Location Based Placement of Whole Distributed Systems

David Spence
University of Cambridge Computer Laboratory
Cambridge, UK
David.Spence@cl.cam.ac.uk

Jon Crowcroft
University of Cambridge Computer Laboratory
Cambridge, UK
Jon.Crowcroft@cl.cam.ac.uk

Steven Hand
University of Cambridge Computer Laboratory
Cambridge, UK
Steven.Hand@cl.cam.ac.uk

Tim Harris
Microsoft Research
Cambridge, UK
tharris@microsoft.com

ABSTRACT

The high bandwidth and low latency of the modern internet has made possible the deployment of *distributed computing platforms*. The *XenoServer* platform provides a distributed computing platform open to all and presents three major new challenges for resource discovery: Firstly, network location is key for effectively provisioning services, to mitigate against high-latency, high-load or component failure. Secondly, many services require a presence on several servers, with inter-related requirements. Finally, as the platform is open with respect to users and servers, large numbers of queries and updates are expected.

To address these requirements we introduce and evaluate *XenoSearch,* a new distributed service for selecting the machines to host components of multi-node distributed systems and which is uniquely able to express and efficiently answer complex queries with inter-related location constraints. We demonstrate that XenoSearch represents a trade-off between accuracy and query time which avoids exhaustive search and supports multiple resources. In addition the performance of the algorithm and the quality of its server selections is investigated and the performance of the distributed service shown to be invariant as the number of nodes or items indexed increases.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed systems—*Distributed applications, Distributed Databases*; E.1 [**Data Structures**]: Distributed data structures; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms

Algorithms

CoNEXT'05, October 24–27, 2005, Toulouse, France.

Keywords

Peer-to-Peer, Resource Discovery, Location Systems

1. INTRODUCTION

Location and resource availability are key factors in the effective use of computing resources for network-centered applications, whether they are owned by the largest multi-national or a home user. The impressive increases in throughput gained by harnessing distributed resources, the commercial advantage of a well-designed e-commerce site or the lucrative revenue available to a new on-line game can all be negated if users experience high-latency, high-load or failure of the hosting machine or network connection.

The advent of distributed computing systems like Planet-Lab [1], XenoServers [2] and the Grid [3] addresses many of these problems by providing platforms on which an organization or user can deploy an application on remote servers for short timescales.

1.1 Requirements

The XenoServer platform requires a resource discovery system that brings together a unique set of features:

1.1.1 Location

Ensuring good quality of service is not simply a case of correctly provisioning hosting machines; latency and reliability must also be considered. As shown in [4] propagation delay is becoming the dominant factor in latency, and therefore the location of application components becomes important: they must be at network locations which are close to the expected users, have sufficient resource provisioning for the number of expected users, and exhibit low correlated probabilities of failure. To achieve this resource discovery systems need to accept *location constraints* as well as the conventional *resource constraints*.

1.1.2 Multiple-Resources

In general we must aim to select an optimal set of nodes for whole service deployment given complex inter-related constraints.

1.1.3 Scalability

The XenoServer platform is open both with respect to users and resources, and therefore its resource discovery system must be scalable with respect to items indexed, query load and, to a lesser extent update load.

The requirement for scalability, in addition to the normal concerns for availability and locality to users, leads us to choose a Peer-to-Peer (P2P) implementation. Recent efforts to provide P2P resource discovery have mostly focused on distributed range searching (e.g. SWORD [5] for Planet-Lab and Mercury [6] for Distributed Game Objects) or unstructured query flooding (e.g. P2P Grid resource discovery [7, 8]). P2P is a wide and varied term encompassing high churn unstructured P2P file-sharing systems and structured overlays running on well-provisioned servers. In our case we are considering the latter type and assume that any "nodes" used for the search system will be well-provisioned server-class machines active over months rather than minutes.

1.2 Contributions

Through the presentation and evaluation of our own resource discovery system, XenoSearch, developed for the XenoServer platform, this paper makes two main contributions applicable to the wider area of general resource discovery:

Firstly, we explore a whole new philosophy of resource discovery which makes location a key concern and allows expressive queries based on location. In XenoSearch we add to the standard notion of "closest" resource discovery the ability to request the distribution of servers for fault-tolerance and low-latency, and the combination of these operators through boolean operations.

Secondly, a new algorithmic approach to distributed resource discovery is taken. In XenoSearch the initial stage of query simplification is greatly enhanced. Previous P2P resource discovery systems have largely just performed a distributed range query for an attribute and then matched the resources in that range to the query. In XenoSearch, global summary information about both the location space and resource attributes are used to intelligently pick the correct nodes to send queries to and hence to greatly improve query accuracy.

XenoSearch is at the convergence of many of the approaches to resource discovery, combining overlays and their semantics from P2P systems with the multi-resource queries of Grid matchmaking; extending the flexible queries of spatial database technology; and using the web-search philosophy of providing the querier with a ranked set of results to choose from.

1.3 Overview

After defining a generalized model for distributed resource discovery in Section 2, we begin our description of XenoSearch. XenoSearch extends the generalized model from Section 2, to use summary information from its spatial index to optimize the simplification of queries. We commence in Section 3 with the novel query language developed to support location-based queries.

The XenoSearch algorithm, described in Section 4, allows the efficient minimization of user-provided cost/benefit functions over the inter-related locations of multiple resources, its spatial index definition allowing both centralized and distributed implementations. While a centralized implementation follows naturally from the definition, a scalable distributed implementation is more involved and is described in Section 5.

The performance of XenoSearch is demonstrated in Section 6, where it is shown to perform well under a variety of workloads. Following this we survey related work in Section 7 and conclude in Section 8.

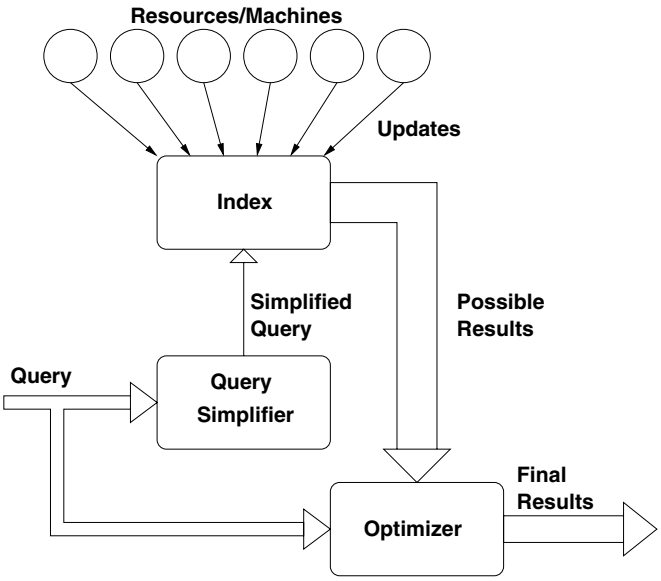

Figure 1: The general structure of resource discovery systems.

2. EXISTING DISTRIBUTED RESOURCE DISCOVERY

There are clear requirements on resource discovery systems within distributed computing platforms: to index the location, current resources and total resources of the machines; design protocols for obtaining updates; and specify a query format which includes support for location-based constraints.

Most resource discovery systems in this area use a three stage process. In the first stage queries are simplified into a form suitable for the index (e.g. ranges of attribute values). Secondly this simple form is used to search the index. In a distributed setting, aggregation occurs if results are obtained from more than one node. In the third stage the results from the index search are combined to give results for the complete query and the most optimal full results are returned that satisfy all the constraints in the query. Normally at this stage an exhaustive search is still infeasible so heuristics are used by and *optimizer* to produce final results. Ranking may also occur based on how well each result fits the constraints.

This setup is shown in Figure 1. The simplification of the query is key to the performance of the algorithm in the distributed case as it dictates how many remote servers are involved in the query resolution and how many superfluous results are returned. These affect the overall system load and network traffic and hence the user-perceived query latency.

Despite differences in implementation (for example push or pull updates, automatic or manual clustering, where the various stages are performed, etc.) this is the model that systems like SWORD [5] and Mercury [6] use. It is also the basic model followed by XenoSearch. In most cases, exhaustive search is likely to be infeasible, as it is characterized by an exponential function in the number of search terms; early investigations performed showed that exhaustive search query time scaled as $O(N^s)$, where N is the number of machines indexed and s is the number of servers in the query.

$$\text{far}(\text{near}(C_1,\ C_2,\ S_1),\ \text{near}(S_3,\ S_4)) \wedge \text{far}(\text{near}(C_1,\ C_3,\ S_2),\ \text{near}(S_3,\ S_4)) \wedge S_5$$

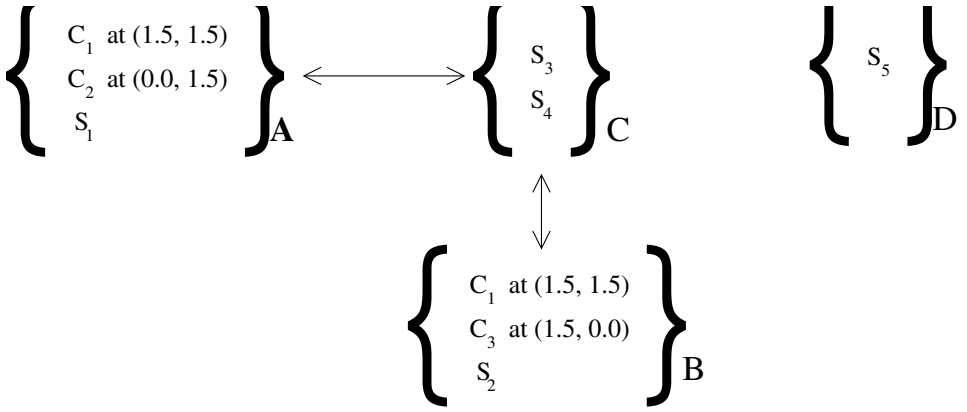

Figure 2: An example query, used for our running example, and the graph generated from it. In the example, client C_1 is located at $(1.5, 1.5)$, C_2 at $(0.0, 1.5)$ and C_3 at $(1.5, 0.0)$.

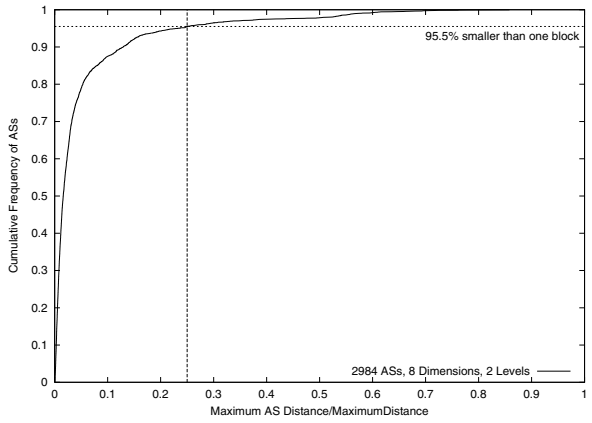

Figure 3: How distance in the location space is correlated to Autonomous System (AS). Here distance is the maximum distance between any two host in the same AS as a fraction of the maximum distance in the location space.

3. QUERY LANGUAGE

Here we define our new query language for the specification of spatial queries. Location requirements are defined recursively using the primitives of disjunction ($\vee$), conjunction ($\wedge$), proximity ($\text{near}(A_1, \ldots, A_n)$), distribution ($\text{far}(A_1, \ldots, A_n)$) and terms representing fixed locations (e.g. clients' positions in the network – C_i) and free servers to locate (S_i) — i.e. the resource request terms to be matched to machines. These queries are preprocessed to a disjunctive normal form in which each disjunction expresses an alternative way to satisfy the query.

3.1 The near predicate

The near predicate is used to encourage a group of servers to be selected between which low-latency communication is possible. For example, this might be used between a machine which is to provide back-end storage and a group of machines which are to provide front-end access. An optional maximum distance can be supplied if it is essential that the chosen servers are sufficiently close. Otherwise, the final ranking will rate selections with very close machines over those in which less proximity has been achieved.

3.2 The far predicate

The far predicate is used to encourage dispersion of a group of servers. By analogy with the near predicate, a minimum separation can be specified if necessary. Note that far may be used for two different purposes: first, it causes machines to be scattered through the network – to reduce mean distance from client to nearest machine for an unknown global workload. Second; we envisage many users employing it to provide protection against correlated failures, due to organizational disconnection, without collecting expensive large-scale uptime statistics.

We justify the latter experimentally by taking 30,000 (IP-address, network-location) pairs calculated, from the Skitter data set, and finding their Autonomous System (AS) number using RouteViews BGP information[1]. From this we selected ASs for which we had more than one location and then calculated the maximum distance between such locations in a common AS. Figure 3 plots the resulting distance as a fraction of the location space. From the graph we see that only 4.5% of ASs in our survey have a distance larger than a quarter of our location space. The results that AS number and location are well-correlated suggests that being far in the location space is a good indication of low correlated network failures.

The far predicate is not ideal for every situation, in a different scenario an application could require different semantics. The definition of far given here is likely to spread the far terms to the furthest edges of the network, this means machines in far places or on slow connections. It would instead be better to also require that they are close to the center of the location space. The behaviors of the far predicate can be modified to this behaviors through the use of a global query option.

3.3 Example

As a running example we use a simple query which contains only one disjunction. This is shown in Figure 2 along with a correspond-

[1]We discuss the Skitter data in more detail in Section 6 where we use it as the basis of further experiments. The RouteViews system is described at http://www.routeviews.org.

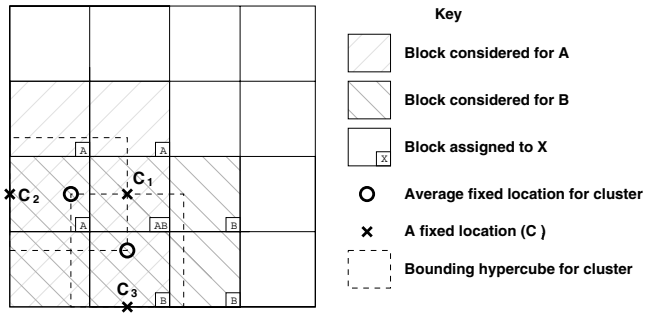

Figure 4: The output from stage one: coarse-grained placement of constrained clusters, to blocks in spatial index.

ing *graph* which summarizes it. Each of the nodes in the graph $(A \ldots D)$ is a *cluster*, which encodes a $\mathrm{near}(A_1, \ldots, A_n)$ relationship. Clusters can contain any number of *fixed locations* (e.g. the first two items in cluster A) and *servers* to be found (e.g. the last item in cluster A). If the cluster contains one or more fixed locations then it is called a *constrained cluster* otherwise a *free cluster*. The links between the various clusters are *far relationships*, encoding the $\mathrm{far}(A_1, \ldots, A_n)$ terms. The conjunction ($\wedge$) operator is encoded by simply combining the graphs that are produced in each of its branches into a single graph. Cluster D has no far links and contains one free server, S_5, indicating that the location of S_5 can be anywhere.

The algorithm tries to assign real *machines* to the free servers in a set of graphs, returning a ranked list of various possible configurations. The algorithm aims to minimize the distance between selected machines and fixed locations within clusters, while maximizing the distance between all clusters separated by $\mathrm{far}(\ldots)$ relationships.

4. THE HIGHER LEVEL ALGORITHM

In this section we introduce the algorithm used for answering queries of the form described in Section 3. The algorithm operates over a spatial index which leads naturally to a centralized implementation. We then detail, in Section 5, our scalable distributed implementation based on the same index structure. The algorithm extends the model from Section 2, using summary information from the index to optimize the simplification of the query and therefore improve the accuracy of the results from complex queries, even when XenoSearch is fully distributed.

Our algorithm consists of four stages, which are performed for each of the graphs produced from the query. Stages one and two deal with coarse-grained assignment of clusters to spatial regions, with stage one dealing with the constrained clusters and stage two the free clusters. These stages correspond to the "Query Simplification" box in Figure 1 and seek to maximize the distances between clusters with far relationships. These stages try to maximize the distance between clusters which are required to be far apart. The third stage deals with the detailed assignment of servers to machines, through an index-search. Finally stage four, seeking to minimize the distances within each cluster aggregates and optimizes the results ("Optimizer" from Figure 1).

4.1 Spatial Data Structure

The algorithm works over a data structure that decomposes the location space at a number of different *levels* into 2^{ld} identical *blocks* of equal hyper-volume, with l the level and d the dimensionality of the co-ordinate based location space. Level 0 covers the whole index, level 1 splits each of the dimensions into two equal halves, level 2 splits each of the resulting sections into half again on each dimension and so on. The output of the second stage of the algorithm, shown in Figure 6, also shows the decomposition of a two dimensional index.

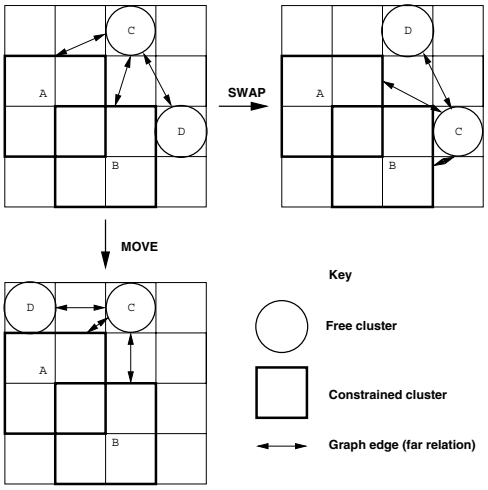

Figure 5: The two kinds of transformation that are used when placing the free clusters C and D around the constrained clusters A and B

As well as storing the machines' locations, the data structure can provide summary information about the number of blocks at each level that are non-empty. These summaries are used during the initial coarse-grained search in order to ensure that the selected blocks contain sufficient servers to match the query.

We have implemented this data model as a quad-tree-based data structure [9] for the centralized case and a gossip based (see Section 5) data structure for the distributed resource discovery system.

4.2 Detailed Algorithm Description

The algorithm proceeds in four stages, repeated for each of the graphs that make up the query.

4.2.1 First Stage

The first stage places the constrained clusters. Each cluster is mapped to a hyper-cube exactly large enough to contain all of the fixed locations within the cluster, and centered at their average. All blocks that intersect this hyper-cube are considered for assignment to the cluster; if a block has already been assigned to another cluster, then we assign it to both clusters iff it contains fixed locations from both. If it contains fixed locations from only one cluster, it is assigned only to that; otherwise the block is assigned to neither cluster. This is demonstrated in Figure 4.

4.2.2 Second Stage

In the second stage of the algorithm we perform *simulated annealing* [10] with the goal of placing free clusters so that they respect the far relationships in the graph. We start with an arbitrary placement and proceed by considering two possible kinds of transformation which can be applied at random in each round. These are shown in Figure 5.

A cost function is used which is the negative of the total distance of all the far relations; hence the annealing attempts to place clus-

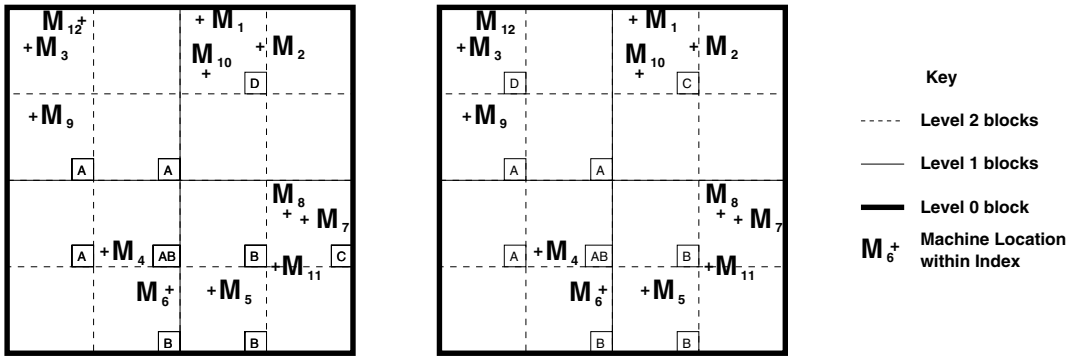

Figure 6: The output from stage two: coarse-grained placement of free clusters with two simulated annealing runs. Each of the two coarse-grained placements shows the blocks A, B, C, D from within which machines for those four clusters will be selected.

A: $S_1 \rightarrow [M_4,\ M_9]$ B: $S_2 \rightarrow [M_6,\ M_4,\ M_5]$
C: $S_3 \rightarrow [M_7,\ M_8,\ M_{11}]$ D: $S_5 \rightarrow [M_2,\ M_1,\ M_{10}]$
$S_4 \rightarrow [M_7,\ M_8,\ M_{11}]$

A: $S_1 \rightarrow [M_4,\ M_9]$ B: $S_2 \rightarrow [M_6,\ M_4,\ M_5]$
C: $S_3 \rightarrow [M_2,\ M_1,\ M_{10}]$ D: $S_5 \rightarrow [M_3,\ M_{12}]$
$S_4 \rightarrow [M_2,\ M_1,\ M_{10}]$

Stage 3: detailed placement

A: $\begin{bmatrix} S_1 \rightarrow M_4 \\ S_1 \rightarrow M_9 \end{bmatrix}$ B: $\begin{bmatrix} S_2 \rightarrow M_6 \\ S_2 \rightarrow M_4 \\ S_2 \rightarrow M_5 \end{bmatrix}$

C: $\begin{bmatrix} S_3 \rightarrow M_7,\ S_4 \rightarrow M_8 \\ S_3 \rightarrow M_8,\ S_4 \rightarrow M_{11} \end{bmatrix}$ D: $\begin{bmatrix} S_5 \rightarrow M_2 \\ S_5 \rightarrow M_1 \\ S_5 \rightarrow M_{10} \end{bmatrix}$

A: $\begin{bmatrix} S_1 \rightarrow M_4 \\ S_1 \rightarrow M_9 \end{bmatrix}$ B: $\begin{bmatrix} S_2 \rightarrow M_6 \\ S_2 \rightarrow M_4 \\ S_2 \rightarrow M_5 \end{bmatrix}$

C: $\begin{bmatrix} S_3 \rightarrow M_2,\ S_4 \rightarrow M_1 \\ S_3 \rightarrow M_1,\ S_4 \rightarrow M_{10} \end{bmatrix}$ D: $\begin{bmatrix} S_5 \rightarrow M_3 \\ S_5 \rightarrow M_{12} \end{bmatrix}$

Stage 4a: Form Server Clusters

$$\begin{bmatrix} S_1 \rightarrow M_4,\ S_2 \rightarrow M_6,\ S_3 \rightarrow M_7,\ S_4 \rightarrow M_8,\ S_5 \rightarrow M_2 \\ S_1 \rightarrow M_9,\ S_2 \rightarrow M_4,\ S_3 \rightarrow M_8,\ S_4 \rightarrow M_{11},\ S_5 \rightarrow M_1 \end{bmatrix}$$

Stage 4b: Form Search Results

Figure 7: The output from stage three and four: detailed Server placement, using the two coarse-grained placements from Figure 6

ters far apart. In fact the distance from the center of the space to all the clusters is added if the query specifies that clusters should be placed away from the edges of the space. As is normal for simulated annealing, if a change increases the cost function by δf then it is accepted with probability $e^{\left(-\frac{\delta f}{T}\right)}$; if it does not increase the cost then it is always accepted. We perform this step a number of times for each graph (fixing the total number of times across all the graphs) to obtain a number of candidate graph assignments and run stages three and four on each of these assignments. The output of stage two for our example is shown in Figure 6.

4.2.3 Third and Fourth Stages

These stages of the algorithm are concerned with mapping the servers to machines in the index. This is done one cluster at a time (in decreasing number of fixed servers per cluster), and initially involves an index-search for the whole cluster. We restrict the blocks searched to the ones assigned to that cluster by coarse-grained placement. This drastically cuts the running time compared with naively forming the cross product of the server-to-machine assignments.

If there are no results for a particular server a different graph assignment may give a result which satisfies all free clusters. In the case where constrained clusters are unsatisfied, a nearest-neighbor search can be used.

In the fourth stage we construct groups of machines for each cluster which that satisfy the query. Each group is stored as a *map*: a mapping from free servers (from the query) to actual machine advertisements in which no machine appears twice within a single group. Several of these groups are constructed by a simple scan through the list of candidate machines returned for each server. Each list is sorted according to distance from the set of cluster centers which have a far relationship with this cluster. These lists are then ranked according to the total distance between all the machines assigned to the cluster.

Finally, these maps are combined over all the clusters and ranked using the global ranking function and the required number re-

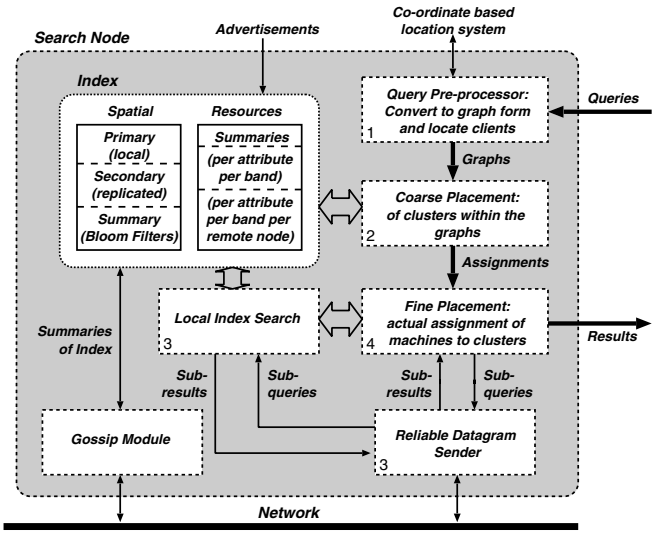

Figure 8: Architecture of a node.

turned. The function is based on that used in simulated annealing in Stage 2, but we add in the total distance of all the near relations and scale all the terms accordingly so comparisons can be made between graphs. In stage 4 other attributes are considered and the maps constructed obey these. These stages are demonstrated in Figure 7. There is the possibility of using a back-tracking depth-first search algorithm instead of the simple optimizer described here, if the constraints are very complex. Such an algorithm would consider more assignments, but experience from SWORD [5] suggests this would greatly increase the query time.

5. PEER-TO-PEER IMPLEMENTATION

The distributed version of our algorithm is implemented based on an index distributed over a set of mutually-trusting nodes. Gossiping techniques are employed to separate the maintenance and distribution of summaries from the implementation of the algorithm using them. Note that we do not consider issues of churn since we expect nodes to be well-provisioned machines remaining active over months rather than minutes. The basic architecture of a node is illustrated in Figure 8, the numbers indicating the algorithm stages from Section 4.2.

Nodes determine the network location [11] of the indexed machines, by the use of a co-ordinate location system. These co-ordinates are then mapped to points in a single-dimensional 128-bit key space by using a Hilbert curve, chosen as it gives better clustering than other space-filling curves [12]. As described in Section 4.1, the location space is partitioned into blocks, which each correspond to as single range in the transformed space. Furthermore, each continuous resource (e.g. CPU time, network bandwidth) is mapped to a separate 128-bit key space that is divided into a series of resource bands.

The nodes are also mapped to positions in the flattened location space, with each node acting as the primary *authoritative repository* for information about the machines occurring in blocks between it and its successor in the key space. A primary repository constructs a summary Bloom filter [13] for each of the blocks and for each of the resource bands for which it is responsible.

As demonstrated in Figure 9, a node also acts as a secondary *replication repository* for the blocks owned by the following k (the *replication factor*) nodes in the key space and holds summaries for the remaining blocks. Nodes acting as replication repositories for a block hold a (possibly delayed) version of the state about machines in that region in order to support fail-over from the primary.

To enable efficient searching, each node maintains and propagates the following state:

- Location Space: for each location space block, the network address of its primary repository and a Bloom filter summary of the machines within it.
- Resources: the per-node resource band summaries, which are aggregated at each node to give an overall summary for each resource band.

The summaries allow nodes to identify blocks in which there are likely to be machines that meet a combination of location and resource based constraints, from the intersection of the Bloom filters held for the block and any bands spanned by resource queries. Summary information is exchanged between nodes using gossip.

Performing the final machine assignment is the only search stage that requires communicating messages to other node(s) in the system in order to return the machines matching the query. This is done on a per cluster basis for efficiency, but is logically done per server. Bloom filters are passed with the query for efficient primary filtering. As the information is replicated, any of the primary or replication repositories can be queried to allow load balancing, and several may be queried in parallel to increase robustness and speed.

The communication substrate uses a reliable datagram service with TCP-style congestion-control built upon a gossiped routing table. Routing information and congestion control information is held for all the other nodes; this information is less than 100 bytes per node so with 10,000 nodes the routing table is still less than 1MB. We only expect hundreds of search nodes.

A further innovation in the routing substrate is for each node to record a "probably dead" flag along with its state for each other node. This flag is set when reliable datagrams are repeatedly unacknowledged and, when set, messages are sent to the node's first predecessor: this causes a node to fail over from using a primary repository to using a replication repository. If the predecessor also considers the node probably dead it will process the request as replication repository otherwise it will forward it on to the node. Only the predecessor of a node can make a real liveness decision and as NodeIds are location based, probing is mostly local and only done by the one node holding up to date gossip timestamps.

A node receiving a message that does not fall within its primary range will normally forward it on to the destination. That node can make sure the sender's routing table is updated, either to clear the probably dead flag or to add a new entry if it is a new node. If the final destination is dead the closest live predecessor will receive the message, realize its successor might be dead, and initiate failure recovery.

6. RESULTS

We evaluate the scalability and fidelity of our search system using three metrics: the time taken to respond to a query, the volume of network traffic generated by queries and the accuracy of query results when compared with a exhaustive search of all possible machine assignments.

We deployed our prototype implementation over 200 Planet-Lab nodes, as well as in local small-scale simulations; k (the replication factor, see Section 5) was set to 5 which gave us the best trade-off between traffic and resilience. The nodes were seeded

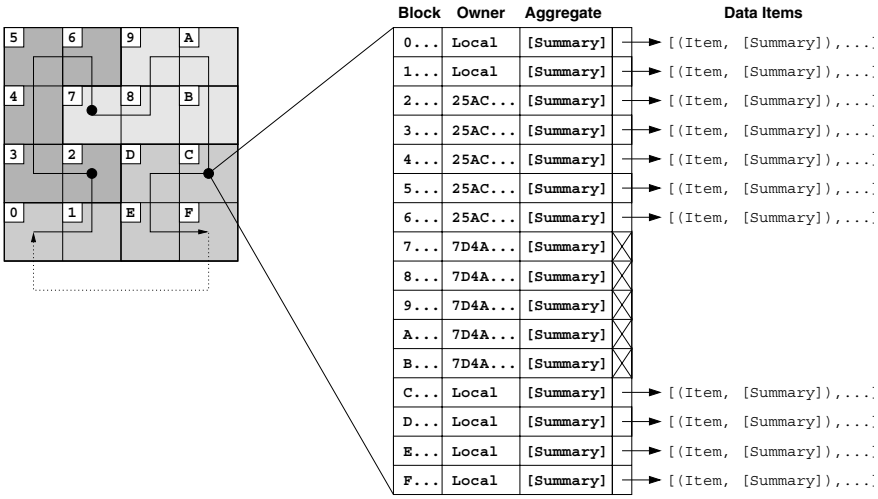

Figure 9: How the index decomposes the location space and replicates it at various nodes.

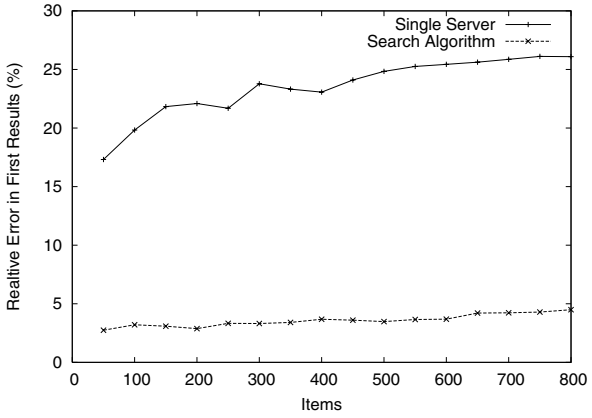

Figure 10: Relative of the heuristic algorithm and single server searches when compared with a exhaustive search.

with synthetic information about 10,000 potential distributed computing platform servers, which we generated by combining the network location of machines from the Skitter data set[2] and resource availability information from Planet Lab[3]. Ten types of query are preformed 100 times each. Each query searches for between one and ten servers, with each server being in a separate cluster.

6.1 Accuracy

Figure 10 compares the accuracy of the search algorithm against an 'optimal' assignment using exhaustive search and a user search through searching for each server in the query individually, for the top ranked result returned. In performing this experiment we seek to justify the use of our new algorithmic approach to resource discovery: using an enhanced query simplification stage.

The optimal assignment minimizes the sum of the distances between terms in near relationships, less the distances between terms in far relationships. The algorithm used to search individually for servers, uses the underlying index to search for each server in the query in turn, starting with the most constrained and proceeding to the less constrained servers, feeding back the locations of the servers already found. This models the traditional single server approach to resource discovery. To make the comparisons clear and only based on algorithm choice (not network characteristics) the centralized version of the XenoSearch algorithm is used.

To make exhaustive searches feasible a small data sets and query was used; the results are averaged over 20 different random selections of 800 items from the Skitter data set. For each set 30 different queries were tried, with equal numbers of one, two and three cluster queries. This gives 600 queries per point. The XenoSearch algorithm looks to constantly be about 5% worse than the ideal solution while searching for each server individually gives just over 25%. This shows that the use of a two stage assignment of free servers; first to blocks, then to a machine within that block does not significantly degrade accuracy. Both the XenoSearch and single server algorithms have a slight linear increase in query time with both the number of clusters in a query and machines indexed (omitted due to space considerations), but the single server is approximately 10 times faster in general (10ths seconds for single server and seconds for XenoSearch). Therefore the XenoSearch algorithm makes a sensible accuracy/time trade-off between the fast but inaccurate single server algorithm and the very accurate but very slow exhaustive algorithm.

6.2 Number of Resources Indexed

If our system is to be scalable in the number of items indexed, then the index structure should give a near constant query time as the number of resources indexed changes. Figure 11 demonstrates that the small linear scaling with the number of items seen in the centralized version is no longer present. This is because query times are dominated by the time of sending messages through the network. Again there is a slight linear increase in the query time with the number of clusters in the query.

In this experiment we look at the effect that varying the clusters in a query and the number of items has on the traffic generated by a query. We expect the network traffic to be constant with the number of indexed items and linear in number of clusters. We measure the total traffic generated by a query at the initiator node.

[2]The data used in this research was collected as part of CAIDA's skitter initiative, http://www.caida.org. Support for skitter is provided by DARPA, NSF and CAIDA membership.
[3]http://www.planet-lab.org/doc/Reference.php#ProcSensor

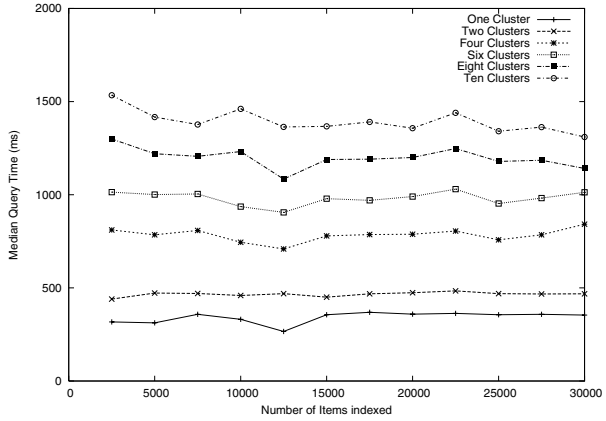

Figure 11: Mean query time for queries performed with 2500...30000 items in the index.

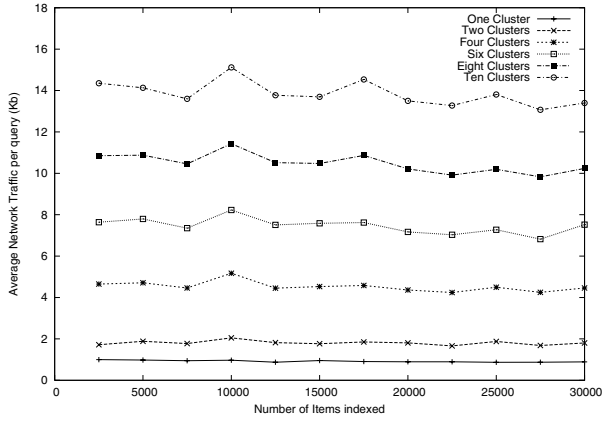

Figure 12: Mean network traffic queries performed with 2500...30000 items in the index.

The results (Figure 12) show that total query traffic does not scale with the number of items, although the amount of traffic is highly dependent on the number of clusters, starting at around 1Kb for single cluster queries and increasing to approximately 13-15Kb for ten clusters, the large variation being due to retries when Planet Lab is highly loaded. We believe these levels of traffic are acceptable for a complex resource discovery query, being of comparable size to a web page displaying the results. The gossip system produced between 5-7Kb/s of background network traffic at the initiator node during these experiments. The accuracy is not affected by the choice of algorithm implementation, therefore these results are not shown.

6.3 Number of nodes Participating

Our system was designed to give constant query time, across systems with a wide number of member nodes. This is achieved by the summary tables in which nodes retain outline information about the full key space. Figure 13 confirms this for our five example query classes. The system should continue to scale in this way so long as the churn rate is low compared with the rate at which summary information is exchanged by gossiping. The results show that queries take less than 2.2 seconds even in the worst case. There is a hint that small networks (< 20 nodes) will be faster. This is

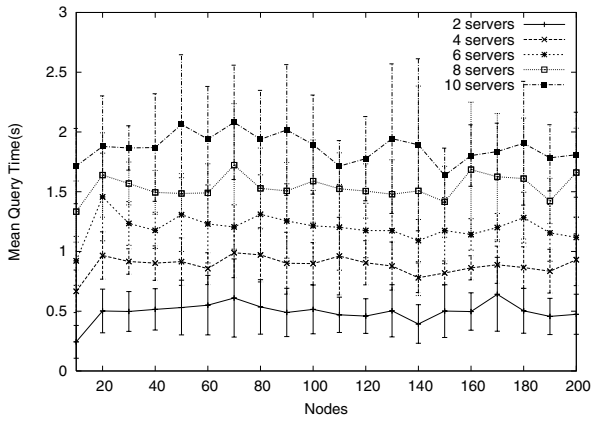

Figure 13: Mean query time for five example query classes performed using 10...200 nodes. Shown with standard deviation.

due to much more of the data being held at the initiating node as the replication factor is close to the number of nodes.

Since the query time is sensitive to external effects, we include error bars. These demonstrate that when the mean query time is high, so is the variance. In particular although minimum query time is relatively stable, we occasionally observe a high maximum query time (about twice the mean), hence we often use the median. These effects are due to high-load at the searching node and high congestion.

The number of clusters is the dominant factor in the time taken, with the total time scaling linearly and each new cluster adding less than a quarter of a second; this agrees with our earlier results. Furthermore, our current implementation trades off the actual query time against factors such as the volume of data transferred between nodes which we limit to prevent congestion.

Again the network traffic (results omitted for space) do not scale with the number of nodes and scale linearly with the number of clusters in the query.

6.4 Query Load

This section presents some results on how the system scales with query load, this is an important consideration in an open platform as there is un-limited number of users of the system. It also allows three set-ups to be compared; centralized, a pair of co-located servers (ie a server cluster) and a distributed system.

The following set up is used; C clients are deployed on different Planet Lab machines and issue 100 queries of a uniform type, in a back-to-back fashion. Each client issues queries to each of the N (again deployed on different Planet Lab machines) nodes under test in a cyclical fashion, randomly picking the node to send the first request to. The cluster pair systems use the most lightly-loaded pair of Planet Lab nodes at a common site.

Figure 14 compares the results for one cluster queries with one client and twenty-five clients. The twenty-five clients case has a minimum (around 5–10 nodes), which has shifted from 1 node in the monotonically increasing one client case. With twenty-five clients the 1 and 2 node systems have high query times, of up to 3 $1/4$ seconds, but even the five node system can exploit enough parallelism to give a query time similar to the one client case.

The query time for the centralized system is slightly lower than the one node distributed system, because of the extra efficiency of the centralized index. The cluster pair system has a query time of about half that of the centralized index, as the processing is shared

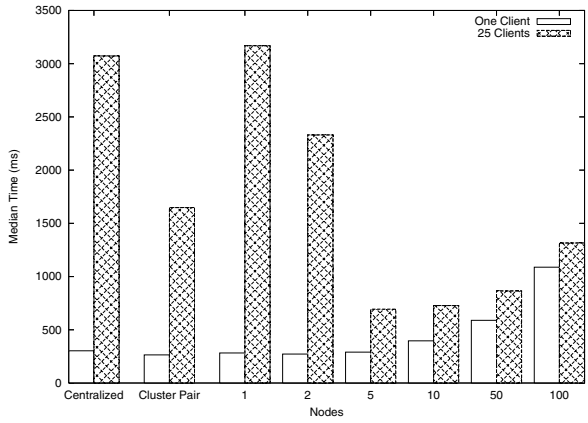

Figure 14: Comparing Query Time with 1 and 25 Clients, for various numbers of nodes (one cluster queries).

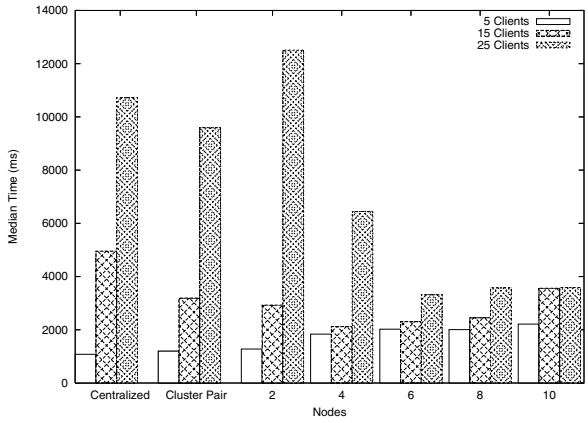

Figure 15: Comparing Query Time with 5, 15 and 25 Clients, for various numbers of nodes (two cluster queries).

between the two nodes. It is also 30% better than the two node distributed system, due to the effect of wide-area network delays. The performance of the twenty-five client system never reaches that of the one client system, as there is always some contention between queries, because the nodes contacted in a query are not disjoint.

These results point to significant amounts of parallelism between concurrent queries, even for a single node system. For example in the one node case the average time for a query with one client is 282ms and for twenty-five clients 3168ms. Naively we might expect the twenty-five client case to be twenty-five times the one client case $282 \times 25 = 7050$ms, but this is well over twice the measured value, suggesting that over 50% of each query is parallelizable. In a large part this is due to TCP connections being set up in parallel. This is different to the parallelism leveraged in the query due to distributing the query processing over many nodes.

We investigate systems with between two and ten nodes more closely to see how the changing query load affects the number of nodes responsible for minimal query time. This is shown in Figure 15, for five, fifteen and twenty-five clients, using results for two cluster queries as they show the results clearly. As expected, the minimum query time does constantly shift, being higher for more clients; for five clients it seems to be around two nodes in a cluster, for fifteen about four nodes and twenty-five around six nodes. In addition the minimum seems to be shallower as the number of clients increases. A third trend is that more clients means a longer query time, even when comparing optimal numbers of nodes in the system, there is an overall trend for slightly higher query times as the number of clients increases.

Figure 15 has a number of features that differ from those in Figure 14. For example, with twenty-five clients the centralized case is not twice that of the cluster-pair. The anomalies are due to search nodes becoming saturated and no longer able to service the query load. In many cases this actually causes the median time to drop as the clients finish at different times and some clients execute some of their queries on a network with a lower query load. This affects the two-node, cluster pair and centralized cases with 25 clients. Further the cluster pair and centralized cases with 15 clients is nearly saturated.

Similar results were also seen as the complexity of queries are increased; as the number of clusters in a query is increased the optimal number of nodes for the distributed set-up increases, for example see Figure 16.

7. RELATED WORK

A wide variety of resource discovery systems have been proposed; encompassing a range of problems beyond those addressed in this paper. Here we classify related work according to the style of query it supports and the implementation used to evaluate queries.

7.1 Centralized and Hierarchical Information Systems

Historically resource discovery has been built around searches through a directory for a single object. Systems may be centralized (e.g. SLP [14]) or hierarchical (e.g. SSDS [15] and DNS [16]) or anycast [17], but still focus on locating only a single server at a time. There has also been considerable work on locating a single 'nearby' server: a selection of approaches are compared by Guyton and Schwartz [18]. In their work on Astrolabe, van Renesse *et al.* use Bloom filters combined hierarchically to summarize interest in publish/subscribe groups [19], providing wide-area hierarchical resource discovery. SDIMS [20] this with DHT style routing and Willow [21] also combines this with publish/subscribe.

7.2 Bi-lateral Matching

Recent work in the grid community has focused on extending the kinds of queries to include bi-lateral matching — allowing both the provider and the consumer to provide descriptive attributes, constraints on the other party or parties and a ranking function. This was pioneered in the Condor matchmaker system [22] which allows single resources to be matched and extended by Gang-Matching [23] to provide multi-resource matching and by Redline [24] to allow set-matching where constraints could also be aggregate over the resources returned. Further work on ontology-based resource matching uses semantic web technology to break the need for all providers and consumers to agree on the names and values of attributes.

7.3 Peer-to-Peer Range Searching

There have been a number of resource discovery systems supporting range-based queries above peer-to-peer networks. The XenoSearch-I system [25] allows a single server to be selected on the basis of attributes such as CPU load, or location within a range of network co-ordinates. [26] is similar work looking at performing multi-dimensional searches using a Hilbert space map to associate servers with locations in a Chord [27] network. Both do not suggest how a series of inter-related servers would be chosen. The Prefix

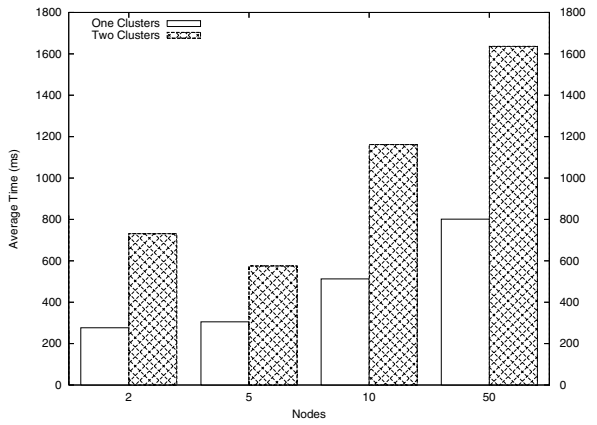

Figure 16: Comparing Query Time with 1 and 2 clusters, for various numbers of nodes (with one client).

Hash Table (PHT) [28] is built upon a Distributed Hash Table and provides a mechanism for building an maintaining a distributed trie on which ranges searches are performed.

SWORD [5] provides resource discovery for Planet Lab. Clients initiate queries by contacting nodes in a distributed hash table; an optimizer on the chosen node tries to find the best set of machines to satisfy a query. SWORD relies on range searches so cannot efficiently search for inter-related servers in the location space. SWORD also requires the user to form their own clusters of servers, our system performs this automatically. The Mercury [6] resource discovery system uses structured peer-to-peer networks called hubs to manage attributes, these are liked by a few inter-hub links. Mercury is used to provide a substrate for a publish-subscribe system for distributed multi-player game objects.

7.4 Spatial Databases

Traditional spatial databases support location based queries using multi-dimensional indexing schemes using trees e.g. [9, 29] or grid-like partitioning e.g. [30]. The VA file [31] in particular is very similar to the data-structure from our scheme, approximating multidimensional indexes by splitting it into 2^b cells, assigning each a unique bit-string and creating a list of these approximations. For a wider survey see [32].

7.5 Peer-to-Peer Databases

Peer-to-peer distribution techniques have also been applied in the database community; work at UCSB [33, 34] has centered on providing caches for range based database searches, assuming static, non-distributed data, whereas our scheme looks to serve a situation in which the data is dynamic. Iris [35] and Sophia [36] provide a more dynamic database like system for querying wide-area sensor network.

Pier [37] seeks to service databases queries over a Peer-to-Peer network, but this system only supports a limited number of query types. Using CAN, Harren *et al.* investigated supporting 'join' operators by the temporary creation of new DHT name-space [38]. Further efforts to apply Peer-to-Peer distribution to databases include [39, 40] all forsaking some of the strict database invariants to allow efficient wide-area distributed databases.

8. CONCLUSION

This paper has presented XenoSearch, a new distributed resource discovery system which provides a novel query language able to express complex inter-related location-based queries. It also demonstrated how to perform searches for resources that match such queries and we have evaluated the performance of the resulting system in a wide range of settings on a wide-area deployment.

XenoSearch has been shown to provide a good trade-off between the exhaustive search (with perfect accuracy but infeasible query time) and the use of single resource algorithms (with fast query times but low accuracy). This justifies the use our new approach of incorporating complex query simplification to improve query accuracy with complex queries. In addition this algorithm still retains good scalability: the performance of the distributed XenoSearch algorithm has been shown in general not to be affected by the number of items in the index or number of search nodes.

Therefore XenoSearch possesses all the features required of a resource discovery system: servicing location-based queries for placing whole distributed systems in a scalable fashion.

The results presented here suggest that the optimal number of nodes to provision for the system must take into account query load. Although there is a clear network size that produces the fastest query time, there is still a penalty for higher loads. Further work to consider is measuring the effect of other system parameters on the optimal number of nodes for the query time metric (or any other metric). It is likely that these results are indicative of other peer-to-peer systems and therefore given a better model of optimal performance, monitoring software could be developed which continually measures a Peer-to-Peer system and uses the model to dynamically provision (over long time-scales to avoid churn) the optimal system size.

In addition XenoSearch provides the equivalent of a lightweight distributed spatial database, giving rise to further work to investigate other applications which could relax some of the strict semantics of databases and take advantage of the service.

Acknowledgments

We are grateful to the States of Jersey Education Department for funding the work of David Spence.

The data used in this research was collected as part of CAIDA's skitter initiative, http://www.caida.org. Support for skitter is provided by DARPA, NSF and CAIDA membership.

9. REFERENCES

[1] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proc. 1st ACM HotNets Workshop*, October 2002.

[2] S. Hand, T. Harris, E. Kotsovinos, and I. Pratt. Controlling the XenoServer Open Platform. In *Proc. 6th Int. Conf. Open Arch. and Network Programming (OPENARCH)*, April 2003.

[3] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The Int. Journal of Supercomputer Application and High Performance Computing*, 11(2):115–128, 1997.

[4] F. Kelly. Models for a self-managed internet. *Philisohical Transactions: Mathematical, Physical and Engineering Sciences, The Royal Society, London*, 358(1773):2335–2348, August 2000.

[5] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable wide-area resource discovery. Technical Report UCB//CSD-04-1334, UC Berkeley, July 2004.

[6] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proc. ACM SIGCOMM*, August 2004.

[7] A. Andrzejak and Z. Xu. Scalable, Efficient Range Queries for Grid Information Services. Technical Report HPL-2002-209, Hewlett-Packard Laboratories, Palo Alto, 2002.

[8] A. Iamnitchi and I. Foster. On Fully decentralized Resource Discovery in Grid Environments. In *Proc. Int. Workshop on Grid Computing*, November 2001.

[9] G. Hjaltason and H. Samet. Speeding up construction of quadtrees for spatial indexing. Technical Report TR-4033, Computer Science Department, University of Maryland, July 1999.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[11] David Spence. An implementation of a coordinate based location system. Technical Report UCAM-CL-TR-576, University of Cambridge, Computer Laboratory, November 2003.

[12] B. Moon, H. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *Knowledge and Data Engineering*, 13(1):124–141, 2001.

[13] B. H. Bloom. Space/Time Tradeoffs in Hash Coding with Allowable Errors. *Comms. of the ACM*, 13(7):422–426, July 1970.

[14] J. Veizades, E. Guttman, C. Perkins, and S. Kaplin. Service location protocal, June 1997. Request for Comments RFC 2165.

[15] T. Hodes, S. Czerwinski, B. Zhao, A. Joseph, and R. Katz. An Architecture for Secure Service Discovery Service. *Wireless Networks*, 8(2–3):213–230, March 2002.

[16] P. Mockapetris. Domain names – concepts and facilities, November 1987. RFC 1034.

[17] C. Partridge, T. Mendez, and W. Milliken. Host Anycasting Service, November 1993. RFC1546.

[18] J. Guyton and M. Schwartz. Locating nearby copies of replicated internet servers. In *Proc. ACM SIGCOMM*, August 1995.

[19] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. 21(2):164–206, May 2003. ACM Trans. Comp. Sys.

[20] Praveen Yalagandula and Mike Dahlin. A scalable distributed information management system. In *Proc. ACM SIGCOMM*, August 2004.

[21] R. van Renesse and A. Bozdog. Willow: Dht, aggregation, and publish/subscribe in one protocol. In *Proc. 3rd Int. Workshop on Peer-to-Peer Systems (IPTPS 2004)*, February 2004.

[22] N. Coleman, R. Raman, M. Livny, and M. Solomon. Distributed policy management and comprehension with classified advertisements. Technical Report TR 1481, University of Wisconsin, Computer Science Department, April 2003.

[23] R. Raman, M. Livny, and M. Solomon. Policy driven hetrogeneous resource co-allocation with gangmatching. In *Proc. 12th IEEE Symp. High Perf. Dist. Comp.*, June 2003.

[24] C. Liu and I. Foster. A constraint language approach to grid resource selection. Technical Report TR-2003-07, Department of Computer Science, University of Chicago, March 2003.

[25] D. Spence and T. Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *Proc. 12th IEEE Symp. High Perf. Dist. Comp.*, June 2003.

[26] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *Proc. 12th IEEE Symp. High Perf. Dist. Comp.*, June 2003.

[27] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM*, August 2001.

[28] S. Ratnasamy, J. Hellerstein, and S. Shenker. Range queries over DHTs. Technical Report IRB-TR-03-009, Intel Research Berkeley, June 2003.

[29] K. Lum Cheung and A. Wai-Chee Fu. Enhanced nearest neighbour search on the R-tree. *SIGMOD Record*, 27(3):16–21, 1998.

[30] J. Nievergelt, H. Hinterberger, and K. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Sys. (TODS)*, 9(1):38–71, 1984.

[31] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. 24th VLDB*, August 1998.

[32] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[33] A. Gupta, D. Agrawal, and A. El Abbadi. Approximate range selection queries in Peer-to-Peer systems. Technical Report UCSB-CS-2002-23, Department of Computer Science, University of California at Santa Barbara, October 2002.

[34] O.D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. Query processing over Peer-To-Peer data sharing systems. Technical Report UCSB-CS-2002-28, Department of Computer Science, UC Santa Barbara, October 2002.

[35] A. Deshpande, S. Nath, P. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proc. SIGMOD*, 2003.

[36] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. Sophia: An Information Plane for Networked Systems. In *Proc. 2nd Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[37] R. Huebsch, J. Hellerstein, N. Lanham, B. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proc. VLDB*, 2003.

[38] M. Harren, J. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex queries in DHT-based peer-to-peer networks. *Lecture Notes in Computer Science*, 2429:242–250, March 2002.

[39] M. Annamalai. Designing an Efficient Distributed Digital Library Database for Image Data, 1997.

[40] M. Stonebraker, P. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1):48–63, 1996.