Searching for Bandwidth-Constrained Clusters

Sukhyun Song, Pete Keleher, and Alan Sussman
UMIACS and Department of Computer Science, University of Maryland
{shsong, keleher, als}@cs.umd.edu

***Abstract*—Data-intensive distributed applications can increase the performance by running on a cluster of hosts with high-bandwidth interconnections. However, there exists no effective method to find such a bandwidth-constrained cluster in a decentralized fashion. Our work is inspired by prior work that treats Internet bandwidth as an approximate *tree metric space*. This paper presents a decentralized, accurate, and efficient system that finds a cluster with constraints of the number of nodes and minimum interconnect bandwidth. We describe a centralized polynomial time algorithm running in a tree metric space, along with the proof of correctness. We then provide a decentralized version. Simulation experiments with two real-world datasets confirm that our clustering approach achieves high accuracy and scalability. We also discuss what the tradeoff of decentralization is and how the treeness of dataset affects the clustering accuracy.**

I. Introduction

Distributed applications can benefit from a decentralized algorithm to find a cluster of hosts that have high-bandwidth connections to each other. For example, a peer-to-peer (P2P) desktop grid [11], [10], [12], [14] can reduce a job execution time by scheduling a data-intensive scientific set of jobs, such as CyberShake workflow [4], on a set of nodes with high-bandwidth interconnections. A content delivery network [17] can quickly distribute data by finding several clusters of high-bandwidth nodes and making a representative of each cluster responsible for distributing the data in the cluster. Unfortunately, however, there exists no effective method to find a bandwidth-constrained cluster in a decentralized fashion. Most of the existing studies only focus on latency-constrained clustering [16], [24], [3]. They even need a centralized sturucture [16] or a fixed set of landmark nodes that every node has to perform measurements with [24].

There are two important reasons why decentralized bandwidth-constrained clustering is not successfully explored yet. First, a clustering problem is difficult to solve, in that it is equivalent to k-Clique problem in a general undirected graph, which is NP-complete. Second, we were lack of effective framework for bandwidth prediction that is indispensable to minimize extra costs of measurements. Fortunately, it is now possible to overcome those difficulties as a consequence of recent research efforts. Ramasubramanian et. al [21] claim that a metric space for Internet bandwidth can be modeled by an approximate tree metric space.If we limit the clustring problem in a tree metric space based on this finding, it is expected that we can develop a polynomial time algorithm. Also, our prior work [25], [26] designed a decentralized framework for bandwidth prediction and successfully embedded bandwidth measurements into a tree metric space with a high accuracy. We expect that the bandwidth prediction framework will enable a clustering algorithm to run without any delay of measurements.

With these motivations, we will design a decentralized algorithm to solve the following specific problem. Given a set of nodes V, a bandwidth function BW on V, and query constraints $k \geq 2$ and b, find a set X such that $X \subseteq V$, $|X| = k$, and $BW(u, v) \geq b \ \forall u, v \in X$.

We feel that five requirements must be considered in decentralized bandwidth-constrained clustering.

- Decentralized Cluster Formation: Nodes must be grouped into clusters without any help of a centralized server.
- Decentralized Query Processing: A query should be able to be submitted to any node in the system, and each node should make a decision with local information.
- High Accuracy: A query result should satisfy constraints.
- Scalable Search: A cost should increase in a scalable way with an increasing number of nodes in the system.
- Dynamic Clustering: Members of each cluster should adatatively change as network condition changes.

Our contributions are fourfold in specific. First, we show that the clustering problem is P in a tree metric space by developing a centralized polynomial time algorithm and proving its correctness. Second, we provide a decentralized clustering algorithm along with the proof of correctness. The key idea is to let each node maintain a simple routing table, so that a query can route towards the direction where the wanted cluster exists. The third contribution is a new dataset that we collected by measuring available bandwidth between PlanetLab nodes. Finally, we present extensive simulation results validating the high accuracy and scalability and showing the tradeoff of decentralization and the effect of treeness of datasets.

The rest of the paper is organized as follows. We first discuss the underlying intuition behind this work in Sec. II. Sec. III describes the details of the algorithm design, and provides a proof of correctness. Sec. IV evaluates our approach experimentally. Finally, Sec V discusses related work, and we conclude and discuss future work in Sec. VI.

II. Terminology and Background

This section defines our terms and provides backgrounds about how to represent bandwidth in a metric space and why bandwidth is approximately a tree metric. We also describe the design of a decentralized bandwidth prediction framework that our clustering algorithm runs on.

A. Definitions

- An *edge-weighted tree* is a connected graph without cycles, and with non-negative edge weights.
- The *distance* between two nodes u and v on an edge-weighted tree T, denoted $d_T(u, v)$, is defined by the sum of weights of edges on the path from u to v.
- 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.
- 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 $d(w, y) + d(x, z) = d(w, z) + d(x, y)$.
- A metric space that satisfies 4PC is called a *tree metric space*.

B. Bandwidth as a Metric

Higher values are considered better for bandwidth while closer is generally more desirable for distance in a metric space. So, we use the *rational transform function* $d(u, v) = \frac{C}{BW(u,v)}$ 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 positive constant. Representing bandwidth as a metric implies four properties:

1) $d(u, v) \geq 0$ (non-negativity)
2) $d(u, v) = 0$ if and only if $u = v$
3) $d(u, v) = d(v, u)$ (symmetry)
4) $d(u, w) \leq d(u, v) + d(v, w)$ (triangle inequality)

The first property is satisfied because C is a positive constant. By setting $BW(u, u) = \infty$, we can also satisfy the second property. Symmetry can be justified by one measurement study [15] that estimates an asymmetry factor $\alpha \in [0, 1]$ such that $\alpha = 0$ when $BW(u, v) = BW(v, u)$ (i.e., complete symmetry). The study shows 90% of bandwidth capacity data in PlanetLab version 3 have α less than 0.5. Nonetheless, we satisfy the third property by setting both $BW(u, v)$ and $BW(v, u)$ to the average bandwidth of forward and reverse directions. Even though there is no effective method found that directly addresses the last assumption, our prior work [25], [26] succeeded in accurately embedding bandwidth into a metric space with several heuristics.

C. Treeness of Bandwidth

There are three evidences to verify that the Internet is close to a tree metric space in terms of bandwidth. First, Ramasubramanian et. al [21] verify that a bandwidth dataset produces a lot of small ε values. ε is introduced by Abraham et. al [1] to quantify how much a set of four nodes satifies 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 at the access link of either end. And it is proved that a metric space for this model is a perfect tree metric space. [20] Last, the attempts of embedding bandwidth into a tree metric space resulted in a high accuracy. [21], [26] Based on Theorem 2.1, proved by Buneman [5], they constructed an edge-weighted tree to embed bandwidth measurements into. The result graph showed low relative errors of embedded bandwidth value compared to the real data.

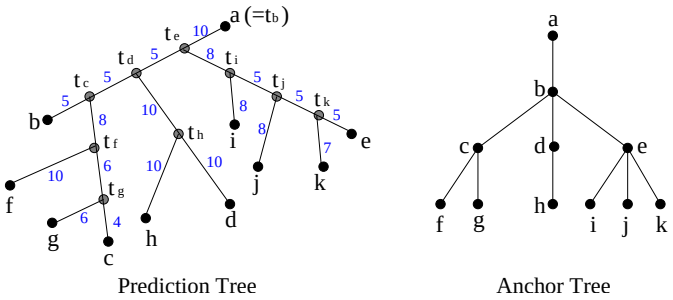


Fig. 1. Example Structures of Decentralized Bandwidth Prediction System

Theorem 2.1: A metric space (V, d) satisfies 4PC if and only if there exists an edge-weighted tree that induces (V, d).

D. Decentralized Bandwidth Prediction System

Our clustering algorithm will be designed to run on top of a decentralized bandwidth prediction framework which is developed in our prior work [25], [26]. It is briefly described as belows how the framework is designed.

An edge-weighted tree embedding bandwidth information is called a *prediction tree* (Fig. 1). The number on each edge represents the weight of the edge. A leaf node in a prediction tree has degree one and represents each participating host in the system. An inner node with degree two or more is created when a new leaf node is added to a prediction tree. The rational transform function $d(u, v) = \frac{C}{BW(u,v)}$ is used to represent bandwidth as a metric, and $BW_T(u, v) = \frac{C}{d_T(u,v)}$ for bandwidth prediction. For example, in Fig. 1 if $C = 100$, the predicted bandwidth value $BW_T(b, c)$ is 77 because $d_T(b, c) = 23$. A prediction tree starts with the first added node as a singleton, and the second node is added along with an edge that connects the two nodes and is weighted by their distance. The tree grows by iteratively adding nodes as follows. To add a new node x to a prediction tree, the algorithm chooses a node z called the *base node*, which can be any leaf node, and selects another node y called the *end node* that maximizes *Gromov product* $(x|y)_z$. The Gromov product of x and y at z, denoted $(x|y)_z$, is defined as $(x|y)_z = \frac{1}{2}(d(z, x) + d(z, y) - d(x, y))$. x's *inner node* t_x is created and located on the path $z \sim y$ where $d_T(z, t_x) = (x|y)_z$. The algorithm then adds x to the prediction tree by creating an edge (t_x, x) with weight $(y|z)_x$. That is, x's position in the graph is determined by the maximized Gromov product.

The prediction framework constructs an overlay network of hosts following a structure of a rooted unweighted tree called *anchor tree*. The first added node in the system becomes the root node of the anchor tree, and the second node becomes the child of the root node. When x is added to the prediction tree, x is also added to the anchor tree by becoming a child of x's *anchor node*. x's anchor node is defined as a node that was

Algorithm 1: $\mathbf{X}$ = **FindCluster**$(\mathbf{V}, \mathbf{d}, \mathbf{k}, \mathbf{l})$: A centralized algorithm to find in a tree metric space (V, d) a set X such that $X \subseteq V$, $|X| = k$, and $diam(X) \leq l$

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

previously added to the prediction tree along with the edge that x's inner node t_x is located on. For example, assuming that nodes in Fig. 1 are added to the system in an alphabetical order, when adding h to a prediction tree, h's inner node t_h is located on edge (t_d, d). Node d is defined as h's anchor node because edge (t_d, d) is created when d was added. The anchor tree is also used to quickly find a global maximizer of Gromov product, so that the framework can be built without performing a full *n-to-n* measurements.

A *distance label* is assigned to each node, so that a prediction tree is constructed in a distributed fashion. Node x's distance label contains all anchor nodes on the path from the root node to x in the anchor tree. The label also contains the corresponding distance values between anchor nodes and inner nodes. For example, node d in Fig. 1 has $(a \xrightarrow[25]{0} b \xrightarrow[20]{10} d)$ as its distance label, with $d_T(a, t_b) = 0$, $d_T(t_b, b) = 25$, $d_T(b, t_d) = 10$, and $d_T(t_d, d) = 20$. Since a distance label is equivalent to a partial prediction tree, the distance between two nodes can be estimated with a simple computation. In other words, a distance label plays a similar role to the network coordinates in Vivaldi.

III. Design

This section describes details of our approach. We first develop a centralized clustering algorithm, and then discuss how to decentralize it with several techniques.

Note that we consider bandwidth as a metric using the rational transform function as described in Sec. II. In the bandwidth-constrained clustering problem defined in Sec. I, we can convert a bandwidth function BW to a distance function d and a bandwidth constraint b to a distance constraint $l = \frac{C}{b}$. As a result, we can define this distance-constrained clustering problem. Given a metric space (V, d) and constraints $k \geq 2$ and l, find a set X such that $X \subseteq V$, $|X| = k$, and $d(u, v) \leq l \; \forall u, v \in X$. By defining the *diameter* of a set X as $diam(X) = \max_{\forall u,v \in X}\{d(u, v)\}$, the distance contraint can be rephrased as $diam(X) \leq l$.

A. Centralized Clustering in a Tree Metric Space

Algorithm 1 describes a simple centralized algorithm to find a cluster in a tree metric space (V, d). We first divide all the clusters that can be considered in (V, d) into several groups, each of which is associated with each node pair in V. The group of a node pair (p, q) includes all the clusters S_{pq} whose diameter is determined by (p, q), which means $p, q \in S_{pq}$ and $diam(S_{pq}) = d(p, q)$ in formal terms. Let S^*_{pq} denote the maximum size cluster in each group. Since S^*_{pq} is the best cluster in each group in terms of cluster size, checking only S^*_{pq} for each group will be satisfactory to find a wanted cluster.

With this intuition, Algorithm 1 iterates every node pair (p, q) in V and 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 3.1, we know that Algorithm 1 correctly creates S^*_{pq}. If S^*_{pq} satisfies the constraints k and l, then the algorithm stops iterating pairs and returns any k nodes in S^*_{pq}. If the algorithm did not find any S^*_{pq} that satisfies the constraints, we can ensure that a wanted cluster does not exist because the algorithm has checked all possible clusters.

Theorem 3.1: (Correctness of Algorithm 1) Given a tree metric space (V, d) and constraint values $k \geq 2$ and l, if Algorithm 1 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 $|S_{pq}| > |S^*_{pq}|$ and $diam(S_{pq}) = d(p, q)$.

Proof of Theorem 3.1: 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 1. Otherwise, three cases can be considered by the order of three distance sums in 4PC of p, q, r, and s.

1) $d(p, q) + d(r, s) \leq d(p, r) + d(q, s) = d(p, s) + d(q, r)$
   By the assumption of 1) $d(r, s) \leq d(p, r) + d(q, s) - d(p, q)$ and $d(r, s) - d(p, q) \leq (d(p, r) - d(p, q)) + (d(q, s) - d(p, q))$. Since $d(p, r) \leq d(p, q)$ and $d(q, s) \leq d(p, q)$ by definition of S^*_{pq}, $d(r, s) - d(p, q) \leq 0$.
2) $d(p, r) + d(q, s) \leq d(p, s) + d(q, r) = d(p, q) + d(r, s)$
   By the assumption of 2), $d(r, s) = d(p, s) + d(q, r) - d(p, q)$ and $d(r, s) - d(p, q) = (d(p, s) - d(p, q)) + (d(q, r) - d(p, q))$. Since $d(p, s) \leq d(p, q)$ and $d(q, r) \leq d(p, q)$ by definition of S^*_{pq}, $d(r, s) - d(p, q) \leq 0$.
3) $d(p, s) + d(q, r) \leq d(p, r) + d(q, s) = d(p, q) + d(r, s)$
   Similarly to 2), $d(r, s) - d(p, q) \leq 0$.

Thus, $diam(S^*_{pq}) = d(p, q)$.

Now let's assume that there exists $S_{pq} \subseteq V$ such that $|S_{pq}| > |S^*_{pq}|$ and $diam(S_{pq}) = d(p, q)$. $|S_{pq}| > |S^*_{pq}|$ implies that there is 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, there exists no $S_{pq} \subseteq V$ such that $|S_{pq}| > |S^*_{pq}|$ and $diam(S_{pq}) = d(p, q)$. ∎

When n is the number of nodes in V, the algorithm takes $O(n^3)$ time because it takes $O(n)$ to create S^*_{pq} for each pair and $O(n^2)$ to iterate every pair. We would not claim that Algorithm 1 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 the real world as described in Sec. V, Algorithm 1 can find

a cluster in a polynomial time by determining S^*_{pq} under the assumption of tree metric space. Since bandwidth is close to a tree metric as described in Sec. II, Algorithm 1 can be applied to find a bandwidth-constrained cluster.

B. Decentralization

The design goal for decentralization is to let users submit a query to any node and to route it towards the direction where the wanted cluster exists. To achieve this decentralized query processing, we first construct an overlay network with all the hosts that participate in clustering as a member. By periodically exchanging messages with neighbors, each node aggregates the information of nodes that are close to itself. Then each node runs Algorithm 1 on the aggregated node information and figures out the maximum size cluster that the node can create. Each node then fill in the entry of a routing table by aggregating the information of the maximum size cluster that exists in each direction of neighbors. The details of these background mechanisms are provided in below.

1) Overlay Construction: Our decentralized clustering algorithm runs on top of the decentralized bandwidth prediction framework that is described in Sec. II. For that reason, all the hosts that are considered as a cluster member should participate in constructing an anchor tree. By directly using the overlay structure of the prediction framework, we can benefit in three aspects. First, we can find clusters quickly. Bandwidth-constrained clustering requires to measure bandwidth between many nodes. Instead of performing measurements at the time of clustering, we use the bandwidth data that are accurately predicted by the framework. Consequently, we can avoid any delay for extensive measurements when searching for a cluster. Second, we can exploit Algorithm 1 whose correctness is proved in a tree metric space. Since the framework accurately embeds bandwidth measurements into a tree metric space, Algorithm 1 is expected to work accurately with the bandwidth data predicted by the framework. Third, we can avoid any extra cost about overlay structuring. Since our approach already needs the prediction framework to achieve two benefits above, we would run our clustering algorithm directly on the framework rather than design a new overlay structure.

2) Dynamic Aggregation of Close Nodes: After becoming a member of the anchor tree, each node starts to periodically perform two types of background mechanisms, one of which is described here, and the other in the next section. By the first mechanism, each node x aggregates the information of the nodes that are close to x. This aggregated information of close nodes will be used as a local system space where each node can create a cluster. Algorithm 2 more specifically describes such a procedure about how x receives from each neighbor m on the anchor tree the information of the nodes that are the closest to x in the set of all the nodes reachable from x via m. Let p.aggrNode[q] denote the information of nodes that a node p receives from its neighbor q through p.DynAggrNodeInfo(q).

m first creates a set candNode by collecting nodes from m.aggrNode[v] for each of m's neighbor v except x. m is also included in candNode. To create another set propNode, m chooses the top n_{cut} nodes that are the closest to x regarding the distance on a prediction tree. Since the underlying framework is designed to predict the real bandwidth accurately [26], the predicted distance $d_T(u,v)$ is quite close to the real distance $d(u,v)$ for two nodes u and v. So, the n_{cut} chosen nodes should be close to x in terms of real distances. Note that the information size is limited to n_{cut} nodes, so that a messaging workload can be controlled in a distributed system. m sends propNode to x, so that x can set x.aggrNode[m]. Note that x.aggrNode[m] will dynamically change over time because the decentralized bandwidth prediction framework automatically restructures itself as network conditions change. Theorem 3.2 states that by running Algorithm 2 x receives the correct information from m, and the proof is also provided in below.

Algorithm 2: x.**DynAggrNodeInfo(**m**)**: Node x's procedure to dynamically aggregate from x's neighbor m the information of nodes that are close to x.

```
1  begin m's propagation to x
2      candNode ← {m}
3      foreach m's neighbor node v except x do
4          candNode ← candNode ∪ m.aggrNode[v]
5      propNode ← n_cut nodes that minimizes d_T(x,u) for
       all u ∈ candNode
6      m sends propNode to x
7  end
8  begin x's aggregation from m
9      x receives propNode from m
10     x.aggrNode[m] ← propNode
11 end
```

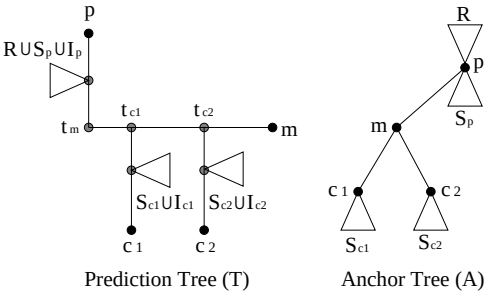


Fig. 2. Structures of the Prediction Framework from Node m's Perspective

Theorem 3.2: (Correctness of Algorithm 2) After the execution of Algorithm 2, x.aggrNode[m] will contain a set of the top n_{cut} nodes in U that minimizes the predicted distance $d_T(x,u)$ $\forall u \in U$ where U is a set of all nodes that are reachable from x via m on an anchor tree.

Proof of Theorem 3.2: Without loss of generality, assume that a prediction tree T and an anchor tree A look as shown in Fig. 2 from the perspective of node m. In A, m has p as its parent, c_1 and c_2 as its children. S_i denotes a set of child nodes of node i. $R = A \setminus (S_p \cup S_{c_1} \cup S_{c_2} \cup \{m, p, c_1, c_2\})$. T contains all nodes in A as leaf nodes. t_i means node i's

inner node. I_i denotes a set of inner nodes in the subgraph that contains a set S_i.

Induction will be used for proof. Considering three cases in terms of what x is, we divide the theorem into three statements and will prove each one. Those statements are included in STMT(m) that is defined as follows:

1) p.aggrNode[m] is a set of the top n_{cut} nodes that minimizes $d_T(p,u)\ \forall u \in U = A \setminus (\{p\} \cup R \cup S_p)$.
2) c_1.aggrNode[m] is a set of the top n_{cut} nodes that minimizes $d_T(c_1,u)\ \forall u \in U = A \setminus (\{c_1\} \cup S_{c_1})$.
3) c_2.aggrNode[m] is a set of the top n_{cut} nodes that minimizes $d_T(c_2,u)\ \forall u \in U = A \setminus (\{c_2\} \cup S_{c_2})$.

Basis: Show that STMT(m) holds when m has only one neighbor.

If the only neighbor x is p, we can focus on the first statement. p.aggrNode[m] $= \{m\}$ holds by the algorithm. Since $U = \{m\}$ and $m \in U$ minimizes $d_T(p,m)$, the first statement is true, so does STMT(m). Similarly, STMT(m) is true when $x = c_1$ or $x = c_2$.

Inductive Step: Show that if STMT(j) holds for each of m's neighbor j, then STMT(m) also holds.

Since all the statements in STMT(m) can be proved in the similar way, we only focus on the first statement. By the induction hypothesis, the first statement of STMT(c_1) is true, and $U = \{c_1\} \cup S_{c_1}$ holds. This implies m.aggrNode[c_1] is equal to a set of the top n_{cut} nodes that minimizes $d_T(m,u)\ \forall u \in U$. As you can see in T of Fig. 2, $d_T(m,u) = d_T(m,t_{c_1}) + d_T(t_{c_1},u)\ \forall u \in U$ with a constant $d_T(m,t_{c_1})$. So m.aggrNode[c_1] should also be equal to a set of the top n_{cut} nodes that minimizes $d_T(t_{c_1},u)\ \forall u \in U$. Since $d_T(p,u) = d_T(p,t_{c_1}) + d_T(t_{c_1},u)\ \forall u \in U$ with a constant $d_T(p,t_{c_1})$ as shown in T, m.aggrNode[c_1] should also be equal to a set of the top n_{cut} nodes that minimizes $d_T(p,u)\ \forall u \in U = \{c_1\} \cup S_{c_1}$.

The first statement of STMT(c_2) is true with the induction hypothesis. So, similarly, m.aggrNode[c_2] is a set of at most n_{cut} nodes that minimizes $d_T(p,u)\ \forall u \in U = \{c_2\} \cup S_{c_2}$.

By Algorithm 2, p.aggrNode[m] is a set of the top n_{cut} nodes that minimizes $d_T(p,u)\ \forall u \in \{m\} \cup m$.aggrNode[$c_1$] $\cup$ m.aggrNode[c_2]. By the above observations about m.aggrNode[c_1] and m.aggrNode[c_2], p.aggrNode[m] must be equal to a set of the top n_{cut} nodes that minimizes $d_T(p,u)\ \forall u \in \{m, c_1, c_2\} \cup S_{c_1} \cup S_{c_2}$, which is $A \setminus (\{p\} \cup R \cup S_p)$. So, the first statement of STMT(m) is true.

Similarly, the other two statements are also true. Thus, Algorithm 2 produces a correct x.aggrNode[m] with a limited set candNode. ∎

3) Dynamic Aggregation of Maximum Cluster Size: By the second background mechanism, each node dynamically aggregates the information about the maximum size of clusters that exists in each direction of neighbors. The aggregated cluster size information is used for each node to construct a *cluster routing table* (CRT) that each node has to maintain in order to forward queries toward the direction where a wanted cluster exists. As a tradeoff for decentralization, we limit the flexibility for the choice of query constraint b. Instead of allowing users to choose any value of b, we let users to choose b from a predetermined set of bandwidth classes. With this limited flexibility of b, we can reduce the size of cluster routing table at each node.

Algorithm 3: x.**DynAggrMaxCluster(**m**)**: Node x's procedure to dynamically aggregate from x's neighbor m the maximum size of cluster in m's direction.

```
1  L ← {l_1, l_2, ..., l_|L|}
2  begin m's propagation to x
3      V_m ← {m}
4      foreach m's neighbor node v do
5          V_m ← V_m ∪ m.aggrNode[v]
6      d_{V_m} ← the distance function on V_m
7      foreach l ∈ L do
8          m.aggrCRT[m][l] ← the maximum k such that
           FindCluster(V_m, d_{V_m}, k, l) returns a non-empty set
9          propCRT[l] ← max_{∀v∈S}{m.aggrCRT[v][l]}
           where S = {m, m's neighbor except x}
10     m sends propCRT[l] ∀l ∈ L to x
11 end
12 begin x's aggregation from m
13     x receives propCRT[l] ∀l ∈ L from m
14     foreach l ∈ L do
15         x.aggrCRT[m][l] ← propCRT[l]
16 end
```

Algorithm 3 describes how a node x aggregates the information of the maximum size of cluster from its neighbor m. L is a fixed predetermined set of distance classes that are transformed from bandwidth classes for query constraints. Let p.aggrCRT[q][l] $\forall l \in L$ denote values that p receives from its neighbor q through p.DynAggrMaxCluster(q). m first defines m's *clustering space* (V_m, d_{V_m}) where m can create a cluster. V_m is defined as the union of $\{m\}$ and m.aggrNode[v] for all neighbors v. The distance function d_{V_m} on V_m is defined from the distance values on a prediction tree. Since the nodes in V_m are close to m, we can expect that they will be also close to each other. Accordingly, we can expect our clustering algorithm to be responsive to a difficult query with large bandwidth constraint b (or small l).

Algorithm 1 is executed several times until finding the maximum size of cluster that we can create with nodes in V_m. The binary search technique can be used for efficient searching. Then m sends to its neighbor x the maximum size of clusters that can be created by the nodes reachable from x via m. m sets propCRT[l] to be the maximum m.aggrCRT[v][l] for all $v \in \{m,\ m$'s neighbors except $x\}$. After m sends propCRT[l] $\forall l \in L$ to x, x sets x.aggrCRT[m][l] to the received propCRT[l] for each l. x's CRT is defined as a set of values of x.aggrCRT[v][l] for all x's neighbor v and all $l \in L$. Theorem 3.3 states that by running Algorithm 3 x receives the correct information from m for the entry of CRT. Since

Algorithm 4: x.**ProcessQuery(**k, l, m**)**: Node x's procedure to process a query (k, l) forwarded from node m

```
if k ≤ x.aggrCRT[x][l] then
    V_x ← {x}
    foreach x's neighbor node v do
        V_x ← V_x ∪ x.aggrNode[v]
    d_{V_x} ← the distance function on V_x
    X ← FindCluster(V_x, d_{V_x}, k, l)
else
    x_next ← any of x's neighbor v except m such that
    k ≤ x.aggrCRT[v][l]
    if x_next exists then X = x_next.ProcessQuery(k, l, x)
    else X = {}
return X
```

Theorem 3.3 can be proved by an induction in the similar way to Theorem 3.2, we omit the proof.

Theorem 3.3: (Correctness of Algorithm 3) After the execution of Algorithm 3, x.aggrCRT[m][l] will be equal to the maximum size of clusters that have diameter l and can be created by all the nodes that are reachable from x via m on an anchor tree.

4) Query Processing: Based on the background mechanisms explained above, we are finally ready to describe how to find a cluster in a decentralized fashion. A clustering query with size constraint k and diameter constraint l is first submitted to any node in the system, then each node forwards the query to its neighbor until finding a desired cluster.

Algorithm 4 describes how a node x processes a query (k, l) that is forwarded by its neighbor node m. A user can initiate searching by invoking x.ProcessQuery($k, l,$ null) at any node x. x first tries to find cluster by running Algorithm 1 on its clustering space (V_x, d_{V_x}). If it fails, x forwards the query to its neighbor x_{next} such that x is sure that there exists a cluster in x_{next}'s clustering space. The query should not be forwarded back to the previous node m to avoid any possibility of routing in an infinite cycle. If x ensures that there does not exist any cluster in any direction, the algorithm returns an empty cluster.

IV. Evaluation

This section evaluates our approach by examining i) accuracy of clustering, ii) tradeoff of decentralization, iii) effect of the treeness of dataset, and iv) scalability of query routing.

Our simulations are based on two datasets. The first dataset is named *HP-PlanetLab*. As described in [21], this dataset contains available bandwidth measurements between PlanetLab nodes collected at HP Labs using pathChirp [22]. Since the raw dataset 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. Then we converted the matrix to a symmetric one by estimating the average of two bandwidth values from forward and reverse directions of each pair of nodes. This symmetric matrix is considered as a set of real-world bandwidth measurements for our simulations. To provide more reliable experimental results, we collected another set of bandwidth measurements between PlanetLab nodes using pathChirp during two weeks starting in the late October in 2010 and named it *UMD-PlanetLab*. We preprocessed this new dataset into a full symmetric matrix of 317 nodes (out of 497) in the same way we did for HP-PlanetLab.

We simulated our clustering algorithm in Java by extending the simulator used to evaluate the decentralized bandwidth prediction system [25], [26]. Our simulator is implemented using the PeerSim [9] as a starting point.

A. Accuracy of Clustering in Tree Metric Space

Since there is no effective system to find a bandwidth-constrained cluster, we designed a new comparison model by combining two algorithms. We first embed bandwidth measurements into 2-d Euclidean coordinate space using Vivaldi [7]. The rational transform function is used to represent bandwidth as a metric as described in Sec. II. We now need a clustering algorithm to run based on the distance data that are predicted by Vivaldi. The centralized algorithm to solve the k-diameter problem which is described in a theoretical work [2], is used as a clustering algorithm on 2-d Euclidean coordinate space. The algorithm is about finding a set of k nodes that has the minimum diameter. By adding a diameter constraint l, the algorithm can be easily modified to find a set of k nodes with diameter at most l so that we can apply it to our bandwidth-constrained problem. Briefly explaining the algorithm, for each pair of nodes (p, q) such that $d(p, q) \leq l$, it first collects a set of nodes x such that $d(x, p) \leq d(p, q)$ and $d(x, q) \leq d(p, q)$,divides the found set into two sets to create a bipartite graph, and finds the maximum independent set X in the bibpartite graph. If $|X| \geq k$, then X is a cluster that satisfies the constraints.

As the correctness of the clustering algorithm is proved, clustering error of this comparison model only comes from imperfect bandwidth embedding of Euclidean space. We used the simulator for Vivaldi that is implemented in C++ by Ledlie [13], and we implemented the clustering algorithm in Python.

For the HP-PlanetLab dataset, we will show the results of three different approaches: **HP-TREE-DECENTRAL**, **HP-TREE-CENTRAL**, and **HP-EUCL-CENTRAL**. HP-TREE-DECENTRAL indicates our "decentralized clustering" algorithm on a tree metric space described in Sec. III, which runs on the bandwidth data estimated by the decentralized bandwidth prediction framework described in Sec. II. HP-TREE-CENTRAL means our "centralized clustering" algorithm on a tree metric space described in Sec. III, which runs on the bandwidth data estimated by the same framework as used in HP-TREE-DECENTRAL. HP-EUCL-CENTRAL represents the comparison model described above. HP-EUCL-CENTRAL uses a "centralized clustering" algorithm on a 2-d Euclidean coordinate space, which runs on the bandwidth data estimated by the Vivaldi framework.

We constructed a bandwidth prediction framework with the HP-PlanetLab dataset and sent 1000 queries, each of which is a pair (k, b) of cluster size constraint $k = 10$ nodes and bandwidth constraint $b = 15 \sim 75$ Mbps. k is decided as 5% of the total number of nodes in the dataset, and b is decided in between 20-th percentile and 80-th percentile of real bandwidth in the dataset. We used such non-difficult queries so that the algorithms could find a cluster for all queries, and we can fairly compare the accuracy of result clusters. 10 rounds are executed as 10 different frameworks are constructed with different random seeds. So, total 10000 queries are examined for each of the three approaches.

We define a performance metric to compare the three approaches. WPR (Wrong Pair Rate) means the ratio of the number of wrong pair of nodes to the number of all pairs in all the clusters returned by a clustering algorithm. As shown in Fig. IV-A, WPR increases as b increases in all three approaches. With larger b, it is more likely that the bandwidth prediction framework of each approach incorrectly concludes $BW(u, v) \geq b$ for a node pair (u, v) when it is actually $BW(u, v) < b$. HP-TREE-CENTRAL and HP-TREE-DECENTRAL show higher accuracy than HP-EUCL-CENTRAL. This is because a tree metric space predicts bandwidth more accurately than a 2-d Euclidean metric space. Fig. IV-A shows CDF of relative errors of bandwidth prediction of two metric spaces. A relative error of a pair of nodes (p, q) is defined by $\frac{|BW(p,q)-BW_T(p,q)|}{BW(p,q)}$ where $BW(p, q)$ is the real bandwidth of (p, q) and $BW_T(p, q)$ is its predicted bandwidth. HP-TREE shows the result of our decentralized bandwidth prediction framework used in HP-TREE-CENTRAL and HP-TREE-DECENTRAL. HP-EUCL shows the result of Vivaldi system about bandwidth prediction. As you can see in Fig. IV-A, pairs in HP-TREE have smaller prediction errors than those in HP-EUCL.

You can see in Fig. IV-A that the clustering accuracy of HP-TREE-DECENTRAL and HP-TREE-CENTRAL is quite similar. This is because both approaches run based on the same bandwidth prediction framework. Since we proved the correctness of the clustering algorithms used in both approaches on a perfect tree metric space, the inaccuracy of clustering can be caused only by the underlying bandwidth prediction framework. Also, only queries with small k are submitted in this experiment, so that both approaches can find a cluster for every query. If a difficult query with large k is used, a decentralized clustering algorithm will be outperformed by a centralized one. This will be explained in the next section.

We executed the same simulations for UMD-PlanetLab except that we used different queries such that $k = 16$ nodes and $b = 30 \sim 110$ Mbps. The same notations are used except that HP is replaced with UMD. Fig. IV-A shows that a tree metric space works more accurately for bandwidth-constrained clustering than a 2-d Euclidean metric space. Also, this difference of clustering accuracy comes from the different embedding accuracy of bandwidth prediction frameworks as shown in Fig. IV-A.

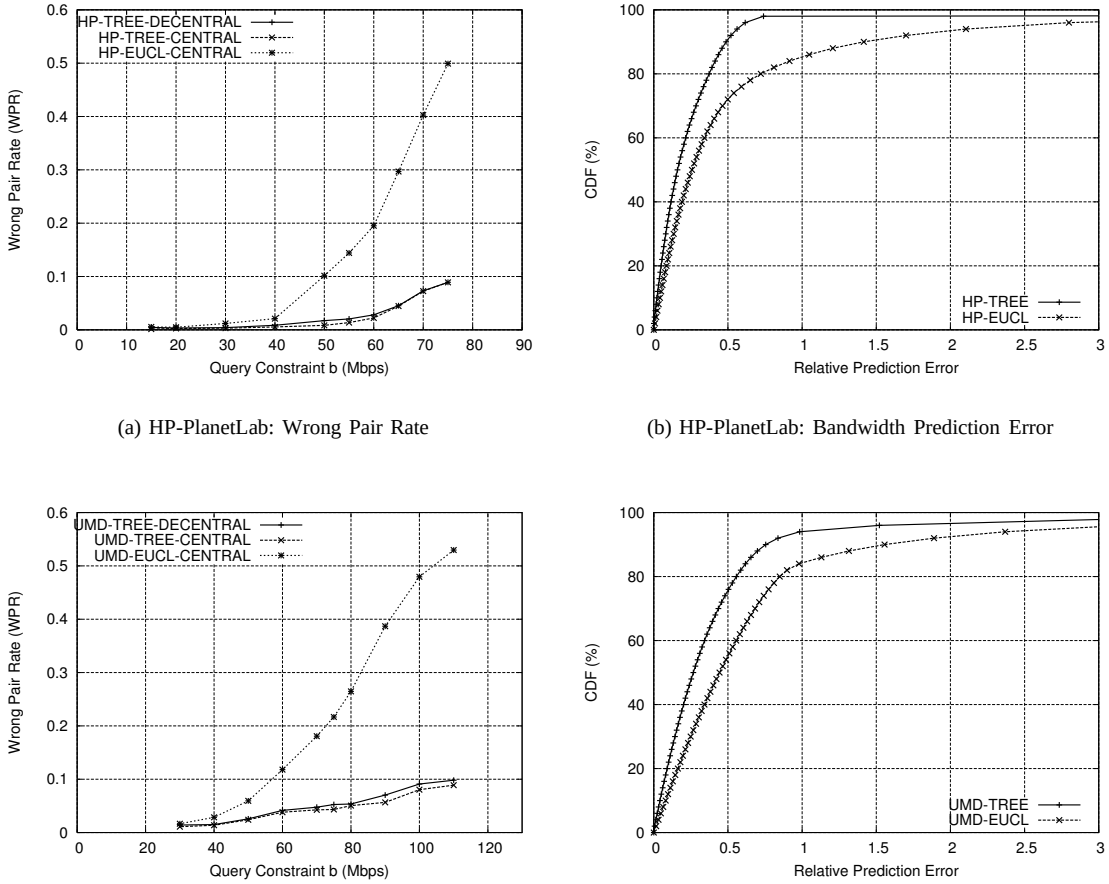
B. Tradeoff of Decentralization

As discussed in Sec. III, our decentralized clustering approach has the downside of a limited flexibility for the choice of query constraint b. This is useful to reduce the size of cluster routing table that each node maintains, and does not make worse the quality of clustering results. However, the second downside of decentralization might *not* allow the decentralized algorithm to find a cluster for some difficult queries with large k. When each node periodically sends the information of nodes to its neighbor by Algorithm 2, the information size is limited upto n_{cut} nodes. As a result, we can control a messaging workload in a distributed system to a desired degree. On the other hand, this results in a small clustering space where each node can create a cluster. Naturally, the decentralized clustering would not be so responsive as the centralized clustering.

We executed a simulation to see how this second downside of decentralization affects the result of clustering. For HP-PlanetLab, we constructed a bandwidth prediction framework and sent 100 queries, each of which is a pair (k, b) differently chosen from $k = 2 \sim 90$ nodes and $b = 15 \sim 75$ Mbps. 100 rounds are executed as 100 different frameworks are constructed with different random seeds. n_{cut} is set to 10 nodes. RR (Return Rate) means the ratio of the number of found clusters to the number of submitted queries. As shown in Fig. IV-B, as a query gets more difficult with larger k, RR gets smaller for both centralized and decentralized clustering algorithms. HP-TREE-DECENTRAL shows RR less than or equal to HP-TREE-CENTRAL at every k. This is because each node only knows about the information of partial system in the decentralized clustering approach. If a decentralized system receives a query with k bigger than $n_{\mathrm{cut}} \times \max\{n_{\mathrm{neigh}}\}$ where $\max\{n_{\mathrm{neigh}}\}$ is the maximum number of neighbors of nodes, a cluster satisfying the size constraint k can never be found. However, it is rare that a user wants to find a cluster of very large size. When k is less than 20% of the total number of nodes in the system, the difference in RR of both approaches is negligible. Moreover, as we already confirmed in the previous experiments, if k is small, WPR is also quite similar in both approaches. Thus, we can claim that our decentralized approach shows a high clustering accuracy and a high responsiveness compared to our centralized approach for queries with reasonably small constraint k. We executed the same simulations for UMD-PlanetLab except that we used different queries such that $k = 2 \sim 150$ nodes and $b = 30 \sim 110$ Mbps. The result in Fig. IV-B shows the similar trend to what we found with HP-PlanetLab.

C. Effect of Treeness

This section describes how the treeness of dataset affects the accuracy of our clustering algorithm. Abraham et. al defined a parameter $\varepsilon \in [0, \infty)$ to indicate how close a dataset is to a tree metric space [1]. Since ε is determined by each set of four nodes in a dataset, we will use the average value $\varepsilon_{\mathrm{avg}}$ to represent the treeness of one dataset. $\varepsilon_{\mathrm{avg}} = 0$ means the



(a) HP-PlanetLab: Wrong Pair Rate

(b) HP-PlanetLab: Bandwidth Prediction Error

(c) UMD-PlanetLab: Wrong Pair Rate

(d) UMD-PlanetLab: Bandwidth Prediction Error

Fig. 3. Clustering Accuracy: Bandwidth-constrained clustering works more accurate on a tree metric space than on a 2-d Euclidean space.

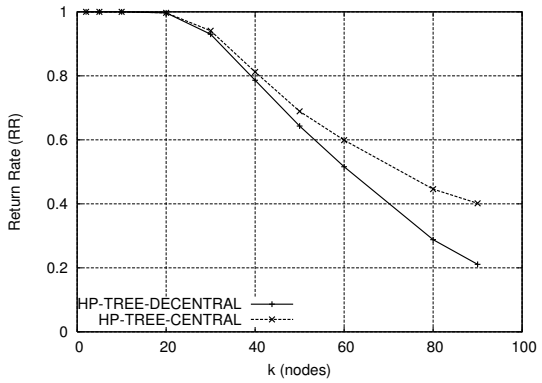
dataset is a perfect tree metric space, and large ε_{avg} indicates a bad treeness of dataset.

There are many factors that can affect WPR. System size N, bandwidth distribution, and ε_{avg} are the characteristics of dataset that can affect WPR. Query constraints k and b are also important factors. To focus on the effect of ε_{avg}, here we will fix N and k. Since it is not easy to have multiple datasets that have the same bandwidth distribution, we define two other variables f_b and f_a by combining two factors of bandwidth distribution and b. f_b is the fraction of node pairs with bandwidth less than b. f_a is the fraction of node pairs with bandwidth around b, in the range of $[b-10, b+10]$. In other words, f_b is CDF value at b, and f_a indicates how steep the slope of CDF at b is.

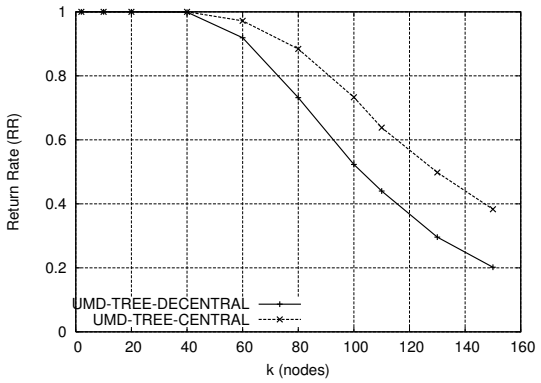
We will see how ε_{avg}, f_b, and f_a affect WPR. Let's assume that the bandwidth prediction error of a node pair, caused by a bandwidth prediction framework, does not affect errors of other pairs. Then WPR should be close to the probability that a bandwidth prediction framework chooses a wrong node pair for a given query to find a single pair with bandwidth bigger than or equal to b. In other words, WPR is the probability that while a bandwidth prediction framework chooses a node pair (p, q) such that the predicted bandwidth $BW_T(p, q)$ is at least b, the real bandwidth $BW(p, q)$ is actually less than b.

It is clear that WPR increases as f_b increases because there are fewer choices of node pairs with bandwidth bigger than or equal to b. We can also expect that large ε_{avg} leads to large WPR because the bandwidth prediction framework will result in high errors of bandwidth embedding for imperfect tree metric space. Let's transform $\varepsilon_{\text{avg}} \in [0, \infty)$ to a bounded variable $\varepsilon^*_{\text{avg}} \in [0, 1]$ for later usage by defining $\varepsilon^*_{\text{avg}} = 1 - \frac{1}{1+\varepsilon_{\text{avg}}}$. Large f_a means that there are many node pairs around b, and it will increase the effect of ε_{avg}. Accordingly, if there are two datasets with the same ε_{avg} values but different f_a, then the dataset with larger f_a will get higher WPR. By defining $f^*_a = (\alpha - \frac{1}{\alpha}) \times f_a + \frac{1}{\alpha}$ with a constant $\alpha > 1$, we can transform $f_a \in [0, 1]$ to a variable with a different bound $f^*_a \in [\frac{1}{\alpha}, \alpha]$. We can multiply $\varepsilon^*_{\text{avg}}$ by f^*_a to strengthen or weaken the effect of $\varepsilon^*_{\text{avg}}$ by α times. Such an adjusted variable for treeness is defined by by $\varepsilon^{\#}_{\text{avg}} = \varepsilon^*_{\text{avg}} \times f^*_a$. And we can bound $\varepsilon^{\#}_{\text{avg}} \in [0, 1]$ by setting $\varepsilon^{\#}_{\text{avg}} = 1$ for $\varepsilon^*_{\text{avg}} \times f^*_a > 1$.

Now let's consider a more concrete mathematical model of



(a) HP-PlanetLab: Return Rate



(b) UMD-PlanetLab: Return Rate

Fig. 4. Tradeoff of Decentralization: In a decentralized approach, each node maintains the partial information of system, so that the algorithm cannot find a cluster for a difficult query with very large k.

WPR with f_b and $\varepsilon^{\#}_{\text{avg}}$. For $f_b \in [0,1]$, $WPR = 0$ when $f_b = 0$, and $WPR = 1$ when $f_b = 1$. For $\varepsilon^{\#}_{\text{avg}} \in [0,1]$, $WPR = 0$ when $\varepsilon^{\#}_{\text{avg}} = 0$ because the bandwidth prediction framework will predict perfectly the bandwidth of pairs around b. When $\varepsilon^{\#}_{\text{avg}} = 1$, we expect $WPR = f_b$. If a dataset is infinitely far from a tree metric space, the prediction framework will be totally confused about how to predict pairwise bandwidth. Accordingly, when the predicion framework tries to choose a node pair with the bandwidth at least b, it will be likely to choose a random pair. Assuming that a uniformly random pair is chosen, WPR should be the same as f_b. Considering these relations between WPR, f_b, and $\varepsilon^{\#}_{\text{avg}}$, we can define the following model.

$$WPR = f_b^{(1/\varepsilon^{\#}_{\text{avg}})} = f_b^{(1/\varepsilon^{*}_{\text{avg}})(1/f^{*}_{a})} \quad (1)$$

We did a simulation to see how effective Equation 1 is. By choosing subsets from HP-PlanetLab, we created six datasets of 100 nodes with different treeness. For each subset, we constructed a bandwidth prediciton framework and sent 2000 queries such that $k = 5$ nodes and $b = 5 \sim 300$ Mbps. And we ran 10 rounds by constructing 10 different frameworks with different random seeds. Fig. IV-C and Fig. IV-C show some curves for each dataset, and the number that is included in a legend represents ε_{avg} of each dataset. And the general shape of all the curves is quite similar to what we expected in Equation 1, where WPR increases as f_b does following the shape of $WPR = f_b^c$ with a constant $c > 1$. However, when we plotted the relation between f_b and WPR in Fig. IV-C, any effect of ε_{avg} was not seen. By computing f^*_a with $\alpha = 3.2$ and normalizing WPR to $(WPR)^{f^*_a}$, we could see that a dataset with large ε_{avg} has a large normalized WPR in Fig. IV-C. Since $(WPR)^{f^*_a} = f_b^{(1/\varepsilon^{*}_{\text{avg}})}$, the effect of ε_{avg} could appear in Fig. IV-C. The dataset of large ε_{avg} is plotted above that of small ε_{avg}.

We did exactly the same experiment for UMD-PlanetLab, and the result is shown in Fig. IV-C and Fig. IV-C. While WPR-f_b curves in Fig. IV-C do not show any effect of ε_{avg}, we could see the effect of treeness in $(WPR)^{f^*_a}$-f_b curves in Fig. IV-C.

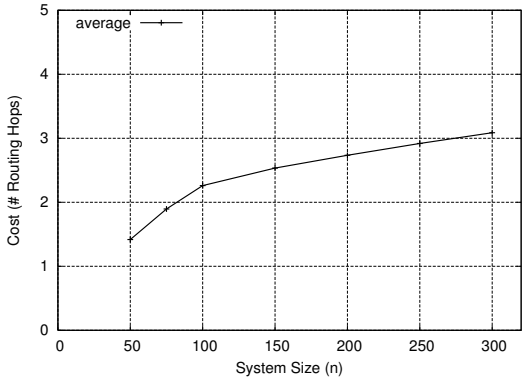
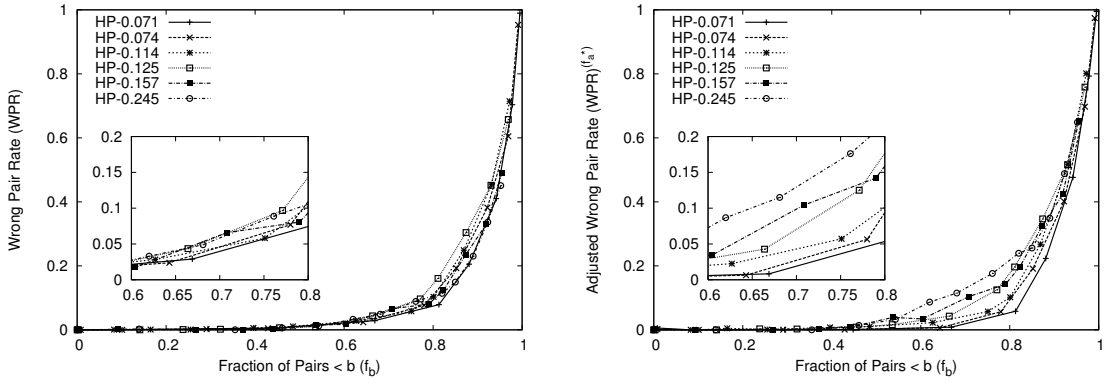


Fig. 6. Scalable Query Routing Cost: The number of query routing hops increases in a scalable way.

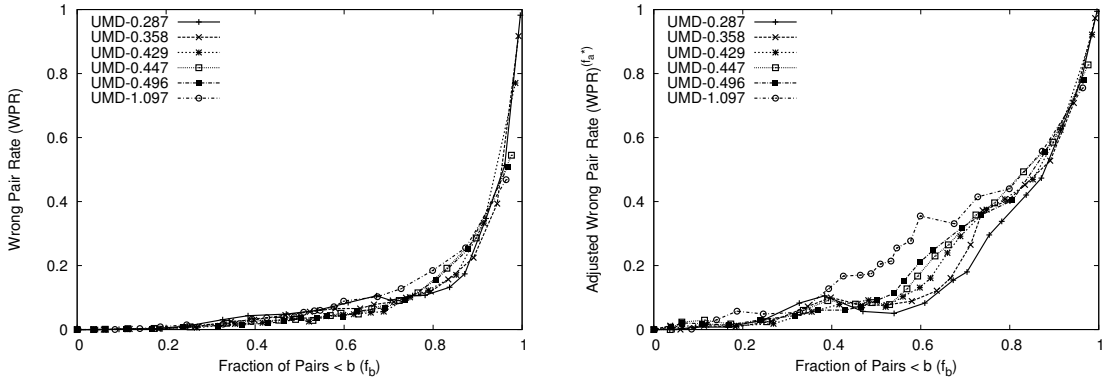
D. Scalability

The last experiment is about how the number of query routing hops increases as the system size n does. We created 10 different datasets with the same n by choosing a random subset from UMD-PlanetLab. Total 70 datasets are created for $n = 50 \sim 300$ nodes. For each dataset, we constructed a bandwidth prediction framework and sent 1000 queries such that $k = 0.05n \sim 0.30n$ and $b = 30 \sim 110$ Mbps. 10 rounds are executed as 10 different frameworks are constructed with different random seeds. Then we computed the average number of query routing hops for each system size n. As shown in Fig. 6, the average hop count is quite small, which is around two or three hops. Also, it increases slowly as n increases, shaping a concave curve.



(a) HP-PlanetLab: WPR vs f_b

(b) HP-PlanetLab: $(WPR)^{f_a^*}$ vs f_b



(c) UMD-PlanetLab: WPR vs f_b

(d) UMD-PlanetLab: $(WPR)^{f_a^*}$ vs f_b

Fig. 5. The Effect of Treeness: A dataset that is close to a tree metric space with low ε results in higher accuracy of clustering. The effect can be shown with WPR normalized by f_a^*.

V. Related Work

There exist several systems that predict end-to-end network performance without performing *n-to-n* measurements. By using the data accurately predicted by such systems, we can expect to find a cluster quickly without spending any delay performing extensive measurements. GNP [18], PIC [6], and Vivaldi [7] successfully embed end-to-end latency into an almost Euclidean space. However, those systems are not good for bandwidth prediction, accordingly an attempt to embed bandwidth measurements using Vivaldi results in poor accuracy [21] when the linear transform function $d(u,v) = C - BW(u,v)$ is used to represent bandwidth as a metric. Even though we found that Euclidean space could show much higher accuracy with our rational transform function $d(u,v) = \frac{C}{BW(u,v)}$, it is still not so much accurate as tree metric space as shown in Sec. IV. Based on the finding of the treeness of bandwidth as described in Sec. II, Sequoia [21] constructs an edge-weighted tree to embed bandwidth measurements with low embedding errors. Our prior works [25], [26] decentralized Sequoia by removing a single fixed measurement bottleneck and succeeded in accurately predicting pairwise bandwidth.

k-Clique is a well-known problem as NP-completeand is about finding a clique of size k in an undirected graph G, where a clique in G is a complete subgraph of G. Our bandwidth-constrained clustering problem is equivalent to k-Clique in the real world because we can create an undirected graph with V by adding an edge (u,v) for nodes u and v such that $BW(u,v) \geq b$ where $BW(u,v)$ is the average of forward and reverse bandwidth between u and v. There are several researches to find a set of k nodes with a maximum diameter in a 2-d Euclidean space. Aggarwal et. al [2] provided $O(k^{2.5}n\log k + n\log n)$ algorithm and Eppstein et. al [8] 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 to find a bandwidth-constrained cluster because bandwidth does not fit Euclidean space very well. Instead, we designed $O(n^3)$ algorithm to solve the clustering problem in a tree metric space and applied it to our decentralized clustering approach.

There have been several research efforts about resource clustering. Liu et. al [16] introduce a hierarchical cluster

structure and propose an approximate algorithm to answer queries for resource clustering. The similarity to our approach is that they support a query with two constraints: the size and network distance of cluster. However, they only consider latency-constrained clustering, and it is not feasible to directly apply their approach to our problem. Their system constructs a centralized hierarchical structure, and communications are also centralized, so that each node must start measurement from the root node of the hierarchical structure. Shen et. al [24] 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 and memory. The difference between HCO and our approach is that HCO 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 [3] designed a distributed approximation algorithm for resource clustering and proved its correctness theoretically. They solved a complicated problem to answer a query with both distance constraint and storage capacity. However, they just provides an approximation, and especially they restricted their work to a 1-d Euclidean space which is definitely not a strong model to embed bandwidth measurements. SWORD [19] 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 of nodes, there is a serious limitation in their clustering scheme. SWORD basically relies on an exhaustive search taking an exponential time, and stops searching when timeout expires. On the other hand, our approach guarantees to answer a query in a polynomial time under the assumption of tree metric space.

We currently consider three possible applications of our decentralized bandwidth-constrained clustering algorithm. Our overall research project on P2P desktop grid systems motivates this work. When a data-intensive scientific set of jobs, such as CyberShake workflow [4], runs in a P2P desktop grid, we can increase the performance by assigning the jobset to a cluster of nodes connected through high-bandwidth connections. We can also expect to distribute large-scale data quickly in a content delivery network [17]. We first divide content subscribers into several high-bandwidth clusters, deploy data only to a few of nodes in each cluster, and finally let the data distributed quickly within each cluster. A P2P storage system [23] can use our clustering algorithm for fast and consistent maintenance of multiple replicas.

VI. Conclusions and Future Work

This paper has presented a decentralized and scalable method that accurately finds a cluster of Internet hosts with two constraints: clsuter size and minimum pairwise bandwidth. We show that our centralized algorithm takes a polynomial time in a tree metric space. A decentralized algorithm lets any node receive a query and routes the query to the direction where the wanted cluster exists. To do that, each node dynamically aggregates the information of other nodes. Simulation results with a newly collected dataset confirm high accuracy and scalability, and show the tradeoff of decentralization and the effect of treeness.

We are currently extending this work in several ways. First, we are working on a different type of node search algorithm. For a given set of multiple nodes, we are investigating approaches to find a single node that has high bandwidth with all the nodes in the input set. Second, we intend to use our system as the underlying technology for resource discovery in a P2P desktop grid [11], [10], [12], [14]. Finally, we will use our clustering approach to find a latency-constrained cluster. Since latency can also be successfully embedded into a tree metric space [21], we expect that our decentralized clustering approach can be directly applied to find a cluster under a latency constraint.

References

[1] I. Abraham, M. Balakrishnan, F. Kuhn, D. Malkhi, V. Ramasubramanian, and K. Talwar, "Reconstructing approximate tree metrics," in *PODC*, I. Gupta and R. Wattenhofer, Eds. ACM, 2007, pp. 43–52.

[2] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, "Fining points with minimum diameter and related problems," in *Symposium on Computational Geometry*, 1989, pp. 283–291.

[3] O. Beaumont, N. Bonichon, P. Duchon, and H. Larchevêque, "Distributed approximation algorithm for resource clustering," in *SIROCCO*, ser. Lecture Notes in Computer Science, A. A. Shvartsman and P. Felber, Eds., vol. 5058. Springer, 2008, pp. 61–73.

[4] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of Scientific Workflows," in *Proceedings of 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS08)*.

[5] P. Buneman, "A note on the metric properties of trees," *Journal of Combinatorial Theory, Ser. B*, vol. 17, pp. 48–50, 1974.

[6] M. Costa, M. Castro, A. I. T. Rowstron, and P. B. Key, "Pic: Practical internet coordinates for distance estimation," in *ICDCS*. IEEE Computer Society, 2004, pp. 178–187.

[7] F. Dabek, R. Cox, M. F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," in *SIGCOMM*, R. Yavatkar, E. W. Zegura, and J. Rexford, Eds. ACM, 2004, pp. 15–26.

[8] D. Eppstein and J. Erickson, "Iterated nearest neighbors and finding minimal polytopes," in *SODA*, 1993, pp. 64–73.

[9] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," `http://peersim.sf.net`.

[10] J.-S. Kim, P. Keleher, M. Marsh, B. Bhattacharjee, and A. 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-16)*. IEEE Computer Society Press, Jun. 2007.

[11] J.-S. Kim, B. Nam, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman, "Resource discovery techniques in distributed desktop grid environments," in *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing - GRID 2006*. IEEE Computer Society Press, Sep. 2006.

[12] J.-S. Kim, B. Nam, M. Marsh, P. Keleher, B. Bhattacharjee, and A. Sussman, "Integrating categorical resource types into a P2P desktop grid system," in *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing - GRID 2008*. IEEE Computer Society Press, Sep. 2008.

[13] J. Ledlie, "The vivaldi simulator," `http://www.eecs.harvard.edu/ syrah/nc/`.

[14] J. Lee, P. Keleher, and A. Sussman, "Decentralized resource management for multi-core desktop grids," in *Proceedings of the 24th International Parallel & Distributed Processing Symposium*. IEEE Computer Society Press, Apr. 2010.

[15] S.-J. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca, "Measuring bandwidth between planetlab nodes," in *PAM*, ser. Lecture Notes in Computer Science, C. Dovrolis, Ed., vol. 3431. Springer, 2005, pp. 292–305.

[16] C. Liu and I. T. Foster, “A scalable cluster algorithm for internet resources,” in *IPDPS*. IEEE, 2007, pp. 1–8.
[17] A. Nandi, A. Ganjam, P. Druschel, T. S. E. Ng, I. Stoica, H. Zhang, and B. Bhattacharjee, “Saar: A shared control plane for overlay multicast,” in *NSDI*. USENIX, 2007.
[18] T. S. E. Ng and H. Zhang, “Predicting internet network distance with coordinates-based approaches,” in *INFOCOM*, 2002.
[19] D. L. Oppenheimer, J. R. Albrecht, D. A. Patterson, and A. Vahdat, “Design and implementation tradeoffs for wide-area resource discovery,” in *HPDC*. IEEE, 2005, pp. 113–124.
[20] V. Ramasubramanian, D. Malkhi, F. Kuhn, I. Abraham, M. Balakrishnan, A. Gupta, and A. Akella, “A unified network coordinate system for bandwidth and latency,” Microsoft Research, Tech. Rep. MSR-TR-2008-124, 2008.
[21] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella, “On the treeness of internet latency and bandwidth,” in *SIGMETRICS/Performance*, J. R. Douceur, A. G. Greenberg, T. Bonald, and J. Nieh, Eds. ACM, 2009, pp. 61–72.
[22] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, “pathchirp: Efficient available bandwidth estimation for network paths,” in *In Passive and Active Measurement Workshop*, 2003.
[23] A. I. T. Rowstron and P. Druschel, “Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility,” in *SOSP*, 2001, pp. 188–201.
[24] H. Shen and K. Hwang, “Locality-preserving clustering and discovery of wide-area grid resources,” in *ICDCS*. IEEE Computer Society, 2009, pp. 518–525.
[25] S. Song, P. J. Keleher, B. Bhattacharjee, and A. Sussman, “Brief announcement: Decentralized network bandwidth prediction,” in *DISC*, ser. Lecture Notes in Computer Science, N. A. Lynch and A. A. Shvartsman, Eds., vol. 6343. Springer, 2010, pp. 198–200.
[26] ——, “Decentralized network bandwidth prediction,” in *INFOCOM*, 2011, submitted.