ABSTRACT

Title of dissertation: DECENTRALIZED NETWORK BANDWIDTH PREDICTION AND NODE SEARCH

Sukhyun Song, Doctor of Philosophy, 2012

Dissertation directed by: Professor Alan Sussman
Department of Computer Science

As modern computing becomes increasingly data-intensive and distributed, it is becoming crucial to effectively manage and exploit end-to-end network bandwidth information from hosts on wide-area networks. Inspired by the finding that Internet bandwidth can be represented approximately in a tree metric space, we focus on three specific research problems.

First, we have designed a decentralized algorithm for network bandwidth prediction. The algorithm embeds the bandwidth information as distance in an edge-weighted tree, without performing full n-to-n measurements. No central and fixed infrastructure is required. Each joining node performs a limited number of sampling measurements. Second, we designed a decentralized algorithm to search for a centroid node that has high-bandwidth connections with a given set of nodes. The algorithm can find a centroid accurately and efficiently using the bandwidth data produced by the prediction algorithm. Last, we have designed another type of decentralized search algorithm to find a cluster of nodes that have high-bandwidth interconnections. While the clustering problem is NP-complete in a general graph, our algorithm runs in polynomial time with the bandwidth data predicted in a tree metric space. We provide proofs that our algorithms for bandwidth prediction and node search have perfect accuracy and high scalability when a network is modeled as a tree metric space. Also, experimental results with real-world data sets validate the high accuracy and scalability of our approaches.

DECENTRALIZED NETWORK BANDWIDTH PREDICTION AND NODE SEARCH

by

Sukhyun Song

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2012

To my wife, Minsun, for her love and support.

To my parents and brothers.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Alan Sussman. He always trusted me and encouraged my academic aspirations. I would also like to thank my co-advisor, Dr. Peter J. Keleher for his excellent feedback and encouragement. Without their guidance and support, my dissertation could not be completed. I am grateful to Dr. Bobby Bhattacharjee for stimulating my enthusiasm of academic research. Thanks are due to Dr. Jeffrey K. Hollingsworth and Dr. Derek C. Richardson for sparing their invaluable time serving on my thesis committee. I give my special thanks to Dr. Kyung Dong Ryu for his advice. I could learn from him how to conduct research for the first time.

In the long journey of my study, I have met many wonderful people. I am especially grateful to my sincere friends Inseok Choi and Sungwoo Park. We have shared many joys and sorrows of life. I would also like to thank my colleagues and friends — Jaehwan Lee, Ilchul Yoon, Gary Jackson, Teng Long, Jik-Soo Kim, Minkyoung Cho, Jae-Yoon Jung, Eunhui Park, Joonghoon Lee, Hyejung Lee, Fatih Kaya, John Hwang, Wei He, Taewoong Kim, Kyungjin Yoo, Sang Chul Song, Dov Gordon, Ananta Tiwari, Nick Rutar, Geoffrey Stoker, and Chris Hayden.

Finally, I extend my hearty thanks to my family. My mother and father have always stood by me. My brothers, Seok-Je and Sukjoon have been a great source of support and motivation in many ways. I am also grateful to my parents-in-law for always being supportive. My deepest thanks must go to my beautiful wife, Minsun for her endless love, support, and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Modern computing applications are increasingly more data-intensive and widely distributed. As the *fourth paradigm* of data-intensive science is emerging [33], grid applications being employed to effectively manage the data flood caused by scientific simulations and experiments. For example, wide-area computing resources in TeraGrid infrastructure [10] are used for earthquake forecasting in the Cyber-Shake project [20]. Scientists schedule data-intensive tasks on grid resources using distributed workflow frameworks such as Pegasus [28] and DAGMan [4] and analyze a large amount of sensor data. Multi-player online games such as Second Life [8] are another example of data-intensive applications. Since central dedicated servers cannot handle huge traffic to send 3-d object data to massive clients, several studies [42, 68] propose to distribute the role of central servers through peer-to-peer (P2P) technologies. Also, file-sharing applications such as BitTorrent [24, 3] are used to share a huge amount of data among globally distributed users.

Network bandwidth is a key performance factor in data-intensive applications because it determines data transfer times for large data sets. Since bandwidth is relatively low and widely varying in wide-area networks, many applications want to utilize high-bandwidth connections.

1.2 Important Problems in Bandwidth Management

This dissertation considers research issues for effective network bandwidth management. The ultimate goal is to support data-intensive wide-area applications to exploit high-bandwidth connections. This section briefly describes three problems to be investigated in this dissertation, and introduces several applications that can benefit from solving the problems.

1.2.1 Decentralized Network Bandwidth Prediction

Data-intensive wide-area applications can increase their performance by identifying end-to-end network bandwidth and transferring data through high-bandwidth connections. Since bandwidth measurements are generally expensive to perform, it would not be desirable for applications to keep track of bandwidth information for all pairs of nodes. Thus, there is a need to design an algorithm to predict a full set of n-to-n bandwidth values from a limited number of sampling measurements. If the prediction algorithm could be executed in a completely decentralized fashion, it would be helpful to preserve scalability and reliability of distributed applications. We can apply a decentralized bandwidth prediction algorithm to peer selection in file sharing systems. In the current implementation of BitTorrent [24], each peer downloads file pieces from random peers. If BitTorrent runs the bandwidth prediction algorithm, peers can estimate quickly the bandwidth of each connection with other peers. Then each downloading peer can selectively choose high-bandwidth peers so that file pieces can be transferred at a high rate. It is expected to be challenging to find an effective method to achieve high accuracy in prediction data with only a small number of sampling measurements. Also, it would not be easy to design a decentralized algorithm that maintains distributed data structures and requires no landmark nodes for sampling measurements.

1.2.2 Decentralized Centroid Search

The second problem that this dissertation explores is searching for a node called a centroid under bandwidth constraints. A centroid is informally defined as a node connected via high-bandwidth connections to a given set of nodes. Suppose that we can design an efficient algorithm that finds centroids in a decentralized fashion. Then by sending or receiving data to or from a centroid node, applications can optimize data locality and thereby increase their overall performance. Applications that use super-peers to distribute system workloads can benefit from a decentralized centroid search algorithm. For example, P2P online games [42, 68] divide a game space and let super-peers coordinate game players in each divided game space. Using the centroid search algorithm, we can choose the centroid of the players in a divided game space as a super-peer. Then the super-peer will be able to send a large amount of object data to the players in the assigned game space at a high rate. It is not desirable to naively perform exhaustive searches in a large-scale wide-area network. So we have to develop an efficient search algorithm that can finish within a small number of network hops. Achieving highly accurate results with limited costs is another challenge. Also, we need to have the search algorithm exploit bandwidth prediction data in order to avoid measurement delays.

1.2.3 Decentralized Cluster Search

The third issue is for another type of node search algorithm. We want to design a decentralized algorithm to find a cluster of hosts that are all interconnected with high bandwidth. Searching for clusters is beneficial to data-intensive distributed applications as nodes in the same cluster can send large amount of data to one another at a high rate. Data-intensive grid computing is one example of such applications of a cluster search algorithm. Current desktop grid computing frameworks such as BOINC [13, 14] and P2P Grid [40, 39, 41, 44] can utilize large-scale wide-area computing resources. But those systems cannot execute data-intensive workflow or parallel jobs such as CyberShake [20] because it is not feasible to transfer large-scale data over low-bandwidth wide-area networks. Local cluster frameworks such as Condor [48] and workflow management systems such as Pegasus [28] and DAGMan [4] can execute data-intensive jobs, require running on grid resources in a high-bandwidth local-area network. The cluster search algortihm can combine advantages of both local-area and wide-area grid technologies: we can schedule data-intensive jobs on virtual clusters of large-scale computing resources in a wide-area network. The clustering problem is difficult to solve, in that it is NP-complete for a general graph. So we have to find an approximate algorithm that guarantees a given degree of accuracy. Also, it is challenging to design a decentralized algorithm that finishes within a small number of network hops.

1.3 Thesis Statement and Contributions

With these motivations, my thesis is that *a decentralized approach can be employed to predict end-to-end network bandwidth and search for nodes under network bandwidth constraints in an accurate and scalable way.* In support of this thesis, I made the following contributions:

- A decentralized bandwidth prediction algorithm

  We can accurately predict bandwidth information with only $O(\log^2 n)$ sampling measurements for each node. The algorithm is completely decentralized: all the data structures are distributed, and there exists no landmark node for sampling measurements. Theoretical proofs for correctness and performance are also provided. We also have confirmed that our approach performed more accurately than the existing approaches with extensive simulations.

- A decentralized algorithm to find centroids

  The algorithm runs based on the prediction data so that we can avoid measurement delays. We theoretically prove that a centroid can be found accurately in a decentralized fashion within $O(\log n)$ network hops. Simulation results are also provided to show the high accuracy and scalability of our approach.

- A decentralized algorithm to find clusters

  We first show that the clustering problem can be solved in polynomial time if we use the bandwidth data produced by the prediction framework. Then we describe a decentralized algorithm with theoretical analyses to show that the algorithm can find clusters accurately within $O(\log^2 n)$ network hops. Simulation results confirm that our approach achieves high accuracy and scalability.

1.4 Thesis Organization

The rest of the dissertation is organized as follows. We first discuss the underlying intuition behind this work in Chapter 2. Chapter 3 presents research related to the dissertation. Chapters 4, 5, and 6 discuss our approaches for each of three research problems described in Section 1.2. Chapter 7 discusses future work. Finally, we conclude in Chapter 8.

Chapter 2

Background

This chapter defines terms and provides background information that is needed to understand our overall approach. Three types of end-to-end network bandwidth measures are first presented. Then we describe how bandwidth can be represented in a tree metric space. We also introduce the skip list data structure that is used as the basis for our approach for bandwidth prediction.

2.1 Types of Network Bandwidth

The term *end-to-end network bandwidth* of a network path can refer to related yet different measures [37]. The *capacity* is the maximum rate that the path can provide to a flow, when there is no other traffic in the path. The *available bandwidth* is the maximum rate that the path can provide to a flow, without reducing the rate of the rest of the traffic in the path. The *bulk transfer capacity* is the maximum rate that a TCP transfer can attain over the path.

Among the three measures, we consider predicting the available bandwidth between the Internet nodes and search for nodes under constraints of available bandwidth. Thus, this dissertation will use the term bandwidth to refer to the available bandwidth unless there is a special reason not to. Although we use the available bandwidth for prediction and node search, we expect that our approach would also work for the capacity and the bulk transfer capacity as all three measures have some degree of correlation. Yalagandula et. al [71] show that there is a strong correlation between the capacity and the available bandwidth, so orderings of nodes for both measures are similar. Also, a high-capacity path tends to have high bulk transfer capacity.

2.2 Network Bandwidth in a Tree Metric Space

As a tree metric space is a kind of metric space, we first discuss how network bandwidth can be represented in a metric space. Then we will discuss why a tree metric space is a good model to embed network bandwidth information.

Definition A *metric space* is an ordered pair (V, d) where V is a set of nodes and d is a *metric* or *distance function* on V such that for any $u, v, w \in V$, the following properties hold:

1. $d(u, v) \geq 0$ (*non-negative*)
2. $d(u, v) = 0$ if and only if $u = v$ (*identity of indiscernibles*)
3. $d(u, v) = d(v, u)$ (*symmetry*)
4. $d(u, w) \leq d(u, v) + d(v, w)$ (*triangle inequality*)

Higher values are considered better for bandwidth while closer is generally more desirable for distance in a metric space. So, Ramasubramanian et. al [58] used the *linear transform function* $d(u, v) = C - BW(u, v)$ to represent bandwidth as a metric, where $BW(u, v)$ is the bandwidth between nodes u and v, $d(u, v)$ is the distance in a metric space, and C is a constant. Representing bandwidth as a metric satisfies the four properties of metrics. The first property is satisfied by having a large value for C, for example, the expected maximum bandwidth. By setting $BW(u, u) = C$, we can also satisfy the second property. We satisfy the third property by setting both $BW(u, v)$ and $BW(v, u)$ to the average bandwidth of the forward and reverse directions. So our research focus on predicting and utilizing the average bandwidth values. Even though no effective method has been found to directly address the last assumption, we provide several heuristics to accurately embed bandwidth information into a metric space in the real Internet, as described in Section 4.3.

For further discussions, the following definitions about tree metric space are provided.

Definition An *edge-weighted tree* is a connected graph without cycles, and with non-negative edge weights.

Definition The *distance* between two nodes u and v on an edge-weighted tree T, denoted $d_T(u, v)$, is defined as the sum of the weights of the edges on the path from u to v.

Definition An edge-weighted tree T *induces* a metric space (V, d) if and only if T contains all nodes in V and $\forall u, v \in V$, $d(u, v) = d_T(u, v)$ holds.

Definition A metric space (V, d) is called a *tree metric space* if there exists an edge-weighted tree that induces (V, d).

Definition The *four-point condition (4PC)* on a metric space (V, d) states that for any set of four nodes $w, x, y, z \in V$, $d(w, x) + d(y, z) \leq d(w, y) + d(x, z) \leq d(w, z) + d(x, y)$ implies $d(w, y) + d(x, z) = d(w, z) + d(x, y)$.

Theorem 2.1. *A metric space* (V, d) *is a tree metric space if and only if* (V, d) *satisfies 4PC.*

There are three pieces of evidence to verify that the Internet is close to a tree metric space for bandwidth. First, Ramasubramanian et. al [58] verify that a bandwidth data set produced many small ε values. ε was introduced by Abraham et. al [11] to quantify how closely a set of four nodes satisfies 4PC. If all ε values in a metric space are zero, the metric space is a perfect tree metric space. Second, there is a theoretical model of network topology such that bandwidth between two nodes is bottlenecked in the first hop of a routing path. This "edge-bandwidth" network model is proposed by Ramasubramanian et. al [57] based on Hu et. al's empirical research [34] showing that 60% of paths between random end hosts in the wide-area Internet have a bottleneck in the first or second hop. And it has been proved that a metric space for this model is a perfect tree metric space [57]. Last, an attempt to embed bandwidth into a tree metric space has resulted in a high accuracy. Theorem 2.1, proved by Buneman [21], shows the relationship between 4PC and a tree metric space. Based on Theorem 2.1, Ramasubramanian et. al [58] constructed an edge-weighted tree to embed bandwidth measurements into. Their results showed low relative errors of the embedded bandwidth values compared to the real data.

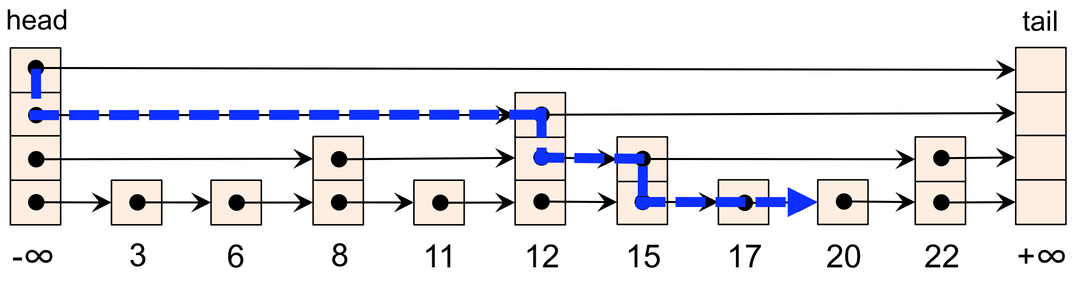


Figure 2.1: An example of skip list data structure

2.3 Skip List

A *skip list* [55, 56] is a randomized data structure to store items represented by keys. The list is organized as multiple levels of linked lists of nodes. Each node has a tower of pointers, each of which points to the next node in a linked list at each level. Each node in level $i-1$ appears in level i with a fixed probability of $\frac{1}{2}$ (or more generally p), which results in a tower height bounded by $O(\log n)$ with high probability. [1] So the linked lists at higher levels are sparser than lists in lower levels, and are used as *express lanes* for traversing a sequence of nodes. A skip list searches for a node with an input key by traversing nodes starting from the head towards the tail, and terminates in $O(\log n)$ time with high probability. The search algorithm starts at the top-level list and moves down to a lower-level list when the next node in the current level has a key larger than the input key.

Figure 2.1 shows an example of skip list data structure. Each node has a skip

[1] An event occurs *with high probability* if, for any $\alpha \geq 1$, the event occurs with probability at least $1 - \frac{c_\alpha}{n^\alpha}$, where c_α depends only on α.

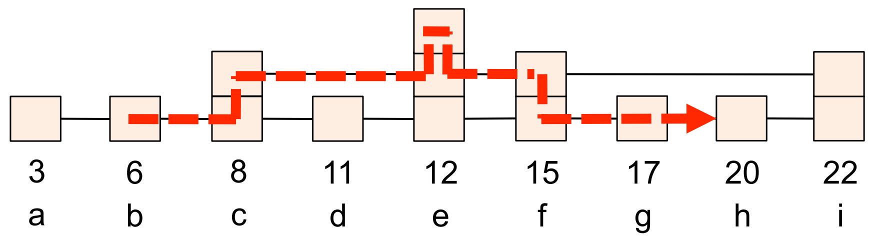


Figure 2.2: A skip list as a distributed structure

list tower that is represented as a set of boxes, each of which contains a pointer to the next nodes at each level. Numbers at the bottom represent node keys. The thick dashed arrow shows a path to search for a node with the key of 20. The search algorithm starts at the top level of the head and “skips” some nodes by utilizing high-level pointers until reaching at the node with key 20.

We use a skip list as a distributed set of doubly-linked lists that connects computers in a network. Each node provides other nodes with a distributed operation, denoted by findPred(k), to search for a *predecessor* of an input key k. A predecessor of key k is defined as a node that has the maximum key among nodes with key $\leq k$ in a skip list. The predecessor search operation starts at any node and traverses either right towards a tail or left towards a head. At each traversed node, if k is equal to the key of the current node, the operation returns the current node as a predecessor of key k. When k is larger than the key of the current node, 1) if the next node in the level-0 list has key $> k$, the current node is determined as a predecessor of k, or 2) the operation moves to the next node in the highest-level list that has a key $\leq k$. Similarly, when k is smaller than the key of the current node, 1) if the previous node in the level-0 list has key $< k$, the previous node at level zero is determined as a predecessor of k, or 2) the operation moves to the previous node in the highest-level list that has a key $\geq k$. This search operation is divided into two phases. The first phase is before the traversal moves down to a lower-level list for the first time, so the traversal level stays the same or increases. The second phase is the rest of the procedure where the traversal level stays the same or decreases until the predecessor is found. Since each phase follows the similar procedure to the original search algorithm of skip list, the predecessor search operation will complete in $O(\log n)$ hop counts. Likewise, each node provides findSucc(k) to search for a *successor* that has the minimum key among nodes with key $\geq k$.

Figure 2.2 shows an example of skip list that is used as a distributed data structure. Each node in a network has a skip list tower that contains pointers to neighbors at each level. Alphabets at the bottom represents node id's, and numbers above the id's are node keys. The thick dashed arrow shows a network path that an operation b.findPred(20) traverses to search for a node with the key of 20 starting at node b. The path from node b to e shows the first phase of non-decreasing skip list level, and the path from node e to h is for the second phase. Note that h.findPred(6) will exactly reverse the path in the figure from h to b.

Chapter 3

Related Work

This chapter discusses other methods for network distance prediction and node search.

3.1 Tree Metric Approaches for Network Distance Prediction

The original theoretical underpinnings of tree metric spaces were provided by Buneman [21], including the first constructive algorithm to induce tree metric spaces. However, Buneman's algorithm does not allow nodes to be incrementally added to existing edge-weighted trees. Since the result edge-weighted tree would not be expandable, we cannot directly apply the algorithm in practice to a dynamically changing real-world system.

The theoretical work of Abraham et. al [11] proposes a tree construction algorithm for an approximate tree metric space and provides upper and lower bounds on the accuracy of tree embedding. Since their approach uses a non-incremental recursive algorithm, it suffers the same problem as Buneman's algorithm for practical uses. Another reason why that algorithm is not feasible in the real world is that the algorithm uses a predetermined parameter ε. The parameter must be calculated for an input metric space before the algorithm is executed, but there is no way to predetermine the parameter in the real world because it requires information about which nodes will join the system. Instead of using such a predetermined parameter, our algorithm uses several heuristics to achieve high accuracy for an imperfect tree metric space.

Our research is inspired by the Sequoia system [58], which uses a tree-embedding model for bandwidth prediction and proposes an incremental iterative tree construction algorithm for the first time. We naturally use the same terms as the Sequoia authors do to explain our algorithm even though some terms have somewhat different meanings. However, our study has several contributions relative to Sequoia. First, our system is decentralized and does not require any fixed infrastructure. To participate in the Sequoia system, each node must measure bandwidth with several nodes starting from a single fixed node called the lever node, and this can cause a load imbalance problem. On the other hand, each node joins our system by performing sampling measurements starting at a random node. Second, Sequoia has a scalability problem as each node needs to perform $O(n)$ sampling measurements in the worst case. On the other hand, our new approach described in this dissertation is highly scalable. Each node in our system performs $O(\log^2 n)$ sampling measurements with high probability. Last, we provide novel heuristics that increase prediction accuracy in a real world network. Sequoia uses an algorithm that fits a perfect tree metric space directly in practice, and results in low accuracy, as shown in Section 4.4. Even though Sequoia addresses this inaccuracy problem by constructing multiple trees, multiple trees will cause a significant amount of extra measurement workload. Our techniques succeed in achieving high accuracy for real-world networks without imposing significant overhead.

Before developing the prediction framework described in Chapter 4, we designed another decentralized system [64, 65] that can predict pairwise bandwidth on a tree metric space. Unlike Sequoia [58], our original approach [64, 65] does not require any landmark nodes for sampling measurements. More specifically, while each node in Sequoia starts sampling measurements at the root node of a data structure called an anchor tree, our old system allows each node to start at a random node. Also, our old system achieves higher accuracy than Sequoia by introducing several heuristics similar to those used by our new approach described in this dissertation. However, our old approach is not highly scalable. Each node needs to store information of size $O(n)$ and perform $O(n)$ sampling measurements in the worst case. which implies an inefficient prediction method. On the other hand, each node in the algorithm described in Chapter 4 only stores information of size $O(\log n)$ and performs $O(\log^2 n)$ sampling measurements with high probability.

There is another system called PathGuru [73] for bandwidth prediction, which embeds bandwidth information in an ultrametric space. Unlike our approach, PathGuru is a landmark-based system. More importantly, PathGuru provides quite poor accuracy [19], even worse than Sequoia. An ultrametric space (V, d) satisfies the *three-point condition (3PC)* that states that for any set of three nodes $x, y, z \in V$, $d(x, y) \leq d(x, z) \leq d(y, z)$ implies $d(x, z) = d(y, z)$. An ultrametric space is a kind of tree metric space because 3PC is a stronger assumption than 4PC of tree metric space. Thus tree metric approaches can cope more accurately than PathGuru with bandwidth data that violate the ultrametric space assumption.

3.2 Non-Tree Metric Approaches for Network Distance Prediction

There have been several research efforts on coordinate-based latency prediction. Each node is assigned synthetic coordinates in an Euclidean space. End-to-end latencies are then estimated as the distance between the coordinates of a pair of nodes. GNP [53], which pioneered this research area, calculates coordinates of each node relying on a small number of landmark nodes. Vivaldi [26] and PIC [25] eliminate such designated landmark nodes and provide a decentralized algorithm to compute network coordinates. All of these systems show high accuracy in latency prediction. However, the coordinates-based approach does not work well for bandwidth prediction, and accordingly, attempts to use Vivaldi for bandwidth prediction result in poor accuracy [57, 58]. On the other hand, approaches [58, 65] based on a tree metric space, including the approach in this dissertation, can accurately predict both latency and bandwidth.

To cope with violations of the triangular inequality, IDES [51] considers non-metric embeddings and predicts end-to-end network latency by using the matrix factorization technique. As the IDES system requires a set of landmark nodes, a system called DMF [46] has been proposed for decentralized matrix factorization. Although DMF is initially designed for latency prediction, one study [19] shows that DMF can predict network bandwidth more accurately than coordinate-based approaches. Despite its successful bandwidth prediction, DMF has two downsides compared to our approach. First, DMF executes multiple iterations to adjust prediction data until reaching convergence. Multiple iterations are needed even when predicting a static snapshot of network data, which results in high communication costs among participating nodes. On the other hand, our approach computes all the prediction data correctly after each node finishes its sampling measurements and performs the join process. Also, search algorithms for centroids or clusters have not been explored based on the prediction results of DMF. It is presumably not easy to find efficient distributed algorithms for node search on non-metric spaces.

Last-mile [19] is a decentralized system that predicts end-to-end network bandwidth, and shows higher accuracy than Sequoia [58] and Vivaldi [26]. Last-mile is based on the assumption of an edge-bandwidth network model that is described in Chapter 2. So each node is characterized by two values of incoming and outgoing bandwidth, and the bandwidth between two nodes is determined as the minimum of the incoming bandwidth of one node and the outgoing bandwidth of the other. The approach of last-mile is too simple to cover many practical cases that violate the edge-bandwidth assumption. It will be difficult to predict all end-to-end bandwidth values accurately with only two simple values per each node. Since motivations of our tree metric approach include the edge-bandwidth assumption as described in Chapter 2, our approach would be able to cover more varied data sets with high accuracy than last-mile. In addition, last-mile needs high communication costs coming from multiple convergence steps just like DMF. Also, unlike our approach, it will not be easy to extend last-mile to search for nodes because last-mile produces prediction data on a non-metric space.

3.3 Node Search Algorithms

There are many approaches to search for nodes with network metric constraints, but most of them [43, 59, 69] are about searching for the nearest neighbors for a given node. As network coordinates put a geometric meaning to node search problems, we can try to find centroids in a distributed fashion by building an overlay network on Euclidean coordinate spaces. For example, Sherpa [50] builds an overlay network on Vivaldi [26] following a Voronoi diagram. Then it finds a node that minimizes a given cost function using the gradient decent method. So Sherpa can be applied to solve the centroid search problem by searching for the nearest node to a virtual centroid point of its input nodes. However, a Euclidean space is a good model only for network latencies, so searching for centroids on a Euclidean space for bandwidth will result in low accuracy. To our best knowledge, there exist no efficient approaches to search for centroids using network bandwidth. A possible centralized approach is to perform exhaustive searches in the entire system based on prediction data provided by Sequoia [58] or last-mile [19]. We provide a decentralized, accurate, and low-cost algorithm to find bandwidth-constrained centroids.

Chapter 6 describes an algorithm to find a cluster of k nodes with minimum interconnection bandwidth b. The cluster search problem is not easy to solve because it is equivalent to the k-clique problem. k-clique is a well-known problem as NP-complete and is about finding a clique of size k in an undirected graph G, where a clique in G is a complete sub-graph of G. We can show that k-clique is equivalent to the cluster search problem by creating an undirected graph with V and adding an edge between nodes $u, v \in V$ such that the average of forward and reverse bandwidth between u and v is greater than or equal to a bandwidth constraint b. Before developing the cluster search algorithm described in Chapter 6, we designed another decentralized algorithm [66]. Our old algorithm employs the similar approach to the new algorithm described in this dissertation, finding clusters based on bandwidth data that are accurately embedded in a tree metric space. So clustering accuracy should be similar in both approaches. However, the old algorithm runs on top of our old prediction framework [65] that is not highly scalable as described in Section 3.1. Accordingly, the old clustering algorithm inherits the scalability problems of the old prediction framework. Each node needs to store additional information of size $O(n)$ and requires $O(n)$ network hops to find a cluster in the worst case. On the other hand, each node in the algorithm described in Chapter 6 only stores information of size $O(\log n)$ and spends $O(\log^2 n)$ network hops with high probability.

There are several theoretical centralized approaches to find a set of k nodes with a maximum diameter in a 2-d Euclidean space. Aggarwal et. al [12] provide an $O(k^{2.5} n \log k + n \log n)$ algorithm, and Eppstein et. al [30] improved it to $O(k^2 n \log^2 k + n \log n)$. In spite of the beauty of these algorithms, we could not successfully use it for our problem because bandwidth does not fit Euclidean space well. Instead, we have designed an $O(n^3)$ algorithm to solve the clustering problem in a tree metric space that fits bandwidth better. It is an open question if the time complexity of our algorithm can be improved.

There have been several research efforts about resource clustering. Liu et. al [49] introduce a hierarchical structure and propose an approximate algorithm to answer queries. Like our approach, they support a query with two constraints of cluster size and diameter. However, they only consider latency-constrained clustering. Also, unlike our decentralized approach, they construct a centralized hierarchical structure. Shen et. al [63] present a hierarchical cycloid overlay (HCO) architecture for locality-preserving clustering. HCO is used to discover wide-area grid resources with multiple attributes such as CPU speed and memory capacity. The difference from our approach is that their work only considers latency-constrained clustering, does not support a distance constraint for queries, and relies on a fixed set of landmark nodes to form clusters. Beaumont et. al [18] designed a distributed approximation algorithm for resource clustering and proved its correctness theoretically. They solved a problem to answer a query with both distance constraint and storage capacity constraint. However, they only provide an approximation, and restrict their work to a 1-d Euclidean space, which is not a good model to embed bandwidth measurements. SWORD [54] provides a decentralized algorithm to discover wide-area resources with multiple inter-node and per-node characteristics. Even though they consider both latency and bandwidth to find a cluster, they exhaustively search for clusters, require an exponential time, and stop searching when a timeout occurs. On the other hand, our approach guarantees answering a query in polynomial time under the assumption of a tree metric space.

Chapter 4

Decentralized Network Bandwidth Prediction

This chapter presents a framework for decentralized network bandwidth prediction. We first describe motivations and requirements of this work. Then the design of algorithm is provided with theoretical proofs of its correctness and performance. We will explain how distributed data structures are used and maintained for effective bandwidth prediction. Last, we will evaluate our approach with extensive experiments.

4.1 Introduction

We investigate a scalable and decentralized method to predict pairwise bandwidth without performing full n-to-n measurements. Bandwidth prediction can help distributed applications identify bandwidth between nodes and choose high-bandwidth connections without performing expensive bandwidth measurements. Unfortunately, however, there exists no effective framework that can predict bandwidth between hosts in a decentralized fashion. Euclidean coordinate spaces are not a good model for embedding bandwidth measurements. Accordingly, attempts [57, 58, 19] to use a network coordinate system do not work well in predicting bandwidth, resulting in poor accuracy. Ramasubramanian et. al [58] developed a system named Sequoia that embeds bandwidth information into a tree metric space, and could achieve higher accuracy than a Euclidean space. However, Sequoia requires centralized data structures, and a single fixed landmark node is used for sampling measurements. There are decentralized approaches [19, 46] that can predict network bandwidth without any landmark nodes. But those systems cause high communication costs to reach convergence of prediction data. Also, unlike our approach, it is not easy to extend those systems to node search problems because prediction data do not fit any kind of metric spaces.

Our approach is to embed bandwidth information into a tree metric space like Sequoia, but in a decentralized fashion. We choose a tree metric approach because we are able to design novel methods to predict bandwidth with high accuracy and low cost. Also, we could develop node search algorithms based on the prediction data produced on a tree metric space, which will be described in Chapter 5 and 6. The goal is to design a prediction algorithm that satisfies the following requirements for a system with n nodes:

- Decentralization: There exists no central component. Data structures are distributed, and no landmark node for sampling measurements is used.
- Scalable Message Complexity: A node join operation requires less than $O(n)$ measurements.
- Scalable Space Complexity: Each node maintains information of size less than $O(n)$.
- High Accuracy: Predicted bandwidth values are close to the real ones.

- Self-Organization: The algorithms handle dynamic network conditions.

The contributions of this work are fourfold. First we describe the design of a decentralized bandwidth prediction system that satisfies all the above requirements. Second, we provide a theoretical proof of the algorithm's correctness. The edge-weighted tree constructed by our algorithm embeds bandwidth measurements with 100% accuracy when we assume that bandwidth measurements can be exactly represented as a tree metric space. The third contribution is a set of heuristics that allow high prediction accuracy in the real Internet. Since a real network cannot be represented as a perfect tree metric space, we need an extra effort to improve the prediction accuracy. Finally, we present extensive simulation results validating the high accuracy, low cost, and scalability for our algorithm.

4.2 Design

There are two main design goals for scalable and decentralized network bandwidth prediction. The first is to store $O(n^2)$ bandwidth values for all pairs of nodes in a network in a distributed data structure, using only a few sampling measurements per node. The other goal is, given two nodes, to compute their bandwidth as stored in the data structure in a completely distributed way. This section describes 1) two distributed data structures used to achieve these goals, 2) how to retrieve bandwidth information from the structures, and 3) how to construct and maintain the structures.

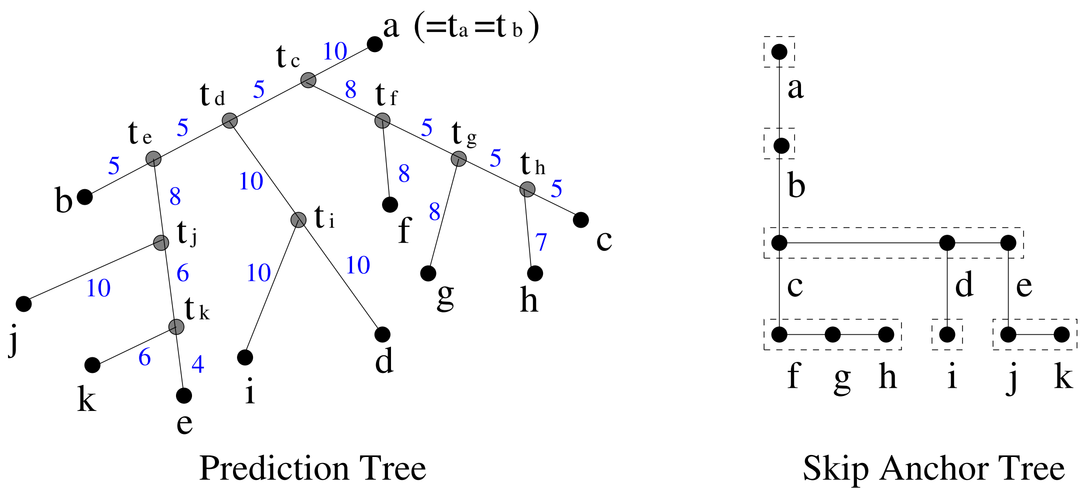


Figure 4.1: Distributed data structures used for bandwidth prediction

4.2.1 Distributed Data Structures

We build two distributed structures called a *prediction tree* and a *skip anchor tree*. This section describes these data structures, their properties, how they are used, and how they are distributed.

4.2.1.1 Prediction Tree

A prediction tree is an edge-weighted tree that embeds bandwidth information. A leaf node represents a real host in the network, and an inner node in the tree is a virtual node created as the prediction tree grows. Each edge is assigned a weight value, so that the bandwidth information is stored as distances in the graph. Given any two nodes u and v in a network, a prediction tree is used to compute their stored bandwidth $BW(u, v)$. The linear transform function $d(u, v) = C - BW(u, v)$ is used to represent bandwidth as a metric, as discussed in Section 2.2, and $BW_T(u, v) = C - d_T(u, v)$ computes a bandwidth measurement from a prediction tree T. For example, in Figure 4.1 if $C = 100$, the predicted bandwidth value $BW_T(b,e)$ is 77 because $d_T(b,e) = 23$. A prediction tree is incrementally constructed by adding nodes one at a time. Each new node x is added along with an inner node t_x and a weighted-edge (t_x, x). We then say that t_x is *owned by x*. *Anchor relationships* are defined during the incremental construction of a prediction tree. If t_x is located on a path between a node a and t_a owned by a, a is called the *anchor parent* of x, and x is called the *anchor child* of a. Assuming nodes are added in alphabetical order in Figure 4.1, node c is the anchor parent of f because t_f is positioned on path $t_c \sim c$. Nodes sharing the same anchor parent are called *anchor siblings*. The top-level anchor parent is called the *root*, denoted by R.

4.2.1.2 Skip Anchor Tree

A skip anchor tree is a hierarchical structure of skip lists that are described as a distributed data structure in Section 2.3. The skip anchor tree contains the connectivity information for the corresponding prediction tree. The skip anchor tree is used for scalable construction of the prediction tree by having each joining node perform only $O(\log^2 n)$ sampling measurements. The skip anchor tree is also used as an overlay network structure.

The structure of a skip anchor tree is determined by anchor relationships. All the anchor child nodes of a node a build one skip list called the *child list* of a. a is connected only to the head b of the child list of a. Then a is called b's *anchor parent neighbor*, and b is a's *anchor child neighbor*. At different levels of skip list tower, each node has *left neighbors* (and *right neighbors*) that are the previous (and next) nodes in linked lists. All the anchor siblings of a form one skip list called the *sibling list* of a. Each node x uses $d_T(t_R, t_x)$ as a key to determine its location in the skip list, where t_R is an inner node owned by the root R. The maximum depth of the anchor relationship is called the *height* of the skip anchor tree. The height is bounded by $O(\log n)$, as we will show in Section 4.2.4. In the skip anchor tree shown in Figure 4.1, there are six separate skip lists, each of which is represented by a dashed box. Each edge connects nodes that are level-0 neighbors to each other in a skip list. Links for the higher-level neighbors are omitted in Figure 4.1. Node c is connected to the head f of its child list, which contains three anchor child nodes f, g, and h. Since a is the root, f has key $d_T(t_a, t_f) = 18$.

We use a skip list [55, 56] as our distributed data structure as described in Section 2.3. Any type of self-balancing binary search tree structure, such as an AVL tree or a treap [15, 62], can be considered as a data structure to store anchor child nodes. We have chosen a skip list because its simplicity made it easier to design and build for a dynamically changing distributed system. Also, our system can be extended to use a distributed version of a skip list, such as SkipNet [32] and Skip Graphs [17].

4.2.1.3 Data Structure Distribution

Each node x maintains the network information shown in Table 4.1. We build a prediction tree in a distributed fashion by having each node maintain graph

Table 4.1: Network information maintained by each node x

Variable	Content
$x.e$	$d_T(t_x, x)$
$x.k$	$d_T(t_R, t_x)$, the key of x used in a skip list (t_R is the inner node owned by the root R.)
$x.P$	the id of anchor parent neighbor of x
$x.C$	the id of anchor child neighbor of x
$x.h_t$	skip list tower height
$x.L[i]$	the id's of left neighbors (with key $\leq x.k$) in the sibling list of x at each level i of skip list
$x.R[i]$	the id's of right neighbors (with key $\geq x.k$) in the sibling list of x at each level i of skip list
$x.k_L[i]$	the keys of left neighbors
$x.k_R[i]$	the keys of right neighbors
$x.label$	the distance label, a list of triplets (a's id, $a.k$, $a.e$) of each successive anchor parent a from x to the root R

distances ($x.e$ and $x.k$ in Table 4.1) and the neighbor information derived from anchor relationships. A skip anchor tree is built from the neighbor information, also in a distributed way. x also maintains a *distance label*, represented by $x.label$, which is a list of triplets (a's id, $a.k$, $a.e$) of all anchor parents a between x and the root. A distance label is then equivalent to a partial prediction tree and is used for distributed bandwidth computations. Since the skip list tower height is bounded by $O(\log n)$ and the size of a distance label is limited by the height of a skip anchor tree, the storage size required in each node is $O(\log n)$.

4.2.2 Distributed Bandwidth Computation

Given two nodes u and v in a prediction tree T, we can compute $d_T(u, v)$ in a distributed fashion by building a partial prediction tree containing u and v with the two following methods. The first option is to collect distance information on demand. Starting at u, we move left towards a head in each skip list and up towards the root in a skip anchor tree, and collect $m.k$ and $m.e$ for each successive anchor parent m of u. We run the same procedure in parallel starting at v, and finish when the algorithm finds the common anchor parent. Then we can build a partial prediction tree containing u and v using the collected information, and compute $d_T(u, v)$. This method returns an accurate result since it computes a distance based on the current information. However, it has some communication costs. We need $O(\log n)$ messages to move to the head of each sibling list through skip list links. Since the height of the skip anchor tree is bounded by $O(\log n)$, collecting the distance information requires $O(\log^2 n)$ messages.

The second option is a one-shot computation using previously collected distance information. We can build a partial prediction tree using *u.label* and *v.label*, and compute $d_T(u, v)$ on it. Unlike the on-demand method, this method sends only $O(1)$ messages for retrieving two distance labels. However, the one-shot method may produce an inaccurate result because distance labels are maintained via a periodic background mechanism, as will be described in Section 4.2.5. Nodes can have stale information in the distance labels as a skip anchor tree restructures itself to recover from node failures. Fortunately, for the same reason as for the on-demand method, only $O(\log^2 n)$ hops are required for each node to propagate its distance information through the entire set of nodes.

4.2.3 Node Join

This section describes a node join algorithm that constructs a distributed prediction tree and a skip anchor tree. The join algorithm does not require any landmark node to be used for performing bandwidth measurements. Also, each joining node performs only $O(\log^2 n)$ bandwidth measurements. Here we assume that a network is represented by a tree metric space that satisfies the four-point condition (4PC), as described in Section 2. In Section 4.3 we will discuss how to improve accuracy in a real network.

Algorithm 1: x**.join(**z**)**

```
1  if $z = null$ then $x.k \leftarrow 0$;  $x.e \leftarrow 0$;  return
2  $y \leftarrow x$.findOpt($z$, null)
3  $T \leftarrow$ a sub-prediction tree containing every node $v$ such that $d(x, v)$ is
   measured so far
4  Add $x$ to $T$ along with $t_x$, where $d_T(z, t_x) = g(y)$ and
   $d_T(t_x, x) = d(x, z) - g(y)$
5  $a \leftarrow x$'s anchor parent node in $T$
6  $x.k \leftarrow d_T(t_R, t_a) + d_T(t_a, t_x)$;  $x.e \leftarrow d_T(t_x, x)$
7  if $a.C = null$ then $x.P \leftarrow a$
8  else
9      $x_L \leftarrow a.C$.findPred($x.k$)
10     if $x_L = null$ then $x.P \leftarrow a$; $x.R[0] \leftarrow a.C$
11     else  $x.L[0] \leftarrow x_L$; $x.R[0] \leftarrow x_L.R[0]$
12 Notify $x.P$, $x.C$, $x.L[0]$, and $x.R[0]$ of the join event
```

4.2.3.1 Overall Algorithm

Algorithm 1 is the join algorithm for a node x. Each new node x is added to a prediction tree along with an inner node t_x and a weighted edge (t_x, x). If x is the first joining node, x becomes the root R and t_x is located at the same position as x, which means $x.k = 0$ and $x.e = 0$. The join algorithm for subsequent nodes starts by contacting any node z in the network, called a *base node*. x finds another

Algorithm 2: x.**findOpt**(m, m_{prev})

```
1  Measure $d(x,m)$ // measure $BW(x,m)$ and convert it to $d(x,m)$
2  $p \leftarrow m$.findSucc$(-\infty).P$; Measure $d(x,p)$
3  if $m_{\text{prev}} = null$ then
4      $y_c \leftarrow x$.findChildOpt$(m.C$.findPred$(+\infty)$, Left$)$
5  else if $m_{\text{prev}} = p$ then
6      $y_c \leftarrow x$.findChildOpt$(m.C$, Right$)$
7  else
8      $y_{cL} \leftarrow x$.findChildOpt$(m_{\text{prev}}.L[0]$, Left$)$
9      $y_{cR} \leftarrow x$.findChildOpt$(m_{\text{prev}}.R[0]$, Right$)$
10     if $g(y_{cL}) \geq g(y_{cR})$ then $y_c \leftarrow y_{cL}$
11     else $y_c \leftarrow y_{cR}$
12 $m_{\text{next}} \leftarrow$ a maximizer of $g(s)$ $\forall s \in \{m, p, y_c\} \setminus \{m_{prev}\}$
13 if $g(m) \geq g(m_{\text{next}})$ then return $m$
14 return $x$.findOpt($m_{\text{next}}$, $m$)
```

node y that is called an *end node*. y is chosen as the maximizer of the *Gromov product* $g(y) = \frac{1}{2}(d(x,z) + d_T(y,z) - d(x,y))$. We provide some intuition into why we maximize $g(y)$. Suppose that, in a prediction tree x and t_x are added at the unique positions that make every distance correct. Given three nodes u, v, and w in a prediction tree, let's define a *joint node* of u, v, and w as an inner node that is located on all the three paths $u \sim v$, $u \sim w$, and $v \sim w$. Then t_x should not be closer

Algorithm 3: x.**findChildOpt(**b**,** dir**)**

```
1  if $b = null$ then return $null$; else Measure $d(x, b)$
2  if dir = Right then $c_1 \leftarrow b.R[0]$; else $c_1 \leftarrow b.L[0]$
3  if $c_1 = null$ then return $b$; else Measure $d(x, c_1)$
4  if $g(b) > g(c_1)$ then return $b$
5  if $g(b) = g(c_1)$ and $b.k \neq c_1.k$ then return $b$
6  for $t \leftarrow b.h_t - 1$ to 0 do
7      if dir = Right then $c_1 \leftarrow b.R[t]$; $c_2 \leftarrow c_1.R[0]$
8      else $c_1 \leftarrow b.L[t]$; $c_2 \leftarrow c_1.L[0]$
9      if $c_1 = null$ then continue; else Measure $d(x, c_1)$
10     if $c_2 = null$ then continue; else Measure $d(x, c_2)$
11     if $g(c_1) \neq g(c_2)$ then break
12 return $x$.findChildOpt($c_1$, dir)
```

to z than any joint node of z, x, and another node. Note that $g(y)$ should be equal to the distance between z and the joint node of z, x, and y. So maximizing $g(y)$ will determine the correct positions of t_x and x, and $d_T(z, t_x)$ should be equal to the maximum $g(y)$. After finding the position of x in a prediction tree, the algorithm determines the anchor parent of x and assigns graph distances $x.k$ and $x.e$. Finally, x is inserted in the child list of the anchor parent of x using the key of $x.k$. Note that x updates only level-0 skip list neighbors to make the join operation fast. The links for higher-level neighbors will be connected by periodic update mechanisms that will be described in Section 4.2.5.

4.2.3.2 How to Find an Optimizer y

x finds an optimizer y by traversing nodes in the skip anchor tree, starting at z. Algorithm 2 shows the procedure executed with the currently visited node m and the previous node m_{prev}. The intuition is that the global optimizer y must exist in the direction of a local optimizer around m because nodes satisfy the four-point condition in a tree metric space. In Algorithm 2, x executes Algorithm 3 to find y_c that maximizes $g(y_c)$ in the child list of m. For the anchor parent p of m, if $g(m)$ is the local maximum among $g(m)$, $g(p)$, and $g(y_c)$, m is chosen as y. Otherwise, the algorithm visits a maximizer p or y_c, and repeats the procedure in Algorithm 2. In this way, x can move toward where the global optimizer y exists. For example, if y_c is a local optimizer, y will be located in a sub-skip anchor tree rooted at y_c. Algorithm 3 finds a local optimizer y_c in the child list of m by traversing either right towards a tail or left towards a head starting at node b. The intuition is that, for two level-0 neighbors c_1 and c_2 in a skip list, the comparison of $g(c_1)$ and $g(c_2)$ can determine the direction where y_c exists and shrink the search space by around half. Algorithm 3 chooses c_1 as a high-level skip list neighbor of b within the search space, and keeps shrinking the search space until finding y_c. For example, when $g(c_1) \neq g(c_2)$, the algorithm ignores b and nodes between b and c_1, and moves to c_1 because y_c must exist in the new range starting at c_1.

4.2.3.3 Proof of Correctness

This section gives several lemmas and proof sketches to show the correctness of the join algorithm given in Theorem 4.1. Lemma 4.1 shows that by comparing the Gromov products, we can exclude part of a prediction tree from end node searches.

Lemma 4.1. *Let S be the set of all leaf nodes in a prediction tree. For three leaf nodes z, y, and w and their joint node t, let S_w be the set of leaf nodes including w in sub-trees rooted at inner nodes between t and w, and S_y between t and y. Then with the base node z, if $g(y) > g(w)$, then $g(y) > g(s)\ \forall s \in S \setminus S_y$. Also, if $g(y) = g(w)$, then $g(y) = g(s)\ \forall s \in S_w$.*

Proof. We first show that $g(y) > g(w) \Rightarrow g(y) > g(s)\ \forall s \in S \setminus S_y$. The proof is divided into the two following parts as $S \setminus S_y$ can be divided into S_w and $S \setminus S_y \setminus S_w$.

- Part 1: Show $g(y) > g(w) \Rightarrow g(y) > g(s)\ \forall s \in S_w$.

  Because $g(y) > g(w)$ and 4PC holds for z, y, w, and x, $d(y, x) + d(z, w) < d(y, w) + d(z, x)$ is satisfied, which implies $d(y, x) + d(z, s) < d(y, w) + d(z, x) - d(z, w) + d(z, s)$. From the location of $s \in S_w$ in the prediction tree (Figure 4.2), $d(y, w) - d(z, w) + d(z, s) = d(y, s)$ holds, so $d(y, x) + d(z, s) < d(y, s) + d(z, x)$. Since 4PC holds for z, y, s, and x, $d(z, y) + d(s, x) > d(y, x) + d(z, s)$. So, $g(y) > g(s)\ \forall s \in S_w$.

- Part 2: Show $g(y) > g(w) \Rightarrow g(y) > g(s)\ \forall s \in S \setminus S_y \setminus S_w$.

  Because $g(y) > g(w)$ and 4PC holds for z, y, w, and x, $d(y, x) + d(z, w) < d(y, w) + d(z, x)$ is satisfied, which implies $d(y, x) + d(z, s) < d(y, w) + d(z, x) -$

$d(z,w) + d(z,s) = d(y,s) + d(z,x) + d(z,s) + d(y,w) - (d(z,w) + d(y,s))$. From the location of $s \in S \setminus S_y \setminus S_w$ in the prediction tree (Figure 4.2), $d(z,s)+d(y,w) \leq d(z,w)+d(y,s)$ holds, so $d(y,x)+d(z,s) < d(y,s)+d(z,x)$. Since 4PC holds for z, y, s, and x, $d(z,y) + d(s,x) > d(y,x) + d(z,s)$. So, $g(y) > g(s)\ \forall s \in S \setminus S_y \setminus S_w$.

Thus, $g(y) > g(w) \Rightarrow g(y) > g(s)\ \forall s \in S \setminus S_y$.

We now show that $g(y) = g(w) \Rightarrow g(y) = g(s)\ \forall s \in S_w$. Because $g(y) = g(w)$ and 4PC holds for z, y, w, and x, $d(y,w) + d(z,x) \leq d(y,x) + d(z,w) = d(z,y) + d(w,x)$ is satisfied, which can be divided into the two following cases.

- Case 1: $d(y,w) + d(z,x) = d(y,x) + d(z,w) = d(z,y) + d(w,x)$

  The assumption of Case 1 implies $d(z,x)+d(w,s) = d(z,y)+d(w,x)-d(y,w)+d(w,s) = d(z,s) + d(w,x) + d(z,y) + d(w,s) - (d(z,s) + d(y,w))$. From the location of $s \in S_w$ in the prediction tree (Figure 4.2), $d(z,y) + d(w,s) < d(z,s)+d(y,w)$ holds, so $d(z,x)+d(w,s) < d(z,s)+d(w,x)$. Since 4PC holds for z, w, s, and x, $d(z,x)+d(w,s) < d(z,s)+d(w,x) = d(z,w)+d(s,x)$ holds. The assumption of Case 1 implies $d(y,x)+d(z,s) = d(z,y)+d(w,x)-d(z,w)+d(z,s) = d(z,s)+d(w,x)+d(z,y)-d(z,w)$. Since $d(z,s)+d(w,x) = d(z,w)+d(s,x)$, $d(y,x)+d(z,s) = d(z,y)+d(s,x)$ is satisfied. So, $g(y) = g(s)\ \forall s \in S_w$.

- Case 2: $d(y,w) + d(z,x) < d(y,x) + d(z,w) = d(z,y) + d(w,x)$

  The assumption of Case 2 implies $d(y,x)+d(z,s) > d(y,w)+d(z,x)-d(z,w)+d(z,s) = d(y,s) + d(z,x) + d(z,s) + d(y,w) - (d(z,w) + d(y,s))$. From the location of $s \in S_w$ in the prediction tree (Figure 4.2), $d(z,s) + d(y,w) =$

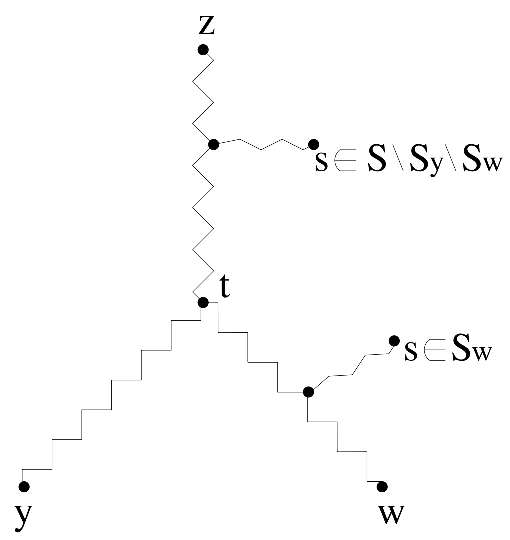


Figure 4.2: The prediction tree used in the proof of Lemma 4.1

$d(z,w)+d(y,s)$ holds, so $d(y,x)+d(z,s) > d(y,s)+d(z,x)$. Since 4PC holds for z, y, s, and x, $d(y,x)+d(z,s) = d(z,y)+d(s,x)$ holds. So, $g(y) = g(s)\ \forall s \in S_w$.

Thus, $g(y) = g(w) \Rightarrow g(y) = g(s)\ \forall s \in S_w$. □

Lemma 4.2. *Algorithm 3 finds y_c that maximizes $g(y_c)$ among nodes from b in the direction of* dir *in a skip list.*

Proof. We assume the traversal goes right toward the tail (dir = Right) as the other case can be proved in the similar way. Also, we assume every node has a unique key in the skip list for simplicity, although the algorithm supports the case of duplicate keys (which should happen rarely in practice). With given c_1 and c_2, the algorithm shrinks the search space in two cases. When $g(c_1) = g(c_2)$, $g(c_1) = g(s)$ for all nodes s from c_2 to the tail by Lemma 4.1, and the algorithm correctly shrinks the search space from the right by moving down to the lower level in b's skip list tower. When $g(c_1) \neq g(c_2)$, y_c must be located among nodes from c_1 to the tail by Lemma 4.1, and the algorithm correctly shrinks the search space from the left by moving to c_1. The algorithm terminates with three different cases. If b is the tail, $b = y_c$. If $g(b) > g(c_1)$ for b's level-0 neighbor c_1, $g(b) > g(s)$ for all nodes s in the search space by Lemma 4.1, so $b = y_c$. Likewise, when $g(b) = g(c_1)$, $b = y_c$. In all three cases, the algorithm correctly returns b as y_c. □

Lemma 4.3. *Algorithm 2 finds y that maximizes $g(y)$.*

Proof. Three cases are possible depending on m_{prev}, as shown by the if statements in Algorithm 2. Here, we only prove the case where $m_{\text{prev}} = p$ as the other two proofs are similar. The algorithm moves from p to m because $g(p) < g(m)$. If $g(m) \geq g(y_c)$, $m = y$ by Lemma 4.1. Otherwise, y must exist in the sub-skip anchor tree rooted at y_c, by Lemma 4.1, and the algorithm correctly moves to y_c. Thus, Algorithm 2 finds a correct y. □

Theorem 4.1. *When x is added to a prediction tree T by Algorithm 1, $d(x, s) = d_T(x, s)$ for all leaf nodes s in T.*

Proof. For a leaf node s in T, let t be the joint node of s, z, and y. As shown in Figure 4.3, two cases for the shape of prediction tree can be considered with respect to the location of t and s.

- Case 1: $d_T(z, t) \leq d_T(z, t_x)$

  Since $g(s) \leq g(y)$ by Lemma 4.3, $d(z, s) + d(x, y) \leq d(z, y) + d(x, s)$. And $d_T(z, t) \leq d_T(z, t_x)$ implies $d(z, s) + d(x, y) \leq d(z, x) + d(y, s)$. 4PC holds for z, y, x, and s, which implies $d(x, s) = d(z, x) + d(y, s) - d(z, y)$. From the

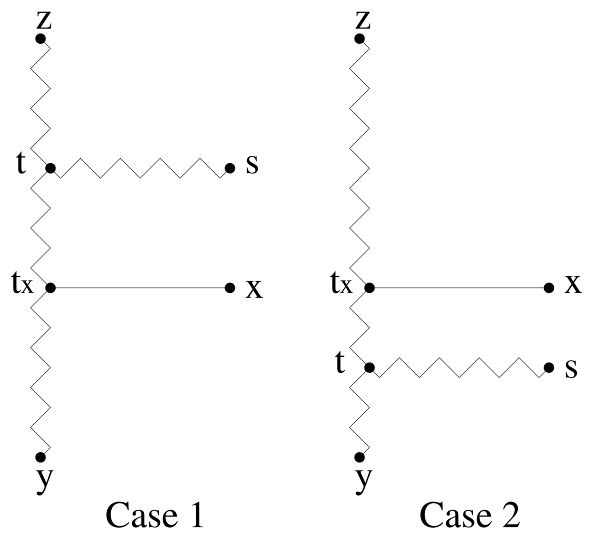


Figure 4.3: Two cases of prediction tree used in the proof of Theorem 4.1

structure of T, $d(z,x) = d_T(z,t_x) + d_T(t_x,x)$, $d(y,s) = d_T(y,t_x) + d_T(t_x,s)$, and $d(z,y) = d_T(z,t_x) + d_T(t_x,y)$. So, $d(x,s) = d_T(t_x,x) + d_T(t_x,s) = d_T(x,s)$.

- Case 2: $d_T(z,t) > d_T(z,t_x)$

  Since $g(s) \le g(y)$ by Lemma 4.3, $d(z,y) - d(x,y) \ge d(z,s) - d(x,s)$. And $d_T(z,t) > d_T(z,t_x)$ implies $d(z,x) - d(x,y) < d(z,s) - d(y,s)$. 4PC holds for z, y, x, and s, which implies $d(x,s) = d(x,y) + d(z,s) - d(z,y)$. From the structure of T, $d(x,y) = d_T(x,t_x) + d_T(t_x,y)$, $d(z,s) = d_T(z,t_x) + d_T(t_x,s)$, and $d(z,y) = d_T(z,t_x) + d_T(t_x,y)$. So, $d(x,s) = d_T(x,t_x) + d_T(t_x,s) = d_T(x,s)$.

Thus, $d(x,s) = d_T(x,s)$ for all leaf nodes s in T. $\square$

4.2.3.4 Performance Analysis

This section analyzes the performance of the join algorithm by proving Theorem 4.2.

Lemma 4.4. *Algorithm 3 performs measurements with $O(\log n)$ nodes with high probability.*

Proof. Let y_c be the optimizer in x's child list. Consider the node traversal path for executing b.findPred($y_c.k$) to find the node y_c with key $y_c.k$ starting at b. This traversal path of findPred must be equal to the traversal path of b in Algorithm 3 when searching for y_c because both algorithms are executed in exactly the same pattern. Both algorithms return the current node b when b is determined to be y_c. Otherwise, both algorithms move to b's highest-level neighbor c_1 such that y_c exists behind c_1.

At each network hop, x performs measurements with two nodes as shown in lines 1 and 3 of Algorithm 3. Also, for each horizontal move to another node and each downward move in a skip list tower, x performs measurements with two more nodes in lines 9 and 10 of Algorithm 3. The findPred operation takes $O(\log n)$ network hops with high probability and the skip list tower height is $O(\log n)$ as described in Section 2.3. Thus, the number of measurements performed by Algorithm 3 should be $O(\log n)$, with high probability. □

Theorem 4.2. *Algorithm 1 performs measurements with $O(\log^2 n)$ nodes, with high probability.*

Proof. Algorithm 3 runs on $O(\log n)$ skip lists as the skip anchor tree height is $O(\log n)$. By Lemma 4.4, Algorithm 1 takes $O(\log^2 n)$ measurements, with high probability. □

4.2.4 Height-Bounding of Skip Anchor Tree

The height of a skip anchor tree should be bounded by $O(\log n)$. Bounding height is important because it limits 1) the cost of the join algorithm, 2) the size of a distance label, and 3) the time taken by periodic information aggregation that will be described in Section 4.2.5.

Let $x.h$ be the height of the sub-skip anchor tree rooted at a node x. Select a node m_1 such that $m_1.h$ is the maximum in the child list of x. m_2 is the node with the second greatest $m_2.h$. Then the *height-bounding factor* of a node x is defined by $(m_1.h - m_2.h)$. Each node x computes its height-bounding factor by periodically aggregating the id's of m_1 and m_2, and $m_1.h$ and $m_2.h$. The aggregation is done from tail to head in each skip list and in a bottom-up way towards the root.

Every node (except the root) should have a height-bounding factor of zero or one, which is called the *height-bounding property.* Then the skip anchor tree height is bounded by $O(\log n)$ as shown in Theorem 4.3. The id of a node x violating the height-bounding property is also periodically aggregated up to the root. Then the root node periodically restructures its trees by requesting x to perform a rotation, as shown in Figure 4.4. x and m_1 are rotated with a pivot t_{m_1} in a prediction tree. x is switched with m_1, and a sub-tree B is switched with C in the skip anchor tree. Anchor relationships are changed in the rotation, but the distance information in the prediction tree remains the same. Also, a rotation requires only $O(1)$ messages to update x, m_1, and their new neighbors: anchor parent, anchor child, level-0 left, and level-0 right. The reason that rotations are managed by the root node is to avoid unnecessary rotations. One node join operation affects the height-bounding factor of all the successive anchor parents from a newly joined node to the root. There are multiple possible violators of the height-bounding property in the set of anchor parent nodes. Those multiple violations can be fixed by executing just one rotation at the violator in the lowest level in the skip anchor tree. So the root aggregates the id of the lowest level violator and performs one rotation instead of allowing unnecessary rotations to be performed at multiple nodes. Since not every node join or leave leads to a rotation, rotations do not need to be executed very frequently. So rotations can be managed relying on the periodic aggregation mechanism.

Theorem 4.3. *The height of a skip anchor tree is* $O(\log n)$.

Proof. The minimum number of nodes N_h in a skip anchor tree with height h can be computed as $N_h = N_{h-1} + N_{h-2}$. ($N_0 = 1$ and $N_1 = 2$)

$N_0 = 1$ is true because a skip anchor tree with only one node has height zero. $N_1 = 2$ is also true because if the root has only one node in its child list, the height is one. Let A_i denote the skip anchor tree with height i and the minimum N_i nodes. And let A'_i denote the remaining skip anchor tree after removing the root from A_i. For $h \geq 2$, we can create A_h with A'_{h-1} and A'_{h-2} in the following way. First, we create a skip list with the root of A'_{h-1} as the head and the root of A'_{h-2} as the tail. Then we create a node p and connect p to the root of A'_{h-1} as a parent neighbor. Finally, we create another node and connect it to p as a parent neighbor. Then the result structure is A_h because all the nodes except the root satisfy the height-bounding property. Also, the structure contains the minimum number of nodes for

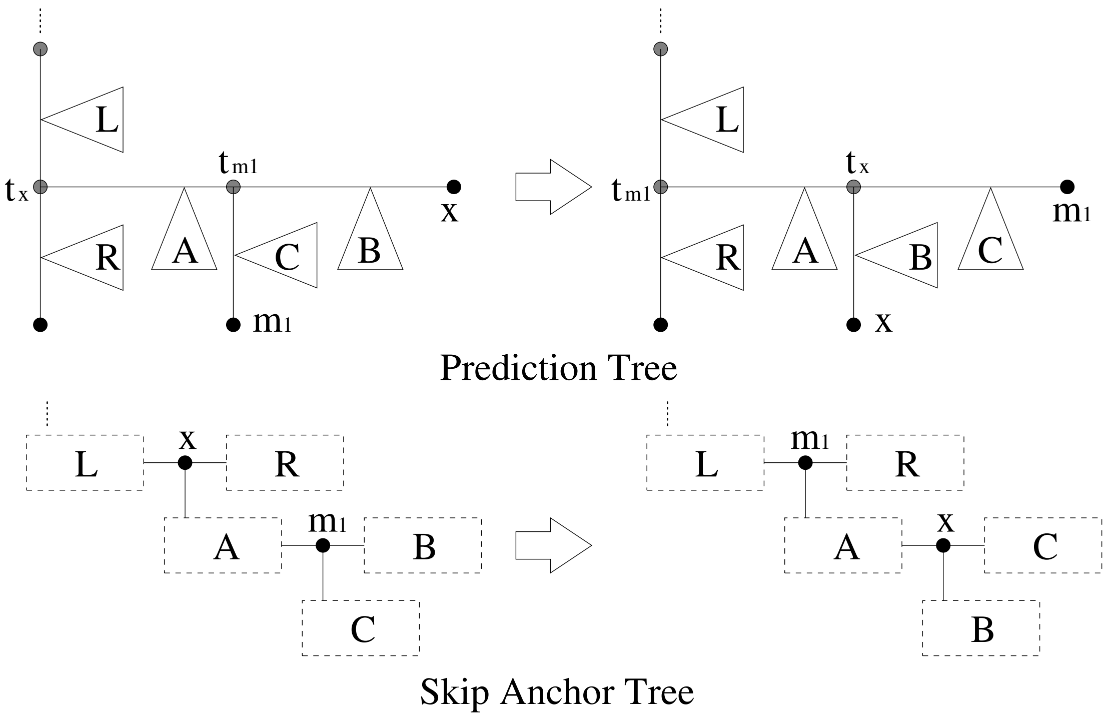


Figure 4.4: Rotation of x and m_1 by the height-bounding operation

given height h because it is constructed using the smallest structures A'_{h-1} and A'_{h-2}. So N_h should be equal to the sum of 1 (for the root), 1 (for p), $N_{h-1} - 1$ (for A'_{h-1}), and $N_{h-2} - 1$ (for A'_{h-2}), which is $N_{h-1} + N_{h-2}$.

N_h is a shifted sequence of Fibonacci numbers, which evaluates to approximately $\frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^{h+2}$. Note that $N_h \leq n$ for the total number of nodes n. Thus, $h \approx 1.44 \log_2 N_h \leq 1.44 \log_2 n = O(\log n)$. □

4.2.5 Failover and Network Changes

Our system reorganizes itself in response to a changing network environment. We first discuss dealing with failover. A node leave operation can be handled similarly to failover. When node x fails, one of its neighbors in the overlay network detects the event from missing periodic heartbeat messages and performs a rotation, in a similar way to the height-bounding mechanism. Let m_1 be the tail of x's child list. The rotation is performed between x and m_1 with pivot t_{m_1}. Imagine the situation where B is empty in Figure 4.4. Restructuring finishes by removing x in both trees. The rotation changes anchor relationships, but all the graph distances in the prediction tree remain exactly the same. The failover operation requires $O(\log n)$ messages to find the tail m_1 in the skip list.

We employ a periodically executing background mechanism that gradually updates high-level skip list neighbor entries after a node joins or fails. Each node performs the background mechanism periodically and independently to propagate high-level neighbor information forward to the tail and backward to the head in each skip list. This allows join and failover operations to finish quickly and makes the system resilient to frequent node joins, leaves and failures. Distance labels are also updated by a periodic background mechanism. After a failover operation for node x, we must remove x from the distance labels of nodes in the sub-skip anchor tree of x. It is not desirable to update distance labels of $O(n)$ nodes (the worst case) whenever a failover operation occurs. So nodes periodically propagates distance labels in a top-down and head-to-tail direction in a skip anchor tree, so that receivers can update their distance labels. This mechanism runs quickly using only $O(\log^2 n)$ network hops with the help of skip list links and the bounded height of the skip anchor tree.

In a real network environment, bandwidth can change dynamically over time. Reconstructing the entire prediction tree to adapt to dynamic changes would have a high cost. We restructure only the part of the system where bandwidth changes occur, by periodic adjustment of the tree. Each node x maintains a fixed-size set M of random nodes whose bandwidth with x has been measured during node join operations. x periodically performs a measurement with each node in M. If x detects significant changes in the measurement data, x leaves the overlay network and joins back at a new position. We leave it as an open problem to determine the proper set size of M. If $|M|$ is large, we can adjust the prediction tree quickly following dynamically changing network environments. However, large $|M|$ implies high maintenance costs that come from many measurements. So, it is necessary to find the proper set size of M so that the prediction tree can adjust itself quickly with low measurement costs.

4.3 Tolerating Imperfect Data

The algorithms described in the previous section can construct a prediction tree that embeds distance information of a tree metric space with no errors. However, since the Internet cannot be modeled as a perfect tree metric space, directly applying the previously described algorithms in practice results in prediction and search errors. We now describe heuristics to improve the accuracy of our algorithms for real networks.

4.3.1 Rational Transform Function

Following Ramasubramanian et. al's approach [58], we used the linear transform function $BW_T(u, v) = C - d_T(u, v)$ for bandwidth prediction as described in Section 2.2. If a network is not modeled as a perfect tree metric space, $d_T(u, v)$ might not be equal to $d(u, v)$. If $d_T(u, v)$ is much larger than $d(u, v)$, that can result in predicting a negative bandwidth value and will decrease overall prediction accuracy. So we introduce a *rational transform function* to overcome this problem. We use $d(u, v) = \frac{C}{BW(u,v)}$ and $BW_T(u, v) = \frac{C}{d_T(u,v)}$ with a positive constant C to transform between bandwidth and distance, instead of the linear transform function. Then the predicted bandwidth will always be positive even when $d_T(u, v)$ is overestimated in a real world scenario. As does the linear function, the rational transform function inverts ordering of bandwidth after performing the transformation, so it can be used as a distance function in a metric space. The first metric space property $d(u, v) \geq 0$, described in Section 2.2, is satisfied by using a positive constant C. The second property which states that $d(u, v) = 0$ if and only if $u = v$, can be satisfied by setting $BW(u, u) = \infty$. The other two properties about symmetry and triangle inequality are also satisfied for the rational transform function, similarly to the linear function as described in Section 2.2.

4.3.2 Error Minimization

The algorithms described in the previous section maximize the Gromov product $g(y)$ to construct a prediction tree. Since we use a graph distance value to compute $g(y)$, the graph distance should be close to the real distance so that we can construct an accurate prediction tree. However, the distance information in a prediction tree must contain some errors in a real network where the tree metric assumption does not hold. Adding each node to a prediction tree using the inaccurate values of $g(y)$ will accumulate errors in graph distances.

So we modify the node join algorithm to directly minimize the relative prediction error rather than maximize the Gromov product. The relative prediction error is defined as $\frac{|BW-BW_T|}{\min\{BW,BW_T\}}$ [53, 19] for the predicted bandwidth BW_T of a node pair and the real bandwidth BW. Unlike the general relative error $\frac{|BW-BW_T|}{BW}$, this metric avoids underestimating BW_T because the error goes to infinity as BW_T approaches zero. We define $e(u)$ as the average relative prediction errors of node pairs (x, t) such that $BW(x, t)$ has been measured so far, based on the position of x computed by a temporary end node u. In Algorithms 2 and 3, we replace $g(u)$ with $e(u)$, for any node u used in $g(u)$ comparisons. Also, we reverse the comparison condition, for example, replacing $g(u) < g(v)$ with $e(u) > e(v)$. The revised join algorithm finds an end node y that minimizes $e(y)$ instead of maximizing $g(y)$. Once an error minimizer is found using the heuristic, x determines its final position using all its collected measurement data. For each pair of nodes in the set of measured nodes, x chooses a temporary base node and an end node, and computes the temporary position of x. The position that minimizes the error is selected as the final position of x. Note that this heuristic also works well in a perfect tree metric space, causing no errors like the original algorithm. The original algorithm finds the position of a new node that makes $e(y) = 0$, so maximizing $g(y)$ produces the same result in a tree metric space as minimizing $e(y)$.

4.3.3 More Sampling Measurements

In the error minimization technique, collecting more sampling measurements results in a more accurate prediction tree. We therefore modify the node join algorithm to collect more samples as follows. First, whenever x performs measurements with c_1 and c_2 in Algorithm 3, x collects more sampling measurements for two successive level-0 neighbors of c_1 and c_2 in their sibling list and their anchor child neighbors. Second, for each visited node m in Algorithm 2, x performs measurements with all neighbors of m in m's sibling list. Last, we modify Algorithm 2 to proceed further to the second optimum even though the currently visited node m is the local optimum. That is, we choose either p or y_c as the next visited node m_{next}, and the algorithm returns m when there is nowhere to move. Despite performing more samples, this heuristic still keeps the measurement cost at $O(\log^2 n)$ per node.

4.4 Evaluation

This section evaluates our approach for bandwidth prediction. The experimental results show the high accuracy and scalable cost of our algorithms compared to prior approaches.

4.4.1 Experimental Setup

Our simulations are based on two bandwidth measurement data sets. The first data set, *HP-PlanetLab* [58], contains bandwidth measurements between PlanetLab [23] nodes, and has been collected at HP Labs using pathChirp [60] in June 2006 for the S-cube project [7, 72]. Since the raw data set is incomplete and has many unmeasured pairs of nodes, we first extracted measurements for the 190 nodes (out of 459) that give a full n-to-n asymmetric matrix containing bandwidth measurements. To represent bandwidth in a metric space, we then converted the matrix to a symmetric one by averaging bandwidth values from forward and reverse directions for each pair of nodes. This symmetric matrix is considered as containing a set of real-world bandwidth measurements for our simulations. The second set, *UMD-PlanetLab* [66], contains measurements between PlanetLab nodes, and was collected at the University of Maryland at College Park using pathChirp during two weeks starting in late October 2010. We preprocessed this new data set into a full symmetric matrix of 317 nodes (out of 497) in the same way as for the HP-PlanetLab data set.

We will show the results of four different bandwidth prediction systems: **SEQ**,

OLD, **NEW**, and **FUL**. The simulator for these algorithms is implemented in Java using PeerSim [38] as a starting point. All four approaches embed bandwidth into a tree metric space. We will also compare our approach with other systems based on Euclidean spaces at the end of this section. SEQ refers to our implementation of the centralized Sequoia algorithm [58]. As described in Section 3.1, SEQ embeds bandwidth information into a centralized prediction tree. Each node in SEQ performs a measurement with a single landmark node. OLD is our original decentralized but not highly scalable approach [65], as described in Section 3.1. NEW is our decentralized and scalable algorithm described in Section 4.2. OLD and NEW are similar to each other in the sense that both construct a distributed prediction tree without using any landmark nodes. But a joining node in OLD performs measurements with all the anchor child nodes at each visited node. On the other hand, NEW chooses only a few anchor child nodes in a skip list for sampling measurements. FUL is a prediction tree constructed by doing exhaustive n-to-n measurements to achieve the possible highest accuracy in a tree metric space, so would be very expensive to create and maintain. For each approach and each bandwidth data set, we run 100 rounds of system construction with different node join orderings.

4.4.2 Accuracy

For each round, we recorded relative bandwidth prediction errors, as defined in Section 4.3, across all possible pairs of nodes in a data set. Figure 4.5(a) shows results from five rounds for the HP-PlanetLab data. (Figure 4.5(b) is for UMD-

PlanetLab.) The y-axis in the graph indicates median, 25-th, and 75-th percentile values over a total of 17955 relative prediction error values across 190 nodes for HP-PlanetLab (total 50086 values across 317 nodes for UMD-PlanetLab). While we show the results of only five rounds, all 100 rounds show similar trends.

First, NEW shows lower error than SEQ. For example in Figure 4.5(a), the median error is 0.1 for NEW while the error for SEQ is around 0.3. This is because NEW applies the heuristics described in Section 4.3 to improve the prediction accuracy in a real network that is not modeled exactly by a tree metric space. On the other hand, SEQ does no extra work to support an imperfect tree metric space. Second, NEW closely tracks FUL. FUL provides an upper bound on the prediction accuracy that we can achieve in a tree metric space. While FUL uses exhaustive measurements, NEW has much less cost (fewer measurements) to achieve slightly lower accuracy. Last, NEW shows similar accuracy to OLD as both approaches use the similar techniques to improve prediction accuracy as described in Section 4.3. However, NEW has more scalable cost than OLD as will be shown below.

4.4.3 Total Measurement Cost

Figure 4.6(a) shows the total number of measurements performed to construct each system in Figure 4.5(a) for HP-PlanetLab. Figure 4.6(b) shows the same data for UMD-PlanetLab. Since FUL uses exhaustive measurements, FUL has the highest cost. NEW has less cost than OLD and SEQ. This is because NEW bounds the join operation cost to $O(\log^2 n)$ measurements with high probability by using the skip list structure and limiting the height of the skip anchor tree, as discussed in Section 4.2. On the other hand, each node join operation for OLD and SEQ requires $O(n)$ measurements in the worst case. The cost of NEW is consistent across multiple rounds while the cost is widely varying for OLD and SEQ. For example, in Figure 4.6(a) NEW consistently requires around 2500 measurements while SEQ varies between 9000 and 12000. Since the best-case join cost is $O(1)$ for all three approaches, the total cost of OLD and SEQ varies between $O(n)$ and $O(n^2)$. However, the total cost of NEW varies only between $O(n)$ and $O(n \log^2 n)$.

4.4.4 Scalable Node Join Cost

We recorded the number of measurements used for each node join operation. Figure 4.7(a) and Figure 4.7(b) show the maximum join cost among the 100 rounds (node join orderings). The cost for FUL and SEQ increases linearly as the system size increases. OLD shows a little lower cost than SEQ, but still increases almost linearly. As expected, the cost for NEW increases most slowly.

4.4.5 Tree Metric Space vs. Euclidean Space

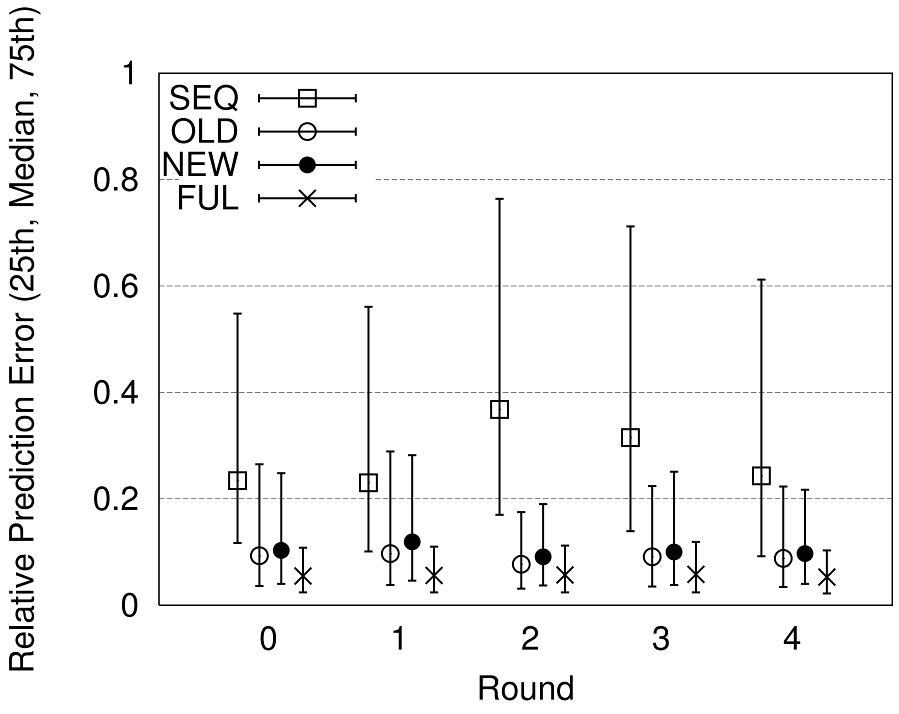
We introduce two more systems that embed network distances in a 2-d Euclidean coordinate space: **VIV-** and **VIV**. VIV- is our simulation of the Vivaldi [26] system with a linear transform function $d(u, v) = C - BW(u, v)$. VIV means the Vivaldi system with a rational transform function $d(u, v) = \frac{C}{BW(u,v)}$. For both cases, we produced the best accuracy that Vivaldi could achieve for bandwidth prediction.

Each node has all other nodes as neighbors. 100000 bandwidth measurements are performed for random pairs of nodes, so that Vivaldi converges to the best result using a large number of measurements. Also, C in VIV- is set to be the maximum bandwidth value in each data set, which results in the best accuracy. ($C = 398$ in HP-PlanetLab and $C = 1308$ in UMD-PlanetLab) $C = 10000$ is set for VIV just like other tree metric space approaches as C does not affect the prediction accuracy. For each approach and each bandwidth data set, we run 100 rounds of system construction with different orderings of sampling measurements.

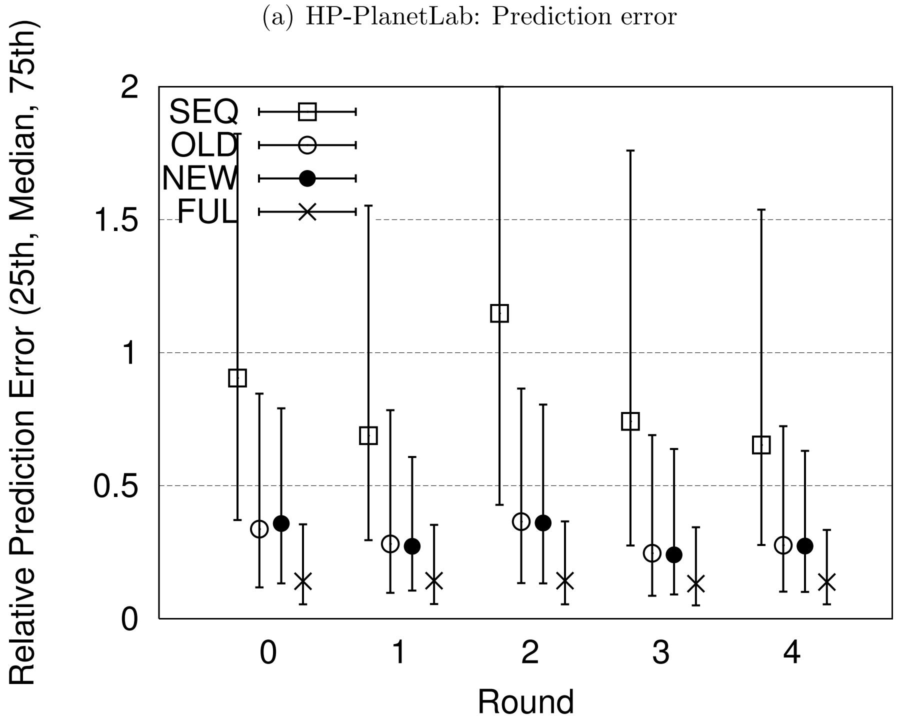
Figure 4.8(a) and 4.8(b) show the cumulative distribution function (CDF) of relative prediction error of all six approaches. While we show the results of only one round, all 100 rounds show similar trends. For the four systems using the rational transform function, FUL, NEW, and OLD show lower prediction error than VIV. Also, for the other two systems using the linear transform function, SEQ has better accuracy than VIV-. This shows a tree metric space is better than a 2-d Euclidean space to embed bandwidth information. More to the point, NEW has a higher accuracy than VIV with much lower measurement cost. Note that VIV shows much higher accuracy than VIV-. This shows the efficacy of a rational transform function in combination with error minimization in Vivaldi. Vivaldi reduces the relative error $\frac{|d-d_T|}{d}$, which is equal to lowering $\frac{|BW-BW_T|}{BW_T}$ with a rational transform function. Although $\frac{|BW-BW_T|}{BW_T}$ is not exactly the same as our metric of relative prediction error $\frac{|BW-BW_T|}{\min\{BW,BW_T\}}$, both metrics have some degree of correlation. Thus, VIV shows quite high accuracy, even slightly higher than SEQ, which uses a linear transform function.

4.5 Summary

This chapter has presented an algorithm to predict end-to-end network bandwidth. Our approach is completely decentralized: no central landmark node exists for sampling measurements, and data structures are distributed. We proved that the algorithm has the perfect accuracy assuming bandwidth can be represented in a tree metric space. We also described heuristics to achieve high accuracy in a real network that is not perfectly modeled as a tree metric space. The algorithm is also highly scalable as each joining node only performs $O(\log^2 n)$ sampling measurements. Simulation results validate the high accuracy and scalability of our approach.

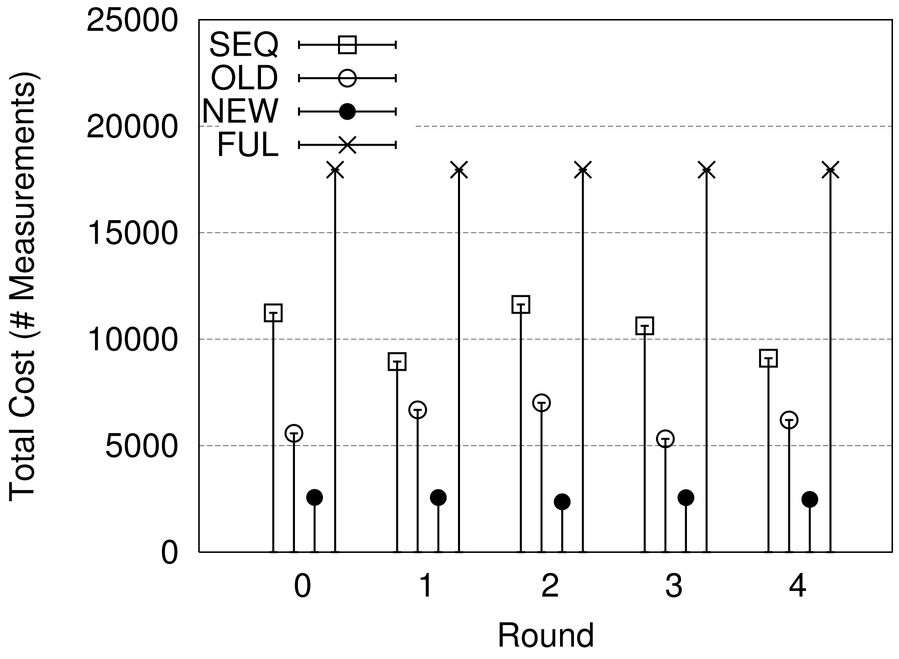


(a) HP-PlanetLab: Prediction error

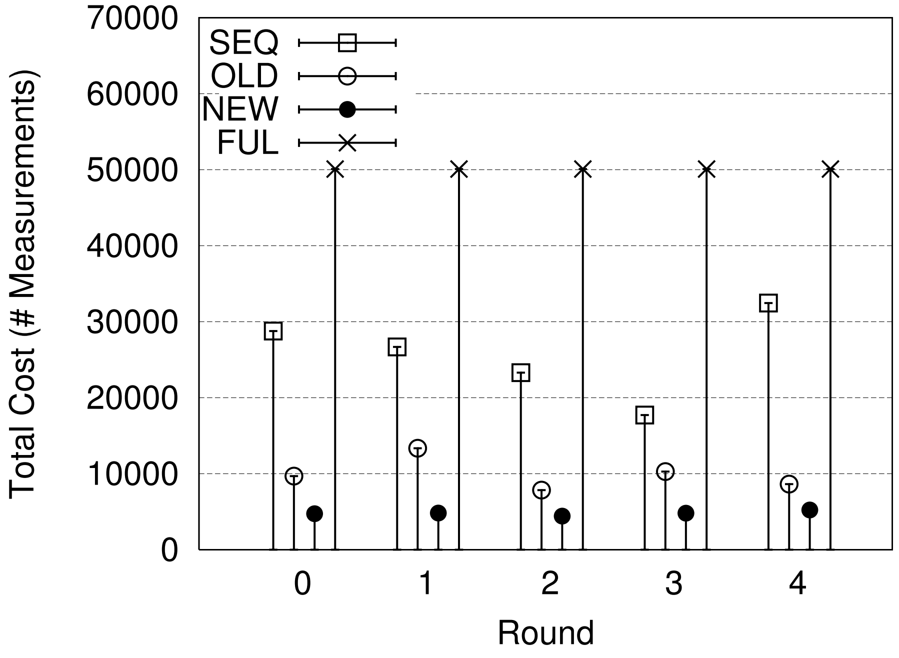


(b) UMD-PlanetLab: Prediction error

Figure 4.5: Bandwidth prediction error

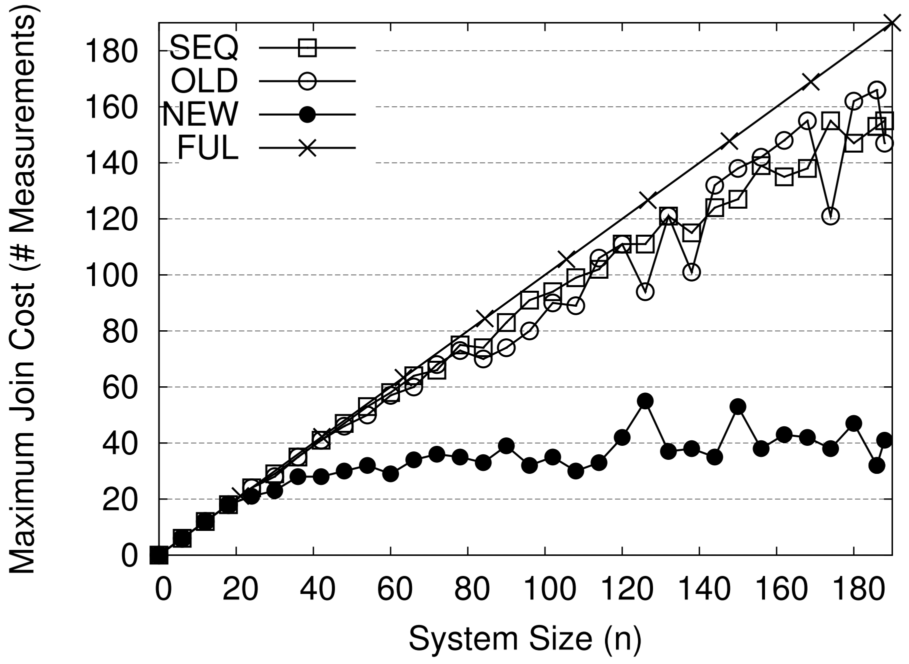


(a) HP-PlanetLab: Total cost

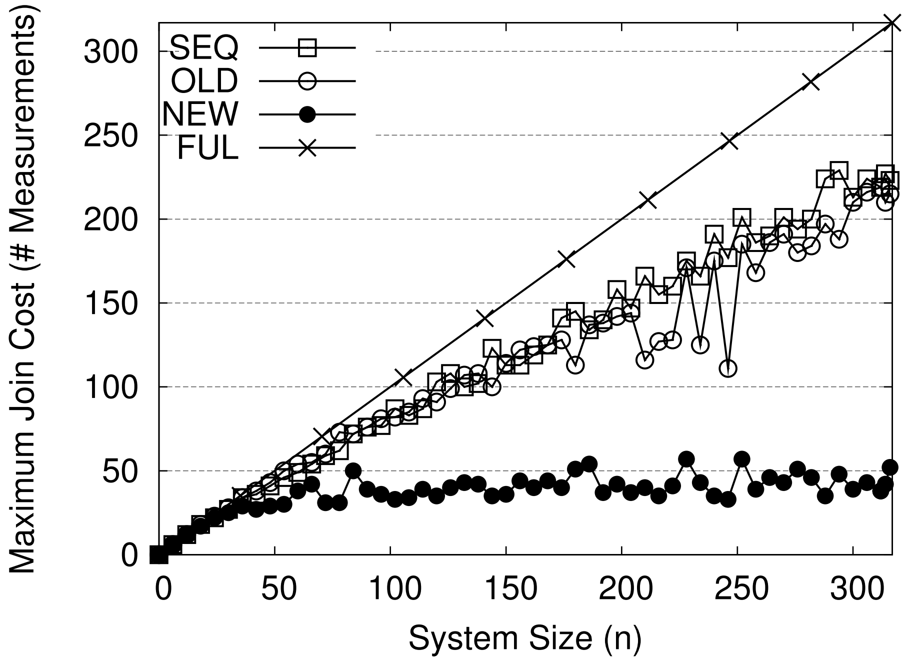


(b) UMD-PlanetLab: Total cost

Figure 4.6: Total measurement cost for bandwidth prediction

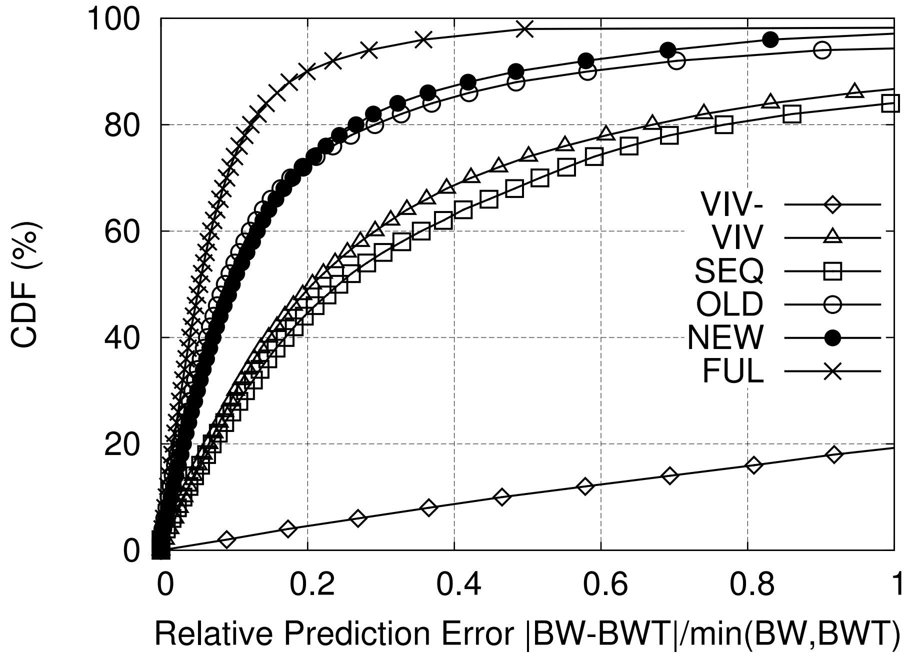


(a) HP-PlanetLab: Worst-case join cost

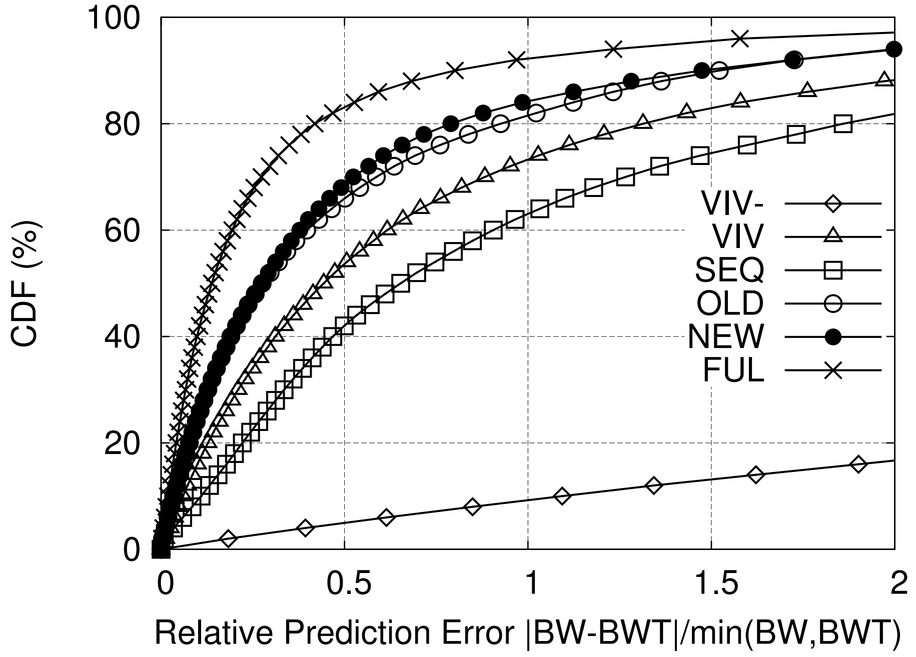


(b) UMD-PlanetLab: Worst-case join cost

Figure 4.7: Worst-case measurement cost for each node



(a) HP-PlanetLab: Prediction error



(b) UMD-PlanetLab: Prediction error

Figure 4.8: CDF of relative prediction errors: Tree metric space approaches are more accurate than 2-d Euclidean space approaches.

Chapter 5

Decentralized Centroid Search

This chapter discusses a decentralized method to find a centroid node that has high-bandwidth connections to a given set of nodes. We first formulate the centroid search problem and describe requirements an algorithm must satisfy. Then the design of algorithms is provided with proofs of correctness. Last, the accuracy and scalability of our approach will be evaluated with several experiments.

5.1 Introduction

We investigate a decentralized algorithm to answer the following question:

> Given a set Q of nodes, find a centroid x that maximizes the objective function $f(x) = \min_{q \in Q}\{BW(x, q)\}$.

By sending or receiving data to or from a centroid node, applications can optimize data locality and thereby increase their overall performance. Unfortunately, there has been little research for decentralized centroid search, especially for bandwidth measurement. There is a distributed approach [50] to find nodes with network distance constraints on a Euclidean coordinate space. But the method only works with network latency constraints rather than bandwidth, as a Euclidean space is not a good model to embed bandwidth as shown in Section 4.4. Although several systems [58, 19] can produce accurate bandwidth prediction data, no effective search algorithms have been studied for those systems. Our approach is to find a centroid using accurate prediction data that are produced on a tree metric space by the algorithm described in Chapter 4. The goal is to design a centroid search algorithm that satisfies the following requirements for a system with n nodes:

- Decentralization: There exists no central component.
- Scalable Message Complexity: A centroid can be found in less than $O(n)$ network hops.
- Scalable Space Complexity: Each node maintains information of size less than $O(n)$.
- High Accuracy: A result centroid should be close to the optimal one.
- Self-Organization: The algorithms handle dynamic network conditions.

This work makes several contributions. First, we provide the design of decentralized centroid search algorithm that has the desired properties. The algorithm avoids significant measurement delays by running on top of the bandwidth prediction framework. The second contribution is a thorough theoretical analysis of our algorithm. The third contribution is a heuristic to increase the accuracy of the algorithm. Finally, we present extensive simulation results validating the high accuracy and scalability of our algorithm.

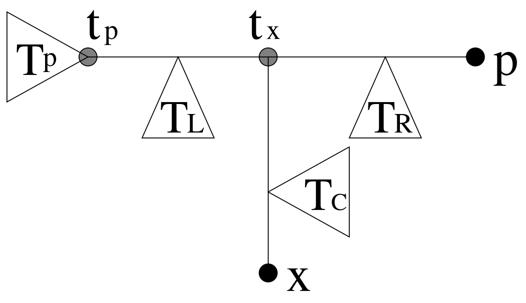


Figure 5.1: Prediction tree from the perspective of x

5.2 Design

To search for a bandwidth-constrained centroid, we define and solve the corresponding distance problem in a metric space:

> Given a set Q of nodes, find a node x that minimizes the objective function $f_d(x) = \max_{q \in Q}\{d(x, q)\}$.

We will use graph distances produced by the algorithms described in Section 4.2, so that we can avoid any measurement delays in centroid search. We first construct the prediction framework, have each node maintain some additional information described in Section 5.2.1, then are able to process a centroid search query as shown in Section 5.2.2.

5.2.1 Dynamic Information Aggregation

Figure 5.1 shows a prediction tree from the perspective of a node x and an inner node t_x owned by x. To support decentralized centroid search, x maintains distance labels for five other nodes: p, s_p, s_L, s_R, and s_C. p is the anchor parent of x. s_p, s_L, s_R, and s_C refer to the closest leaf node to t_x in the sub-trees T_p,

T_L, T_R, and T_C, respectively. Since the size of a distance label is $O(\log n)$, the space complexity of each node remains $O(\log n)$. This information is maintained by a periodic aggregation mechanism in the skip anchor tree. Each node x receives distance labels for p, s_p, and s_L from $x.P$ or $x.L[0]$, s_C from $x.C$, and s_R from $x.R[0]$. For example, after aggregating s_C and s_R, x will send to $x.L[0]$ the closest node to t_x in the set $\{x, s_C, s_R\}$. Then $x.L[0]$ will update $x.L[0].s_R$ with the received information. Distance labels are used to compute distance and determine the closest node as described in Section 4.2.2.

Since it takes some time for each node to propagate the information to other nodes, the aggregated information in each node can be stale. The stale information will decrease the accuracy of centroid search. Fortunately, the information propagation time is reasonably short in our approach, so the possibility of the stale information is low. By utilizing high-level links in a skip list and from the bounded height of the skip anchor tree, it takes only $O(\log^2 n)$ hops to propagate the information across all nodes in the system.

5.2.2 Query Processing

Algorithm 4 finds a centroid for a query set Q. When $|Q| = 1$, the algorithm returns the only node in Q because every node is the closest to itself. For the case of $|Q| \geq 2$, the algorithm first determines the longest path $q_1 \sim q_2$ in a prediction tree. To do that, the algorithm retrieves the distance labels of nodes in Q and computes the distances of all-to-all node pairs in Q. Lemma 5.1 shows that a centroid is the closest node to the midpoint t_m of $q_1 \sim q_2$. So the algorithm finds the owner node u of the inner node t_u that is adjacent to t_m in the prediction tree. Then the closest node to t_m in $\{u, u.p, u.s_p, u.s_L, u.s_R, u.s_C\}$ is chosen as a centroid. Note that Algorithm 4 requires only $O(\log n)$ messages to find u in a skip list.

Lemma 5.1. *The centroid of Q is the closest node in the prediction tree T to the midpoint t_m of the longest path $q_1 \sim q_2$ between nodes in Q.*

Proof. $(\forall q_i \in Q) \;\; d_T(t_m, q_i) \leq d_T(t_m, q_1) = d_T(t_m, q_2)$. For a leaf node x in T, $(\forall q_i \in Q) \;\; d_T(x, q_i) \leq d_T(x, t_m) + d_T(t_m, q_i) \leq \max\{d(x, q_1), d(x, q_2)\}$. So $f_d(x) = \max\{d(x, q_1), d(x, q_2)\}$. Let x^* be a leaf node in T such that $(\forall x) \;\; d_T(x^*, t_m) \leq d_T(x, t_m)$. $(\forall x)$ $f_d(x^*) = \max\{d(x^*, q_1), d(x^*, q_2)\} = d_T(x^*, t_m) + d_T(t_m, q_2) \leq d_T(x, t_m) + d_T(t_m, q_2) = f_d(x)$. Thus, x^* is the centroid. □

Theorem 5.1. *Algorithm 4 correctly finds a centroid for Q.*

Proof. Since the case with $|Q| = 1$ is obvious, we only show the case of $|Q| \geq 2$. Rename x and t_x in Figure 5.1 to u and t_u, respectively. Then put t_m between t_u and T_R. Since $d_T(s, t_m) = d_T(s, t_u) + d_T(t_u, t_m) \; \forall s \in T_p$, $u.s_p$ is the closest to t_m in T_p. Likewise, $u.s_L$, $u.s_R$, and $u.s_C$ are the closest to t_m in T_L, T_R, and T_C, respectively. Thus, the closest node x^* to t_m in $\{u, u.p, u.s_p, u.s_L, u.s_R, u.s_C\}$ must be the closest leaf node to t_m in the whole prediction tree. By Lemma 5.1, x^* is the correct centroid of Q. □

Algorithm 4: **findCentroid**(Q)

```
1  if $|Q| = 1$ then
2      return the only node in $Q$
3  $q_1 \sim q_2 \leftarrow$ the longest path between nodes in $Q$
4  $T \leftarrow$ a partial prediction tree containing $q_1$ and $q_2$
5  Add a midpoint $t_m$ of the path $q_1 \sim q_2$ to $T$
6  $a \leftarrow$ a node in $T$ such that $t_m$ is located on a path $t_a \sim a$
7  if $a.C = null$ then  $u \leftarrow a$
8  else
9      $u \leftarrow a.C.\text{findPred}(d_T(t_R, t_m))$
10     if $u = null$ then  $u \leftarrow a.C$
11 Add $u$, $u.p$, $u.s_p$, $u.s_L$, $u.s_R$, and $u.s_C$ to $T$
12 return the closest node to $t_m$ among the six added nodes
```

5.2.3 Probing for Higher Accuracy

Since the bandwidth prediction information contains errors, the result of a centroid search may not optimize the objective function. To increase the accuracy of centroid search, we modify the algorithm to return the top-k candidate nodes that optimize the objective function based on graph distances. We then perform extra probing measurements between the k candidates and all nodes in Q, and choose the optimal node as the result centroid based on the real measurement data. If we use more candidates with large k, the result of centroid search will become more accurate. However, we cannot use a very large value of k because the probing measurement costs will increase in proportion to k. In Section 5.3, we will see how k affects the centroid search accuracy with experimental results.

To support this technique, we first generalize the dynamic aggregation mechanism described in Section 5.2.1 so that each node aggregates the information of k nodes from each neighbor direction. Then the query processing algorithm (Algorithm 4) is modified as follows. When $|Q| \geq 2$, line 12 of Algorithm 4 is changed to return the top-k nodes from the set of u and k aggregated nodes from each neighbor direction, which are the closest to t_m. When $|Q| = 1$ and q is the only node in Q, the centroid search algorithm should choose the top-k nodes from the set of q and all aggregated nodes that are closest to q.

5.3 Evaluation

This section evaluates our centroid search algorithm. Simulation results will show the high accuracy and scalable cost of our approach.

5.3.1 Experimental Setup

The two data sets described in Section 4.4 to evaluate the prediction algorithms are also used to evaluate centroid searches. We show results from five different approaches for centroid search: **VIV**, **SEQ**, **NEW**, **NEW-HEU-2**, and **NEW-HEU-5**. VIV refers to a centralized and exhaustive search for centroids based on the bandwidth prediction data produced by the Vivaldi simulation named VIV in Section 4.4. SEQ refers to a centralized and exhaustive search for centroids based on the prediction data produced by the Sequoia system named SEQ in Section 4.4. NEW is our decentralized centroid search algorithm running on the prediction data produced by our bandwidth prediction framework named NEW in Section 4.4. NEW-HEU-2 is the improved version of NEW where the probing heuristic is applied with $k = 2$, as described in Section 5.2.3 for higher centroid search accuracy. NEW-HEU-5 also uses the probing heuristic with $k = 5$. As described in Section 4.4, for each approach and each bandwidth data set, we run 100 rounds of system construction with different node join orderings and sampling measurements.

5.3.2 Centroid Search Error

For each round of system construction, 200 queries are submitted to search for a centroid of three randomly selected nodes ($|Q| = 3$). Therefore the total number of queries submitted for each data set is 20000. Although we only show the results for the case of $|Q| = 3$, the experiments with $|Q| = 5$ and $|Q| = 10$ produce the similar results.

The real network cannot be modeled as a perfect tree metric space. So, finding the optimal centroid in a tree metric space does *not* also imply finding the optimal node when real bandwidth values are considered. For each query, we order all nodes in the system by the value of the objective function for centroid search in terms of real bandwidth, and identify the rank of the centroid found by an algorithm. Then we measure the centroid search error for each query by a *relative rank error*, defined by a rank normalized to the total number of nodes. The relative rank error would be zero if an algorithm has found the correct centroid, and 0.1 means the result centroid ranks in the top 10% across all nodes. Figure 5.2(a) shows the cumulative distribution function (CDF) of the relative rank error values for all 20000 queries into the HP-PlanetLab data set, and Figure 5.2(b) is for the UMD-PlanetLab data set.

NEW shows smaller rank error than SEQ and VIV, even though SEQ and VIV employ exhaustive search methods. This is because NEW searches for centroids using more accurate predicted bandwidth information than SEQ and VIV, as shown in Figures 4.5(a) and 4.5(b). Better prediction accuracy results in better centroid search accuracy. SEQ is more accurate for centroid search than VIV, while VIV shows slightly higher accuracy than SEQ for bandwidth prediction in Figures 4.5(a) and 4.5(b). This is because the error minimization technique of VIV can overestimate a predicted bandwidth value BW_T. As described in Section 4.4, VIV minimizes $\frac{|d-d_T|}{d}$ which is equal to $\frac{|BW-BW_T|}{BW_T} = |1 - \frac{BW}{BW_T}|$. So when it is difficult to predict bandwidth accurately, VIV tends to produce very large BW_T and make errors close to one. Then VIV can mistakenly choose a centroid node with a low rank when the centroid node has many overestimated BW_T values for its connections to other nodes.

Despite the high accuracy of NEW, we found that there is still much room to improve NEW. For example, in Figure 5.2(a), only 60% of queries return centroids that rank within the top 20% of nodes that could be selected. This is because the bandwidth data predicted by our NEW framework is still not perfect. So we applied the probing heuristic in NEW-HEU-2 and NEW-HEU-5. NEW-HEU-2 returns two candidate nodes instead of only one node. Then NEW-HEU-2 performs six measurements between each of the two candidate nodes and each of the three nodes in a query input set. The best node is chosen from the two candidate nodes based on the real measurement data. In Figures 5.2(a) and 5.2(b), NEW-HEU-2 shows higher accuracy than NEW. Since NEW-HEU-5 performs more probing measurements using five candidates, NEW-HEU-5 improves the centroid search accuracy even more than NEW-HEU-2. In Figures 5.2(a) and 5.2(b), 90% of queries for NEW-HEU-5, compared to less than 80% for NEW-HEU-2, return centroids that rank within around the top 20% of nodes.

5.3.3 Relative Error of Objective Function Values

The goal of centroid search is to find the top-ranker that maximizes the objective function $f(x) = \min_{q \in Q}\{BW(x,q)\}$. We have shown that our algorithm successfully finds highly-ranked nodes that are associated with large $f(x)$ values for real bandwidth. In addition to confirming low rank errors, it is also important to check how close the quality of a result centroid is to the optimal. So we computed the relative errors of objective function values. Let x^* is the optimal top-ranked centroid that maximizes $f(x)$. For each query result x, we define the relative error of $f(x)$ as $\frac{f(x^*)-f(x)}{f(x^*)}$. Since $f(x^*) \geq f(x)$ must be true for every node x, the error values should range in $[0,1]$.

Figures 5.3(a) and 5.3(b) show the CDF of the relative errors of $f(x)$. The trend among the five approaches is similar to that in relative rank errors. For both data sets, NEW shows lower error than SEQ and VIV. Also, the approaches using probing heuristics help lowering errors. In HP-PlanetLab, NEW shows quite high accuracy in that more than 70% of queries only produce errors less than 0.2. However, all the approaches in UMD-PlanetLab show higher errors than HP-PlanetLab. For example, only 20% of queries in NEW show errors less than 0.2 in UMD-PlanetLab. Also, there is a large gap between the perfect accuracy and all approaches in UMD-PlanetLab. We can find the reason of the high errors in UMD-PlanetLab from the high variance of bandwidth in the data set. Figure 5.3(c) shows the CDF of bandwidth of all node pairs in the two data sets. Bandwidth ranges widely from 0.01 Mbps to 1308 Mbps in UMD-PlanetLab while HP-PlanetLab shows bandwidth between 5 Mbps and 398 Mbps. Because of the long tail of UMD-PlanetLab, the failure to find the optimal centroid x^* can result in very high relative error of $f(x)$. In other words, a result centroid can have a very high error for $f(x)$ even though it has a low rank error.

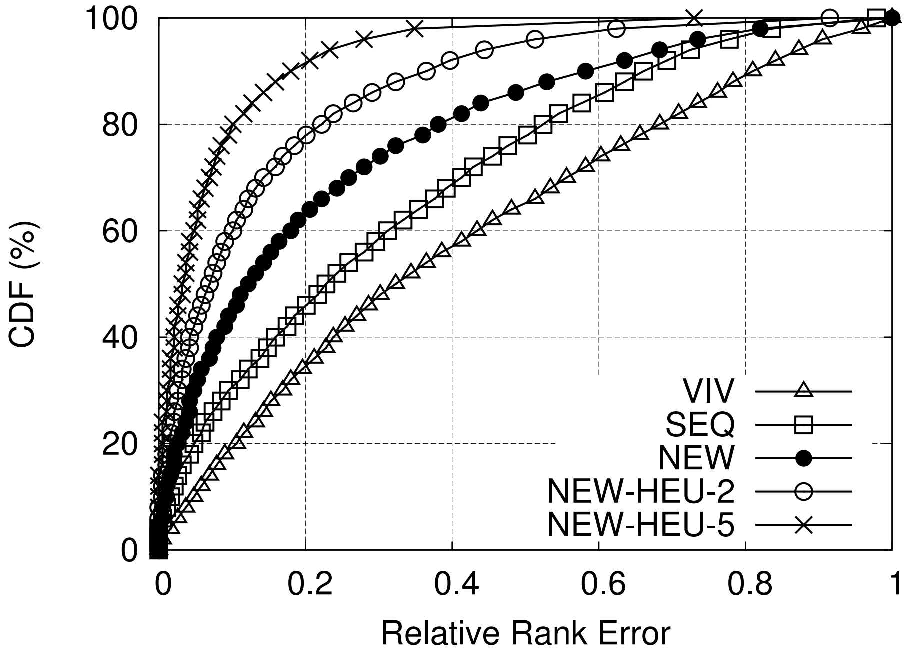
We will leave it as an open problem to optimize the relative error of $f(x)$ regardless of bandwidth distribution. Note that the primary goal is to find a top-ranker for $f(x)$, and our algorithm can find high-ranked nodes correctly. The only open problem is how to reduce the relative error of $f(x)$ in a network condition where bandwidth shows high variance and a long tail in CDF.

5.3.4 Scalable Query Cost

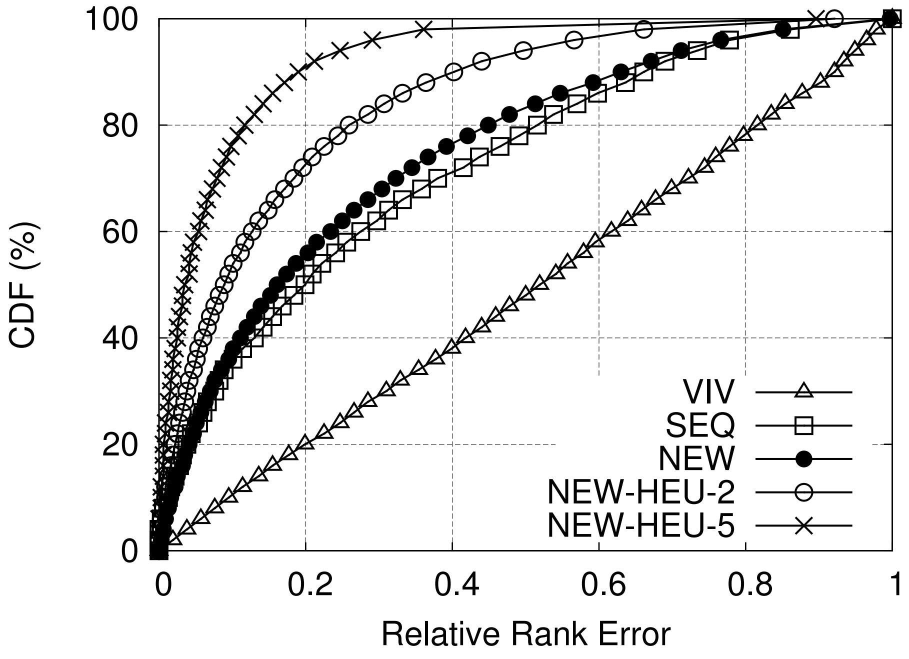
For each round of system construction, we submitted 200 search queries for several different system sizes n. We measured the query cost as the number of network hops that each query needed to find a centroid. Figures 5.4(a) and 5.4(b) show the average, 95-th percentile, and maximum of the centroid search costs across 20000 queries. Since the query costs for NEW, NEW-HEU-2, NEW-HEU-5 are the same, we only show NEW in the figure. Note that VIV and SEQ are not compared here as neither of those is a distributed approach. As expected, the cost of our approach increases in a scalable way as n increases. The cost of our centroid search algorithm should be bounded by $O(\log n)$ hops with high probability, as discussed in Section 5.2. Thus, we confirm that our algorithm finds centroids accurately with scalable query cost.

5.4 Summary

This chapter has presented a decentralized, accurate, and scalable algorithm that can support data-intensive widely distributed applications. The algorithm can find a centroid node with high-bandwidth connections to the desired set of nodes. Our approach requires no centralized component or data structure. The algorithm runs on top of the bandwidth prediction framework, which provides accurate prediction data. We proved the correctness of our centroid search algorithm assuming bandwidth can be represented in a tree metric space. We also described a heuristic to achieve higher accuracy in a real network. Our algorithm requires only $O(\log n)$ network hops to process a search query. Simulation results show the high accuracy and scalability of our approach.

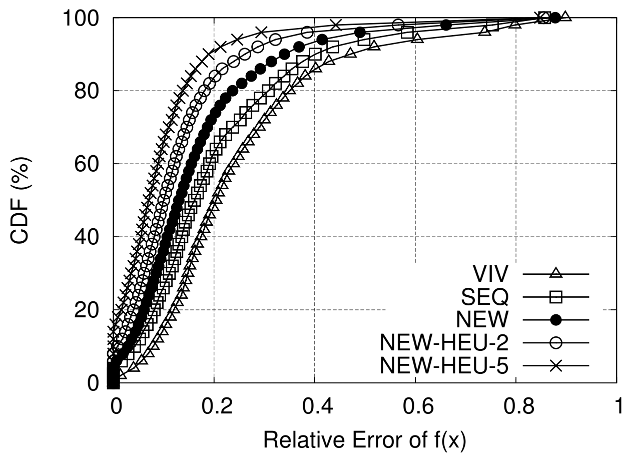


(a) HP-PlanetLab: Search error

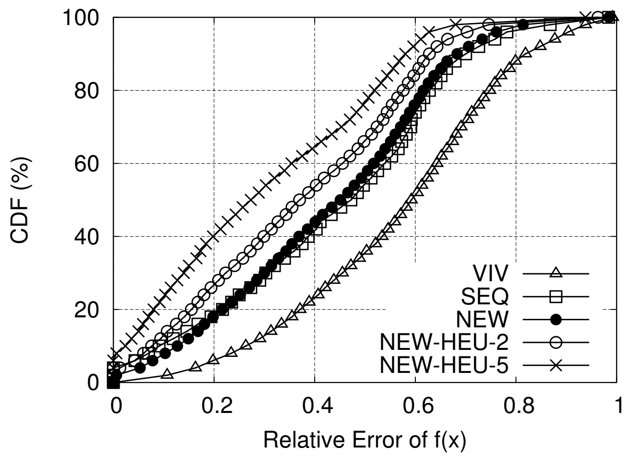


(b) UMD-PlanetLab: Search error

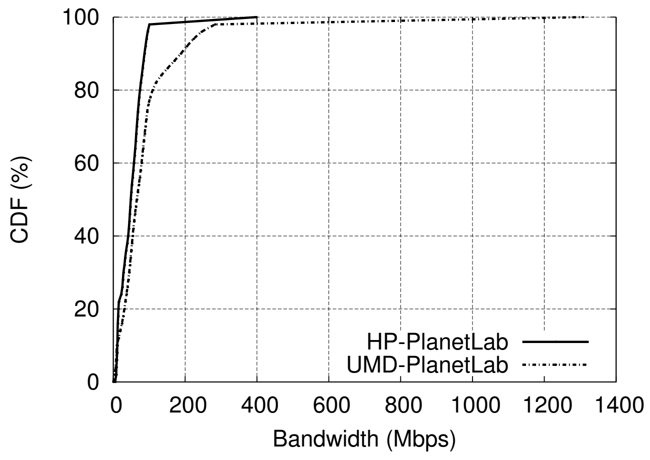
Figure 5.2: Centroid search error



(a) HP-PlanetLab: Relative error of $f(x)$

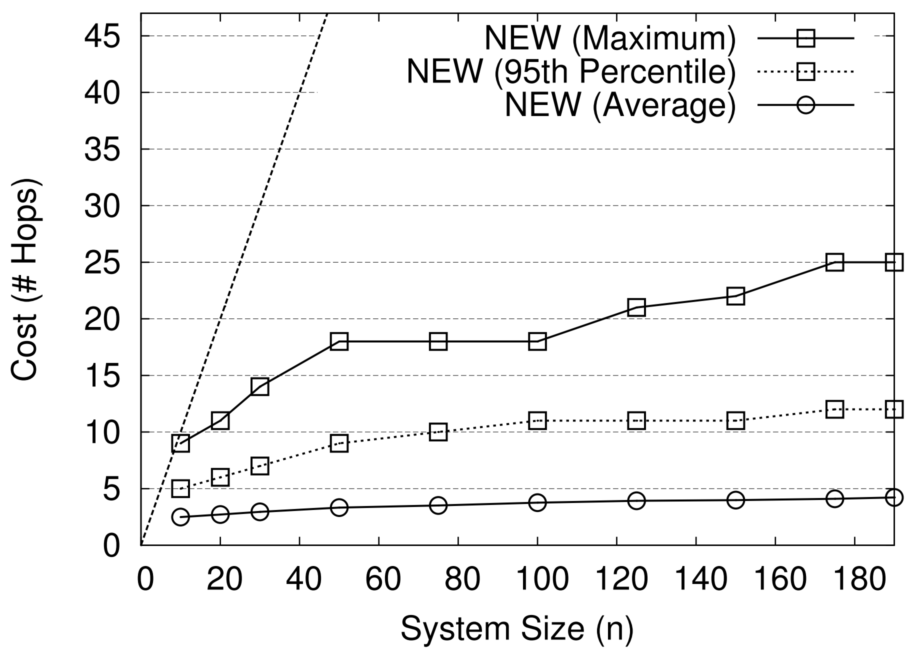


(b) UMD-PlanetLab: Relative error of $f(x)$

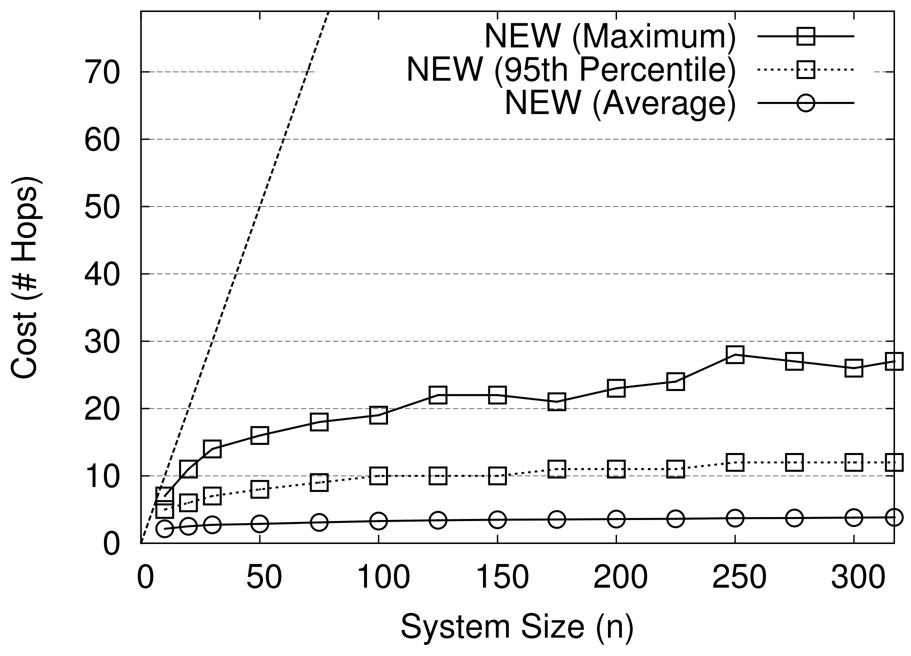


(c) Bandwidth distribution

Figure 5.3: Relative error of objective function values and bandwidth distribution



(a) HP-PlanetLab: Query cost



(b) UMD-PlanetLab: Query cost

Figure 5.4: Query cost of centroid search

Chapter 6

Decentralized Cluster Search

This chapter describes another node search algorithm that works on top of the decentralized bandwidth prediction framework described in Chapter 4. We consider finding a cluster of nodes with high-bandwidth interconnections. We first formulate the cluster search problem and describe the requirements it must satisfy. Then we describe what additional information is needed to support decentralized cluster search and how the information is maintained. Last, the accuracy and scalability of our approach will be evaluated throughout extensive experiments.

6.1 Introduction

We investigate a scalable and decentralized method to answer the following question:

> Given two constraints, a cluster size k and a minimum bandwidth b, find a cluster X of nodes such that $|X| = k$ and $\forall u, v \in X, BW(u, v) \geq b$.

Data-intensive distributed applications can benefit from a cluster search algorithm as nodes in a cluster can send or receive large-scale data to or from one another at a high rate. Unfortunately, there has not been an effective method to find bandwidth-constrained clusters in a decentralized fashion. There are two important reasons for the lack of a solution for cluster search. First, the clustering problem is difficult to solve, in that it is equivalent to the k-clique problem in a general graph, which is NP-complete. Second, there has been no effective framework for bandwidth prediction that can reduce the costs of performing bandwidth measurements. Accordingly, most previous studies only focused on designing heuristics for latency-constrained clustering [49, 63]. Those systems also require a centralized structure [49] or a fixed set of landmark nodes that every node has to perform measurements with [63]. There are theoretical studies [12, 30] that have presented polynomial-time algorithms to find clusters in 2-d Euclidean spaces. We can search for clusters by applying those clustering algorithms to network distances produced by network coordinate systems [26]. However, those algorithms only work with network latency rather than bandwidth, as a Euclidean space is not a good model to embed bandwidth we showed in Section 4.4.

Fortunately, it is now possible to overcome those difficulties as a consequence of our recent research efforts. The decentralized framework for bandwidth prediction described in Chapter 4 accurately embeds bandwidth measurements into a tree metric space. Also, while the clustering problem is NP-complete in a general graph, we could design a polynomial-time algorithm to find clusters by assuming that bandwidth can be represented in a tree metric space. So our approach for decentralized cluster search is to find a cluster using accurate bandwidth prediction data that are produced on a tree metric space by the algorithm described in Chapter 4. Then we can reduce the measurement delays for clustering. The goal is to design a cluster search algorithm that satisfies the following requirements for a system with n nodes:

- Decentralization: There exists no central component.
- Scalable Message Complexity: A cluster can be found in less than $O(n)$ network hops.
- Scalable Space Complexity: Each node maintains information of size less than $O(n)$.
- High Accuracy: Nodes in a result cluster should satisfy the input constraints.
- Self-Organization: The algorithms handle dynamic network conditions.

This work makes several contributions. We first show that the clustering problem can be solved in polynomial time in a tree metric space, by presenting a centralized algorithm and proving its correctness. Then we describe a decentralized polynomial time algorithm that satisfies all the requirements. The key idea is to have each node maintain a routing table on an overlay network so that a query can be routed towards where the desired cluster exists. Finally, we present extensive simulation results validating the high accuracy and scalability, also measuring the cost of the decentralized algorithm.

6.2 Design

This section describes details of our approach for clustering. We first develop a centralized algorithm, and then discuss how to decentralize it with several techniques. To search for a bandwidth-constrained cluster, we define and solve the corresponding distance problem in a tree metric space. Using the rational transform

Algorithm 5: **centralizedFindCluster($\mathbf{V}, \mathbf{d}, \mathbf{k}, \mathbf{l}$)**: A centralized algorithm to find in a tree metric space (V, d) a set $X \subseteq V$ such that $|X| = k$ and $diam(X) \leq l$

```
1 $X \leftarrow \{\}$
2 foreach node pair $p, q \in V$ such that $d(p, q) \leq l$ do
3     $S^*_{pq} \leftarrow \{x \in V : d(x, p) \leq d(p, q) \wedge d(x, q) \leq d(p, q)\}$
4     if $|S^*_{pq}| \geq k$ then
5         $X \leftarrow$ a set of any $k$ nodes in $S^*_{pq}$
6         break
7 return $X$
```

function described in Section 4.3, we can convert the bandwidth function BW to a distance function d and the bandwidth constraint b to a distance constraint $l = \frac{C}{b}$. We can also define the *diameter* of a node set X as $diam(X) = \max_{\forall u,v \in X}\{d(u, v)\}$. As a result, we can define this distance-constrained clustering problem:

> Given a tree metric space (V, d) and constraints k and l, find a set $X \subseteq V$ such that $|X| = k$ and $diam(X) \leq l$.

6.2.1 Centralized Clustering in a Tree Metric Space

Algorithm 6 describes a simple centralized algorithm to find a cluster in a tree metric space (V, d). To explain the underlying intuition of the algorithm, we first present a brute-force approach for the cluster search problem. For two nodes

$p, q \in V$, let S_{pq} denote a cluster of nodes such that $S_{pq} \subseteq V$, $p, q \in S_{pq}$, and $diam(S_{pq}) = d(p, q)$, and let G_{pq} be a set containing all S_{pq}'s. Then the union of G_{pq} for all $p, q \in V$ should be equal to a set of all possible non-empty clusters that can be created in (V, d) regardless of constraints k and l. So, we can find a cluster by iterating through all sets S_{pq} in G_{pq} for all $p, q \in V$ and checking if each set satisfies the constraints k and l. This brute-force algorithm requires exponential time to iterate through all sets in G_{pq}.

We can reduce the exponential time of the brute-force approach by checking just the one maximum-sized cluster S^*_{pq} for each G_{pq}. Since all clusters in G_{pq} have the same diameter determined by the same node pair (p, q), we do not have to iterate over all S_{pq}'s in each G_{pq} to find a cluster with a desired size. Instead, we can just compute the maximum-sized set S^*_{pq} and return any k nodes from S^*_{pq}.

With this intuition, for each node pair $p, q \in V$ such that $d(p, q) \leq l$, Algorithm 6 determines S^*_{pq} by collecting all nodes $x \in V$ such that $d(x, p) \leq d(p, q)$ and $d(x, q) \leq d(p, q)$. From the proof of Theorem 6.1, we will show that Algorithm 6 correctly creates S^*_{pq}. If $|S^*_{pq}| \geq k$, then the algorithm stops iterating over node pairs and returns any k nodes in S^*_{pq}. If such S^*_{pq} is not found, we can be sure that a cluster does not exist because all the maximum size clusters S^*_{pq} in each group G_{pq} of clusters with the same diameter are examined.

Theorem 6.1. (Correctness of Algorithm 6) *Given a tree metric space (V, d) and constraint values k and l, if Algorithm 6 creates S^*_{pq} for a pair of nodes $p, q \in V$, then i) $diam(S^*_{pq}) = d(p, q)$ and ii) there exists no $S_{pq} \subseteq V$ such that $p, q \in S_{pq}$,*

$diam(S_{pq}) = d(p,q)$, and $|S_{pq}| > |S^*_{pq}|$.

Proof. To prove $diam(S^*_{pq}) = d(p,q)$, we will show $d(r,s) \leq d(p,q)\ \forall r,s \in S^*_{pq}$. If $r \in \{p,q\}$ or $s \in \{p,q\}$, it is clear that $d(r,s) \leq d(p,q)$ by definition of S^*_{pq} in Algorithm 6. Otherwise, three cases are considered by the order of three distance sums among the four nodes p, q, r, and s: $d(p,q)+d(r,s)$, $d(p,r)+d(q,s)$, and $d(p,s)+d(q,r)$. Note that, by the definition of the four-point condition, two large sums out of those three sums must be equal to each other.

- Case 1: $d(p,q)+d(r,s) \leq d(p,r)+d(q,s) = d(p,s)+d(q,r)$

  By the assumption of Case 1, $d(r,s) \leq d(p,r)+d(q,s)-d(p,q)$ holds. So, it is true that $d(r,s)-d(p,q) \leq (d(p,r)-d(p,q))+(d(q,s)-d(p,q))$. Since $d(p,r) \leq d(p,q)$ and $d(q,s) \leq d(p,q)$ is satisfied by the definition of S^*_{pq}, $d(r,s)-d(p,q) \leq 0$.

- Case 2: $d(p,r)+d(q,s) \leq d(p,s)+d(q,r) = d(p,q)+d(r,s)$

  By the assumption of Case 2, $d(r,s) = d(p,s)+d(q,r)-d(p,q)$ holds. So, it is true that $d(r,s)-d(p,q) = (d(p,s)-d(p,q))+(d(q,r)-d(p,q))$. Since $d(p,s) \leq d(p,q)$ and $d(q,r) \leq d(p,q)$ is satisfied by the definition of S^*_{pq}, $d(r,s)-d(p,q) \leq 0$.

- Case 3: $d(p,s)+d(q,r) \leq d(p,r)+d(q,s) = d(p,q)+d(r,s)$

  Similarly to Case 2, $d(r,s)-d(p,q) \leq 0$.

In all three cases, $d(r,s) \leq d(p,q)\ \forall r,s \in S^*_{pq}$ holds. Thus, $diam(S^*_{pq}) = d(p,q)$.

Now let's assume that there exists $S_{pq} \subseteq V$ such that $p,q \in S_{pq}$, $diam(S_{pq}) =$

$d(p,q)$, and $|S_{pq}| > |S^*_{pq}|$. Then there must exist a node $x \in V$ such that $x \in S_{pq}$ and $x \notin S^*_{pq}$. For such a node x, $d(x,p) \leq d(p,q) \wedge d(x,q) \leq d(p,q)$ holds because $diam(S_{pq}) = d(p,q)$. However, by the definition of S^*_{pq}, $x \notin S^*_{pq}$ implies $d(x,p) > d(p,q) \vee d(x,q) > d(p,q)$, which causes a contradiction. Thus, $S^*_{pq} \subseteq V$ is the maximum set such that $p,q \in S_{pq}$ and $diam(S_{pq}) = d(p,q)$. □

When $|V| = n$, the algorithm takes $O(n^3)$ time because it takes $O(n)$ to create S^*_{pq} and $O(n^2)$ to iterate over every pair. We do not claim Algorithm 6 is the fastest algorithm to find a cluster in a tree metric space. The point is that there actually exists an effective algorithm to solve the clustering problem in a tree metric space. While the clustering problem is NP-complete in a general graph as described in Chapter 3, Algorithm 6 can find a cluster in polynomial time by determining S^*_{pq} under the assumptions of a tree metric space. Since bandwidth can be embedded accurately into a tree metric space as described in Chapter 2 and 4, Algorithm 6 can be applied to find a bandwidth-constrained cluster.

6.2.2 Decentralization

Our basic strategy for decentralized cluster search is to use graph distances produced by the algorithms described in Section 4.2, so that we can avoid any measurement delays. We construct an overlay network for the prediction framework and have each node maintain two additional types of information. A *clustering space (CS)* of a node is a set of nodes that are close to the node in a prediction tree, and a *cluster routing table (CRT)* is a table that stores (1) the maximum size of clusters that can be formed on clustering spaces of other nodes, and (2) the direction of the maximum-sized clusters in an overlay network. Then nodes are able to process a cluster search query by routing a query towards a direction where a desired cluster exists on an overlay network.

The rest of this section describes how a clustering space and a cluster routing table are created and maintained, and how they are used to process a cluster search query.

6.2.2.1 Clustering Space and Cluster Routing Table

A clustering space of a node x is a set of close nodes to x in a prediction tree. Recall Figure 5.1 used in Chapter 5 that shows a prediction tree from the perspective of a node x. p refers to the anchor parent of x. Given a user-defined parameter n_{cut}, let S_p, S_L, S_R, and S_C refer to the n_{cut} closest leaf nodes to t_x in a sub-tree T_p, T_L, T_R, and T_C, respectively. And let S'_C refer to the n_{cut} closest leaf nodes to x in a sub-tree T_C. Then the clustering space x.CS is defined as $\{x, p\} \cup S_p \cup S_L \cup S_R \cup S'_C$.

A cluster routing table of a node x stores the maximum size of clusters that can be formed on clustering spaces of other nodes in each direction of neighbors of x in a skip anchor tree. Each row of the cluster routing table x.CRT of a node x represents each neighbor m of x from $\{P, C, L[0], L[1], \cdots, L[h_t - 1], R[0], R[1], \cdots, R[h_t - 1]\}$, and each column represents a diameter constraint l in a predetermined discrete set of diameter constraints. Let U_m denote a set of nodes that x can only reach via m on the skip anchor tree. Then x.CRT$[m][l]$ means the maximum cluster size that nodes in U_m can form for a diameter constraint l in their clustering space. For example, U_C is equal to a set of leaf nodes in T_C in Figure 5.1, and x.CRT$[C][l]$ is the maximum size of clusters with diameter $\leq l$ that can be formed in clustering spaces of leaf nodes in T_C. Note that there is also a row for x to represent the maximum cluster size that can be formed in the clustering space of x. Table 6.1 shows an example of a cluster routing table of a node x. Each column represents a diameter constraint l in a set $\{10000, 1000, 100, 10\}$ and the corresponding bandwidth constraint b assuming constant $C = 10000$ is used for $l = \frac{C}{b}$ conversion. Each row represents the neighbors of x in a skip anchor tree. x is the head of a skip list in the example, so left neighbors do not exist. x.CRT$[P][1000]$ is equal to 18, which means there exists a set of 18 nodes, in the direction of the anchor parent neighbor P in a skip anchor tree, such that every distance between nodes is less than or equal to 1000.

As the size of each of the four node sets S_p, S_L, S_R, and S'_C is bounded by n_{cut}, our decentralized approach limits the size of the clustering space up to $4n_{\text{cut}}+2$ nodes. This limitation potentially allows each node to create a cluster with only a small number of nodes. Accordingly, our decentralized algorithm might not be so responsive as the centralized one for difficult queries with large k. However, it is presumably rare that a user wants to find a cluster of very large size. As long as k is small, our decentralized approach becomes as responsive as the centralized one whatever a diameter constraint l is. This is because the quality of the clustering space is good in that the clustering space is formed with the closest nodes in each neighbor direction.

Having a predetermined diameter set limits the freedom of choosing a band-

Table 6.1: An example of cluster routing table of a node x

Constraint l Constraint b	$\leq$ 10000 $\geq$ 1Mbps	$\leq$ 1000 $\geq$ 10Mbps	$\leq$ 100 $\geq$ 100Mbps	$\leq$ 10 $\geq$ 1000Mbps
x	20	15	8	5
P	25	18	10	2
C	32	26	15	3
$L[0]$	N/A	N/A	N/A	N/A
$L[1]$	N/A	N/A	N/A	N/A
$L[2]$	N/A	N/A	N/A	N/A
$L[3]$	N/A	N/A	N/A	N/A
$R[0]$	40	28	13	8
$R[1]$	29	22	8	4
$R[2]$	22	21	7	4
$R[3]$	15	12	5	2

width constraint in a cluster search query, which is another cost of our decentralized approach. However, we believe that users will be satisfied with coarse-grained specification of bandwidth constraints as long as they can find clusters in a scalable and decentralized way.

6.2.2.2 Dynamic Aggregation

To form x.CS, a node x maintains distance labels for p and nodes in six sets S_p, S_L, S_R, S_C, S'_C, and S'_R. S'_R refers to the n_{cut} closest leaf nodes to p in a subtree T_R in Figure 5.1. This information is maintained by a periodic aggregation mechanism in the skip anchor tree. x receives p, S_p, and S_L from $x.P$ or $x.L[0]$, S_C and S'_C from $x.C$, and S_R from $x.R[0]$. For example, after aggregating S_C and S_R, x will send to $x.L[0]$ the n_{cut} closest nodes to t_x in the set $\{x\} \cup S_C \cup S_R$. Then $x.L[0]$ will update $x.L[0].S_R$ with the received information. Note that S_C and S'_R are not directly used to form a clustering space, but S_C is used to update S_p, S_L, and S_R, and S'_R is used to update S'_C. By utilizing high-level links in a skip list, we can propagate the information to all the nodes in a skip list within $O(\log n)$ hops with high probability. Thus, it takes $O(\log^2 n)$ hops for all the nodes in a system to receive the information.

x periodically fills in the row x.CRT$[x]$ by running Algorithm 6 in $x.CS$ for each l in the diameter set. And x receives the information for a row x.CRT$[m]$ from each neighbor $m \in \{P, C, L[0], L[1], \cdots, L[h_t - 1], R[0], R[1], \cdots, R[h_t - 1]\}$. For example, after aggregating x.CRT$[R[0]]$ and x.CRT$[C]$, x will send to $x.L[0]$ the entry $x.L[0]$.CRT$[x][l]$ for each l in a diameter set. The entry is equal to max$\{x$.CRT$[x][l]$, x.CRT$[R[0]][l]$, x.CRT$[C][l]\}$. For fast aggregation, x utilizes high-level links in a skip list. Like the aggregation for clustering spaces, it takes $O(\log^2 n)$ hops for all the nodes in a system to update cluster routing tables.

6.2.2.3 Query Processing

A user can initiate a distributed cluster search by sending a query (k, l) to any node. Then the query routes towards a direction in a skip anchor tree where there exists a node that can form a desired cluster in its clustering space. Algorithm 6 shows how a node x processes a query (k, l) that is forwarded from a node x_{prev}. If $x.CRT[x][l] \geq k$ holds, then x runs the centralized cluster search algorithm (Algorithm 6) in $x.CS$ and return the result. Otherwise, x collects candidate neighbors m such that $x.CRT[m][l] \geq k$ holds, and forwards the query to a random node among the candidates. If x knows that there does not exist any cluster in any direction, the algorithm returns an empty set. We can avoid any possibility of routing in an infinite cycle with the following strategies. First, a query should not be forwarded back to the previous node. Also, a query should move in one-way direction in each skip list, so if a node receives a query from a left (or right) neighbor, the query should not be forwarded back to any left (or right) neighbors. The query routing algorithm utilizes high-level links in skip lists for fast search. Among neighbors in a skip list, the highest-level neighbor m such that $x.CRT[m][l] \geq k$ holds is only considered as a candidate node to be forwarded the query to. For example, if a node x has a CRT as shown in Table 6.1 and receives a query ($k = 20$, $l = 1000$), then x forwards the query to either C or $R[2]$. Although $x.\text{CRT}[R[1]][1000] \geq 20$ and $x.\text{CRT}[R[0]][1000] \geq 20$, the query routing algorithm only chooses the highest-level neighbor $R[2]$ as a possible forwarding direction.

The query routing takes $O(\log n)$ hops in each skip list with high probability, and a query should traverses $O(\log n)$ skip lists in a skip anchor tree, so $O(\log^2 n)$ hops are needed to find clusters if one exists.

Algorithm 6: x.**findCluster(**k, l, x_{prev}**)**: Node x's procedure to process a query (k, l) forwarded from node x_{prev}

```
1  if $k \leq x.\text{CRT}[x][l]$ then
2      $d_{\text{CS}} \leftarrow$ the distance function on the clustering space $x.\text{CS}$
3      $X \leftarrow \text{centralizedFindCluster}(x.\text{CS},\ d_{\text{CS}}, k, l)$
4  else
5      $S_{\text{next}} \leftarrow \{\}$
6      Add the anchor parent neighbor $P$ to $S_{\text{next}}$ if $k \leq x.\text{CRT}[P][l]$
7      Add the anchor child neighbor $C$ to $S_{\text{next}}$ if $k \leq x.\text{CRT}[C][l]$
8      Add the highest-level left neighbor $L[i]$ to $S_{\text{next}}$ s.t. $k \leq x.\text{CRT}[L[i]][l]$
9      Add the highest-level right neighbor $R[i]$ to $S_{\text{next}}$ s.t. $k \leq x.\text{CRT}[R[i]][l]$
10     $S_{\text{next}} \leftarrow S_{\text{next}} \setminus \{x_{\text{prev}}\}$
11     if $S_{\text{next}} = \{\}$ then
12         $X = \{\}$
13     else
14         $x_{\text{next}} \leftarrow$ a random node from $S_{\text{next}}$
15         $X = x_{\text{next}}.\text{findCluster}(k, l, x)$
16 return $X$
```

6.3 Evaluation

This section evaluates our approach by examining i) the accuracy of our clustering approach, ii) scalability of the query routing, and iii) the cost of decentralization. The two data sets described in Section 4.4 to evaluate the prediction algorithms are also used to evaluate the cluster search algorithm.

6.3.1 Accuracy

Since there are no extant systems to find bandwidth-constrained clusters in the literature, we designed a comparison model by combining two algorithms: Vivaldi [26] and the 2-d clustering algorithm [12]. We first embed bandwidth measurements into 2-d Euclidean coordinate space using Vivaldi network coordinates, along with the rational transform function described, as VIV in Section 4.4. To find a cluster in 2-d Euclidean coordinate space, we use a centralized 2-d clustering algorithm that is described in the literature [12]. Since the 2-d clustering algorithm was originally designed to find a set of k nodes that has minimum diameter, we slightly changed it to apply to our problem by adding a diameter constraint l. As the correctness of the 2-d clustering algorithm is known, clustering error in this comparison model only comes from imperfect bandwidth embedding in Euclidean space. This is analogous to our centralized clustering approach, where error comes only from imperfect bandwidth embedding into the tree metric space. We used the C++ Vivaldi simulator [9], and implemented the 2-d clustering algorithm in Python.

We show the results of three different approaches: **VIV**, **SEQ**, and **NEW**.

VIV is the result of the comparison model described above. It uses the 2-d clustering algorithm running with the bandwidth data predicted by Vivaldi in a 2-d Euclidean space. SEQ refers to the result of our centralized cluster search algorithm applied to the prediction data produced by the Sequoia system. NEW indicates the result of our decentralized clustering algorithm running on our decentralized bandwidth prediction framework. Note that both NEW and SEQ show the result of our contributions for finding bandwidth-constrained clusters in a tree metric space.

We first constructed a clustering system for each approach with the HP-PlanetLab data set. Then we sent 200 easily satisfied queries with small cluster size constraint k such that all three approaches could always find clusters. With these easy queries, we could fairly compare the three approaches with respect to clustering accuracy. In each query, k is set at 10 nodes, which is 5% of the total number of nodes in the data set. Since k is already small, we choose bandwidth constraint b in a large range. b is set at $15 - 75$ Mbps that is between the 20-th percentile and 80-th percentile of real bandwidth in the data set. We evaluated 100 distinct clustering systems with different random seeds, so the result is a total of 20000 queries examined for each of the three approaches.

We define the *wrong pair rate (WPR)* metric to compare the three approaches. For a cluster search problem with a bandwidth constraint b, a wrong pair is a pair of nodes in a result cluster with interconnection bandwidth less than b. WPR is the ratio of the number of wrong pairs to the number of all pairs in all the returned clusters. Figure 6.1(a) shows WPR with increasing b for all three approaches. There are more choices of wrong pairs for queries with large b, which results in a high WPR.

Note that the inaccuracy of the result clusters only comes from the inaccuracy of the underlying prediction framework for all three approaches. So it is natural that NEW shows lower WPR than SEQ and VIV because NEW has higher prediction accuracy than the other two approaches, as shown in Section 4.4. SEQ shows higher accuracy in cluster search while VIV is slightly more accurate than SEQ for bandwidth prediction. This is because VIV can overestimate prediction bandwidth value, as described in Section 5.3.

We executed the same simulations for UMD-PlanetLab except that we used different queries such that $k = 16$ nodes and $b = 30 - 110$ Mbps. Such k and b are chosen using the same criteria as in the simulations for HP-PlanetLab. As shown in Figure 6.1(b), the results for UMD-PlanetLab show the same trends as those for HP-PlanetLab.

6.3.2 Scalable Query Cost

For each round of system construction, we submitted 200 search queries for several different system sizes n. Then we measured the query cost as the number of network hops that each query needed to find a cluster. Figures 6.2(a) and 6.2(b) show the average, 95-th percentile, and maximum of the cluster search costs across 20000 queries. Note that VIV and SEQ are not compared here as neither of those is a distributed approach. As the system size increases, the cluster search cost increases in a scalable way. This is because the cost is bounded by $O(\log^2 n)$ with high probability, as described in Section 6.2. Thus, we confirm that our algorithm finds clusters accurately with scalable query cost.

6.3.3 Cost of Decentralization

One of the costs of decentralizing the clustering algorithm is the need to limit the size of clustering spaces. Each node only aggregates up to n_{cut} number of nodes from each neighbor direction. Accordingly, the decentralized algorithm might not be as responsive as the centralized one for difficult queries with large k. This limitation lets us tune the messaging workload to any desired degree. However, it potentially allows each node to create a cluster with only a small number of nodes. Accordingly, the decentralized algorithm might not be as responsive, or accurate, as the centralized one for difficult queries with large k.

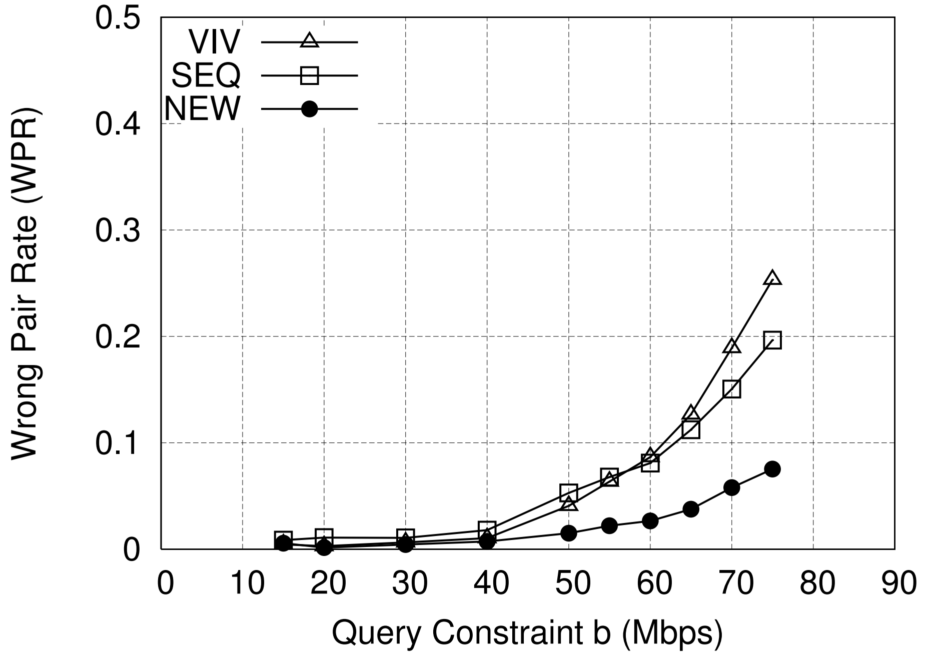
Figure 6.3(a) shows this cost for the HP-PlanetLab data set. We constructed clustering systems with different n_{cut} values of 10, 15, 20, and 1000, and sent 200 queries, each of which is a pair (k, b) chosen from $k = 5 - 45$ nodes and $b = 15 - 75$ Mbps. When $k \leq n_{\text{cut}}$, our decentralized cluster search algorithm is as responsive as the centralized algorithm. And if $k > 4n_{\text{cut}} + 2$, our decentralized algorithm cannot find any cluster with k nodes. To see the whole change in responsiveness of our decentralized algorithm for $n_{\text{cut}} = 10$, we chose k between $0.5n_{\text{cut}} = 5$ and $4.5n_{\text{cut}} = 45$. Also, we wanted to choose b in a large range, so we set b between the 20-th percentile (15 Mbps) and 80-th percentile (75 Mbps) of real bandwidth in the data set. In Figure 6.3(a), NEW-X refers to the system with $n_{\text{cut}} = X$. As each node in NEW-1000 maintains a clustering space containing all the nodes in the system, the result of NEW-1000 represents a centralized clustering approach.

We define the *return rate (RR)* as the ratio of the number of found clusters to the number of submitted queries. As shown in Figure 6.3(a), RR increases as n_{cut} increases because a large value of n_{cut} means that each node can have a large clustering space. Also, as a query gets more difficult with larger k, RR gets smaller for all approaches. RR becomes zero when k is large ($> 4n_{\text{cut}} + 2$) because each node can have a clustering space of size at most $4n_{\text{cut}}+2$. However, when k is small, near n_{cut}, RR is close to centralized clustering (NEW-1000) regardless of b. This is because each node creates a clustering space by aggregating the highest-bandwidth nodes. Thus, we can conclude that our decentralized approach shows a high responsiveness compared to our centralized approach for queries with a reasonably small constraint k relative to the parameter n_{cut}.

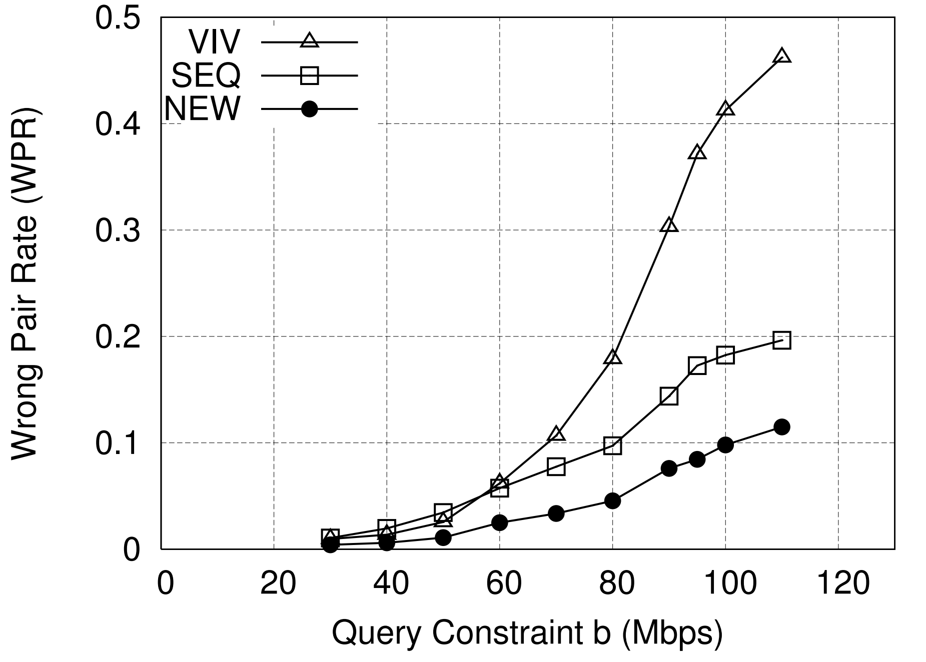
We executed the same simulations for UMD-PlanetLab except that we used different n_{cut} values and queries such that $k = 8 - 72$ nodes and $b = 30 - 110$ Mbps. Such k and b are chosen using the same criteria as in the simulations for HP-PlanetLab. The result in Figure 6.3(b) shows a similar trend to what we found with HP-PlanetLab.

6.4 Summary

This chapter has presented a decentralized algorithm to find a cluster of Internet hosts that are connected via high-bandwidth interconnections. We presented a polynomial-time solution for solving the cluster problem when mapped onto a tree metric space, along with the proof of correctness. We also described a decentralized solution that routes queries along a distributed prediction framework. The decentralized algorithm by dynamically aggregating and propagating local information along the prediction framework, limiting overhead by constraining the amount of information so communicated. Simulation results show that the accuracy of the decentralized approach nonetheless closely approximates that of the centralized approach when the desired cluster sizes are not a large fraction of total system size. We also show that the decentralized algorithm is highly scalable: the query routing only takes $O(\log^2 n)$ network hops with high probability.

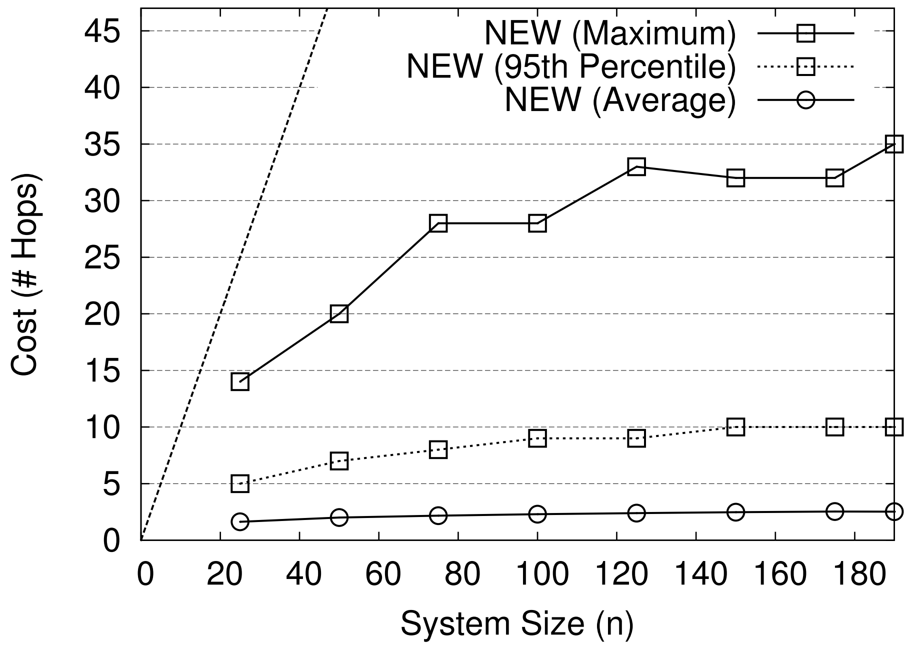


(a) HP-PlanetLab: Wrong pair rate

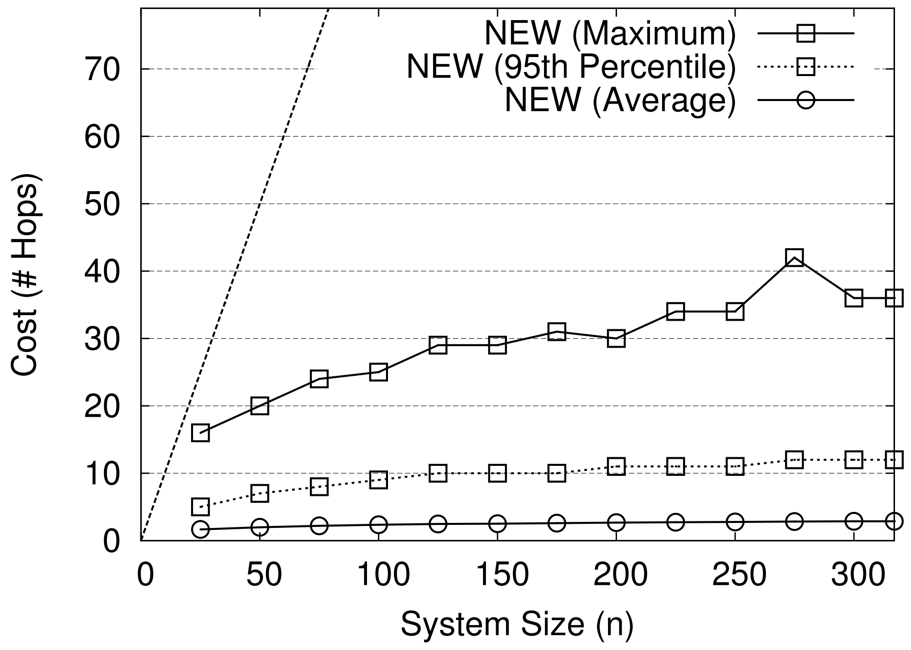


(b) UMD-PlanetLab: Wrong pair rate

Figure 6.1: Cluster search error

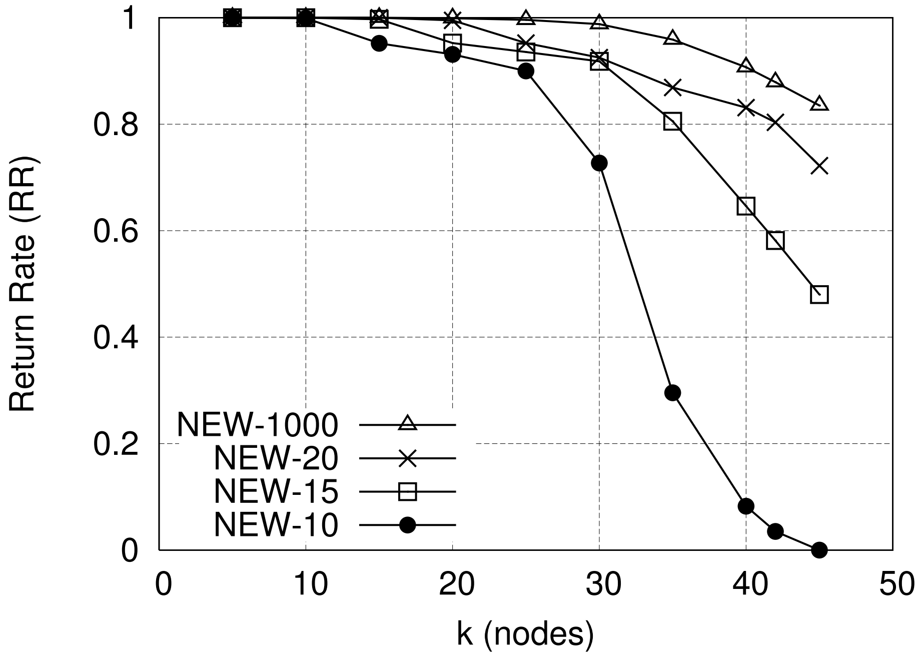


(a) HP-PlanetLab: Query cost

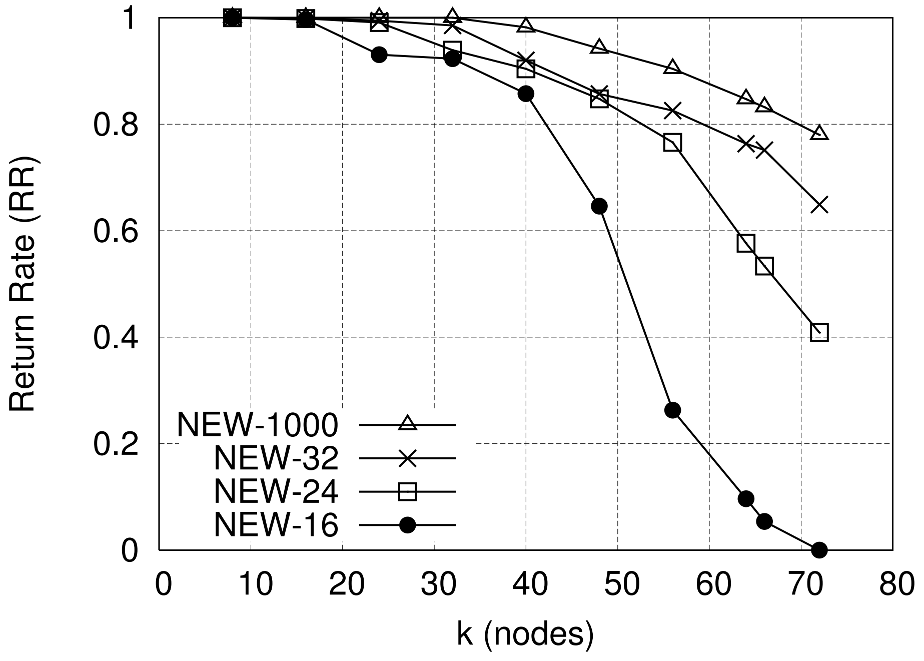


(b) UMD-PlanetLab: Query cost

Figure 6.2: Query cost of cluster search



(a) HP-PlanetLab: Return rate



(b) UMD-PlanetLab: Return rate

Figure 6.3: Cost of decentralization

Chapter 7

Future Work

This chapter describes possible extensions to the work presented in this dissertation.

7.1 Empirical Study for Dynamic Environments

Network bandwidth can dynamically change over time. We described the mechanisms to support such dynamic network environments. However, we still need to conduct experiments to investigate the behaviors of our algorithms in practice. We can collect bandwidth data sets at different time frames for a long period of time. Then we can see how our algorithms organize themselves and produce prediction data. After that, we can also implement a real system and run it over wide-area Internet hosts.

7.2 Prediction of Asymmetric Bandwidth

In this dissertation, we considered bandwidth in a metric space where the symmetric property of metric should be satisfied. Accordingly, we focused on predicting and utilizing the average bandwidth of the forward and reverse directions. However, the forward routing path between two nodes in the Internet can be different from the reverse path. The forward capacity of a physical link can also be different from

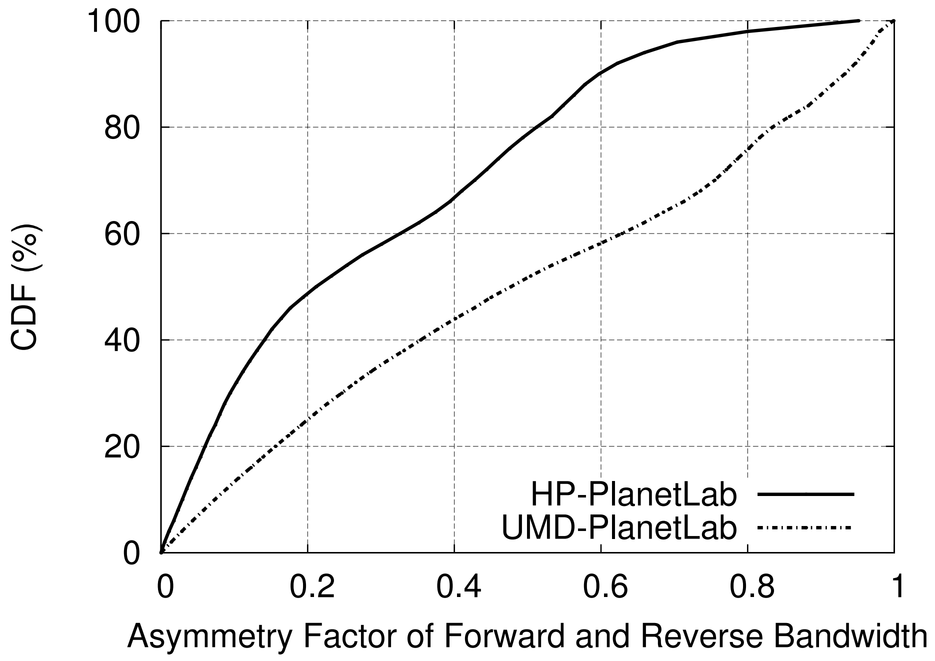


Figure 7.1: Asymmetry factor $\alpha \in [0, 1]$ of forward and reverse bandwidth

the reverse capacity. Because of those asymmetries, there should exist some degree of asymmetry in forward and reverse bandwidth between two network nodes.

To quantify the asymmetry of bandwidth, we can compute the *asymmetry factors* [45] for bandwidth data sets. The asymmetry factor $\alpha \in [0, 1]$ of a pair of nodes is defined as $\frac{|BW_{\mathrm{FWD}} - BW_{\mathrm{REV}}|}{\max\{BW_{\mathrm{FWD}}, BW_{\mathrm{REV}}\}}$, where BW_{FWD} means the forward bandwidth value of the node pair, and BW_{REV} means the reverse bandwidth value. So $\alpha = 0$ indicates $BW_{\mathrm{FWD}} = BW_{\mathrm{REV}}$ (i.e., complete symmetry). Figure 7.1 shows the cumulative distribution function of the α values for the two bandwidth data sets used in Sections 4.4, 5.3, and 6.3. HP-PlanetLab shows high symmetry of bandwidth in that around 80% of node pairs have $\alpha \leq 0.5$. On the other hand, only 50% of node pairs have $\alpha \leq 0.5$ for UMD-PlanetLab. We can find the reason for this difference from different data collection times of the two data sets. HP-PlanetLab was collected four years earlier than UMD-PlanetLab. So, the network conditions of PlanetLab such as link capacity and participating hosts had surely changed before UMD-PlanetLab was collected.

For some applications, it may be important to utilize bandwidth values in both directions of a network path. Our metric-based approach for bandwidth prediction will be useful if such an application runs in a network condition, where bandwidth is quite symmetric like in HP-PlanetLab. However, we need to support prediction of asymmetric bandwidth in case applications want to utilize both forward and reverse bandwidth values in asymmetric network environments like UMD-PlanetLab. So, we leave it as a future work to extend our approach and predict asymmetric bandwidth information.

7.3 Latency Prediction

Because of the inherent nature of network topology, an edge-weighted tree is also a good data structure to embed network latency information [58]. So we should be able to apply the approach for bandwidth prediction to latency prediction. Then no transform function is required, and round-trip time can be directly used as a distance in a tree metric space. We can compare the accuracy and sampling measurement costs with network coordinate approaches such as Vivaldi [26] and GNP [53]. As existing coordinate systems already have high accuracy, we do not expect our approach to seriously outperform those systems in terms of prediction accuracy. Nonetheless, it is meaningful to apply our approach to latency prediction

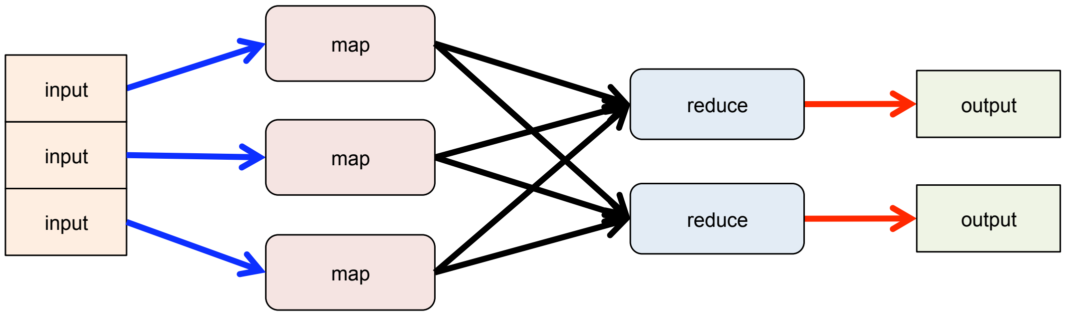


Figure 7.2: The overview of MapReduce workflow execution

because we support decentralized search algorithms for centroids and clusters unlike network coordinate systems.

7.4 Wide-Area MapReduce

In addition to designing the prediction and node search algorithms as discussed in the previous chapters, it is also important to study how the algorithms can be actually used in practice. We are planning to develop a new type of MapReduce system that exploits wide-area computing resources. In this section, we provide the preliminary design of a decentralized job scheduling algorithm that optimizes data locality among wide-area hosts by utilizing the algorithms described in the previous chapters. The following sections describe motivation for this work, details of the design of the job scheduling algorithm, and open questions for this work.

7.4.1 Introduction

MapReduce [27] is a programming model that enables automatic parallelization and distribution of large-scale data-intensive computations. It allows users to execute a workflow that contains data-intensive *tasks*. Figure 7.2 shows an overview of MapReduce workflow execution. Computation in MapReduce is divided into two major phases, which execute *map tasks* and *reduce tasks*, respectively. Input data are first divided into several splits to be assigned to map tasks. A map task "maps" each input split to a list of key-value pairs. The intermediate outputs are then shuffled, so key-value pairs with the same key are grouped together and passed to a single reduce task. Finally, a reduce task "reduces" each group of key-value pairs into a final output value.

Many different data-intensive problems can be solved efficiently with this two-phase computation. So MapReduce is successfully being used as a computation framework in institutions of science and other data-intensive applications. MapReduce solves various data-intensive problems, for example, for data analysis related to astronomical image analysis [70], bioinformatics [61], and high energy physics data analysis [29], also for industrial applications such as web indexing, data mining, and spam detection.

However, current implementations of MapReduce, including Hadoop [2], are targeted to the execution of jobs in a local-area network within one data center. This is the biggest motivation for developing a wide-area MapReduce system. By exploiting wide-area computing resources, we can have two advantages over current local-area systems.

- Scalability: A huge amount of computing resources all over the world will be able to participate in a single instance.

- Cost-efficiency: We can create a wide-area cluster with volunteer resources that are available for free or at very low cost. Federating small local-area clusters without maintaining a large data center is also possible.

Developing a wide-area MapReduce algorithms implementation is not an easy problem in that directly applying the architecture for the local-area clusters [27, 35] will cause two serious problems. The problem is the overhead of a central server for resource allocation. Maintaining a huge amount of information for wide-area resources in a single server will cause problems for scalability and reliability. The second problem is long data transmission time through low-bandwidth network connections. Data locality must be considered when scheduling tasks among wide-area resources. Since resources are widely dispersed, we cannot use the locality-aware scheme of the traditional MapReduce system such that a task is assigned to a node in a local rack where the input data exists. There have been several studies [67, 52, 22] about developing a wide-area MapReduce system with a similar motivation to ours. However, none of them could effectively overcome the challenges above.

In this dissertation, we provide our preliminary design of a job scheduling algorithm that resolves the two challenges of decentralized resource allocation and data locality optimization. Our approach is to utilize the decentralized algorithms for bandwidth prediction and node search described in the previous chapters. The design of a job scheduling algorithm satisfies the following requirements:

- Exploiting Wide-Area Resources: The system can be constructed with a large number of geographically dispersed computing resources.
- Decentralized Resource Allocation: There must exist no single centralized server to maintain information about the entire set of available resources to allocate resources for jobs.
- Locality-Awareness: A task should be assigned to a node that has a high-bandwidth interconnection with data storage node for the input data for the task.
- Load balancing: The number of executed tasks should be evenly distributed across all the nodes in the system.
- Robustness: Node failures should not impact the overall functionality of the services deployed on the system.

7.4.2 Preliminary Design

This section describes a job scheduling algorithm that can be used in a wide-area MapReduce system. We first present an overview of the scheduling procedure, then describe two different approaches for optimizing data locality among data-intensive tasks.

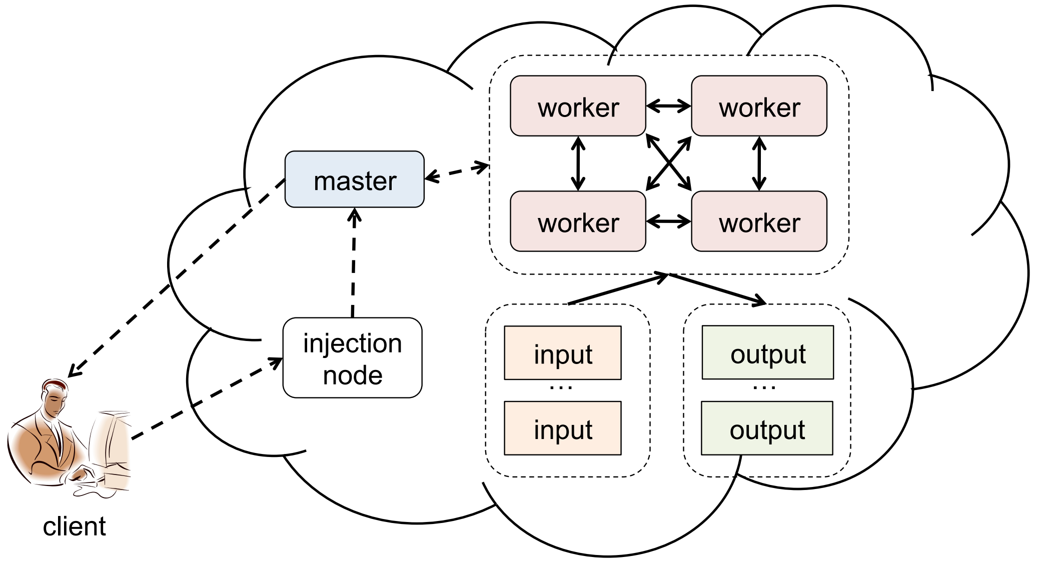


Figure 7.3: The overview of job scheduling in a wide-area MapReduce

7.4.2.1 Overview

All the hosts that will be utilized in a MapReduce computation first construct a bandwidth prediction framework as described in Chapter 4. Then our scheduling algorithm can optimize data locality in terms of bandwidth by using the node search algorithms discussed in Chapters 5 and 6.

Figure 7.3 shows the overall procedure of job execution in a wide-area MapReduce system. First, a client submits a MapReduce job to any node, called an injection node in the system. Each job should specify i) the locations of the input data, ii) the *map function* that each map task executes, iii) the *partition function* that determines which intermediate outputs are grouped together and assigned to the same reduce task, and iv) the *reduce function* that each reduce task executes. Second, the job is routed to a random node which is designated as a *master node* of the job. Third, the master node schedules map and reduce tasks as specified for the job on several *worker nodes*. Fourth, each worker node executes an assigned task by retrieving input data from other nodes. Finally, the output data are stored in several nodes, and the master node maintains the locations of output data until the client retrieves it.

While all the submitted jobs are managed by a single master node in local-area MapReduce implementations [2], our design can have multiple master nodes to handle different jobs. So the workload for managing job execution in large-scale networks can be distributed over multiple master nodes. With the following strategy, we can choose a node uniformly at random and give every node an equal chance to be a master node for each job. Each node first aggregates the number of nodes that exist in the skip anchor tree (described in Chapter 4) in each direction of the anchor parent neighbor P, the anchor child neighbor C, the level-0 left neighbor $L[0]$, and the level-0 right neighbor $R[0]$. An injection node starts a random walk process, and each node chooses a direction with the probability of the ratio of *(the number of nodes in the direction)* to *(the total number of nodes in the system except in the direction of the previously visited node during the random walk process)*. Like the dynamic information aggregation mechanisms described in the previous chapters, each node utilizes high-level links in a skip list, and takes only $O(\log^2 n)$ hops with high probability to propagate the information across all nodes in the system.

A master node watches the status of worker nodes to ensure reliable computations. If a failure happens to a worker node that has been executing a task, the master node assigns the failed task to another worker node. We should also consider the failure of a worker node after it finishes executing a reduce task and before a client retrieves output data from the failed worker node. Then we can possibly lose the output data and need to execute the entire job all over again. To avoid the possible loss of output data, we replicate the output data of each reduce task to another random node, and the master node keeps track of the location of the replica. We can choose a uniformly random replica node in the same way an injection node chooses a master node.

In case of the failure of the master node, we designate a secondary master node that maintains a copy of the information in the master node. So the secondary master node takes over the control of job execution when it detects the failure of the primary master node. A client should know which node is designated as the secondary master nodes, so that it can retrieve output data if the master node fails.

7.4.2.2 Task Scheduling with Data Locality

We now describe how each master node chooses worker nodes to schedule tasks on. The key issue for task scheduling in a wide-area MapReduce system is data locality optimization among data-intensive tasks. It is also important to choose worker nodes in a decentralized fashion and distribute the workloads of task execution over computing resources. We provide two task scheduling approaches that resolve the issues of data locality optimization and decentralization by utilizing the node search algorithms described in the previous chapters.

As shown in Figure 7.4, the first approach uses the centroid search algorithm

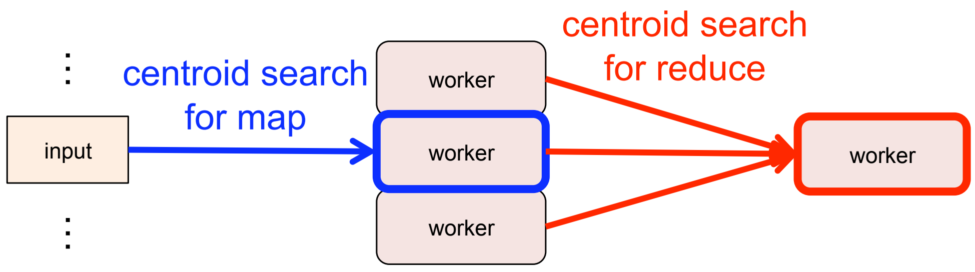


Figure 7.4: Task scheduling with centroid search

designed in Chapter 5. A master node schedules each map task to the node that contains the input split, so that we can optimize data locality by avoiding unnecessary data transfers. For scheduling a reduce task, a master node first finds a centroid of worker nodes that have executed the corresponding map tasks and contain the intermediate output data of the map tasks. The centroid node is designated as a worker node to execute the reduce task. Then the worker node that is assigned the reduce task will be able to retrieve the intermediate output data from other worker nodes at a high rate through high-bandwidth connections.

Although we can optimize data locality using the centroid search algorithm, there exist two possibilities of load imbalance. First, if a small number of nodes are connected to most other nodes in a network via high-bandwidth links, the small set of nodes can be always chosen as worker nodes. To reduce the load imbalance problem, we limit the number of currently running tasks in each node to stay under a user-defined threshold value. We can exclude overloaded nodes from running additional tasks by slightly modifying the centroid search algorithm. Nodes overloaded by

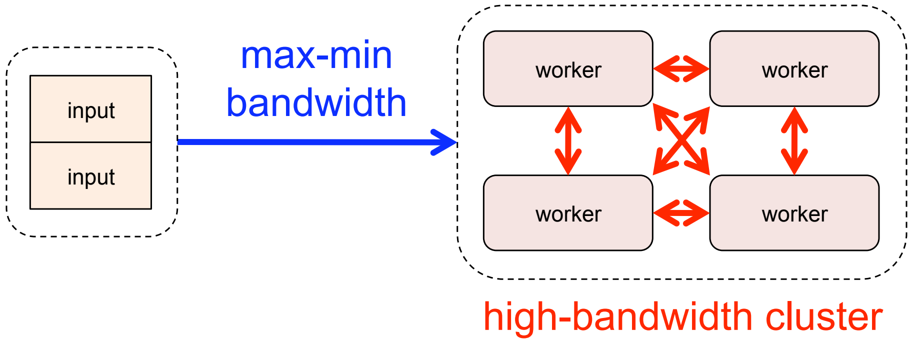


Figure 7.5: Task scheduling with cluster search

too many running tasks are not allowed to participate in the dynamic aggregation mechanism for centroid search described in Section 5.2.1, so that information about the overloaded nodes is not propagated to other nodes. Then the centroid search algorithm will be able to find nodes with fewer running tasks than the threshold value. The second possibility of load imbalance comes from the cases such that many map tasks have input splits contained in the same node, or many reduce tasks are associated with the same set of map tasks. Then every time a task is scheduled, the centroid search algorithm will find the same centroid, and the single centroid will suffer from running many tasks. To resolve this problem, a master node finds multiple centroid nodes and assigns a task to the best centroid node that is not occupied by another task in the same job. We can have the centroid search algorithm return multiple centroid nodes in the same way that the algorithm returns k candidates for the probing heuristic described in Section 5.2.3.

Figure 7.5 shows the second approach for task scheduling, where a master node finds a set of worker nodes by running the cluster search algorithm designed in

Chapter 6. The cluster size constraint k is set to be the number of map tasks. The bandwidth constraint b starts at the maximum value in the bandwidth constraint set and is adjusted down until any cluster is found. If no cluster exists with k nodes, we keep dividing k by two until finding a cluster. Then each worker node in the found cluster will be able to retrieve intermediate output data quickly from other worker nodes at a high rate through high-bandwidth connections. So we can achieve higher data locality among reduce tasks than a naive scheduling method that chooses random worker nodes without considering network bandwidth. We can also increase data locality for map tasks with the following strategy. The master node finds clusters multiple times and picks one that maximizes the minimum of bandwidth (*max-min*) between nodes containing input data and nodes in the cluster. In this way, worker nodes to execute map tasks can quickly retrieve input data splits. Once a cluster of worker nodes is determined, the master node receives heartbeats from the worker nodes in the cluster, and schedules tasks to worker nodes in a FIFO fashion.

Like the centroid search approach, this cluster search approach potentially has a load imbalance problem. Let's say that there exists a small set of nodes that are interconnected via higher bandwidth than the bandwidth between any other node pairs in a network. Then the nodes with high-bandwidth interconnections will be frequently chosen as worker nodes. To eliminate the load imbalance problem, we limit the number of different clusters that each node participates in at the same time, below a user-defined threshold value. Similarly to the solution to the load imbalance problem in the centroid search approach, we can modify the cluster search algorithm.

If a node is participating in too many clusters, the information for the node is not propagated in the overlay network, so that the node cannot be used in the clustering spaces of other nodes and the node does not appear in cluster routing tables. Then the cluster search algorithm will be able to find nodes that are not overloaded with many running tasks.

There exists a trade-off between these two approaches. The cluster approach has less overhead for node search than the centroid approach. The cluster approach executes the cluster search algorithm only a few times to determine a set of worker nodes before executing tasks, while the centroid approach requires searching for a centroid each time a task is scheduled on a worker node. However, the centroid search can possibly optimize data locality better than the cluster approach. The clustering approach relies on the high-bandwidth interconnections among worker nodes while the centroid approach finds a good worker node for every task.

7.4.3 Other MapReduce Job Scheduling Approaches

Hadoop [2] is the most popular implementation of MapReduce, and is being used as a computation framework for a public cloud computing [16] service [1]. Hadoop employs the speculative execution technique, where multiple copies of a MapReduce task are executed at different nodes in case a task is scheduled on a slow node called a straggler. LATE scheduler [75] improves the speculative execution by executing multiple copies of tasks with the longest approximate time to end. Delay scheduling [74] considers both fairness and data locality. For a given idle node and a task queue, if the task at the queue head does not have input data in the idle node, the scheduling of the task is delayed. Dryad [35] is an infrastructure similar to MapReduce and is used to run data-parallel programs in a computer cluster. Quincy [36] schedules Dryad tasks by periodically solving a min-cost flow problem. Like delay scheduling [74], Quincy also considers both fairness and data locality. Moon [47] is a combination of MapReduce and volunteer computing, and executes MapReduce tasks on opportunistic environments. Moon employs a hybrid architecture of reliable and non-reliable nodes, so that important data can be stored in reliable nodes. MapReduce, Dryad, and Moon are all targeted to run in a high-bandwidth local-area network. Accordingly, all the task scheduling techniques used in those systems optimize data locality with a simple strategy: if possible, a task is scheduled on a node that contains the input data for the task. On the other hand, we consider network bandwidth to optimize data locality to schedule MapReduce tasks on nodes dispersed in a wide-area network.

There are several approaches [67, 52, 22] to execute MapReduce tasks in a wide-area network. However, unlike our approach, they do not consider network bandwidth and just employ the simple scheduling approach that is used in local-area MapReduce systems.

7.4.4 Open Questions

Although the job scheduling algorithms described in the previous section can be an important building block, there is still much work to do to design a wide-area

MapReduce system. First, the job scheduler should support fairness among users so that at a given moment computing resources should be distributed well among users. Second, we should employ a wide-area distributed file system to store input and output data. The current MapReduce implementations use distributed file systems such as HDFS [5] and GFS [31]. Since those file systems are also targeted to run in a high-bandwidth network, there will likely be data locality problems when we directly use those systems in a wide-area network. Last, we should work on empirical studies about job scheduling. We can first conduct simulations to evaluate the performance of the job scheduling algorithm. MapReduce workloads [6] such as word-count and sort will be used along with network bandwidth data sets. Then we can implement a real system and run it over wide-area Internet hosts.

Chapter 8

Conclusion

In this chapter, we conclude this dissertation by reviewing the thesis and our contributions.

My thesis is that *a decentralized approach can be employed to predict end-to-end network bandwidth and search for nodes under network bandwidth constraints in an accurate and scalable way.* We made contributions in resolving three research problems that are important in supporting data-intensive widely distributed applications.

First, we designed a highly scalable, decentralized, and accurate algorithm for end-to-end network bandwidth prediction. The algorithm is decentralized, so that data structures are distributed, and no landmark node is required for sampling measurements. We proved that the algorithm can accurately predict bandwidth information with only $O(\log^2 n)$ sampling measurements for each node. We also have confirmed high accuracy and scalability through extensive simulations.

Second, we designed a decentralized algorithm to find centroids. A centroid is a node connected via high-bandwidth connections to a given set of nodes. By running on top of the bandwidth prediction framework, the algorithm can avoid any measurement delays. Through theoretical analyses and extensive simulations, we showed that the algorithm finds a centroid accurately within only $O(\log n)$ network hops.

Last, we investigated the cluster search problem. We found a polynomial time algorithm to find clusters in a tree metric space. Then we designed a decentralized algorithm that can find clusters accurately with $O(\log^2 n)$ network hops, and provided proofs of correctness. Simulation results are also provided to show the high accuracy and scalability of our approach.

Bibliography

[1] Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`.

[2] Apache Hadoop. `http://hadoop.apache.org/`.

[3] Azureus bittorrent. `http://azureus.sourceforge.net/`.

[4] Dagman. `http://www.cs.wisc.edu/condor/dagman/`.

[5] Hadoop Distributed File System. `http://hadoop.apache.org/hdfs/`.

[6] MapReduce benchmarks. `http://web.ics.purdue.edu/~fahmad/benchmarks.htm`.

[7] S3: Scalable sensing service. `http://networking.hpl.hp.com/s-cube/`.

[8] Second life. `http://secondlife.com/`.

[9] A simulator for Vivaldi. `http://www.eecs.harvard.edu/~syrah/nc/`.

[10] Teragrid. `https://www.xsede.org/tg-archives`.

[11] Ittai Abraham, Mahesh Balakrishnan, Fabian Kuhn, Dahlia Malkhi, Venugopalan Ramasubramanian, and Kunal Talwar. Reconstructing approximate tree metrics. In *Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM Press, August 2007.

[12] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Fining points with minimum diameter and related problems. In *Proceedings of the 5th Annual Symposium on Computational Geometry (SoCG)*. ACM Press, 1989.

[13] David P. Anderson. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Conference on Grid Computing (GRID)*, 2004.

[14] David P. Anderson, Carl Christensen, and Bruce Allen. Grid resource management - designing a runtime system for volunteer computing. In *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing (SC)*, 2006.

[15] Cecilia R. Aragon and Raimund Seidel. Randomized search trees. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 540–545. IEEE Computer Society Press, November 1989.

[16] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[17] James Aspnes and Gauri Shah. Skip graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*. SIAM, January 2003.

[18] Olivier Beaumont, Nicolas Bonichon, Philippe Duchon, and Hubert Larchevêque. Distributed approximation algorithm for resource clustering. In *Structural Information and Communication Complexity, 15th International Colloquium (SIROCCO)*. Springer, June 2008.

[19] Olivier Beaumont, Lionel Eyraud-Dubois, and Young J. Won. Using the last-mile model as a distributed scheme for available bandwidth prediction. In *Proceedings of the 17th international conference on Parallel processing (Euro-Par) - Volume Part I*, pages 103–116, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of Scientific Workflows. In *Proceedings of 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS)*, November 2008.

[21] Peter Buneman. A note on the metric properties of trees. *Journal of Combinatorial Theory, Ser. B*, 17:48–50, 1974.

[22] Michael Cardosa, Chenyu Wang, Anshuman Nangia, Abhishek Chandra, and Jon Weissman. Exploring mapreduce efficiency with highly-distributed data. In *Proceedings of the second international workshop on MapReduce and its applications (MapReduce)*, pages 27–34. ACM Press, 2011.

[23] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003.

[24] Bram Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003.

[25] Manuel Costa, Miguel Castro, Antony I. T. Rowstron, and Peter B. Key. Pic: Practical internet coordinates for distance estimation. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society Press, 2004.

[26] Frank Dabek, Russ Cox, M. Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. ACM Press, August 2004.

[27] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, December 2004.

[28] Ewa Deelman, Gurmeet Singh, Mei hui Su, James Blythe, A Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13, 2005.

[29] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. Mapreduce for data intensive scientific analyses. In *Proceedings of the Fourth IEEE International Conference on eScience*, pages 277–284, Washington, DC, USA, 2008. IEEE Computer Society Press.

[30] David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. In *Proceedings of the 4th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA)*. ACM Press, 1993.

[31] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems (SOSP)*. ACM Press, 2003.

[32] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of the 4th USENIX Symposiums on Internet Technologies and Systems (USITS)*. USENIX, March 2003.

[33] Tony Hey, Stewart Tansley, and Kristin Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[34] Ningning Hu, Li Erran Li, Zhuoqing Morley Mao, Peter Steenkiste, and Jia Wang. A measurement study of internet bottlenecks. In *Proceedings of the 24th Annual IEEE Conference on Computer Communications (INFOCOM)*, pages 1689–1700. IEEE Computer Society Press, 2005.

[35] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd European conference on Computer systems (EuroSys)*. ACM Press, March 2007.

[36] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the 22nd ACM Symposium on Operating Systems (SOSP)*. ACM Press, 2009.

[37] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication.* ACM Press, 2002.

[38] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim simulator. `http://peersim.sf.net/`.

[39] Jik-Soo Kim, Peter Keleher, Michael Marsh, Bobby Bhattacharjee, and Alan Sussman. Using content-addressable networks for load balancing in desktop grids. In *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing (HPDC).* ACM Press, June 2007.

[40] Jik-Soo Kim, Beomseok Nam, Peter Keleher, Michael Marsh, Bobby Bhattacharjee, and Alan Sussman. Resource discovery techniques in distributed desktop grid environments. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing (GRID).* IEEE Computer Society Press, September 2006.

[41] Jik-Soo Kim, Beomseok Nam, Michael Marsh, Peter Keleher, Bobby Bhattacharjee, and Alan Sussman. Integrating categorical resource types into a P2P desktop grid system. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (GRID).* IEEE Computer Society Press, September 2008.

[42] Bjorn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceedings of the 23rd Annual IEEE Conference on Computer Communications (INFOCOM).* IEEE Computer Society Press, 2004.

[43] Christopher Kommareddy, Narendar Shankar, and Bobby Bhattacharjee. Finding close friends on the internet. In *Proceedings of 9th International Conference on Network Protocols (ICNP)*, pages 301–. IEEE Computer Society Press, November 2001.

[44] Jaehwan Lee, Pete Keleher, and Alan Sussman. Decentralized resource management for multi-core desktop grids. In *Proceedings of the 24th International Parallel & Distributed Processing Symposium (IPDPS).* IEEE Computer Society Press, April 2010.

[45] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca. Measuring bandwidth between planetlab nodes. In *Proceedings of the 6th Passive and Active Measurement Workshop (PAM).* Springer, 2005.

[46] Yongjun Liao, Pierre Geurts, and Guy Leduc. Network distance prediction based on decentralized matrix factorization. In *Proceedings of the 9th International IFIP TC 6 Networking Conference (NETWORKING)*, volume 6091 of *Lecture Notes in Computer Science*, pages 15–26. Springer, May 2010.

[47] Heshan Lin, Xiaosong Ma, Jeremy S. Archuleta, Wu chun Feng, Mark K. Gardner, and Zhe Zhang. Moon: Mapreduce on opportunistic environments. In *Proceedings of the 19th IEEE International Symposium on High Performance Distributed Computing (HPDC)*. ACM Press, June 2010.

[48] Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society Press, 1988.

[49] Chuang Liu and Ian T. Foster. A scalable cluster algorithm for internet resources. In *Proceedings of the 21st International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, March 2007.

[50] Cristian Lumezanu, Dave Levin, Bo Han, Neil Spring, and Bobby Bhattacharjee. Don't love thy nearest neighbor. In *Proceedings of the 9th International Workshop on Peer-to-Peer Systems (IPTPS)*. USENIX, April 2010.

[51] Yun Mao, L.K. Saul, and J.M. Smith. Ides: An internet distance estimation service for large networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 24(12):2273 –2284, December 2006.

[52] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. A peer-to-peer framework for supporting mapreduce applications in dynamic cloud environments. In Nick Antonopoulos and Lee Gillam, editors, *Cloud Computing: Principles, Systems and Applications*, chapter 7, pages 113–125. Springer, 2010. ISBN 978-1-84996-240-7.

[53] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings of the 21st Annual IEEE Conference on Computer Communications (INFOCOM)*. IEEE Computer Society Press, June 2002.

[54] David L. Oppenheimer, Jeannie R. Albrecht, David A. Patterson, and Amin Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC)*. ACM Press, 2005.

[55] William Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Proceedings of the Workshop on Algorithms and Data Structures (WADS)*. Springer-Verlag, August 1989.

[56] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[57] Venugopalan Ramasubramanian, Dahlia Malkhi, Fabian Kuhn, Ittai Abraham, Mahesh Balakrishnan, Archit Gupta, and Aditya Akella. A unified network coordinate system for bandwidth and latency. Technical Report MSR-TR-2008-124, Microsoft Research, 2008.

[58] Venugopalan Ramasubramanian, Dahlia Malkhi, Fabian Kuhn, Mahesh Balakrishnan, Archit Gupta, and Aditya Akella. On the treeness of internet latency and bandwidth. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance)*. ACM Press, June 2009.

[59] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of the 21st Annual IEEE Conference on Computer Communications (INFOCOM)*. IEEE Computer Society Press, 2002.

[60] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Proceedings of the 4th Passive and Active Measurement Workshop (PAM)*. Springer, April 2003.

[61] Michael C. Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[62] Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464–497, 1996.

[63] Haiying Shen and Kai Hwang. Locality-preserving clustering and discovery of wide-area grid resources. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society Press, June 2009.

[64] Sukhyun Song, Pete Keleher, Bobby Bhattacharjee, and Alan Sussman. Brief announcement: Decentralized network bandwidth prediction. In *Proceedings of the 24th International Symposium on Distributed Computing (DISC)*. Springer, September 2010.

[65] Sukhyun Song, Pete Keleher, Bobby Bhattacharjee, and Alan Sussman. Decentralized, accurate, and low-cost network bandwidth prediction. In *Proceedings of the 30th Annual IEEE Conference on Computer Communications (INFOCOM)*. IEEE Computer Society Press, April 2011.

[66] Sukhyun Song, Pete Keleher, and Alan Sussman. Searching for bandwidth-constrained clusters. In *Proceedings of the 31st IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society Press, June 2011.

[67] Bing Tang, Mircea Moca, Stephane Chevalier, Haiwu He, and Gilles Fedak. Towards mapreduce for desktop grid computing. In *Proceedings of International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. IEEE Computer Society Press, November 2010.

[68] Matteo Varvello, C. Diout, and Ernst W. Biersack. P2p second life: Experimental validation using kad. In *Proceedings of the 28th Annual IEEE Conference on Computer Communications (INFOCOM)*. IEEE Computer Society Press, 2009.

[69] Marcel Waldvogel and Roberto Rinaldi. Efficient topology-aware overlay network. *Computer Communication Review*, 33(1):101–106, 2003.

[70] Keith Wiley, Andrew Connolly, Jeffrey P. Gardner, Simon Krughof, Magdalena Balazinska, Bill Howe, YongChul Kwon, and Yingyi Bu. Astronomy in the cloud: Using mapreduce for image coaddition. *CoRR*, abs/1010.1015, 2010.

[71] Praveen Yalagandula, Sung-Ju Lee, Puneet Sharma, and Sujata Banerjee. Leveraging correlations between capacity and available bandwidth to scale network monitoring. In *Proceedings of the Global Communications Conference (GLOBECOM)*. IEEE Communications Society Press, December 2010.

[72] Praveen Yalagandula, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Sung-Ju Lee. S3: a scalable sensing service for monitoring large networked systems. In *Proceedings of the 2006 SIGCOMM workshop on Internet network management (INM)*, pages 71–76, New York, NY, USA, 2006. ACM Press.

[73] Chang you Xing, Ming Chen, and Li Yang. Predicting available bandwidth of internet path with ultra metric space-based approaches. In *Proceedings of the Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE Communications Society Press, November 2009.

[74] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems (EuroSys)*. ACM Press, April 2010.

[75] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy H. Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, December 2008.