A Scalable Cluster Algorithm for Internet Resources

Chuang Liu[1], Ian Foster[2,3]
[1]Microsoft, [2]University of Chicago, [3]Argonne National Laboratory
[1]chuangl@microsoft.com, [2]foster@cs.uchicago.edu

Abstract

Applications such as parallel computing, online games, and content distribution networks need to run on a set of resources with particular network connection characteristics to get good performance. To locate such resource sets, we introduce a scalable algorithm to compute a hierarchical cluster structure for a large number of Internet resources such that resources in a cluster have much smaller latency with each other than with other resource. Using the hierarchical cluster structure, we propose an approximate algorithm to answer queries for a resource set with desired network connections. We evaluate this method in a large distributed Internet environment including 2500 DNS servers, and show that our algorithm can locate required resources with high accuracy in much shorter time than traditional methods.

1 Introduction

Applications such as parallel computing, online games, and content distribution networks need to run on a set of resources with particular network connection characteristics to get good performance. Suppose we have a large number of geographically distributed, Internet-connected resources. We want an algorithm that, for any N and L, can provide efficient and accurate answers to a query for a set of N resources with the property that the network latency between any pair of those resources is less than L milliseconds. This is not a trivial problem because:

- It is computationally expensive to locate a set of resources with desired network connections among a large number of available resources. This problem is NP complete because it can easily be transferred to a K-clique problem [7].
- It is expensive to measure and store the latency information among resources. To locate resources with desired network connections, the search algorithm needs to know the pair-wise latency information between resources. Because the number of possible latency measurements is N^2 with N as the number of resources, it becomes difficult to measure and store pair-wise latency when dealing with thousands of resources.
- Resources join the resource pool incrementally. A resource pool becomes bigger and bigger over time as more resources join. For example, the Gnutella [8] network has grown to more than one million nodes in several years.

Many schemes, such as landmark [5] and network coordinates [3, 4, 11, 12, 13], have been proposed for representing resource network locations, and a lot of services, such as SWORD [1], XenoSearch [14], and Meridian [2], have been built for locating resources based on their network locations However, none of them combine the representation and location of resources to provide a scalable solution for locating a set of resources based on their pair-wise network connections.

These considerations lead us to propose an algorithm that first uses a scalable cluster algorithm to partition resources into clusters based on end-to-end network latency such that resources in a cluster have much smaller latency with each other than with other resources, and then performs searches within this reduced cluster graph to answer queries. We show experimentally that this algorithm performs well and generates robust results in practical settings. The primary contributions of this paper are:

- A scalable method for determining a hierarchical cluster structure for resources. This method works without knowledge of resources' locations or network topologies. In addition, this method can modify the cluster structure incrementally to reflect changes when new resources join the resource pool.
- A method to organize and store latency measurements among resources. This method organizes resources into a hierarchical cluster structure. Instead of storing pair-wise latency for all resources, we store only the

1-4244-0910-1/07/$20.00 ©2007 IEEE.

average latency for each cluster. We show this information is enough to answer queries accurately, and needs only O(N) space.

- An approximate algorithm to answer queries for a set of resources with desired network connections. This algorithm uses no pair-wise latency between resources but instead the average intra-cluster latency to provide approximate answers. We show that this algorithm can improve search performance remarkably without loss of accuracy and search capability.

The paper is organized as follows. In Section 2, we discuss related work. In Section 3, we present an algorithm for computing the cluster structure of networked hosts based on latency measurements. In Section 4, we propose an approximate algorithm for finding resource sets with desired network connections, and confirm via experiments that this algorithm has fast response time, high accuracy, and a strong ability to find query results. In Section 5, we summarize our work.

2 Related Work

One approach to answering queries for resource sets with desired network connections is by direct computation [1, 6], based on experimentally determined pair-wise inter-resource latencies. Because the possible number of resource pairs is $O(N^2)$, this approach not only requires much network traffic to measure pair-wise latency, but requires a large amount of space to store this information when the number of resources is large. Instead of storing the pair-wise latency, virtual-coordinates based solutions assign a virtual coordinate to each resource such that the latency between any resources can be calculated as the Euclidian distance based on their coordinates [3, 4, 11, 12, 13]. Because this approach stores a virtual coordinates for each resource, it reduces spaces for storing pair-wise latency from $O(N^2)$ to O(N). However, Wong [2] shows that these virtual coordinates introduce significant errors.

Second, given a set of nodes and their latency measurements, it is a NP-hard problem to find a subset of resources with particular connection properties. Therefore, direct computation will not scale because the complexity of direct computation may be exponential to the number of resources in the worse case. In landmark-based solutions [6], each landmark keeps track of its distance to all resources. This approach can find only the closest resource to a given resource by querying all landmarks for resources that are roughly the same distance away from the landmarks. Choosing the number of landmarks and selecting landmarks are non-trivial and have significant impact on the accuracy of the approach. Instead of using statically configured landmarks, Meridian [2] proposes a more general peer-to-peer structure in which each resource keeps track of a set of resources with distances within given ranges called *rings*. This approach can support queries for one resource satisfying multiple constraints. In comparison, we focus on queries for a resource set satisfying multiple constraints.

NICE [15] organizes resources in the Internet into a balanced tree structure based on pair-wise latency to provide a scalable multicast service. Although the created tree structure could help locate resource sets with desired network connections, no work has been done to apply this technique to solve the resource location problem considered in this paper. More importantly, NICE always creates a balanced hierarchical cluster structure. However, the distribution of resources in an Internet-based test bed (like Planetlab [16]) is often unbalanced. Part of Internet may contain many resources, while other parts contain few resources. We create a hierarchical cluster structure that represents the distribution of resources in the Internet, and could answer queries for resource sets more accurately.

3 An Incremental Cluster Algorithm

Intuitively, resources in the Internet form a hierarchical cluster structure. For example, the connection latency between resources from the same university campus is usually less than 1 millisecond; the connection latency between resources from different universities in the same area (such as computers from different campuses of University of California system) may be tens of milliseconds; and the connection latency between resources on different continents may be hundreds of milliseconds. We show an example of such a hierarchical structure in Figure 1. In this graph, computers from the same city form a tight cluster; clusters from the same state forms a looser cluster; and all resources form a top-level cluster.

We introduce an algorithm to find the hierarchical cluster structure of resources based on latency measurements. Although resource clusters have a strong correlation with resources' geographical locations, as shown by the example in Figure 1, we argue that using geographical locations to create a hierarchical cluster structure is problematic when these computers are connected through different networks. For example, computers connected to the Internet through different ISPs may have large connection latency even though they are close to each other geographically, and should be put in different clusters.

A hierarchical structure can be represented as a tree with each cluster as an internal node and each resource as a leaf, as shown in Figure 1. Each cluster in this structure contains several children, which can be resource nodes or sub clusters. We call these children *direct members*. Except for the root cluster, each cluster is a direct member of its parent cluster in the hierarchical structure.

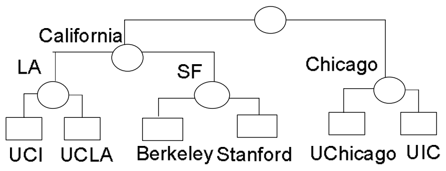

Figure 1 An example of hierarchical structure

We verified in previous work [9] that resources on Planetlab have a cluster structure. We will use DNS servers in the Internet to show that resources in a larger Internet environment also have a cluster structure. We admit that resources in a particular test bed may have no cluster structure, which means that network connections among resources are mostly homogenous. In this case, the selection of resource sets is not important because it does not make big difference anyway. In this paper, we focus on a more general case in which network connections are heterogeneous.

3.1 The Algorithm

Our algorithm allows us to construct the hierarchical cluster structure for resource pools whose resources join incrementally. The algorithm modifies the existing hierarchical cluster structure when a new resource node N joins the resource pool. Starting from the root, the algorithm recursively asks the new node to measure its latency to all direct members in the current cluster. Because direct members of a cluster could be subclusters, we choose one node in each such subcluster as its representative, and use the distance between a node and a representative (or between representatives) as the distance to the cluster.

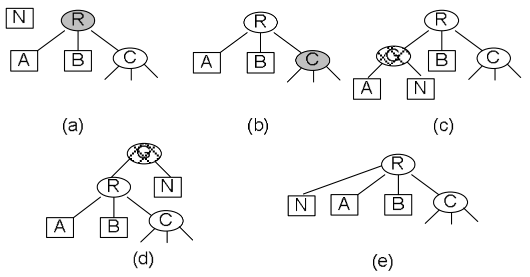

Figure 2 Four operations in the hierarchical cluster algorithm

Based on measurements between new node N and direct members of the current cluster R, the algorithm modifies the cluster structure in the following way. We use *Avg(R)* to represent the average value of known distance between direct members of cluster R, and use *g* to represent a configurable constant.

- If N is closest to a subcluster C and the distance between N and C is g-times smaller than Avg(R), the algorithm recursively inserts the node N into C, as shown in Figure 2 (b).
- If N is closest to a resource node A and the distance between them is g-times smaller than Avg(R), it means that N and A form a tighter cluster. The algorithm creates a new subcluster G containing N and A, and adds this subcluster as a new member in cluster R, as shown in Figure 2(c).
- If the closest distance between N and direct members of R is g-times larger than Avg(R), the algorithm creates a super cluster G that contains the new node N and the cluster R as its two members. If cluster R was a member of a super cluster T before the join of node N, the algorithm replaces R with G as the member of cluster T. The procedure is shown in Figure 2(d).
- If the closest distance between N and direct members of R is between avg(R)/g and avg(R)*g, the algorithm considers the distance of this node to direct members of R to be similar to Avg(R), and inserts this node as a member of this cluster, as shown in Figure 2(e).

In this way, the algorithm creates a hierarchical structure with following characteristics.

- Avg(A) is g-times smaller than Avg(B) if A is a child of B in the hierarchical cluster structure.
- Direct members in a cluster have similar latencies between them. We consider direct members in a cluster to have similar latency if their distances are between Avg(R)/g and Avg(R)*g.

By using different values of g, our algorithm can get hierarchical cluster structures with different granularities. We will study the effect of g in a subsequent section.

Because resources in a cluster have similar latency, we could just store the average latency to represent pair-wise latency in a cluster, and use it to answer queries for resource sets with desired connections without checking pair-wise latency (shown in Section 4). In comparison, NICE [15] constructs a hierarchical structure by splitting a cluster when this cluster contains too many resources. It could create a cluster containing resources with very different network connections, which makes it less efficient to handle queries for resource sets.

This hierarchical cluster algorithm enables the incremental modification of the cluster structure when resources join the resource pool. We will show that this algorithm scales to handle large number of resources. Other works [15,17] propose techniques to maintain a hierarchical cluster structure when resources leave in a dynamic distributed environment. Therefore, we will not discuss this issue in this paper. Instead, we will focus on studying the scalability of the proposed cluster algorithm, and its application on the resource selection problem.

3.2 Scalability

We use two metrics to evaluate the scalability of our cluster algorithm: number of latency measurements and number of recursions.

Number of latency measurements. To create the hierarchical cluster structure, the algorithm (Figure 2) measures the latency between each new node and some existing members in the resource pool. Because measuring latency among resources causes network traffic, a scalable algorithm should construct the cluster structure using only a small number of measurements.

Number of recursions. The algorithm needs to check multiple clusters recursively before it finds the right cluster for the new node. The number of checked clusters determines the execution time of this algorithm. Also, for each checked cluster, the algorithm requires the new node to measure its latency to direct members in this cluster, which will cause network traffic. Therefore, a scalable algorithm should have a small number of recursions.

Results. We use the meridian data collected by Wong [2] to evaluate the scalability of our algorithm. The Meridian data includes 2500 randomly chosen DNS servers at unique IP addresses, spanning 6.25 million node pairs. It is the biggest dataset we found, and has been used as a benchmark to evaluate Internet-based systems.

To simulate the process of resources joining the resource pool incrementally, we start with an empty resource pool, add these resources one by one, and measure the number of latency measurements and the number of recursive calls caused by each resource. As with other incremental cluster algorithms, these values are affected by the order in which resources join the resource pool. To reduce the effect of resource ordering, we repeat the experiment 100 times using different random orders for resource joins, and calculate the average numbers of latency measurements and recursive calls per resource join.

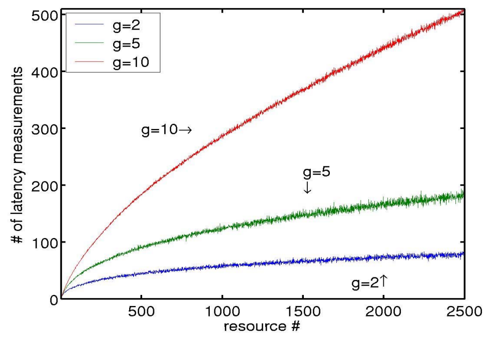

Figure 3 Number of latency measurements incurred by each resource join with g=2, 5, and 10

Figure 3 shows the average number of latency measurements incurred by the cluster algorithm as resources are added over 100 experiments. X axis is the order of resource joins, and Y axis is the number of latency measurements caused by a particular resource join. The first resource join is marked as 0 in the X axis, and the last resource join is marked as 2499.

The three curves in this graph compare the number of latency measurements incurred by the algorithm with g equal to 2, 5, and 10 respectively. We can see that the latency measurements incurred by a particular resource join increases logarithmically with the number of resources in the resource pool.

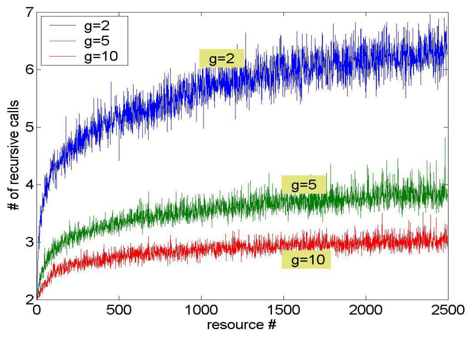

Figure 4 Number of recursive calls incurred by each resource join with g=2, 5, and 10

Similarly, Figure 4 shows the average value of the number of recursive calls incurred by the cluster algorithm as resources are added over 100 experiments. We can see that the number of recursive calls tends to flatten out as the number of resources increases, especially for the algorithm with g equal to 5 and 10. Because the average latency in a cluster tends to g-times smaller than the latency in its parent cluster, the depth of the hierarchical tree will not exceed the $\log_g L$ with L as the maximal latency in the Internet. Inserting a new node is a process of checking clusters in the path from the root to the cluster where the node will be inserted. Therefore, the number of recursive calls will not exceed $\log_g L$.

By measuring two metrics, we conclude that our algorithm has better scalability than previous approaches [1, 6] that require a large number of latency measurements $O(N^2)$.

Effect of g value. The parameter *g* affects scalability of the algorithm, and quality of the calculated cluster structure. Figure 3 shows that we incur the fewest latency measurements when g=2, while Figure 4 shows that we incur the fewest recursive calls when g=10. Therefore, the choice of g value is a tradeoff between the number of latency measurements and the number of recursive calls.

We use g equal to 2 in our algorithm for two reasons. First, the difference in the number of recursive calls is relatively smaller between algorithms with different g, compared to the the number of latency measurements (shown in Figure 4). And the difference in latency measurements between algorithms becomes much bigger with the increase of the number of resources (shown in Figure 3).

Effect of landmarks. We consider here a variation of the cluster algorithm. When a new resource joins the resource pool, instead of measuring its latency to all direct members in the current cluster, it only measures its distance to a given number K of resources called *landmarks* in this cluster. This method has been used to reduce the number of latency measurements required to create and maintain a topology structure of resources (such as ring structure in Meridian [2]). In this paper, we randomly choose K resources from a cluster as the landmarks. Although a more sophisticated landmark selection is possible, we will show that this method work reasonably well for locating resource sets.

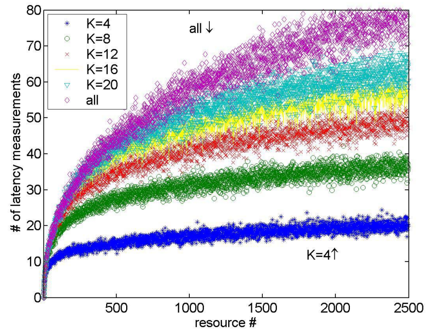

Figure 5 Number of latency measurements incurred by each resource join with K=4, 8, 12, 16, 20

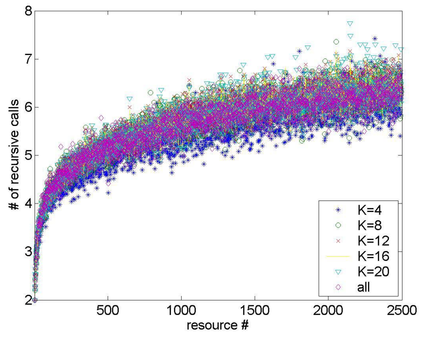

Figure 6 Number of recursive calls incurred by each resource join with K=4, 8, 12, 16, 20

To study the effect of K (the number of landmarks) on the cluster algorithm, we vary the value of K, and measure the performance of the algorithm.

Figure 5 shows the number of latency measurements incurred by each resource join with K=4, 8, 12, 16, and 20 respectively. For comparison, we also show results of the original algorithm that uses all direct members as landmarks (marked as *all* in the figure). Similar to the previous section, we repeat the experiment 100 times and show the average value. We can see that using a smaller number of landmarks reduces the number of latency measurements, and shows better scalability than the original method.

Figure 6 shows the number of recursive calls does not change much with various K values. Because the number of recursive calls is related to the length of the path from root to a cluster where a new resource will be inserted, it shows that the depth of the hierarchical structure does not change much.

Although these algorithms show good scalability, a remaining problem is whether they create good cluster structures. We will evaluate the quality of created cluster structures by their capability to handle considered queries in Section 4.2.

```
// R is the root of the cluster structure
// Q represents a query
Search(R, Q)
1. Add R into a queue S
2. While(S is not empty)
3.   C is the first cluster in queue S
4.   If (Avg(C) satisfies Q)
5.     Construct a result R by randomly
       picking N members from C
6.   End If
7.   Remove C from queue S
8.   Get sub-clusters in C and add them
     to S
9. End While
```

Figure 7 An approximate search algorithm

4 An Approximate Search Algorithm

We use the cluster structure to develop an efficient approximate algorithm searching for a resource set with desired network connections. As mentioned in Section 3, instead of storing pair-wise latency between resources, we store only the average latency between direct members for each cluster. We show in this section that this average value can be used to answer queries with high accuracy.

4.1 An Approximate Algorithm

In the hierarchical cluster structure created by the algorithm, members in each cluster have similar latency. Therefore, if the average latency satisfies the requirements in a query, it is likely that all pair-wise latencies in this cluster satisfy this query. Therefore, instead of checking pair-wise latency in a cluster, we propose an approximate algorithm (shown in Figure 7) that checks the average latency of clusters.

The search algorithm takes the root of a hierarchical cluster structure and a query as inputs. Starting from the root (line 2), this algorithm checks every cluster in the hierarchical cluster structure in preorder (line 1, 2, and 8). If average latency of a cluster satisfies the requirements in a query (line 4), the algorithm randomly picks the required number of resources from the cluster as a query result. The algorithm returns a set of query results after all clusters have been checked.

Comparing with the tree search algorithm [6] whose complexity is $O(2^N)$, our algorithm has complexity O(N). The main operation in our algorithm is to check the average latency of each cluster. Because the number of clusters is less than the number of resources, the complexity of this algorithm is O(N) with N as the number of resources in the worse case.

The reduced complexity comes at the cost of search capability and accuracy. First, our algorithm can only find a resource set consisting of resources from the same cluster. However, although our approach may miss many results, it can locate some results in a very short time, which it is important in a practical setting where users require their queries to be answered in real time, and they usually are not interested in finding all results. We show in the next section that our approach can find more results than other approaches within a short deadline.

Because our algorithm considers only the average latency value when deciding if all pair-wise latencies in a cluster satisfy the query condition, it might return results that do not satisfy the requirements. We call these results *false positives*. We show in Section 4.2 that false positives are only a small fraction of all results returned by our approach.

For applications requiring higher accuracy, we could maintain the maximal latency for a cluster and return results from clusters whose maximal latency value satisfies query requirements. Although this method could remove false positives, it is overly conservative. Part of resources in a cluster may satisfy a query. If only considering the maximum latency, this method could not find these query results.

Instead, we propose a configurable conservative way to use the average latency to answer queries. The algorithm considers a cluster a query result if its average latency is better by a factor of a parameter S than a query's requirements. We call S *secure factor*. For example, for a query for resources with pair-wise latency smaller than 50ms, the algorithm with S=5 only returns resources from clusters with average latency less than 10ms. By adjusting the value of S, We can get a balance between accuracy and search capability of the algorithm. We study this tradeoff in the next section.

4.2 Performance Evaluation

In this section, we use experimental results to illustrate that our approximate algorithm can find query results quickly without sacrificing either accuracy or the ability to find results. We compare our algorithm with the tree search algorithm [6] that has been widely used to answer queries for a set of resources with desired network connections [1].

Experimental settings. We implement our search algorithm and the tree search algorithms [6] in C++ and run the benchmark queries on a computer with an AMD Athlon 2 GHz CPU, 512 Mbytes memory, and Linux version 2.4.27-2-k6.

We create a set of benchmark queries to evaluate the performance of our approach. The query considered in this paper is a search for R resources with latency between each possible pair smaller than L. By varying the value of R and L, we have a set of different queries. We argue that most queries in practice are for a small set of resources with small latency. To simulate these queries, we build 1,000 queries with R as a random positive value smaller than 10% of the total number of resources, and L as a random positive value smaller than the median value of the latency among all resources.

We run the benchmark queries on Meridian data collected by Wong [2], which is one of the largest dataset available on the network latency of Internet resources. This data consists of 2500 randomly chosen DNS servers in the Internet. We run the original cluster algorithm (seen Section 3.1) with g=2 to create a hierarchical cluster structure, and use it in the approximate algorithm to answer queries. We will also study the effect of landmarks at the end of this section.

We use three metrics to evaluate the performance of an algorithm.

- *Response time* is the time needed by an algorithm to find results for a query. We use this metric to evaluate the speed of an algorithm.
- *Return Ratio (RR)* is the number of queries whose results are found by an algorithm. We use this metric to evaluate the search capability of an algorithm.
- *False positive rate (FPR)* is the percentage of results returned by an algorithm that actually do not satisfy a query. We use this metric to evaluate the accuracy of an algorithm.

Response time. We run the benchmark queries on Meridian data and measure the response time of both our approximate algorithm (with S=1) and the tree-based algorithm for one result. Because it might take the tree search algorithm a long time to return results, we set a deadline at three minutes.

We show the cumulative distribution of the execution time in Figure 8. The X axis is time, and the Y axis is the fraction of queries returned before that particular time. From this graph, we can see that our approximate algorithm responds to 80% of queries in less than 30 ms, and responds to all queries in less than 200 ms. In comparison, the tree search algorithm needs much more time, and responds to only 59% of queries before the deadline.

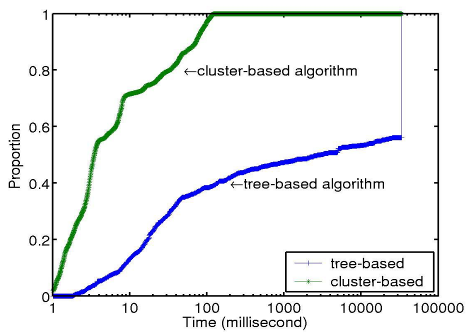

Figure 8 Cumulative distribution of query processing time for 1,000 benchmark queries

Return ratio (RR). Because our algorithm searches for results only in clusters, it may fail to find certain results, namely those that span two or more clusters. To evaluate our algorithm's ability to find query results, we measure the percentage of benchmark queries whose results are found by an algorithm (see Table 1).

Table 1 Return ratio as function of secure factor

S	1	1.5	2	2.5	3	Tree
RR	0.80	0.72	0.63	0.56	0.49	0.59

Among all 1000 benchmark queries, our algorithm can find results for 85% of queries with S=1. The ratio drops as the value of S increases. For comparison, the tree search algorithm returns only 59% of all queries before the deadline. Thus, we find that even though the tree search algorithm can, in theory, find results for all queries, in practice it finds results for fewer queries when subject to a deadline. For S equal to 1, 1.5, 2, our algorithm finds results for more queries than the tree algorithm.

Table 2 FPR as function of secure factor S

S	1	1.5	2	2.5	3
FPR	0.32	0.21	0.15	0.1	0.06

False positive rates (FPR). Because our algorithm uses average latency to answer queries, it might return false results due to individual pair-wise latencies being more than the average value. To evaluate the accuracy of our algorithm, we use the approximate algorithm to return all results that it can find, and measure the fraction of false positives among those results. Table 2 shows the false positive ratio for all results returned by our algorithm with different values of S. For the results returned by our algorithm with S=1, 32% of results violate the query. This number drops rapidly as the secure factor increases.

Table 3 RR & FPR of the ranking-based algorithm

RR	FPR
0.85	0.06

From Table 2, we can see that choosing the secure factor involves a tradeoff between accuracy and search capacity. For queries asking for just a few results, we can balance the algorithm's accuracy and search capacity by ranking all found results based on their accuracy. For each cluster, instead of testing the satisfiability using a particular S value, we check if it is 3-time better, 2-time better, or 1-time better than query requirements, and associate the S value (1, 2, or 3) with the results. For a query for K results, this algorithm returns the K results with highest S value.

We call this modified algorithm the *ranking-based algorithm*, and use the benchmark queries to evaluate its performance. Assuming that each query asks for just one result, the accuracy and number of results returned by ranking-based algorithm is shown in Table 3. We see this algorithm achieves a high return ratio and low false positive ration.

Effect of landmarks. As mentioned in Section 3, using a given number of landmarks can reduce the cost to create a hierarchical cluster structure. To measure the quality of created cluster structures, we run the approximate algorithm on these structures based on benchmark queries. We use a security factor 1, and search for all results. To avoid the effect of randomly chosen landmarks, we run each experiment 50 times, and show the median value, 10 and 90 percentile value for each measured metric.

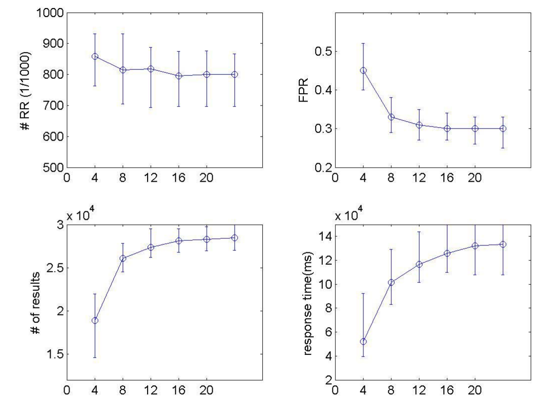

Figure 9 Effect of the number of landmarks on the search algorithm

Figure 9 shows the experimental results. X axis is the number of landmarks K; Y axis is the return ratio (RR), the false positive rate (FPR), the total number of results for all benchmark queries, and the aggregated response time for all benchmark queries respectively. For each graph in the figure, the first five points represent values produced by landmark-based cluster algorithms with K=4, 8, 12, 16, and 20; and the last one is the value produced by the original cluster algorithm.

First, from the graph on the left bottom and the graph on the right top, we can see that algorithms with more landmarks find more results and provide more accurate answers. It shows that more landmarks help build a better hierarchical structure. At the same time, we could see that algorithms with fewer landmarks have similar RR and responds queries quicker.

From these graphs, we could also see that algorithms with K= 12, 16, and 20 have very similar query performance as the original algorithm. Because they have better scalability (shown in Section 3), the landmark-based cluster algorithm is a better choice to answer considered queries. We will do more research on this issue in the future.

5 Summary

We have introduced an algorithm to compute a hierarchical cluster structure for a large and dynamic resource pool. This algorithm can modify the cluster structure incrementally as resources join. The number of latency measurements required by this algorithm is O(N.log(N)). In addition, our algorithm stores only the average latency for each created cluster, and thus requires only O(N) space. This average value can be used to answer queries for a resource set with desired network latency with high accuracy.

We have also proposed an approximate algorithm to answer queries for a resource set with desired network connections by using the hierarchical cluster structure. Experimental results show that our algorithm not only improves the search performance remarkably, it also finds results within a given deadline with high accuracy.

6 Acknowledgements

We are grateful to Anne Rogers and Lingyun Yang for comments on a draft of this paper.

References

[1] D. Oppenheimer, J. Albrecht, D. Patterson, and A.Vahdat, “Distributed resource discovery on planetlab with SWORD”, In *WORLDS*, San Francisco, 2004.

[2] B.Wong, A.Slivkins, E.G.Sirer, “Meridian: a lightweight network location service without virtual coordinates”, in *SIGCOMM*, Philadelphia, 2005.

[3] T. S. E. Ng and H. Zhang, "Predicting Internet distance with coordinates-based approaches," in *IEEE Infocom*, 2002.

[4] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for scalable distributed location," in *IPDPS*, 2003.

[5] L. Tang and M. E. Crovella, "Virtual landmarks for the internet," in *Internet Measurement Conference*, 2003.

[6] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, pp. 263-313, 1980.

[7] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*: The MIT Press, 1989.

[8] Gnutella, "The Gnutella web site: http://gnutella.wego.com," 2003.

[9] C. Liu, I. Foster, Robust Computation of Resource Clusters in the Internet, Chuang Liu, Ian Foster, in *Cluster*, Boston, 2005.

[10] H. Burch and B. Cheswick, "Mapping the Internet," *IEEE Computer*, 1997.

[11] R. Cox, F. Dabek, F. Kaahoek, J. Li, and R. Morris, Practical, distributed network coordinates, in *HotNets-II*, Cambridge, Massachusetts, 2003.

[12] P. Pietzuch, J. Ledlie, and M. Seltzer, “Supporting Network Coordinates on PlanetLab”, In *WORLDS*, San Francisco, 2005.

[13] F. Dabek, R. Cox, F. Kaahoek, R. Morris, “Vivaldi: A Decentralized Network Coordinate System”, In *SIGCOMM*, Portland, 2004.

[14] D. Spence and T. Harris, "XenoSearch: distributed resource discovery in the XenoServer open plantform," in *HPDC*, 2003.

[15] S. Banerjee, B. Bhattacharjee, C. Kommareddy, “Scalable Application Layer Multicast”, In *SIGCOMM*, Pittsburgh, 2002

[16] L. Peterson, S. Muir, T. Roscoe, A. Klingaman, “Planetlab Architecture: An Overview”, in *PDN-06-031*, 2006

[17] H. Yamaguchi, A. Hiromori, T. Higashino, K. Taniguchi, “An Autonomous and Decentralized Protocol for Delay Sensitive Overlay Multicast Tree”, in *ICDCS*, 2004