Self-Stabilized Distributed Network Distance Prediction

Yongquan Fu and Xu Xiaoping

***Abstract*— The network distance service obtains the network latency among large-scale nodes. With increasing numbers of participating nodes, the network distance service has to balance the accuracy and the scalability. The network-coordinate methods scale well by embedding the pairwise latency into a low-dimensional coordinate system. The prediction errors are iteratively optimized by adjusting the coordinates with respect to neighbors. Unfortunately, the optimization process is vulnerable to the inaccurate coordinates, leading to destabilized positions. In this paper, we propose RMF, a relative coordinate-based distributed sparse-preserving matrix-factorization method to provide guaranteed stability for the coordinate system. In RMF, each node maintains a low-rank square matrix that is incrementally adjusted with respect to its neighbors' relative coordinates. The optimization is self-stabilizing, guaranteeing to converge and not interfered by inaccurate coordinates, since the relative coordinates do not have computational errors. By exploiting the sparse structure of the square matrix, the optimization enforces the L_1-norm regularization to preserve the sparseness of the square matrix. Simulation results and a PlanetLab-based experiment confirm that RMF converges to stable positions within 10 to 15 rounds, and decreases the prediction errors by 10% to 20%.**

***Index Terms*— Latency sensitive applications, stability, network distance, network coordinate, relative coordinate, matrix factorization.**

I. Introduction

LATENCY-SENSITIVE network applications, e.g., distributed storage, online social networks, online network games [1], multimedia streaming [2], interactive data publish and query applications have attracted tens of millions of simultaneous online users. Users access personalized contents, with high demands of the quality of experiences (QoE): The interactive process should be low-latency, otherwise, the QoE degrades, yielding decreasing revenues for networking applications. As a result, latency-oriented optimization becomes increasingly popular [1], [3]. For example, online network games recommend nearby players to form the game groups [1]; multimedia-streaming applications construct low-latency overlay topologies and redirect the users to the nearest servers [2].

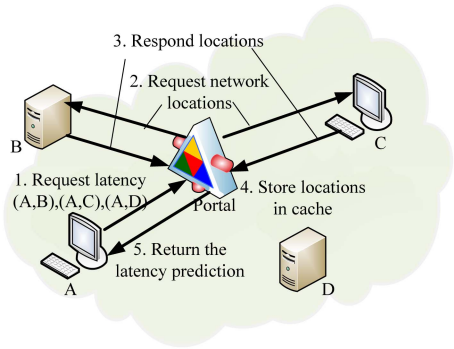


Fig. 1. The diagram of the network distance service. The portal provides the query interface for end hosts. The network distances can be stored in the portal for fast response.

The pairwise latency is a fundamental metric for the latency optimization. Accordingly, the network distance service that measures the pairwise network latency has been extensively studied [1], [4]–[9]. Given the identifiers (for example, the IP address) of a pair of nodes, the network distance service returns the pairwise latency value from one node to the other node. Figure 1 shows an illustrative example. A network distance service approximates the *near real-time network latency* for decentralized nodes. In many cases, the Round Trip Time (RTT) is stationary, since most of the routing paths are relatively stable over tens of minutes to hours [10]–[12]. The network distance service needs scalable latency-measurement techniques, since the numbers of clients and servers are usually on the orders of millions. Collecting all-pair latencies requires a quadratic number $O(N^2)$ of probing packets and $O(N^2)$ storage space, where N denotes the number of participating nodes.

In contrast, network coordinate methods provide a scalable approach to estimate network distances without all-pair direct measurements, for N hosts, $O(N)$ coordinates suffices to predict the all-pair RTTs. GNP [4] and IDES [5] require a set of centralized landmark nodes to serve as reference nodes for calculating the coordinates, which could cause performance bottlenecks as the system increases. Vivaldi [6], DMF [7] and DMFSGD [9] directly let each node adjust its coordinate with respect to the coordinates of a number of sampled neighbors, which avoids the single points of failures. Generally, each coordinate is initialized as a random vector and iteratively adjusted using neighbors' coordinates that have a degree of errors.

Manuscript received June 30, 2015; revised April 7, 2016; accepted June 13, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Wang. This work was supported by the National Natural Science Foundation of China under Grant 61402509.

Y. Fu is with the Science and Technology Laboratory of Parallel and Distributed Processing, College of Computer Science, National University of Defense Technology, Changsha 410073, China (e-mail: yongquanf@nudt.edu.cn).

X. Xiaoping is with the National University of Defense Technology, Changsha 410073, China (e-mail: 53478237@qq.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2016.2581592

The distributed distance estimation has to be consistently accurate under varying system conditions. For example, end hosts may join or leave the system at any time, yielding churns. Existing network coordinate methods face the destabilization issue: some neighbors' coordinates can be arbitrarily inaccurate due to the churns of the distributed systems, accordingly, the coordinate optimization process is easily tampered by these neighbors' positions, as shown in section III-B. Researchers [1], [6], [8], [9] propose to use weights to estimate the accuracy of neighbors' coordinates, and to adjust the coordinate movements scaled by weights. The weights are correlated with the accuracy of neighbors' coordinates. However, calculating exact weights is challenging, since no ground-truth optimal coordinates exist in the system. As a result, the weights based methods [1], [6], [8], [9] still suffer from the destabilization problem.

In this paper, we define a rigorous self-stabilized network coordinate model in section III-C. The **self stabilization** guarantees to converge to a "legitimate" state in a bounded amount of time, regardless of the initial state [13]. We propose to map the "legitimate" state to the local-minimum position of each network coordinate that is not interfered by system churns.

We propose a novel distributed network coordinate method named RMF that guarantees to be self-stabilizing under churns. RMF addresses the destabilization problem in a very different way. Whereas prior network coordinate methods use weights to adapt to the effects of inaccurate coordinates, which are sensitive to the accuracy of weights, RMF avoids the uncertainty using the relative coordinate [14] to adjust each node's coordinate. Each node has a relative coordinate that is constructed as a vector of RTT values from itself to a set of distributed neighbors. Our key observation is that the relative coordinates have no computing errors, as a result, the coordinate optimization process converges to the local minimum irrespective of neighbors' coordinate states.

To represent the pairwise coordinate distance, we present a novel distributed relative coordinate based matrix factorization model. Each node maintains a relative coordinate and a square non-negative parameter matrix whose dimension amounts to the dimension of the relative coordinate. The coordinate distance of a node pair amounts to the product of a node's relative coordinate, parameter matrix and the other one's relative coordinate. The relative coordinates are exchanged among neighbors to optimize each node's parameter matrix, while the parameter matrix of each node is not exchanged. Our three-factor matrix factorization model differs significantly from the two-factor matrix factorization used in DMF [7] and DMFSGD [9], since the latter does not use relative coordinates to represent the coordinate distances.

We propose decentralized algorithms to maintain the relative coordinates and the parameter matrices. First, each node randomly selects a small number of neighbors (16 by default) as the decentralized landmarks of its relative coordinate, in order to avoid the poor scalability of the centralized landmarks. Then, each node measures the RTTs to its neighbors and use them to construct its own relative coordinate. Second, by observing the sparse structure of the parameter matrix (see section V-A), we propose a novel distributed sparse-regularized least-square optimization approach to optimize each parameter matrix. Accordingly, each node initializes its parameter matrix as a random non-negative matrix and updates the parameter matrix in a fully distributed manner, which guarantees to converge to the local minimum. Our experiments from both simulations and a PlanetLab deployment confirm that RMF is robust to the errors of neighbors' parameter matrices, and improves the prediction accuracy by 10-20%.

TABLE I
NOTATIONS

Notation	Meaning
d	The pairwise RTT matrix
$\hat{d}$	The coordinate-distance matrix
r	The rank of $\hat{d}$
N	The number of nodes
Y_i	The relative coordinate of node i
Φ_i	The parameter matrix of node i
m	The dimension of the relative coordinate
τ_r	The time interval of updating relative coordinates
τ	The time interval for adjusting the parameter matrix
α	The regularized parameter

The rest of the paper is organized as follows. We present related network-coordinate studies in section II. Then we formulate the self-stabilization problem in section III. We next introduce the design of the RMF method in section IV. Then, we present a mathematical framework to provide the self-stabilizing network coordinates in section V. We introduce the implementation of RMF in section VI. Next, we evaluate the performance of RMF and compare it with related studies using comprehensive simulations in section VII. We present the performance of RMF on the PlanetLab in section VIII. Finally, we conclude in section IX. The symbols are summarized in Table I.

II. Related Work

We next survey existing network-coordinate methods, focusing on the stability of the coordinate optimization process. A more comprehensive survey is referred to [15].

Relative Coordinate The relative coordinate assumes a set of centralized nodes called landmarks to be available. Then each node obtains a vector of distance values to these landmarks, and uses this vector as its relative coordinate. The order of landmarks should be the same among all nodes. We can see that the relative coordinates themselves do not contain computational errors, since each coordinate component amounts to the distance from a node to a landmark. Moreover, the computing processes of relative coordinates of different nodes do not correlate with each other, therefore, each node's relative coordinate is independent from others.

We can use the upper and lower bounds of the triangle inequality of the distances to these landmarks [16]–[18] to predict the distance intervals for two nodes. Unfortunately, the intervals have approximation errors, since the Internet delay space contains TIVs. Further, centralized landmarks easily lead to single points of failures and performance bottlenecks. In this paper, we use distributed neighbors as the decentralized

landmarks to scale well, and combine the relative coordinates with the matrix factorization to compute exact distances that tolerate the TIVs.

Multidimensional Scaling (MDS) GNP [4] pioneers the study of network coordinates, using a small number of centralized nodes as the landmarks to guide the coordinate computation. These landmarks embed their coordinates into a geometric coordinate space, then other non-landmark nodes compute their coordinates by minimizing the errors between the RTTs and coordinate distances to landmarks. NPS [19] and PIC [20] use decentralized neighbors to compute the coordinates, which therefore increases the scalability. Unfortunately, NPS and PIC are agnostic of neighbors' coordinate errors, as a result, the coordinate-optimization process is vulnerable to inaccurate coordinates. Moreover, MDS methods suffer from the Internet's TIV phenomenon, since the coordinate system assumes the triangle inequality to hold.

Force-Field Simulation It predicts RTTs using the simulations of the physical-force fields. Each node is regarded as one particle in a physical system; and the RTT values amount to the force strengths between particles in the steady positions. Adjusting the coordinate is analogous to moving a particle in a field, which has an interesting connection with the well-studied force-field theory. The Big-Bang [21] method assigns each coordinate at the origin, and iteratively moves all nodes' coordinates in a centralized manner. Vivaldi [6] and its following works like Non-metric Vivaldi [22] and Htrae [1] initialize each coordinate at a random position and incrementally optimize the coordinate in a distributed manner. Further, Vivaldi and its subsequences maintain a weight parameter w to represent the accuracy of each coordinate according to the historical prediction errors. Then each node i adjusts its coordinate respect to the error towards a neighbor j scaled by the weights as $\frac{w_i}{w_i+w_j}$. The weights smooth the coordinate movement towards neighbors. Unfortunately, since no global-truth errors of each coordinate exist in the network coordinate system, the weight parameter has uncertainty that destabilizes the optimization process (see section III-B).

Matrix Factorization IDES [5] assumes centralized landmarks to compute their coordinates first using the two-factor matrix factorization defined in Eq (1), while non-landmark nodes calculate closed-form coordinates with respect to landmarks based on the least-square optimization. DMF [7], Phoenix [8] and DMFSGD [9] let each node adjust its own coordinate with respect to neighbors' coordinates defined in Eq (1). The coordinate components in Eq (1) are initialized as randomized vectors, which have to be iteratively optimized by reducing the errors between the RTTs and the coordinate distances towards its neighbors. IDES and DMF are agnostic of neighbors' coordinate errors. Each node just computes the closed-form coordinate with respect to neighbors' coordinates. As a result, IDES and DMF become destabilized when neighbors' coordinates have errors. Phoenix [8] and DMFSGD use a weight parameter to scale the coordinate optimization analogous to Vivaldi [6]. Unfortunately, since no ground-truth coordinate positions are available, there exist inherent uncertainty in the weights, leading to degraded performance.

Our work has three major differences from DMF and DMFSGD. First, we propose a novel three-factor matrix factorization (see Eq (4)) to compute the coordinate distances using relative coordinates and a low-dimensional parameter matrix. As discussed in the relative-coordinate sub-section, the relative coordinate has no computational errors and therefore does not need to be optimized. However, in DMF [7] and DMFSGD [9], they did not study the relative coordinate, instead, they studied the two-factor matrix factorization defined in Eq (1), where U and V are initialized as the random matrices and need to be optimized. Therefore, both matrices U and V have computational errors.

Second, we observe that the parameter matrix has many zero (close to zero) items, as a result, we formulate a distributed sparse-regularized optimization objective function to preserve the sparseness of the parameter matrix of each node. In contrast, DMF and DMFSGD use the Frobenius-norm regularized objective to optimize the coordinate components U_i and V_i for each node i. DMFSGD regularizes both coordinate components U_i and V_i, while we only need to regularize the parameter matrix, leaving the relative coordinates unchanged.

Third, we propose a nonnegativity-preserving multiplicative rule to adjust the coordinate for each node. We can use the relative coordinates free-of-computation-error benefit to stabilize the network coordinate. We guarantee to find the local minimum for the optimization objective. While DMF uses the least-squared optimization, and DMFSGD uses the stochastic gradient descendent (SGD) optimization. Unfortunately, our experiments show that DMF and DMFSGD are affected by the churns: neighbors' erroneous coordinates destabilize the coordinate optimization process.

III. Problem Definition

A. Prediction Model

The pairwise RTT values of N nodes can be represented as a N-by-N matrix d. Then the network distance problem is transformed to obtain the unobserved items in the matrix d. The network coordinate methods seek to compute a *low-dimensional coordinate-distance matrix* $\hat{d}$ that approximates the RTT matrix d well. Each unobserved pairwise RTT value d_{ij} for a pair of nodes i and j is estimated using the coordinate distance $\hat{d}_{ij}$.

It is well known that [23]–[25] the latency space has a fraction of triangle inequality violations (TIV), as a result, the metric-space based coordinate spaces such as the Euclidean space, the Hyperbolic space or the spherical space have latent distortions due to the assumption of the triangle inequality to hold. The vector-space based matrix factorization tolerates the TIVs via products of vectors. For example, the two-factor matrix factorization model is adopted by existing studies [5], [7]–[9]:

$$\hat{d} = UV^T \tag{1}$$

where $U \in R^{N\times r}$ and $V \in R^{N\times r}$. The i-th row vectors U_{i*} and V_{i*} altogether form the coordinate for node i. The coordinate distance $\hat{d}_{ij}$ from node i to node j amounts to $\hat{d}_{ij} = \sum_{k=1}^{r} U_{ik}V_{jk}$. For each pair (i, j) of nodes, the coordinate

distance $\hat{d}_{ij}$ may differ from $\hat{d}_{ji}$. When the RTT matrix d is complete, we can calculate the closed-form optimal solution of U and V by the Singular Value Decomposition (SVD) technique [26]; while when the matrix d has missing items, computing the optimal matrices U and V is NP-hard [26].

Coordinate Optimization Finding the optimized coordinates requires to form an objective function. A centralized objective function minimizes the sum of pairwise prediction errors:

$$L(\{\overrightarrow{x}_i\}) = \sum\nolimits_{i,j\in S} \left\| d_{ij} - \hat{d}_{ij} \right\| \quad (2)$$

where the norm $\|\bullet\|$ represents some metric, e.g., $\|d\|_F^2 = \sum_{i,j\in S} d_{ij}^2$. Eq (2) requires global pairwise latencies to be available, which does not scale well. Instead, the state-of-the-art methods approximate Eq (2) using decomposed sub-objective functions, where each node i independently minimizes a sub-objective:

$$L(\vec{x}_i) = \sum\nolimits_{j\in S_i} \|d_{ij} - \text{PairCoordDist}\,(\vec{x}_i, \vec{x}_j)\| \quad (3)$$

where $\overrightarrow{x}_i$ denotes a coordinate, S_i represents the **neighbors** of node i, and PairCoordDist $(\overrightarrow{x}_i, \overrightarrow{x}_j)$ represents the coordinate distance from node i to node j.

When each node i joins the network-coordinate system, its coordinate is initialized as a random vector; afterwards, node i iteratively optimizes its own coordinate in rounds with respect to the coordinates of neighbors S_i. Although this distributed optimization process scales well, it is vulnerable to several challenges, as discussed in the related-work section II.

B. Self-Stabilization Comparison

Modeling the churns is challenging, since nodes usually have a variety of decentralization: some nodes in the data centers may be quite stable, while some nodes in home networks or desktop grids could join or leave the system at will. Most of prior network-coordinate simulators usually assume that neighbors are always online during the simulation period [6]–[8]. As a result, each round of the distributed coordinate adjustment procedure is equivalent to a centralized iterative optimization process that converges fast.

We use a simple asynchronous model to represent the dynamics of nodes. The simulation consists of a number of rounds. At each round, every node independently samples its neighbors and updates its own coordinate. At each round, a fraction p of nodes add one neighbor that just joins in the coordinate system with a random coordinate to their neighbor sets. By controlling the fraction p of nodes meeting dynamic neighbors, we are able to study the effects of churns of nodes on the convergence of coordinates.

1) Existing Methods' Stability: We select two representative methods NonMetric [22] and DMF [7], and compare their performance. We compute the well-known **Relative Error** of coordinates, defined as the absolute difference of the coordinate distance and the RTT distance divided by the RTT distance $f_{RE} = \frac{|\hat{d}_{ij} - d_{ij}|}{d_{ij}}$. We also test the stability of other methods. The results are consistently similar as NonMetric and DMF, which are omitted for brevity.

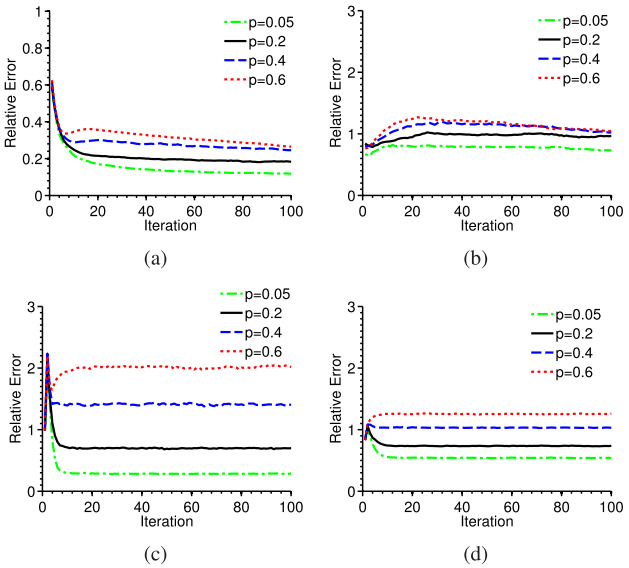


Fig. 2. The convergence of NonMetric and DMF methods for nodes with dynamic neighbors. (a) NonMetric on P2P1143. (b) NonMetric on Meridian2500. (c) DMF on P2P1143. (d) DMF on Meridian2500.

We first test whether nodes that have dynamic neighbors degrade the coordinates' accuracy. We vary the percent p of nodes meeting dynamic neighbors from 0.05 to 0.6. In each round, we compute the mean relative errors for those nodes having dynamic neighbors. Finally, we plot the CCDFs of the relative errors of coordinates for these nodes after 100 rounds.

From Figure 2, we see that NonMetric and DMF degrade their prediction accuracy due to dynamic neighbors. DMF converges to much worse local minimum than NonMetric. As DMF is agnostic of neighbors' errors, while NonMetric tunes uncertainty weights of neighbors' coordinates. NonMetric converges to instable positions when more nodes have dynamic neighbors, since the uncertainty weights only locally approximate the accuracy of each coordinate.

Having shown that dynamic neighbors degrade the convergence, we next study the accuracy of coordinates for nodes whose neighbors are stable throughout the simulation periods. Figure 3 shows the results. We see that increasing the percents of dynamic neighbors degrades the prediction accuracy of stable nodes. This is because although their neighbors are stable, the neighbors of these nodes' neighbors may be dynamic. The coordinate errors of the dynamic neighbors are indirectly propagated along the progress of coordinate updates.

Summary of Findings: Our experiments show that even a small fraction of dynamic nodes degrade the overall accuracy. Not only nodes with dynamic neighbors have degraded coordinates, but also those having stable neighbors also decrease the prediction accuracy. We can also see that the weights have limitations in mitigating the destabilization of random coordinates. Individual nodes may be misled by the random coordinates significantly.

C. Stability of Network Coordinates

Having presented the backgrounds of network coordinate methods, we next introduce the stabilization model for the network coordinate methods. We first state our assumptions. First, the distributed system may have churns, where new

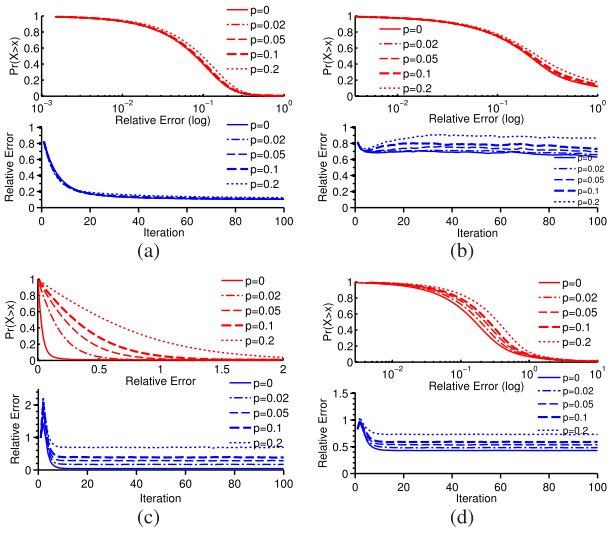


Fig. 3. The convergence of NonMetric and DMF methods for nodes with stable neighbors. (a) NonMetric on P2P1143. (b) NonMetric on Meridian2500. (c) DMF on P2P1143. (d) DMF on Meridian2500.

nodes may join the system and existing nodes may leave the system at any time. Second, the pairwise latency matrix d is assumed to be stationary, where small perturbations of latencies are smoothed using the filters [27]–[29]. Our assumptions are reasonable, as the churns match well with the dynamics of the end hosts, while the stationary assumption is the basis for the network-coordinate studies.

Self-stabilization: A classic self-stabilizing system should satisfy two properties [13]:

- **Convergence**: it guarantees to converge to a legitimate state within finite rounds, irrespective of its starting state;
- **Closure**: it stays in the legitimate state after convergence, and converges to the legitimate state after external perturbations.

Due to the convergence and closure properties, a self-stabilization system is insensitive to the initial state, and tolerates transient events like failures of nodes, joins of new nodes, perturbations of external environments.

Local minimum state: The self stabilization has not been studied in the network-coordinate field. Therefore, we first define the analogous notations in a network coordinate system. We define the **state of a node** as its current coordinate position, and define the **state of a network** coordinate system as the whole set of states of all online nodes.

We next motivate the definition of the legitimate state of the system. When no churns happen, the distributed coordinate-optimization procedure converges to the local-minimum states. Let the **static local minimum** of each node be the ones computed when all nodes are assumed to be online with respect to Eq (3). We can see that the static local minimum is the best state for each node. Due to the churns of the system, some neighbors may be offline. As a result, new neighbors must be re-sampled from the rest of online nodes. The newly sampled neighbors may have arbitrarily inaccurate coordinates, however, existing coordinate-optimization methods tend to move the coordinates towards neighbors having inaccurate coordinates, which increases the overall errors, as shown in section III-B. As a result, it is desirable to ensure the network coordinate system to reach the static local minimum even under churns. Therefore, we define the **legitimate state of a node** as the static local-minimum coordinate.

Self-stabilizing network coordinate: Having presented the legitimate states of network coordinates, we next define the self-stabilizing network coordinate system:

*Definition 1: A network coordinate system is **self-stabilizing**, iff each node's coordinate satisfies two conditions:*

- ***Convergence**: Each node's coordinate should converge to the static local minimum that minimizes Eq* (3) *in a finite rounds of optimization.*
- ***Closure**: Once the coordinate converges to the static local-minimum state, new rounds of optimization should put the coordinate into the static local minimum. If churns happen, new rounds of optimization should converge each node's coordinate to the static local minimum within a finite rounds of optimization.*

To realize the self-stabilizing objective, each node's coordinate optimization process has to be *invariant to the erroneous coordinates of neighbors*, which is the focus of this paper.

IV. Designing Stable Coordinates

Having analyzed the stability model, we next present RMF, a method that completely avoids the instability caused by inaccurate coordinates and guarantees to converge to the self-stabilizing states, using a novel relative coordinate matrix factorization based coordinate structure.

A. Overview

We propose a novel relative coordinate based matrix factorization named RMF to predict the pairwise latency. In RMF, each node i maintains a list of online nodes as its **neighbors** (required to be at least $\geq m$, where $m = 16$ by default). Each node i maintains a relative coordinate Y_i of length m and a $m \times m$-sized square **parameter matrix** Φ_i constructed as follows:

- Node i probes the latencies to these neighbors and generates a vector of length m of probed latencies to neighbors. This vector serves as node i's **relative coordinate** Y_i.
- The parameter matrix Φ_i stands for node i's **coordinate**, which is initialized as a random matrix and needs to be optimized. We enforce each item in the parameter matrix to be non-negative to preserve the nonnegativity of the latency values.

Next we present a three-factor matrix factorization based distance function to calculate pairwise coordinate distances. The **coordinate distance** $\hat{d}_{ij}$ from a node i to another node j amounts to the product of i's relative coordinate Y_i, its parameter matrix Φ_i and node j's relative coordinate Y_j, which is calculated by:

$$\hat{d}_{ij} = Y_i \Phi_i Y_j^T \tag{4}$$

The pairwise coordinate distances $\hat{d}_{ij}$ and $\hat{d}_{ji}$ can be different, since the parameter matrices Φ_i and Φ_j vary.

B. TIV Tolerance and Stabilization

TIV tolerance: Our coordinate structure tolerates TIVs. In section VI-A, we present such an example. Here we give intuitions. For a triple of three nodes A, B and C, we compute

the ratio between the sum of two edges and the third edge

$$\frac{\hat{d}_{AB}+\hat{d}_{BC}}{\hat{d}_{AC}} = \frac{Y_A\Phi_A Y_B^T + Y_B\Phi_B Y_C^T}{Y_A\Phi_A Y_C^T} = \frac{Y_A\Phi_A Y_B^T}{Y_A\Phi_A Y_C^T} + \frac{Y_B\Phi_B Y_C^T}{Y_A\Phi_A Y_C^T}$$

If every item in Y_B is larger than every item in Y_C, then the ratio $\frac{\hat{d}_{AB}+\hat{d}_{BC}}{\hat{d}_{AC}}$ will be always larger than one, i.e., satisfying the triangle inequality. Otherwise, $\frac{\hat{d}_{AB}+\hat{d}_{BC}}{\hat{d}_{AC}}$ may be smaller than one, therefore, a TIV happens.

As the neighbors of each node are randomly selected, the latency values to these neighbors tend to be randomly distributed. As a result, the probability that every item in Y_B is larger than every item in Y_C is very low. Therefore, the TIV may happen.

Stabilization: We ensure the stabilization by decoupling the correlation of parameter matrices of different nodes. Recall that existing methods must obtain neighbors' coordinates to optimize its own coordinate. While in RMF, each node does not need neighbors' parameter matrices, but only requires their relative coordinates. To see why, each node's parameter matrix Φ_i is only useful for predicting network distances from itself to other nodes, while its relative coordinate Y_i is required for predicting network distances from other nodes to node i.

The independence of coordinates helps RMF keep stable against erroneous coordinates. The relative coordinate's free-of-computation-error advantage is translated to decouple the interference of inaccurate coordinates among online nodes. Accordingly, each parameter matrix guarantees to converge by itself, which completely avoids the instability caused by inaccurate coordinates. We provide a formal proof of the self stabilization in Theorem 2 in section V-E.

C. Coordinate Optimization

Maintaining the Relative Coordinate: Each node's relative coordinates are required to be of the same length for interoperability. Selecting centralized landmarks does not scale well. To overcome this shortcoming, each node uses its own neighbors as the landmarks to compute the relative coordinates. The relative coordinates should be updated infrequently, since they are references for optimizing the parameter matrices.

Optimizing the Parameter Matrix: We optimize parameter matrices for each node in a fully distributed manner. Each node i requests the relative coordinates of these neighbors and then iteratively adjusts its own coordinate in cycles. Node i initializes a randomized parameter matrix, then in each cycle, node i updates the parameter matrix Φ_i iteratively that guarantees to reach the local minimum (see section V-C).

V. Mathematical Analysis

Having presented the basic ideas of RMF, we next present mathematical results to optimize the parameter matrix of each node. We first formulate the optimization problem, then derive an multiplicative optimization rule, finally, we prove the stabilization of the optimization process. In contrast, the relative coordinates can be directly maintained by probing the distributed neighbors.

A. L_1-Norm Regularized Optimization

To optimize the parameter matrices, we adopt the well-known least-squared optimization technique [4], [6], [7]. To keep the nonnegativity of coordinate distances, we enforce each node's parameter matrix to be nonnegative. The least-squared optimization minimizes the squared difference between the coordinate distances and pairwise RTT values:

$$L(\Phi) = \min \sum_{i,j\in[1,N],i\neq j} \left\| d_{ij} - Y_i\Phi_i Y_j^T \right\|_F^2 \tag{5}$$

where $\|\mathbf{x}\|_F^2 = \sum_{i,j}\mathbf{x}_{ij}^2$.

Enforcing each coordinate distance $\hat{d}_{ij}$ to match the RTT d_{ij} is likely to fail, since the RTT matrix is only approximately low-rank. Accordingly, the **overfitting** phenomenon may occur: Although we can decrease the prediction errors for pairwise RTTs of nodes in the training set, the errors for nodes that are out of the training set may increase significantly. Therefore, we next refine Eq (5) to avoid the overfitting issue.

A common approach is to incorporate regularized items to Eq (5) that tries to minimize the difference, but does not strictly require the difference to be zero. We propose a regularization to Eq (5) using one important observation: *Many items in each parameter matrix are (close to) zeros*. To see why, the one-way coordinate distance $\hat{d}_{ij}$ from node i to node j can be rewritten as:

$$\hat{d}_{ij} = \sum_{c_1\in[1,m]}\sum_{c_2\in[1,m]} Y_{ic_1}Y_{jc_2}\Phi_i[c_1,c_2]$$

which can be transformed to:

$$\sum_{c_1\in[1,m]}\sum_{c_2\in[1,m]} \frac{Y_{ic_1}Y_{jc_2}}{\hat{d}_{ij}}\Phi_i[c_1,c_2] = 1 \tag{6}$$

As the latency values $\hat{d}_{ij}$, Y_{ic_1} and Y_{jc_2} are uniformly distributed due to the randomly sampling of neighbors, we see that $Y_{ic_1}\times Y_{jc_2} > \hat{d}_{ij}$ holds in many cases. As a result, many items in the parameter matrix Φ_i are much smaller than one with increasing numbers of neighbors. Our example in section VI-A clearly shows that many items in the parameter matrices are close to zeros.

As a result, we use the well-known L_1 norm [30] to enforce the sparsity of each parameter matrix:

$$\begin{aligned} L(\Phi) &= \min \sum_{i,j\in[1,N],i\neq j} \left\|\left(d_{ij}-Y_i\Phi_i Y_j^T\right)\right\|_F^2 + \sum_{i\in[1,N]} \|\Phi_i\|_{L_1} \\ &= \min \sum_{i,j\in[1,N],i\neq j} \left\|\left(d_{ij}-Y_i\Phi_i Y_j^T\right)\right\|_F^2 \\ &\quad + \sum_{i\in[1,N]}\sum_{c_1=1}^{m}\sum_{c_2=1}^{m} |\Phi_i[c_1,c_2]| \\ &= \min \sum_{i,j\in[1,N],i\neq j} \left\|\left(d_{ij}-Y_i\Phi_i Y_j^T\right)\right\|_F^2 \\ &\quad + \sum_{i\in[1,N]}\sum_{c_1=1}^{m}\sum_{c_2=1}^{m} \Phi_i[c_1,c_2]. \end{aligned} \tag{7}$$

due to the nonnegativity of the parameter matrices

B. Distributed Decomposition

We next decompose Eq (7) to a set of problems that can be solved in a distributed manner. For each node i, let the set S_i denotes node i's neighbors, we transform Eq (7) to a new objective that only contains the RTT measurements from node i to its neighbors:

$$L(\Phi_i) = \min \sum_{j \in S_i} \left\| d_{ij} - Y_i \Phi_i Y_j^T \right\|_F^2 + \alpha \sum_{c_1=1}^{m} \sum_{c_2=1}^{m} \Phi_i [c_1, c_2] \tag{8}$$

Now each node i is able to locally compute its parameter matrix with neighbors' relative coordinates, while neighbors' parameter matrices are not needed.

Eq. (8) belongs to the well-studied tri-factor matrix factorization [31], which has the same optimal solution as the traditional two-factor matrix factorization, but increases the freedom of the computation process.

C. Distributed Optimization

We next propose a distributed algorithm to optimize Eq (8). Let $d_i = \left(d_{ij_1}, \ldots, d_{ij_{|S_i|}}\right)$, $Y = \left[Y_{j_1}; \ldots, Y_{j_{|S_i|}}\right]$, where $j_1, \ldots, j_{|S_i|} \in S_i$.

We transform Eq (8) as

$$\begin{aligned} L(\Phi_i) &= \left\| d_i - Y_i \Phi_i Y^T \right\|_F^2 + \alpha \sum_{c_1=1}^{m} \sum_{c_2=1}^{m} \Phi_i [c_1, c_2] \\ &= tr\left(d_i^T d_i - 2 Y^T d_i^T Y_i \Phi_i + Y_i Y_i^T \Phi_i Y^T Y \Phi_i^T \right) \\ &\quad + \alpha \sum_{c_1=1}^{m} \sum_{c_2=1}^{m} \Phi_i [c_1, c_2] \end{aligned} \tag{9}$$

with the trace norm of the matrix, where $tr()$ denotes the trace of a matrix. The optimal solution that minimizes the objective in Eq (9) must satisfy the Karush-Kuhn-Tucker complementarity condition (KKT condition):

$$\left(-Y_i^T d_i Y + Y_i Y_i^T \Phi_i Y^T Y + \alpha \right)_{gh} \Phi_i [g, h] = 0 \tag{10}$$

which implies that the optimal parameter matrix Φ_i must satisfy:

$$\Phi_i [g, h] = \frac{\left(Y_i^T d_i Y \right)_{gh}}{\left(Y_i Y_i^T \Phi_i Y^T Y + \alpha \right)_{gh}} \Phi_i [g, h] \tag{11}$$

We next present a multiplicative rule to adjust the parameter matrix Φ_i for each node i. We first compute the gradient of each element $\Phi_i [g, h]$ with respect to Eq (9) (we omit the scalar "2" in the right side of Eq (12) to simplify the derivations):

$$\nabla_{\Phi_i [g,h]} L(\Phi_i) = \left(-Y_i^T d_i Y + Y_i Y_i^T \Phi_i Y^T Y + \alpha \right)_{gh} \tag{12}$$

By the gradient-descent method, we set the update rule for $\Phi_i [g, h]$ as:

$$\begin{aligned} \Phi_i [g, h] &\leftarrow \Phi_i [g, h] - \eta_{gh} \nabla_{\Phi_i [g,h]} L(\Phi_i) \\ &= \Phi_i [g, h] - \eta_{gh} \Phi_i [g, h] \\ &\quad \times \left(-Y_i^T d_i Y + Y_i Y_i^T \Phi_i Y^T Y + \alpha \right)_{gh} \end{aligned} \tag{13}$$

where η_{gh} denotes the adjustment parameter. To keep $\Phi_i [g, h]$ to be nonnegative, we set η_{gh} as: $\eta_{gh} = \frac{\Phi_i [g,h]}{\left(Y_i Y_i^T \Phi_i Y^T Y + \alpha \right)_{gh}}$.

Then, Eq (13) can be written as:

$$\begin{aligned} \Phi_i [g, h] &\leftarrow \Phi_i [g, h] \left(1 - \frac{1}{\left(Y_i Y_i^T \Phi_i Y^T Y + \alpha \right)_{gh}} \right. \\ &\quad \left. \times \left(-Y_i^T d_i Y + Y_i Y_i^T \Phi_i Y^T Y + \alpha \right)_{gh} \right) \\ &= \Phi_i [g, h] \frac{Y_i^T d_i Y}{\left(Y_i Y_i^T \Phi_i Y^T Y + \alpha \right)_{gh}} \end{aligned} \tag{14}$$

At convergence, the solution of Eq (14) satisfies Eq (11), which guarantees the correctness of Eq (14). We next prove that the distributed objective Eq (9) is non-increasing under Eq (14), yielding a local minimum.

Theorem 1: Eq (9) *reaches the local minimum under the update rule*

$$\Phi_i^{t+1} [g, h] \leftarrow \Phi_i^t [g, h] \cdot \frac{\left(Y_i^T d_i Y \right)_{gh}}{\left(Y_i Y_i^T \Phi_i^t Y^T Y + \alpha \right)_{gh}} \tag{15}$$

Proof: We use the auxiliary-function method for the proof inspired by Ding et al. [31].

Definition 2: A function $g(H, \tilde{H})$ *is called an* ***auxiliary function*** *of another function* $f(H)$ *if the function* $g(H, \tilde{H})$ *satisfies that:*

$$g\left(H, \tilde{H}\right) \geq f(H), g(H, H) = f(H) \tag{16}$$

for any variables $H, \tilde{H}$.

Auxiliary-function Example*: Let*

$$H^{t+1} = \arg\min_{H} g\left(H, H^t\right) \tag{17}$$

be the minimum for the function $g(H, H^t)$*. We have*

$$f\left(H^t\right) = g\left(H^t, H^t\right) \geq g\left(H^{t+1}, H^t\right) \geq f\left(H^{t+1}\right) \tag{18}$$

Therefore, the function $f(H^t)$ *monotonically decreases for a sequence of variables* $\left(H^1, \cdots, H^t\right)$ *as the index* $t \to \infty$.

Let

$$\begin{aligned} g(\Phi_i, \Phi_i') &= \left(d_i^T d_i - 2tr\left(Y^T d_i^T Y_i \Phi_i \right) \right. \\ &\quad \left. + \sum_{g,h} \frac{\left(Y_i Y_i^T \Phi_i' Y^T Y \right)_{gh} \Phi_i^2 [g, h]}{\Phi_i' [g, h]} \right) \\ &\quad + \alpha \sum_{g,h} \frac{\Phi_i^2 [g, h]}{\Phi_i' [g, h]} \end{aligned} \tag{19}$$

We next show that the function $g(\Phi, \Phi')$ is an auxiliary function of Eq (9).

First, $d_i^T d_i$ and $-2tr\left(Y^T d_i^T Y_i \Phi_i\right)$ are preserved in Eq (9). Second, the terms $\sum_{g,h} \frac{\left(Y_i Y_i^T \Phi_i' Y^T Y \right)_{gh} \Phi_i^2 [g,h]}{\Phi_i' [g,h]}$ and $\alpha \sum_{g,h} \frac{\Phi_i^2 [g,h]}{\Phi_i' [g,h]}$ in Eq (19) are proved to be always bigger than the items $Y_i Y_i^T \Phi_i Y^T Y \Phi_i^T$ and $\alpha \sum_{c_1=1}^{m} \sum_{c_2=1}^{m} \Phi_i [c_1, c_2]$ in Eq (9), respectively (see [31, p. 6]). Therefore, it follows that

$$g(\Phi, \Phi') \geq J_{\Phi}$$

which means that $g(\Phi, \Phi')$ is an auxiliary function of the objective function J_Φ of calculating the parameter matrices for all nodes.

(ii) We next prove that the variable Φ^{t+1} in Eq (15) minimizes the auxiliary function $g(\Phi, \Phi^t)$ in Eq (19). Setting the derivative of $g(\Phi, \Phi')$ in Eq (19) to zero leads to:

$$0 = \frac{\delta g(\Phi, \Phi')}{\delta \Phi_i[g,h]} = -2\left(Y_i d_i Y^T\right)_{g,h} + 2 \times \frac{\left(Y_i Y_i^T \Phi_i' Y^T Y\right)_{gh} \Phi_i[g,h]}{\Phi_i'[g,h]} + 2\alpha \frac{\Phi_i[g,h]}{\Phi_i'[g,h]} \quad (20)$$

which means that

$$\Phi_i[g,h] = \Phi_i'[g,h] \times \frac{\left(Y_i d_i Y^T\right)_{gh}}{\left(Y_i Y_i^T \Phi_i' Y^T Y\right)_{gh} + \alpha} \quad (21)$$

Combining Eq (21) and (15), we see that variables $\Phi^{t+1} = \Phi, \Phi^t = \Phi'$ hold. As a result, Eq (15) monotonically decreases the function (9), yielding the local minimum. □

D. Static Local Minimum Under Churns

Assuming that the latency values are stationary during the optimization, we next show that RMF can yield the static local minimum under churns due to the free of computational errors of the relative coordinates.

1) Neighbor Departs: We construct an optimization process that is invariant to the departure of neighbors. Given a node i with a neighbor set S_i, at the first round, node i retrieves the relative coordinates of all neighbors and obtains the latencies to these neighbors. Next, node i initializes its parameter matrix Φ_i as a positively random matrix. Then node i performs the optimization according to Eq (15). From the first round to the $(t-1)$-th round ($t > 1$), let all neighbors in S_i be online. Node i performs the optimization according to Eq (15). From Theorem 1, the optimization monotonically decreases the prediction error for node i.

While at the t-th round, let some neighbors S_{i_o} in the set S_i be offline. For the t-th and the following rounds, node i removes nodes in S_{i_o} from the set S_i, and continues the optimization according to Eq (15) using online neighbors in $(S_i - S_{i_o})$. since Eq (15) only needs the relative coordinates that are static throughout the optimization, node i's optimization process is equivalent to that using the neighbor set $(S_i - S_{i_o})$ when all nodes are always online. As a result, despite the departure of some neighbors, the optimization is equivalent to that when neighbors are online, yielding the static local minimum.

Further, due to the invariance of the relative coordinates, the departures of neighbors can occur multiple times, while the same conclusions still hold.

2) Neighbor Joins: Second, we consider adding new neighbors into the neighbor set. If at time t, some new nodes V_t are added as a node i's neighbors $S_i^t = S_i \cup V_t$, the objective in Eq (8) may deviate from the current local minimum. However, applying the multiplicative rule in Eq (15) monotonically decreases Eq (8), leading to a new local-minimum solution. We can see that this new local minimum is identical to that using the neighbors S_i^t assuming all nodes are online.

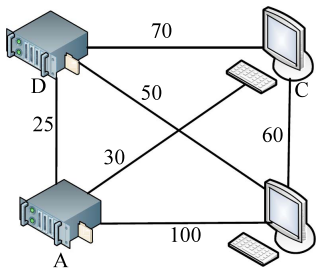


Fig. 4. A simple network topology consisting of four edge hosts A, B, C and D. The edge weight represents RTTs between these nodes.

Therefore, each node i's optimization is also identical to that when all nodes are always online, which yield the static local minimum. Analogous to the departure of neighbors, the same conclusions hold when new neighbors join the system in multiple times.

3) Putting it All: As we can obtain the local minimum that is equivalent to the one when no churns happen, each node's coordinate reaches the static local minimum under churns.

E. Self Stabilization of RMF

We next show that Eq (15) based optimization leads to the self stabilization that is defined in Definition 1.

Theorem 2: Applying Eq (15) *to optimize each parameter matrix leads to the self-stabilizing network coordinates.*

Proof: According to the Definition 1, for each node i, we analyze the convergence and closure of optimizing its parameter matrix by Eq (15). (i) **Convergence**: According to Theorem 1, the multiplicative rule in Eq (15) guarantees that each node i's parameter matrix converges to the local minimum. From section V-D, the local minimum is one of the static local minimum. (ii) **Closure**: If no neighbors are offline, when node i's parameter matrix reaches the static local minimum, further optimization guarantees that the objective in Eq (8) stays in the static local minimum due to the monotonic decreasing optimization objective in Theorem 1. Further, section V-D shows that we can reach the static local minimum under churns. The convergence and closure complete the proof. □

VI. Details of the RMF Method

A. Example of RMF

We next use an example to illustrate how RMF is able to predict RTTs well. From Figure 4, there are four hosts A, B, C and D that need to calculate pairwise RTTs via RMF. Assume that four nodes' neighbors are selected as: $A \leftarrow (B, C)$, $B \leftarrow (A, C)$, $C \leftarrow (A, B)$ and $D \leftarrow (A, C)$. Then these nodes' relative coordinates are calculated as: $Y_A = (100, 30)$, $Y_B = (100, 60)$, $Y_C = (30, 60)$ and $Y_D = (25, 70)$.

After applying 20 rounds of multiplicative rule in Eq (15) for each node's parameter matrices, we obtain a list of two-by-two sized parameter matrices of four hosts:

$$\Phi_A = \begin{bmatrix} 0.0090 & 0.0002 \\ 0.0025 & 0.0005 \end{bmatrix}, \quad \Phi_B = \begin{bmatrix} 0.0056 & 0.0037 \\ 0.0045 & 0.0037 \end{bmatrix}$$

$$\Phi_C = \begin{bmatrix} 0.0000 & 0.0113 \\ 0.0008 & 0.0095 \end{bmatrix}, \quad \Phi_D = \begin{bmatrix} 0.0000 & 0.0116 \\ 0.0000 & 0.0116 \end{bmatrix}$$

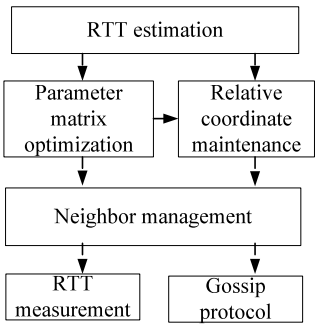


Fig. 5. The diagram of RMF's architecture.

We can calculate the coordinate-distance matrix with Eq (4) as (The diagonal items are set to be zeros):

$$\hat{d} = \begin{bmatrix} 0 & 99.5544 & 31.4116 & 26.9122 \\ 100.0740 & 0 & 59.8762 & 61.6043 \\ 31.7980 & 59.0307 & 0 & 64.6842 \\ 33.0003 & 66.0000 & 65.9996 & 0 \end{bmatrix}$$

We list the RTT matrix d for comparison:

$$d = \begin{bmatrix} 0 & 100 & 30 & 25 \\ 100 & 0 & 60 & 50 \\ 30 & 60 & 0 & 70 \\ 25 & 50 & 70 & 0 \end{bmatrix}$$

We can see that the predicted distances match the RTT distances quite well. For example, the RTT distance $d_{AB} = 100$, while the predicted value $\hat{d}_{AB} = 99.5544$. Further, the TIV triples are well preserved in the prediction. For example, there is a TIV for the triple (A, C, D): $d_{DA} + d_{AC} = 55$, while $d_{DC} = 70$, i.e., $d_{DA} + d_{AC} < d_{DC}$. The TIV is preserved in the predicted distance matrix $\hat{d}$: $\hat{d}_{DA} + \hat{d}_{AC} = 33.0003 + 31.7980 = 64.7983$, and $d_{DC} = 65.9996$. The same conclusion holds for another triple (A, B, C) with a TIV.

Further, the pairwise distances in $\hat{d}$ are asymmetric, as each node's parameter matrix is independently optimized. For example, $d_{CD} = 64.6842$ while $d_{DC} = 65.9996$. This is useful for representing asymmetric distance metrics.

B. Overall Architecture

We have implemented RMF in a distributed network distance service. Figure 5 plots main components in this program: (i) The RTT estimation module. It receives the latency-query requests consisting of the source and destination addresses via the XML RPC interface, and predicts the pairwise RTT value by Eq (4). If the source or the destination are not cached at the local node, it pulls the corresponding relative coordinates and the parameter matrices through the XML RPC interface. (ii) The relative coordinate maintenance module. It maintains the current node's relative coordinate, which will be introduced in section VI-E. (iii) The parameter-matrix maintenance module. It incrementally adjusts a node's parameter matrix, as introduced in section VI-F. (iv) The neighbor management module. It maintains a set of neighbors via the standard gossip protocol [32], [33], introduced in section VI-C.

C. Gossip Based Neighbor Management

The gossip protocol is light-weight and robust, allowing RMF to tolerate failures of nodes and to keep up-to-date membership. Our gossip protocol follows the push-pull based gossip protocol [33]. The objective of the gossip protocol is to maintain a number of randomly-sampled neighbors for updating the parameter matrices.

The gossip protocol consists of two procedures:

(1) **Initialization**: When a node joins the system, it contacts some bootstrapping server that records the addresses of each joined node, obtains the addresses of a number of joined nodes from the bootstrapping node and stores them into a candidate-neighbor set. The candidate-neighbor set stores nodes that are to be neighbors and may contain offline nodes due to churns. Further, each node maintains a neighbor set that stores the neighbors that have exchanged messages.

The neighbor set is used to update the parameter matrix. For each neighbor j, the neighbor set records the address and relative coordinate of node j as well as the RTT value to node j.

(2) **Sampling**: Algorithm 1 shows the process of sampling neighbors. Each node i runs two threads, namely GossipActive and GossipPassive. The GossipActive thread periodically sends gossip messages to other nodes, allows each node to pull the relative coordinates of neighbors and to compute the RTTs through the piggyback messages. The GossipPassive thread continuously waits for the gossip messages from others, stores the pushed information of neighbors, and sends back response messages. To smooth the measured latencies, we use the median filter [27] on the measured RTT samples to filter out transient RTT perturbations due to changing network traffic or machine loads.

D. RTT Estimation

To estimate the RTT to other nodes, each node needs to compute the coordinate distance from itself to others. For example, when a node A needs to predict the RTT value to another node B, node A only requests node B's relative coordinate Y_B. Then node A computes the coordinate distance $\hat{d}_{AB}$ as $\hat{d}_{AB} = Y_A \Phi_A Y_B^T$ according to Eq (4).

E. Computing Decentralized Relative Coordinates

We next introduce the process of maintaining the relative coordinates for each node. To avoid the performance bottlenecks of maintaining centralized landmarks, each node independently chooses its landmarks to construct the relative coordinate, so as to amortize probing loads on each landmark. We do not require that all nodes share the same landmarks, since we use relative coordinates to update each node's coordinate, while prior relative-coordinates target for predicting the proximity between nodes.

Each node i maintains the relative coordinate as the vector of RTTs to its neighbors to obtain its relative coordinate. Node i probes the RTTs from itself to its neighbors in S_i and saves the successfully returned RTTs into a vector Y_i of length m. The ordering sequence of neighbors in the vector Y_i can be arbitrary, but the length of the vector Y_i must be

Algorithm 1 A Gossip based Neighbor Sampling Process

```
GossipActive(i)
while TRUE do
    Node i selects a node j uniformly at random from nodes in the candidate-neighbor set and the neighbor set;
    Node i creates a gossip message, whose payload contains the relative coordinate of node i and the address of a sampled neighbor from the nonempty neighbor set of node i;
    Node i sends the gossip message to node j. Node i records the time-stamp of the transmission and waits for the response from node j.
    if Node i receives node j's response message in L_h seconds (L_h = 10 by default) then
        Node i saves node j's relative coordinate, and updates the RTT value to node j using the timestamp of sending the gossip message to node j;
    else
        Node i removes node j from its neighbor set;
    Sleep(τ);
GossipPassive(j)
while TRUE do
    if Node j receives the gossip message from node i then
        Node j reads the payload, stores the sampled neighbor's address into the candidate-neighbor set, saves node i's relative coordinate into j's neighbor set;
        Node j pings the latency to node i and saves the RTT value in the neighbor set;
        Node j sends a response message to node i, whose payload consists of node j's relative coordinate, a randomly sampled neighbor from its neighbor set;
```

of size m for the matrix factorization between decentralized nodes.

After initializing the relative coordinate, to adapt the dynamics of network latencies, each node i periodically measures RTTs to neighbors in S_i and updates its relative coordinate Y_i accordingly. If a neighbor is temporally offline, we replace the failed landmarks with randomly selected online ones.

F. Distributed Parameter-Matrix Optimization

We then present an algorithm to optimize the parameter matrix that guarantees to converge, based on the results in section V.

We use the neighbor set maintained in section VI-C as the set S_i of nodes to adjust node i's parameter matrix. Thanks to the robustness of the gossip process, offline neighbors are periodically removed and online neighbors are preserved. To keep modest computation costs, we bound the maximal size of the set S_i to be $\Delta_m + \Delta_s$, where Δ_s ($\Delta_s = 2$, $\Delta_m = 16$ by default) denotes the slack parameter to avoid the oscillations of the neighbor set. When a new neighbor is stored in S_i, original neighbors are kept as long as the size of S_i is not larger than $\Delta_m + \Delta_s$, otherwise, we randomly pick a number $|S_i| - \Delta_m$ of online neighbors in S_i and move them to the candidate-neighbor set, therefore, we avoid repeatedly removing the neighbors from the neighbor set until Δ_s new neighbors are added.

Algorithm 2 Decentralized Algorithm to Update the Parameter Matrix for Node i

```
UpdateParameterMatrix(i, Y_i, S_i, d_{S_i}, Y_{S_i}, τ)
input: i: node index, Y_i: node i's relative coordinate, S_i: node i's neighbors, d_{S_i}: the RTT vector from node i to neighbors in S_i, Y_{S_i}: relative-coordinate matrix of neighbors, in which each row vector denotes a relative coordinate of one node i's neighbor in S_i, τ: coordinate update period.
Initialize the parameter matrix Φ as a random matrix: each item of Φ is randomly taken from the interval [0, 1];
while TRUE do
    Let G^d = Y_i^T (d_{S_i}) Y_{S_i};
    Let H^Φ = α + Y_i^T (Y_i Φ Y_{S_i}^T) Y_{S_i} ;
    for g = 1 → m do
        for h = 1 → m do
            tmp ← Φ_gh · G^d_gh / H^Φ_gh;
            if tmp ≠ 0 then
                Φ_gh = tmp;
    Sleep(τ);
```

The multiplicative rule (15) is sensitive to zero values: *if one item Φ_{gh} becomes zero after one round, then this item Φ_{gh} will always be zero afterwards*. However, the coordinate updating process may not converge to the local minimum since we cannot adjust the parameter any more. Skipping the zeros avoids the parameter matrix being stuck at fixed points, consequently, the coordinate updating process converges to the local minimum.

Therefore, in order to increase the robustness of the optimization, we skip the zeros. First, we calculate a temporary value Φ'_{gh} with Eq (15). Second, we conditionally update the parameter matrix with non-zero items. If the temporary value $\Phi'_{gh} \neq 0$, then we set $\Phi_{gh} = \Phi'_{gh}$; otherwise, we skip the item Φ_{gh}.

Algorithm 3 summarizes the main steps. Step 2 initializes node i's parameter matrix Φ_i. Steps 3 to 11 iteratively adjust node's parameter matrix based on Eq (15). Step 4 and 5 cache the local variables of adjusting the parameter matrix Φ_i: Step 4 needs $O(m^2|S_i|)$ time complexity, while Step 5 needs $O(m^3 + m^2|S_i|)$ time complexity. Step 6 to 10 consist of a nested loop to adjust each item in the parameter matrix Φ_i, with time complexity $O(m^2)$. As a result, the overall time complexity of Algorithm 3 amounts to $O(m^3 + m^2|S_i|)$. Besides, if one item becomes zero in Step 8, then Step 9 to 10 will skip this round of update for this item. Step 11 sleeps for a while and waits for the next round.

TABLE II
PARAMETER SETUP

Metric	Value
Topology	Pairwise RTT matrix
Regularized constant α	0.05
Update interval of the relative coordinate τ_r	30 Sec
Update interval of the parameter matrix τ	10 Sec
Number of neighbors Δ_m	16

Remark: We next discuss how to enable the timely response for large-scale distributed systems. First, the coordinate updating process has a bounded time complexity as the system size increases. We choose modest numbers of landmarks and neighbors in order to ensure the scalability: a round of the coordinate updating process requires only several milliseconds (see Figure 12). While the interval between every two coordinate updating processes is usually on the order of tens of seconds [1], [9], [28], [29]. Consequently, the coordinate updating process incurs a negligible delay. Second, we decouple the distance-query component with the coordinate updating process in the network distance service. The distance query component is a front-end service in the portal server, while the coordinate updating process is a background process running in each of the participating servers. Calculating the coordinate distance requires $O(m^2)$ time, which is more than one order of magnitude less than that of the coordinate updating process. Therefore, the distance query process interacts with the end users in a timely manner.

VII. SIMULATION

We next evaluate RMF's performance and compare it with state-of-the-art methods.

A. Experimental Setup

We seek to answer three questions: (i) Is RMF insensitive to the propagation of coordinate errors under churns? (ii) How does RMF vary its performance as we change its parameters? (iii) Is RMF stable against the failures of distributed neighbors used for relative coordinates or neighbors for updating the parameter matrices?

For RMF, we set its default parameters to trade off the accuracy and the computing overhead. We analyze RMF's sensitivity in section VII-D, and found that RMF is robust against the parameter settings, therefore, we set a modest number of neighbors for the RMF method. The size Δ_m of each node's neighbor set is 16. Consequently, the relative coordinate dimension amounts to 16. We set the time interval τ_r to 30 seconds and τ to 10 seconds for each node, since the RTTs are stationary for short-term periods. We set the constant parameter α for the matrix factorization to 0.05. Table II summarizes the parameter configuration.

We report our simulations based on two well-known real-world RTT data sets: (i)P2P1143: a RTT matrix between 1143 DNS servers provided by the P2PSim project [34]. (ii) Meridian2500: a RTT matrix of 2500 DNS servers provided by the Meridian project [35]. We have also tested the performance on other data sets, the same conclusions hold consistently.

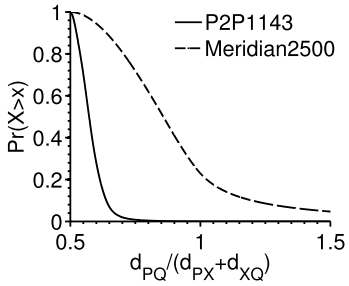


Fig. 6. The CCDFs of the ratios between one edge and the sum of the other two edges in each triple (P, X, Q).

We show the severity of TIVs of these two data sets. For each triple that consists of three nodes denoted as (P, X, Q), where $P, X, Q \in S$, we compute the ratio between an edge and the sum of the other two edges in the triple (P, X, Q), we then plot the complementary cumulative distribution functions (CCDF) of the ratios. When the ratio is bigger than one, a TIV occurs. From Figure 6, we see that over 99% of all triples in P2P1143 satisfy the triangle inequality, while for the Meridian2500 data set, over 23% of the triples violate the triangle inequality.

The TIVs correlate with the latency-prediction accuracy. The low dimensional model of the network coordinate methods assumes that the latency measurements of different nodes are correlated. A TIV occurs for a triple of nodes (P, X, Q) when the sum of two edges d_{PX} + d_{XQ} is smaller than the third edge d_{PQ}, consequently, an edge may be much larger than the sum of the other two edges in this triple. As a result, the TIV in a triple of nodes relaxes the correlation between the edges in this triple. Increasing the ratios of the TIVs in the data set implies that more latencies are less correlated with each other, therefore, the low dimensional coordinate system incurs a higher prediction error. For example, we can see that the Meridian2500 data set has more TIVs than the P2P1143 data set, while from Figure 7 (a) and (b), we see that the prediction accuracy on the Meridian2500 data set is worse than that on the P2P1143 data set in most cases.

B. Stabilization Comparison With RMF

We next compare the stabilization of RMF with state-of-the-art methods including Vivaldi [6], NonMetric [22], DMF [7] and DMFSGD [9]. We set the same number of neighbors for all network coordinate methods for fair comparison. We set the coordinate dimension to ten for compared methods. Further increasing the dimensions of these coordinate methods does not increase the accuracy significantly.

We compute the stress [7], [9] of the network coordinates after each round of coordinate updates: $stress = \sqrt{\frac{\sum_{i,j,i\neq j}\left(d_{ij}-\hat{d}_{ij}\right)^2}{\sum_{i,j,i\neq j} d_{ij}^2}}$ to compute the overall fitness of the coordinates. The stress value illustrates well the convergence of algorithms. We collect the performance of different methods when they converge to stable positions. We repeat the simulations in ten times and report the average results and the 95-th confidence intervals.

Figure 7 plots the stress values as we vary the percent p of nodes having neighbors with randomized coordinates.

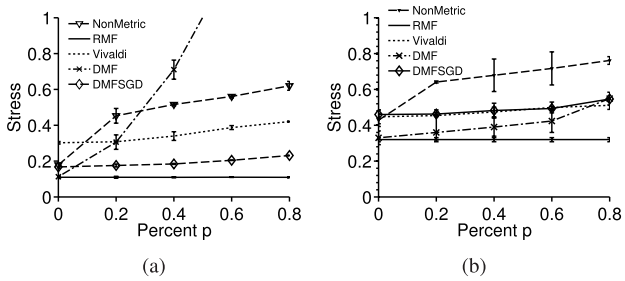


Fig. 7. The stress values of compared methods as we vary the percent p of nodes having neighbors with random coordinates. (a) P2P1143. (b) Meridian2500.

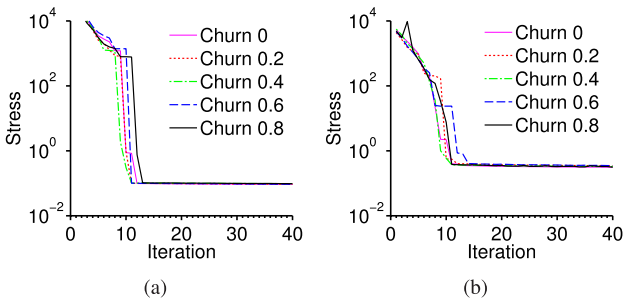


Fig. 8. The convergence of RMF as we change the percent of non-landmark neighbors that have churns from 0 to 0.8. The y-axis is shown in a logarithmic scale. (a) P2P1143. (b) Meridian2500.

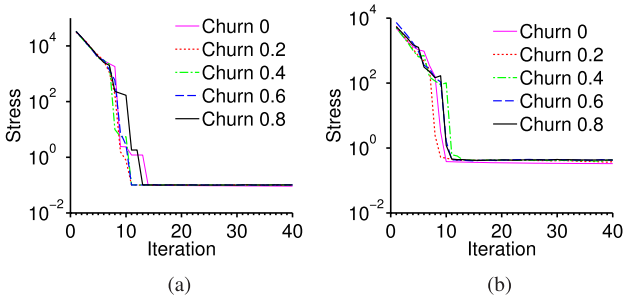


Fig. 9. The convergence of RMF as we change the percent of landmarks that have churns from 0 to 0.8 with respect to the whole number of landmarks. The y-axis is shown in a logarithmic scale. (a) P2P1143. (b) Meridian2500.

RMF does not degrade its accuracy with increasing neighbors having randomized coordinates, since RMF only exchanges the relative coordinates that do not contain prediction errors.

DMF, DMFSGD, NonMetric and Vivaldi become less inaccurate with increasing percent of nodes having neighbors with randomized coordinates, since each node uses neighbors' coordinates to adjust its own position. Vivaldi, Non-Metric and DMFSGD use a weight parameter tune the impact of neighbors' coordinates. However, we see that the accuracy of Vivaldi, Non-Metric and DMFSGD degrade severely when neighbors have randomized coordinates, since the weight parameters may not truthfully approximate the accuracy of coordinates due to the lack of ground-truth coordinates.

C. Convergence Under Churns

We next validate RMF's performance under churns. Generally, there may be two kinds of neighbors in RMF, i.e., the landmarks that are used to create the relative coordinate and non-landmark nodes that are used to adjust the parameter matrix. Both landmarks and non-landmark neighbors may have churns, thus we evaluate RMF's performance in two cases. We set the number of landmarks to 16 and the number of non-landmark nodes to 16. Varying the parameters does not change the conclusions.

(1) **Non-landmark churn**: First, we evaluate RMF's stability under the churns of the non-landmark neighbors. We assign a number of randomly sampled nodes from the data set as the landmarks for each node in the data set and keep the landmarks of each node to be online during the experiments. We set the relative coordinate of each node in the data set as the vector of RTTs towards the corresponding landmark neighbors.

We randomly sample a node i from the data set and let node i join the system at the first round. We sample a number of nodes from the data set as the non-landmark neighbors of node i at the first round. To model the churns of non-landmark neighbors in each round, we choose a fraction of node i's non-landmark neighbors uniformly at random and remove these neighbors from node i's neighbor set. Then, we sample the same number of nodes from the data set into node i's neighbor set. We vary the choice of node i in the data set and calculate the average stress values of the coordinate system. Varying the fraction of offline neighbors allows us to model fine-grained degrees of churns for each node.

Figure 8 shows the average stress values in each round as we vary the percent of non-landmark neighbors that have churns from 0 to 0.8. We see that RMF converges to approximately the same stress values after 10 to 15 rounds and keeps the coordinates at the converged state. This is because the coordinate updating process stochastically performs the gradient descendent optimization process that has nice convergence guarantees [36]. As we change the percent of neighbors that have churns, the number of rounds required to converge the coordinates varies. This is because of the variation of the updating steps with respect to different neighbors.

(2) **Landmark churns**: Having shown that RMF is stable under churns of non-landmark neighbors, we next test the convergence of RMF when the landmarks have churns. At the first round, we initialize the landmarks for each node in the data set as those randomly sampled from the data set. We construct the corresponding relative coordinate for each node. In order to model the churns of the landmarks in each round, we set a fraction of landmarks for each node to be offline, and replace these offline landmarks using the same number of nodes that are randomly sampled from the data set. Then we update the components of each relative coordinate that correspond to offline landmarks using the latency values from the current node to newly sampled landmarks.

We vary the fraction of landmarks that have churns for each node from 0 to 0.8 and calculate the average stress value of the whole coordinate system in each round. From Figure 9, we see that RMF converges to approximately the same stress values within 10 to 15 rounds and is stably accurate afterwards. This is because the components of each relative coordinate are drawn from the same RTT distributions from the current node to all other nodes, consequently, the expected values of the relative-coordinate components do not vary much due to the churns of the landmarks. As a result, the coordinate updating

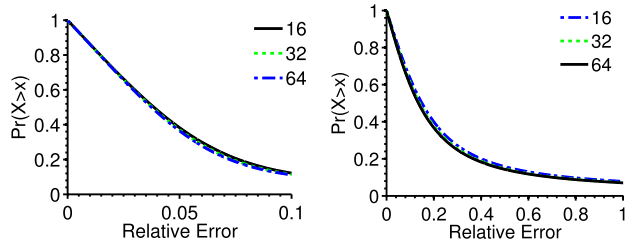


Fig. 10. The CCDFs of the relative errors as we vary the number of neighbors. (a) P2P1143. (b) Meridian2500.

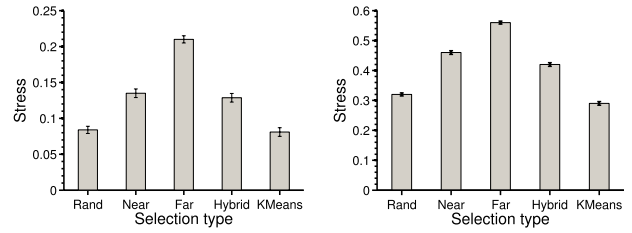


Fig. 11. The stress values of RMF with different neighbor-selection methods. For each node i, we consider five methods: (i) "Rand": select neighbors for node i uniformly at random from all nodes. (ii) "Near": select neighbors nearest to node i. (iii) "Far": select neighbors farthest to node i. (iv) "Hybrid": select neighbors for node i half via the "Near" approach and the other half via the "Far" approach. (v) "KMeans": select neighbors for node i uniformly from each cluster created by applying the K-means clustering method on the RTT matrix. (a) P2P1143. (b) Meridian2500.

process monotonically decreases the prediction error of the parameter matrix with respect to the relative coordinates.

In summary, despite of the churns caused by the non-landmark and landmark neighbors, RMF converges to stable positions within a small number of updating rounds and keeps to be stably accurate after the convergence.

D. Sensitivity Analysis

We next test the variation of RMF's accuracy as we change its parameters. We also found that varying the constant α in Eq (15) does not significantly change the accuracy.

1) Size of Neighbor Set: Figure 10 plots the CCDFs of RMF's relative errors as we increase the size of the neighbor set. We see that RMF keeps stable accuracy when we set the number of neighbors among 16, 32 and 64. This is because the RTT matrix is approximately low rank and we can approximate it with low-dimensional coordinate matrices [15]. As a result, we can use a modest number of neighbors with low bandwidth costs.

2) Neighbor Selection Methods: We next study whether RMF's accuracy can be improved by selecting different set of neighbors. Figure 11 plots the stress values as we vary the choices of neighbors. We see that the Rand and KMeans approaches provide much higher accuracy than the other methods. This is because by choosing the neighbors among a wider range of the network latency space, we can avoid the correlations of RTTs from a node to its neighbors.

E. CPU Efficiency of the RMF Method

We evaluate the CPU efficiency of calculating the parameter matrix as we increase the number of neighbors. We calculate

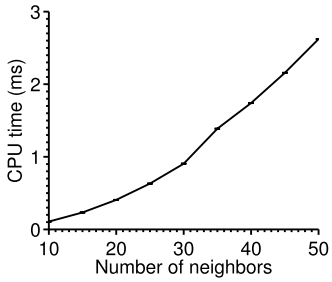


Fig. 12. The CPU time of updating one round of the parameter matrix as we vary the number of landmarks on the Meridian2500 data set.

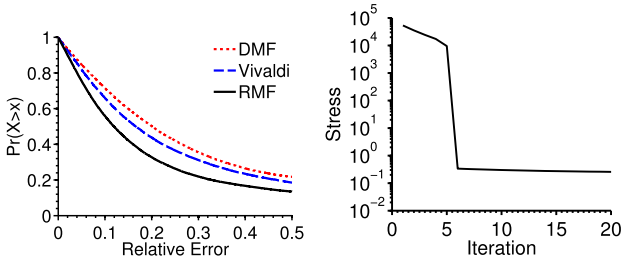


Fig. 13. Latency prediction on the PlanetLab. (a) Accuracy comparison. (b) Convergence of RMF.

the average CPU time of updating one round of the parameter matrix. Figure 12 shows that the CPU time as we increase the number of neighbors. We can see that the mean CPU time increases, since increasing the number of neighbors enlarges the dimension of the vectors used in updating the parameter matrix.

VIII. PlanetLab Experiments

Having presented RMF's performance via simulations, we next deploy RMF on the PlanetLab to conduct real-world deployment experiments. We have implemented a prototype program for RMF in Java. This program works as a daemon thread to predict the network latencies. We select 225 PlanetLab servers and install the RMF prototype program on them. We use a PlanetLab server to store the pairwise RTT values between PlanetLab servers as the ground-truth RTT values. We also implemented the Vivaldi and DMF network coordinate methods for comparison. All comparison methods use the same parameters in section VII. Each node joins the coordinate system at the beginning of the experiment through a set of bootstrap nodes and updates its coordinates in an asynchronous manner.

We first compare the accuracy of predicting the pairwise RTTs. Figure 13(a) plots the CCDFs of the relative errors of three methods. We see that RMF improves the prediction accuracy for more than 90% of node pairs compared to DMF and Vivaldi. Bypassing the propagation of coordinate errors helps RMF avoid bad local minimum.

We next show the convergence of RMF. Figure 13(b) plots the stress values for each round of coordinate updates for RMF. We see that RMF converges with less than ten rounds, which is consistent with the simulation results.

IX. Conclusion

We have addressed the instability issue of the network coordinate methods caused by churns. We propose a distributed

network-coordinate method named RMF that uses the relative coordinate method to avoid the instability due to erroneous coordinates. Further, RMF utilizes the matrix factorization method to tolerate the TIV phenomenon. RMF exchanges only relative coordinates between neighbors ensuring the convergence among decentralized nodes. Simulation results and the PlanetLab experiments show that RMF converges under churns and reduces the relative errors of the RTT estimations for more than 90% of all node pairs. RMF scales well and is robust to failures of neighbors. As a result, we believe RMF is suitable for decentralized latency prediction under churns.

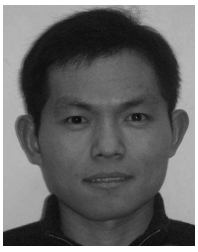
Although RMF guarantees to stabilize in large-scale and dynamic distributed systems, RMF may be destabilized if the relative coordinates are manipulated by adversaries, since RMF relies on the relative coordinates to update the coordinates. Guaranteeing the coordinate stabilization in the adversary environment is an open problem.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their very constructive comments.

REFERENCES

[1] S. Agarwal and J. R. Lorch, "Matchmaking for online games and other latency-sensitive P2P systems," in *Proc. SIGCOMM*, 2009, pp. 315–326.
[2] B. Wong and E. G. Sirer, "ClosestNode.com: An open-access, scalable, shared geocast service for distributed systems," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 62–64, 2006.
[3] Y. Fu, Y. Wang, and E. Biersack, "HybridNN: An accurate and scalable network location service based on the inframetric model," *Future Generat. Comput. Syst.*, vol. 29, no. 6, pp. 1485–1504, 2013.
[4] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. INFOCOM*, 2002, pp. 170–179.
[5] Y. Mao, L. K. Saul, and J. M. Smith, "IDES: An Internet distance estimation service for large networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2273–2284, Dec. 2006.
[6] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. SIGCOMM*, 2004, pp. 15–26.
[7] Y. Liao, P. Geurts, and G. Leduc, "Network distance prediction based on decentralized matrix factorization," in *Proc. IFIP Netw.*, 2010, pp. 15–26.
[8] Y. Chen *et al.*, "Phoenix: A weight-based network coordinate system using matrix factorization," *IEEE Trans. Netw. Service Manage.*, vol. 8, no. 4, pp. 334–347, Dec. 2011.
[9] Y. Liao, W. Du, P. Geurts, and G. Leduc, "DMFSGD: A decentralized matrix factorization algorithm for network distance prediction," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1511–1524, Oct. 2013.
[10] A. Mukherjee, "On the dynamics and significance of low frequency components of Internet load," *Internet-Working, Res. Exper.*, vol. 5, no. 4, pp. 163–205, Dec. 1992.
[11] K. C. Claffy, G. C. Polyzos, and H.-W. Braun, "Measurement considerations for assessing unidirectional latencies," *Internet-Working, Res. Exper.*, vol. 4, no. 3, pp. 121–132, Sep. 1993.
[12] Y. Zhang and N. Duffield, "On the constancy of Internet path properties," in *Proc. 1st ACM SIGCOMM Workshop Internet Meas.*, 2001, pp. 197–211.
[13] S. Dolev, *Self-Stabilization*. Cambridge, MA, USA: MIT Press, 2000. [Online]. Available: http://books.google.de/books?id=UPdnRDX-ygQC
[14] S. Hotz, "Routing information organization to support scalable interdomain routing with heterogeneous path requirements," Ph.D. dissertation, Dept. Comput. Sci., Univ. Southern California, Los Angeles, CA, USA, 1994.
[15] B. Donnet, B. Gueye, and M. A. Kaafar, "A survey on network coordinates systems, design, and security," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 4, pp. 488–503, 4th Quart. 2010.
[16] J. D. Guyton and M. F. Schwartz, "Locating nearby copies of replicated Internet servers," in *Proc. SIGCOMM*, 1995, pp. 288–298.
[17] L. Tang and M. Crovella, "Virtual landmarks for the Internet," in *Proc. IMC*, 2003, pp. 143–152.
[18] H. Lim, J. C. Hou, and C.-H. Choi, "Constructing Internet coordinate system based on delay measurement," *IEEE/ACM Trans. Netw.*, vol. 13, no. 3, pp. 513–525, Jun. 2005.
[19] T. S. E. Ng and H. Zhang, "A network positioning system for the Internet," in *Proc. USENIX Annu. Tech. Conf.*, 2004, p. 11.
[20] M. Costa, M. Castro, A. Rowstron, and P. Key, "PIC: Practical Internet coordinates for distance estimation," in *Proc. IEEE ICDCS*, 2004, pp. 178–187.
[21] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in Euclidean space," *IEEE/ACM Trans. Netw.*, vol. 12, no. 6, pp. 993–1006, Dec. 2004.
[22] P. Key, L. Massoulié, and D.-C. Tomozei, "Non-metric coordinates for predicting network proximity," in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 11–15.
[23] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality and routing policy violations in the Internet," in *Proc. PAM*, 2009, pp. 45–54.
[24] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality variations in the Internet," in *Proc. IMC*, 2009, pp. 177–183.
[25] G. Wang, B. Zhang, and T. S. E. Ng, "Towards network triangle inequality violation aware distributed systems," in *Proc. IMC*, 2007, pp. 175–188.
[26] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 1996.
[27] J. Ledlie, P. Pietzuch, and M. Seltzer, "Stable and accurate network coordinates," in *Proc. IEEE ICDCS*, Jul. 2006, p. 74.
[28] J. Ledlie, P. Gardner, and M. Seltzer, "Network coordinates in the wild," in *Proc. USENIX NSDI*, 2007, p. 22.
[29] G. Wang and T. S. E. Ng, "Distributed algorithms for stable and secure network coordinates," in *Proc. IMC*, 2008, pp. 131–144.
[30] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statist. Soc. B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.
[31] C. Ding, T. Li, W. Peng, and H. Park, "Orthogonal nonnegative matrix Tri-factorizations for clustering," in *Proc. ACM KDD*, 2006, pp. 126–135.
[32] A. Demers *et al.*, "Epidemic algorithms for replicated database maintenance," in *Proc. ACM PODC*, 1987, pp. 1–12.
[33] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, 2007, Art. no. 8.
[34] (Oct. 2010). *Pairwise Latencies Between DNS Servers*. [Online]. Available: http:/pdos.csail.mit.edu/p2psim/kingdata/
[35] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A lightweight network location service without virtual coordinates," in *Proc. SIGCOMM*, 2005, pp. 85–96.
[36] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*, D. Saad, Ed. Cambridge, U.K.: Cambridge Univ. Press, Oct. 2012. [Online]. Available: http://leon.bottou.org/papers/bottou-98x



Yongquan Fu is currently a Lecturer with the Science and Technology Laboratory of Parallel and Distributed Processing, National University of Defense Technology. His current research interests lie in the areas of network measurement, cloud performance measurements, and distributed systems.

Xu Xiaoping is currently a Senior Engineer with the National University of Defense Technology. His research interests include software theory and engineering.