The loss rate is calculated by the ratio between the number of acknowledge packets to the total number of measurement packets, $\left(1 - \frac{L_{success}}{L_{total}}\right)$, where L_{total} is the overall number of measurement packets.

Furthermore, as the performance measurements to neighbors keep changing due to dynamic network conditions, we use the exponential moving average filter with a coefficient at 0.05 to smooth out short-term fluctuations. After completing a performance measurement to a neighbor, node A updates its coordinate vector based on available performance samples of all neighbors, in order to optimize the convergence of its coordinate vector.

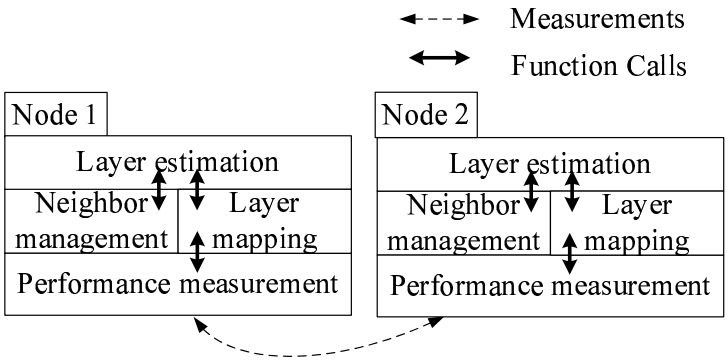


Figure 17: The HPM architecture.

8.2. PlanetLab Experiments

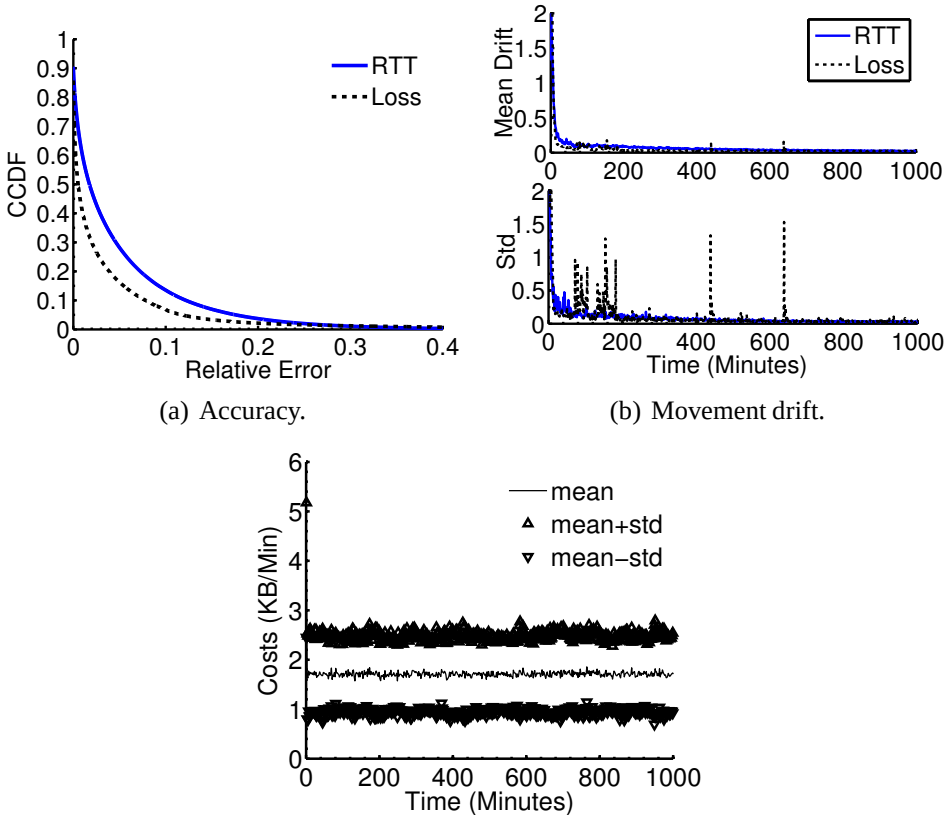
We selected 269 physical machines on PlanetLab and installed the HPM program. We update the cluster centroids and issue the performance measurement based on the gossip communication. The inter-gossip interval T_g is 30 seconds. Accordingly, the coordinates are updated approximately per 30 seconds. Each node independently maintains two coordinates based on HPM, one coordinate for the RTT metric and the other one for the loss metric. We choose other parameters for HPM as those shown in Table 2 in the Simulation section.

The evaluation metric for the prototype include:

- **Relative Error**, Each node i calculates the relative error with each neighbor j between the estimated level and the ground-truth level as $\frac{|Y_{ij}-\hat{Y}_{ij}|}{Y_{ij}}$, where Y_{ij} denotes the level number computed by the distributed K-means clustering method and $\hat{Y}_{ij}$ denotes the estimated level number by Algorithm 2. Then each node i updates the relative error towards the neighbor j when node i's coordinate changes or node i obtains new performance measurements to node j. The updating rule is based on the exponentially moving average with a coefficient at 0.05.

- **Coordinate Drift**, We calculate the coordinate drift per minute based on Eq (26) in order to quantify the stability of each node's coordinate.

- **Bandwidth Costs**, We collect the bandwidth costs of HPM per minute, including the gossip messages and measurement costs incurred by the packet trains.



(a) Accuracy.

(b) Movement drift.

(c) Bandwidth costs.

Figure 18: Performance statistics of HPM on PlanetLab. Std denotes the standard deviation.

Relative Error: Figure 18(a) plots the CCDF of the relative errors of level estimations by HPM. For the RTT based level estimation, HPM incurs low estimation errors for each node pair, where the maximum relative error is around 0.4. For the loss based level estimation, HPM is even more accurate and predicts accurate level values for most node pairs, where in around 95% of the cases of the median relative errors are below 0.1. Therefore, HPM predicts representative metric quite accurately.

Coordinate Drifts: We next plot the dynamics of the coordinate drifts in Figure 18(b). The results show that the coordinate drifts are relatively high at the bootstrap phase, with mean and standard deviation at around two. This is because the initial coordinates move at large steps to converge to accurate positions. On the other hand, after ten minutes, the coordinates keep to be stable with close-to-

zero coordinate drifts. We can see that ten minutes correspond to twenty rounds of coordinate updates, since the coordinate update interval is around 30 seconds. The convergence speed of the PlanetLab deployment is consistent with that in the simulation results.

Bandwidth Costs: Figure 18(c) depicts the dynamics of the system overhead. The mean control overhead stay around 1.8 KB per minute during the experiment period, which is quite modest. However, the standard deviations of the control overhead at the beginning are relatively large, since each node needs to contact multiple potential neighbors returned from the bootstrap node. After the initial contact process, the overhead becomes steadily low.

Summary of Results: We confirm that HPM can converge to stable positions with accurate level predictions at low bandwidth costs. Furthermore, HPM can still estimate levels very accurately for skewed network metric such as the loss rates where most of them are zeros.

9. Application

In this section, we illustrate several service provision that can benefit from HPM in the context of the Nano Data Center (NaDa) [6], which is a kind of hybrid clouds that uses Nano data centers to reduce the energy consumption of traditional data centers. NaDa comprises geographically distributed in-house gateways. NaDa can allow ISPs to host Internet applications and content on residential gateways to reduce the access time for end hosts.

We assume that each node in NaDa computes its coordinate using HPM, and that each node learns the coordinates of all nodes in the system through a coordinate propagation scheme such as the anti-entropy gossiping procedure. Since each coordinate requires $(2d+L)$ space, storing the coordinates of N nodes incurs $O\left(N{\cdot}(2d+L)\right)$ space, which is quite modest.

9.1. K nearest neighbor search

The K nearest neighbor search aims to find K nano servers having the lowest delays or the highest bandwidth to the target. K nearest neighbor search helps optimize the streaming applications and the content backup service, since redirecting host requests to nearest nano servers can reduce the delays and increase the transmission throughput. Furthermore, locating K nearby nano servers ($K \geq 1$) can be used for parallel connections in the content backup service to avoid the performance bottlenecks of some nano servers.

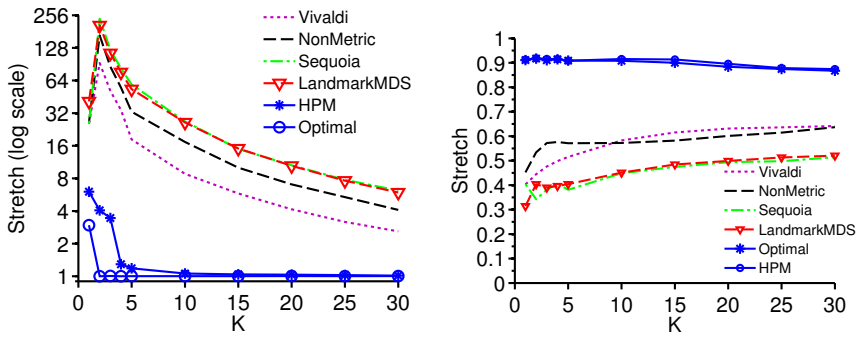
We simulate K nearest neighbor search using HPM as follows: first, we randomly choose a target from the whole set of nodes, and configure the other nodes as nano servers; second, we randomly choose nano servers that have the smallest levels to the target to be the closest nodes to the target, since lower levels correspond to lower delays or higher bandwidth. For indirect methods including Vivaldi, NonMetric, Sequoia and LandmarkMDS, we determine the closest nano servers to the target using estimated delays or bandwidth. Besides, in order to quantify the effectiveness of the K nearest neighbor search, we calculated the **stretch of found closest nodes** for each target i as $\frac{\sum_{j \in \widehat{C}_i} \mathbf{D}_{ij}}{\sum_{j \in C_i} \mathbf{D}_{ij}}$, where $\widehat{C}_i$ denotes the closest nodes found by HPM, and C_i represents the ground-truth closest nodes. The ground-truth closest nodes correspond to nodes that have the lowest delays or the highest bandwidth to the target.

According to the definition of the stretch, for RTT, the stretch is ≥ 1, the lower the stretch, the better HPM performs; on the other hand, for the bandwidth metric, the stretch is ≤ 1, the closer to one the better HPM performs. Figure 19 shows the evaluation of mean stretch with increasing number K of required nano nodes. The Optimal method and HPM significantly improve the stretch compared to other indirect methods including Vivaldi, NonMetric, Sequoia and LandmarkMDS. Moreover, HPM has similar stretch as the Optimal method for the bandwidth metric and the delay metric (when K exceeds 5), implying that HPM can find closest nodes that are as good as those using the Centralized method. On the other hand, indirect methods including Vivaldi, NonMetric, Sequoia and LandmarkMDS have much worse stretch than Optimal and HPM on the delay and bandwidth data sets.

For the delay metric, the stretch values of Optimal and HPM decrease from around 3 and around 6 to nearly 1 as K reaches 2 and 5, respectively, and stay close to 1 afterwards. This means that the use of Optimal and HPM may select inaccurate closest nodes when K is low, since the level values are only coarse-grained proximity metric. On the other hand, Optimal and HPM become very accurate with increasing K. On the other hand, for the bandwidth metric, Optimal and HPM have a stretch close to 1, which decreases slightly with increasing K, indicating that HPM could find high bandwidth nodes, but may miss some of the best ones.

9.2. Proximity-aware matchmaking

Proximity-aware matchmaking finds nodes that have the best proximity [10], by locating hosts that have **lowest pairwise** delays or highest bandwidth to each other. The proximity-aware matchmaking is useful for finding groups of nodes in



(a) RTT. (b) Bandwidth.

Figure 19: The stretch of K nearest neighbor search with varying number of required nodes.

networked games in order to increase the responsiveness between game levels.

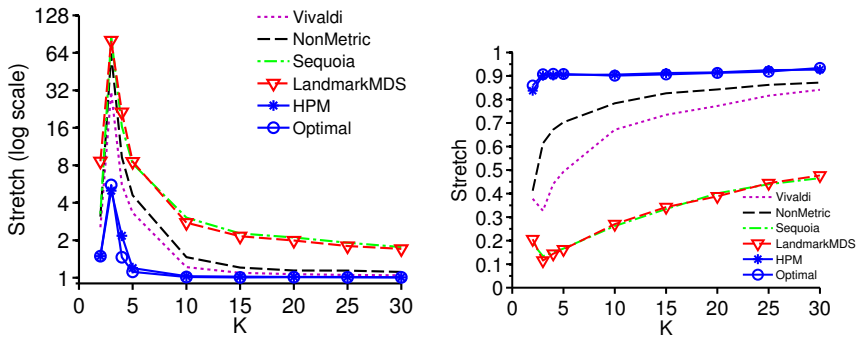
The proximity-aware matchmaking differs from the K nearest neighbor search, since now we need to select nodes for each node i such that *the selected nodes and node i have the lowest delays or the highest bandwidth to each other*, however, in the K nearest neighbor search we only need to find $K-1$ nodes have the lowest delays or the highest bandwidth to node i.

Furthermore, in order to quantify the network quality of matchmaking nodes, we redefine the **stretch of the matchmaking** as the ratio between the averaged delays or bandwidth of the found nodes and those of the ground-truth closest nodes, i.e., $\frac{\sum_{j\neq k,j,k\in\widehat{C}_i}\mathbf{D}_{jk}}{\sum_{j\neq k,j,k\in C_i}\mathbf{D}_{jk}}$, where $\widehat{C}_i$ denotes the set of found nodes plus i, and C_i denotes the set of ground-truth nodes plus i.

To simulate the proximity-aware matchmaking for K nodes using HPM (indirect methods including Vivaldi, NonMetric, Sequoia and LandmarkMDS) with low computation overhead, for each node i, we locate $(K-1)$ nodes that minimize the sum of pairwise levels of these K nodes (minimize the sum of pairwise delays or maximize the sum of pairwise bandwidth for indirect methods) through $Nlog\,(N)$ randomized combinations of nodes, where ties are broken arbitrarily.

Figure 20 shows the mean stretch of matchmaking as a function of the number of required nodes. Optimal and HPM can find nodes that have much higher proximity with each other than those found by Vivaldi, NonMetric, Sequoia and LandmarkMDS. Furthermore, HPM has nearly identical stretch as the Optimal method.

For the delay metric, HPM and Optimal increase the stretch when K reaches 3, and decrease the stretch to close to 1 afterwards, implying that the level based



(a) RTT. (b) Bandwidth.

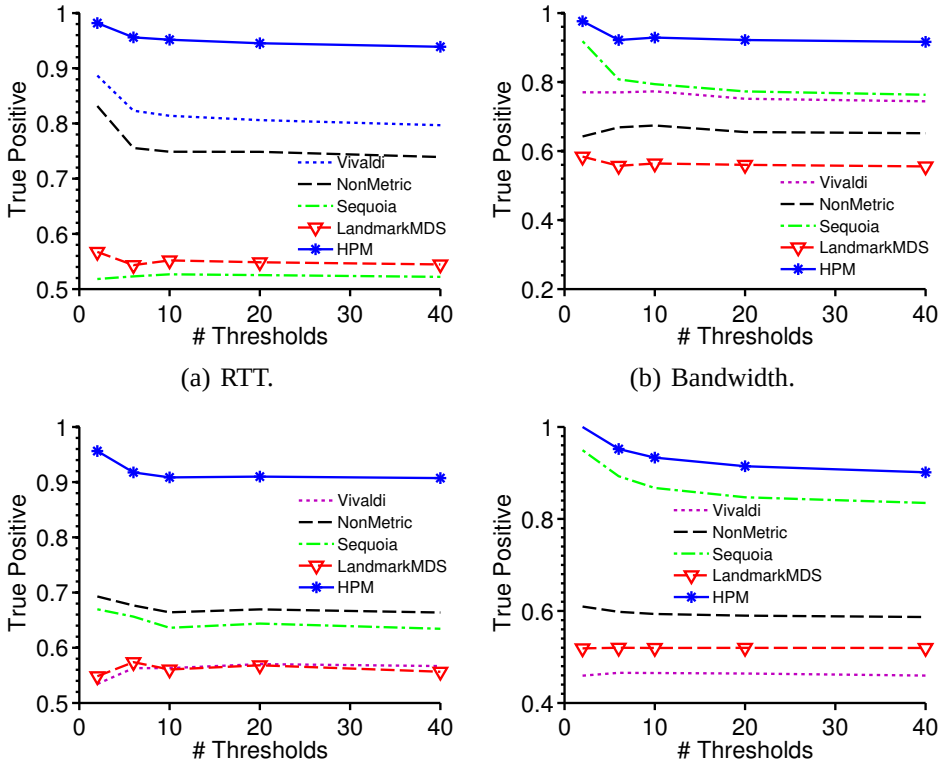
Figure 20: The stretch of proximity-aware matchmaking with increasing number of required nodes.

matchmaking produces less accurate results when participating nodes are very few. However, in an interactive game, the number of players is typically around ten, in which case HPM will do a very good job in match-making. On the other hand, for the bandwidth metric, HPM and Optimal improve the stretch from 0.85 to 0.9 when the number of nodes reaches 4, and stay around 0.9 afterwards, indicating that HPM and Optimal increase the matchmaking accuracy with increasing number of participating nodes.

9.3. Network performance anomaly detection

Finding the occurrence of performance anomaly (such as high delay or loss events) becomes increasingly critical for network infrastructures. Using thresholds to detect anomalous and nearly anomalous network performance is a popular anomaly detection approach [48], which determine whether the performance measurements violate the thresholds. Accordingly, the anomaly detection aims to find all node pairs whose end to end performance measurements are above (or below for the bandwidth) these anomaly thresholds.

Similar as Barford et al. [48], we explicitly introduce performance thresholds as a set of percentile values of the measurement distribution that separate equally with each other. To simulate the threshold based anomaly detection using HPM, we use the anomaly thresholds as additional level separation points and a performance measurement mapped to one of these level separation points is regarded as an anomaly. For indirect methods including Vivaldi, NonMetric, Sequoia and LandmarkMDS, we compare the estimated delays or bandwidth with the thresholds to detect anomalies. For comparison, we vary the number of anomaly thresh-



(a) RTT. (b) Bandwidth.

(c) Hop. (d) Loss.

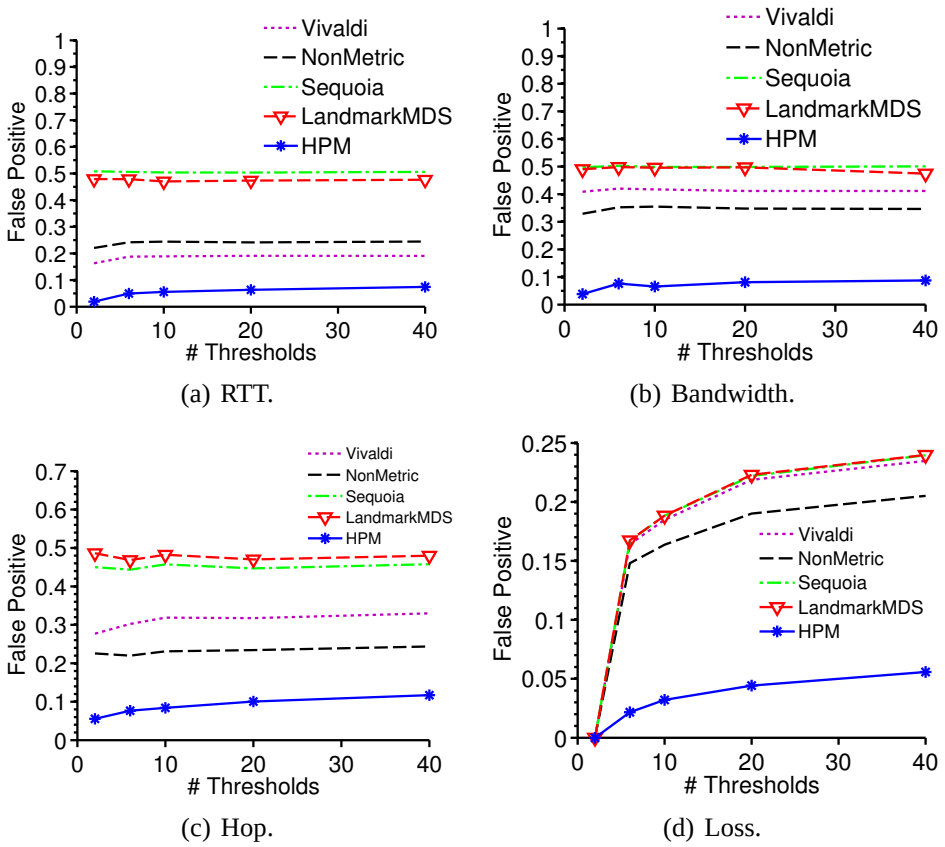
Figure 21: True positive statistics.

olds, and compute the mean percentage of anomaly that is detected successfully (**true positive**) and the percentage of estimated anomaly that do not belong to the true anomaly (**false positive**).

Figure 21 and 22 depict the true positive and false positive statistics by varying the number of anomaly performance thresholds. The Optimal approach has the highest detection precision. HPM has slightly lower anomaly detection accuracy than the Optimal method due to the low-dimensional approximations. Furthermore, HPM has higher detection accuracy than Vivaldi, NonMetric, Sequoia and LandmarkMDS.

10. Conclusion

Hybrid cloud computing provides promisingly elastic, flexible and secure service provision for diverse cloud services. Due to the geographically distributed



(a) RTT. (b) Bandwidth.

(c) Hop. (d) Loss.

Figure 22: False positive statistics.

and heterogeneous participating nodes, metering the network conditions between nodes is increasingly important for optimizing the performance of service provision. However, the large scale and dynamic nature of hybrid-cloud nodes cause challenges for the measurement process. HPM solves this challenging problem in a scalable and decentralized manner. It offers a powerful primitive: given any performance metric, it constructs a hierarchical structure with tunable levels of proximity, and does so scalably and accurately. In order to preserve the asymmetry in the hierarchy, we propose a distributed K-means clustering method [43] based level mapping method that maps performance measurements into levels that are separable for dissimilar measurements and coherent for similar ones. Next, in order to reduce the performance measurement overhead of level mappings for all node pairs, each node measures the level values to a small number of nodes, then

maintains a low-dimensional coordinate with these level measurements by a novel distributed conjugate gradient optimization scheme, and uses the coordinate distances to extrapolate the level values to other nodes.

Simulation results and PlanetLab experiments confirm that HPM can achieve close to optimal performance, and is quite robust with respect to the choice of the parameter values. Furthermore, we show how to use HPM in the context of a novel Nano data center architecture [6].

11. Acknowledgements

This work was supported by the National Grand Fundamental Research 973 Program of China (Grant No.2011CB302601), the National High Technology Research and Development 863 Program of China (Grant No.2013AA010206), the National Natural Science Foundation of China (Grant No.60873215), the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (Grant No.S2010J5050), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No.200899980003) and the Collaborative Project FIGARO supported by the European Commission under the 7th Framework Program (Grant No. 258378). We thank Xingkong Ma for helpful discussions.

References

[1] LACIE, Wuala Secure Online Storage, `http://www.wuala.com/`, 2011.

[2] Microsoft, Halo game, `http://halo.xbox.com/en-us`, 2011.

[3] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, R. Buyya, The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds, Future Gener. Comput. Syst. 28 (2012) 861–870.

[4] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, R. Buyya, Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka, Future Gener. Comput. Syst. 28 (2012) 58–65.

[5] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, Z. Zhang, Moon: Mapreduce on opportunistic environments, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, ACM, New York, NY, USA, 2010, pp. 95–106.

[6] N. Laoutaris, P. Rodriguez, L. Massoulie, ECHOS: Edge Capacity Hosting Overlays of Nano Data Centers, SIGCOMM Comput. Commun. Rev. 38 (2008) 51–54.

[7] G. Mateescu, W. Gentzsch, C. J. Ribbens, Hybrid computing-where hpc meets grid and cloud computing, Future Gener. Comput. Syst. 27 (2011) 440–453.

[8] T. S. E. Ng, H. Zhang, Predicting Internet Network Distance with Coordinates-Based Approaches, in: Proc. of IEEE INFOCOM 2002.

[9] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: a Decentralized Network Coordinate System, in: Proc. of ACM SIGCOMM 2004, pp. 15–26.

[10] S. Agarwal, J. R. Lorch, Matchmaking for Online Games and Other Latency-sensitive P2P Systems, in: Proc. of ACM SIGCOMM 2009, pp. 315–326.

[11] D. Milic, Milic, T. Braun, Braun, NetICE9: A stable landmark-less network positioning system, in: Proc. of LCN '10, pp. 96–103.

[12] Y. Liao, P. Geurts, G. Leduc, Network Distance Prediction Based on Decentralized Matrix Factorization, in: M. Crovella, L. Feeney, D. Rubenstein, S. Raghavan (Eds.), NETWORKING 2010, volume 6091 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2010, pp. 15–26.

[13] Y. Chen, X. Wang, C. Shi, E. K. Lua, X. Fu, B. Deng, X. Li, Phoenix: A weight-based network coordinate system using matrix factorization, IEEE Transactions on Network and Service Management 8 (2011) 334–347.

[14] P. B. Key, L. Massoulié, D.-C. Tomozei, Non-Metric Coordinates for Predicting Network Proximity, in: Proc. of IEEE INFOCOM, pp. 1840–1848.

[15] O. Beaumont, L. Eyraud-Dubois, Y. J. Won, Using the Last-mile Model as a Distributed Scheme for Available Bandwidth Prediction, in: Proc. of Euro-Par'11, pp. 103–116.

[16] J. R. Douceur, J. Mickens, T. Moscibroda, D. Panigrahi, Collaborative Measurements of Upload Speeds in P2P Systems, in: Proc. of INFOCOM 2010.

[17] C. Xing, M. Chen, L. Yang, Predicting Available Bandwidth of Internet Path with Ultra Metric Space-based Approaches, in: Proc. of GLOBECOM'09, pp. 584–589.

[18] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. E. Anderson, A. Krishnamurthy, A. Venkataramani, iPlane: An Information Plane for Distributed Services, in: Proc. of USENIX OSDI 2006, pp. 367–380.

[19] H. V. Madhyastha, E. Katz-Bassett, T. E. Anderson, A. Krishnamurthy, A. Venkataramani, iPlane Nano: Path Prediction for Peer-to-Peer Applications, in: Proc. of USENIX NSDI 2009, pp. 137–152.

[20] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, A. Akella, On the Treeness of Internet Latency and Bandwidth, in: Proc. of ACM SIGMETRICS 2009, pp. 61–72.

[21] S. Song, P. J. Keleher, B. Bhattacharjee, A. Sussman, Decentralized, accurate, and low-cost network bandwidth prediction, in: INFOCOM, pp. 6–10.

[22] Y. Chen, D. Bindel, H. H. Song, R. H. Katz, Algebra-based Scalable Overlay Network Monitoring: Algorithms, Evaluation, and Applications, IEEE/ACM Trans. Netw. 15 (2007) 1084–1097.

[23] M. Coates, Y. Pointurier, M. Rabbat, Compressed Network Monitoring for IP and All-Optical Networks, in: Proc. of ACM IMC 2009, pp. 241–252.

[24] S. Qazi, T. Moors, On the impact of routing matrix inconsistencies on statistical path monitoring in overlay networks, Comput. Netw. 54 (2010) 1554–1572.

[25] O. Beaumont, N. Bonichon, P. Duchon, H. Larchevêque, Use of Internet Embedding Tools for Heterogeneous Resources Aggregation, in: IEEE (Ed.), Heterogeneity in Computing Workshop (HCW) - in IPDPS 2011, Anchorage, États-Unis, pp. 114–124.

[26] H. Eom, D. I. Wolinsky, R. J. O. Figueiredo, SOLARE: Self-Organizing Latency-Aware Resource Ensemble, in: HPCC, pp. 229–236.

[27] S. Malik, F. Huet, D. Caromel, Latency Based Dynamic Grouping Aware Cloud Scheduling, in: Proc. of WAINA '12, pp. 1190–1195.

[28] R. Xu, I. Wunsch, D., Survey of Clustering Algorithms, IEEE Transactions on Neural Networks 16 (2005) 645 –678.

[29] S. Banerjee, C. Kommareddy, B. Bhattacharjee, Scalable Peer Finding on the Internet, in: In Global Internet Symposium.

[30] S. Wieser, L. Böszörményi, Decentralized topology aggregation for qos estimation in large overlay networks, in: Proc. of NCA, pp. 298–301.

[31] H. Balakrishnan, R. H. Katz, V. N. Padmanbhan, The Effects of Asymmetry on TCP Performance, Mob. Netw. Appl. 4 (1999) 219–241.

[32] Y. He, M. Faloutsos, S. Krishnamurthy, B. Huffaker, On routing asymmetry in the internet, in: Proc. of IEEE GLOBECOM 2005, volume 2.

[33] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, Z. M. Mao, A Measurement Study of Internet Delay Asymmetry, in: Proc. of PAM 2008, pp. 182–191.

[34] P. Sharma, Z. Xu, S. Banerjee, S.-J. Lee, Estimating Network Proximity and Latency, Computer Communication Review 36 (2006) 39–50.

[35] A.-J. Su, D. Choffnes, F. E. Bustamante, A. Kuzmanovic, Relative Network Positioning via CDN Redirections, in: Proc. of ICDCS '08, pp. 377–386.

[36] H. Shen, K. Hwang, Locality-Preserving Clustering and Discovery of Resources in Wide-Area Distributed Computational Grids, IEEE Trans. Computers 61 (2012) 458–473.

[37] Y. Liao, W. Du, P. Geurts, G. Leduc, Decentralized prediction of end-to-end network performance classes, in: Proc. of CoNEXT '11, pp. 14:1–14:12.

[38] J. D. M. Rennie, N. Srebro, Fast Maximum Margin Matrix Factorization for Collaborative Prediction, in: Proc. of ICML 2005, pp. 713–719.

[39] M. Weimer, A. Karatzoglou, A. Smola, Improving Maximum Margin Matrix Factorization, Mach. Learn. 72 (2008) 263–276.

[40] J. Stribling, All Pairs of Ping Data for PlanetLab, `http://pdos.csail.mit.edu/~strib/pl_app`, 2005.

[41] S.-J. Lee, P. Sharma, S. Banerjee, S. Basu, R. Fonseca, Measuring Bandwidth Between PlanetLab Nodes, in: Proc. of PAM 2005, pp. 292–305.

[42] Y. A. Wang, C. Huang, J. Li, K. W. Ross, Queen: Estimating Packet Loss Rate between Arbitrary Internet Hosts, in: Proc. of PAM 2009, pp. 57–66.

[43] S. Datta, C. Giannella, H. Kargupta, Approximate Distributed K-Means Clustering over a Peer-to-Peer Network, IEEE Trans. Knowl. Data Eng. 21 (2009) 1372 –1388.

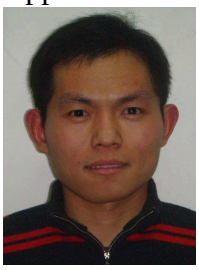
[44] J. R. Shewchuk, An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, Technical Report, Pittsburgh, PA, USA, 1994. http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail\&id=oai\%3Ancstrlh\%3Acmucs\%3ACMU\%2F\%2FCS-94-125.

[45] R. R. Sokal, F. J. Rohlf, The Comparison of Dendrograms by Objective Methods, Taxon 11 (1962).

[46] B. Eriksson, P. Barford, R. D. Nowak, Estimating Hop Distance Between Arbitrary Host Pairs, in: Proc. of IEEE INFOCOM, pp. 801–809.

[47] B. Wong, A. Slivkins, E. G. Sirer, Meridian: a Lightweight Network Location Service Without Virtual Coordinates, in: Proc. of SIGCOMM'05, pp. 85–96.

[48] P. Barford, N. Duffield, A. Ron, J. Sommers, Network Performance Anomaly Detection and Localization, in: Proc. of IEEE INFOCOM 2009, pp. 1377 –1385.



Yongquan Fu received the B.S. degree in computer science and technology from the School of Computer of Shandong University, China, in 2005, and received the M.S. in Computer Science and technology in the School of Computer Science of National University of Defense Technology, China, in 2008. He is currently a Ph.D. candidate in the School of Computer Science of National University of Defense Technology. He is a student member of CCF and ACM. His current research interests lie in the areas of network measurement, Peer-to-Peer network and distributed system.