# Location-Based Resource Co-Allocation

David Spence and Tim Harris

University of Cambridge Computer Laboratory
{firstname.lastname}@cl.cam.ac.uk

**Abstract.** As computers become faster and networks' bandwidth becomes higher it is increasingly important to consider the latency between nodes when selecting where to deploy an application in a distributed computing system.

This paper describes a new algorithm for performing server selection for complex multi-node tasks. Our major focus is on selecting several servers simultaneously, with the location of the servers being constrained inter-dependently, for example '$\mathrm{far}(\mathrm{near}(S_1,\ S_2),\ \mathrm{near}(S_3,\ S_4),\ C_1)$' to request server $S_1$ near to server $S_2$, server $S_3$ near to $S_4$ and both pairs of chosen servers far from one another and far from client $C_1$. Our system is built over a database of server locations determined by a network location algorithm such as GNP.

We present results showing that the algorithm performs more accurate and faster searches than a user could hope to achieve by selecting one server at a time. We also show that the algorithm is many times faster than a brute-force search for the best possible solution, without a drop in accuracy. Further analysis is presented regarding the best parameter settings for the heuristic algorithm, and investigating the trade-offs between the dimensionality of the co-ordinate based location space, the accuracy of the query results and query processing time.

## 1 Introduction

In the XenoServers [2, 14] project we are building a public infrastructure for wide-area distributed computing, creating a world in which XenoServer execution platforms are scattered across the globe at well-connected points in the network and available for use by any organization and by any member of the public. The platform will host a diverse range of networked tasks, but underpinning all of these is the expectation that *network location* is the key factor in deciding which XenoServer to use. This is because selecting a suitable location allows the user to minimize the latency between the service that they deploy and the systems with which it interacts: as computers become faster and network bandwidths grow higher then latency will come come to dominate the performance of distributed systems.

In a previous paper we introduced XenoSearch [29], a distributed system for selecting a server that is close to a specified point in the network: XenoSearch uses a network location service [30] to map servers to points in a multi-dimensional space and then distributes parts of this space between a number of *XenoSearch* nodes for load-balancing and replication. XenoSearch is sufficient to deal with deploying simple services in which only a single server needs to be selected – for instance placing an interactive game server at a point which is equidistant between the players, or placing a proxy for a mobile device close to its current point of attachment.

In this paper we look at the more complex problem of locating groups of machines to host jobs which require multiple XenoServers on which to operate, such as hosting a set of back-end and front-end components in a distributed web service. Network location remains the key requirement in selecting the servers – for instance, in this example, choosing widely distributed machines to host replicas of a web site.

We proceed, in Section 2, to introduce our query language. In outline, the user provides an expression such as '$\mathrm{far}(\mathrm{near}(S_1,\, S_2),\, \mathrm{near}(S_3,\, S_4),\, C_1)$' to indicate that four servers are to be found ($S_1$–$S_4$), that $S_1$ should be located close to $S_2$, $S_3$ close to $S_4$ and that these two pairs should be far from one another and also far from a specified client $C_1$. These two primitives, '$\mathrm{near}$' and '$\mathrm{far}$' aim to capture many of the requirements which arise when deploying distributed systems: '$\mathrm{near}$' terms are used to cause clustering (e.g. for low-latency communication), whereas '$\mathrm{far}$' terms are used to encourage dispersion (e.g. to make replica-failures independent).

In Section 3 we present a new algorithm for selecting servers which match queries. Rather than returning a single binding from $S_1$–$S_4$ to actual servers we return a ranked set of bindings suggesting a number of possibilities. As with Internet search engines, this allows the user to select from the list, or to explore the options presented according to some other metric – for instance, the cost that the server operators would charge.

In Section 4 we assess the performance of the new algorithm. We compare it against two alternatives. The first is a brute-force approach which performs an exhaustive search of all possible bindings – although impracticable, this lets us compare the performance of our algorithm against an 'optimal' server selection. Our second comparison is with a server-by-server approach, representing the best that a client could achieve by making a series of invocations on our original single-server-selection XenoSearch system. Our results show that (i) we can achieve search qualities comparable with the brute-force approach while using only a fraction of the computational time, and that (ii) results obtained by the server-by-server approach are substantially poorer, confirming the need for co-allocation of multiple servers.

Section 5 details related work drawn from the Grid computing, resource discovery and spatial databases communities, highlighting in particular the differences between the kinds of query that our language can express and the kinds of query that can be performed using extensions to SQL. The key difference is that existing spatial query languages tend to search for data which meets boolean criteria, such as "find two points within 10 units of one another", whereas our system tries to capture the user's goals at a higher level and to allow effective ranking of results.

Finally, Section 6 concludes with an implementation status report and describes our plans for future work in this area.

## 2 Query Language

Here we define our query language for the specification of spatial queries. Location requirements are defined recursively using the primitives of disjunction ($\vee$), conjunction

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

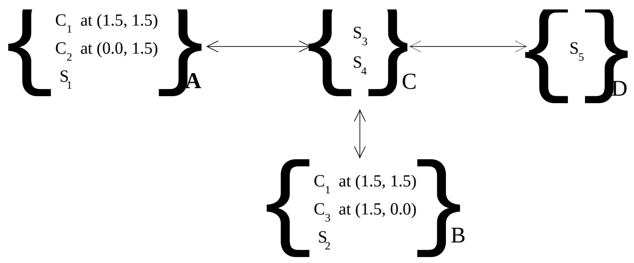

**Fig. 1.** An example query, used for our running example, and the graph generated from it. In the example, client $C_1$ is located at $(1.5, 1.5)$, $C_2$ at $(0.0, 1.5)$ and $C_3$ at $(1.5, 0.0)$.

($\wedge$), proximity ($\mathrm{near}(A_1, \ldots,\ A_n)$), distribution ($\mathrm{far}(A_1, \ldots,\ A_n)$) and terms representing fixed locations (e.g. clients' positions in the network – $C_i$) and free servers to locate ($S_i$). These queries are preprocessed before the evaluation by our algorithm into a disjunctive normal form, with each disjunction giving an alternative way to satisfy the original query.

As a running example we use a simple query which gives only one disjunction. This is shown in Figure 1 along with a corresponding *graph* which summaries it. Each of the nodes in the graph ($A \ldots\ D$) is known as a *cluster*, these encode the $\mathrm{near}(A_1, \ldots,\ A_n)$ terms. Clusters can contain any number of *fixed locations* (e.g. the first two items in cluster A) and *servers* to be found (e.g. last item in cluster A). If the cluster contains 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.

The algorithm takes a set of these graphs and tries to assign real *machines* from its index to the free servers, returning a ranked list of various possible configurations. The algorithm aims (i) to minimize the distance between the selected machines and the fixed locations in each cluster, while (ii) maximizing the distance between each pair of clusters separated by a $\mathrm{far}(\ldots)$ relationship. We return to the exact nature of the ranking function used in the next section.

In this paper we concentrate on location-based server selection and therefore exclude the discussion of how additional conditions on machine attributes are handled – this can be incorporated by simply filtering the set of machines considered for assignment. We refer the interested reader to our exposition in XenoSearch [29].

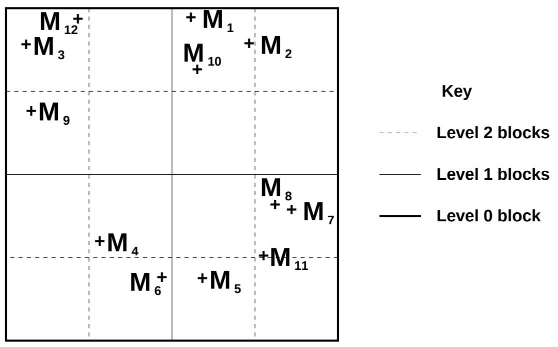

**Fig. 2.** The example two-dimensional index used in our examples

## 3 The Algorithm

The algorithm consists of three stages, which are performed for each graph. 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 try to maximize the distance between clusters which are supposed to be far apart. The third stage deals with the detailed assignment of servers to machines and seeks to minimize the distances within each cluster.

### 3.1 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. For instance, the current implementation uses a quad-tree-based data structure [16] to store the locations of machines within this space. Level 0 summarizes the whole index, level 1 splits each of the dimensions into two equal halves, level 2 splits each of the dimensions into half again and so on. An example of a two-dimensional index is shown in Figure 2.

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.

The data structure provides two operations for selecting individual servers: `singleServerNear` and `singleServerFar`. These take as arguments the number of servers required, a list of blocks to search and a list of any points we wish the servers to be near to or far from. These methods are used in the final stage to create the assignments from servers to machines.

### 3.2 Algorithm Parameters

There are two main parameters which control the results returned by a search. The first of these, **totalTimes**, sets the number of separate coarse-grained placements to consider.

$$\mathrm{cost}(C,\ FR) = \left( \frac{\sum\limits_{k \in C} \sum\limits_{\substack{i,j \in k, \\ i \neq j}} \mathrm{distance}(i,\ j)}{\sum\limits_{k \in C} |k|(|k| - 1)} \right) + \left( \sqrt{d} - \frac{\sum\limits_{(i,\ j) \in FR} \mathrm{distance}\left( \frac{\sum_{x \in i} x}{|i|},\ \frac{\sum_{x \in j} x}{|j|} \right)}{|FR|} \right)$$

**Fig. 3.** The Cost Function: C is a set of clusters (represented as sets containing fixed locations and servers) and FR is the set of far relations, between the clusters in C

Increasing this parameter leads to more possible server assignments being tested at the expense of a longer running time. The second parameter, **numberPerNearGroup**, sets the maximum number of machines to consider when performing fine-grained placement within each cluster.

The cost function we use to rank the possible assignments is shown in Figure 3. This aims to give a high cost to assignments in which machines in the same cluster cannot be placed close together, or in which clusters separated by a far relationship cannot actually be placed far apart.

The first term calculates the mean distance between the contents of each cluster. The second term calculates the mean distance between each pair of clusters which are supposed to be far apart. The scaling factors, such as $|FR|$, allow costs to be compared between different graphs, independently of the number of clusters or far relationships involved. The quantity $\sqrt{d}$ is the maximum distance possible between two points if all co-ordinates are normalized to the range $[0 \ldots 1]$. The function $\mathrm{distance}()$ calculates the distance between two points (fixed locations or machines satisfying servers).

### 3.3 First Stage

The algorithm proceeds in three stages, repeated for each of the graphs that make up the query. The first stage places the constrained clusters. For each constrained clusters we find a hyper-cube such that its center is the average point of all the fixed locations in the cluster and it is exactly large enough to contain all the fixed locations in that cluster. The blocks that intersect this hyper-cube are considered for assignment to the cluster. Sometimes one of these blocks will have already been assigned to another cluster; if that block contains fixed locations from both clusters then we must assign the block to both clusters. However if it only contains fixed locations from one of the clusters we assign the block to that cluster and otherwise we assign the block to neither cluster. This is demonstrated in Figure 4.

### 3.4 Second Stage

In the second stage of the algorithm we perform *simulated annealing* [20] with the goal of minimizing the cost function shown in Figure 3 (estimating server positions, at this stage, by the center of the block within which they may be placed). At all times, both

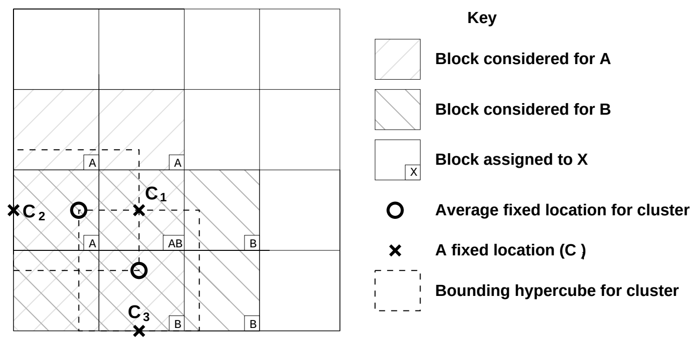

**Fig. 4.** The output from stage 1: coarse-grained placement of constrained clusters

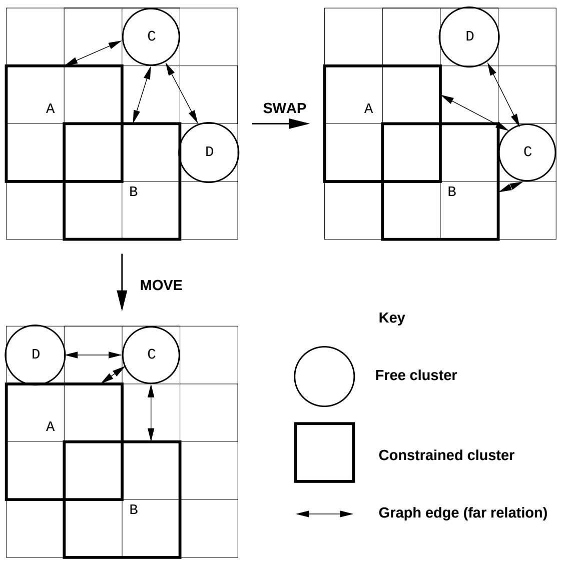

**Fig. 5.** The two kinds of transformation that are used when placing the free clusters C and D around the constrained clusters A and B

in the initial arbitrary placement and the perturbation steps, we cannot assign a cluster to a block also assigned to a cluster to which it has a far relation, or to a block with no machines in the index.

The two transformations which are applied at random in each round are shown in Figure 5. As is normal for simulated annealing, if a change increases the cost function by $\delta f$ then it is accepted with probability $e^{\left(-\frac{\delta f}{T}\right)}$, if it does not increase it then it is always accepted. We perform this step a number of times for each graph to obtain a number of graph assignments and run stage three on each of these assignments. We do a

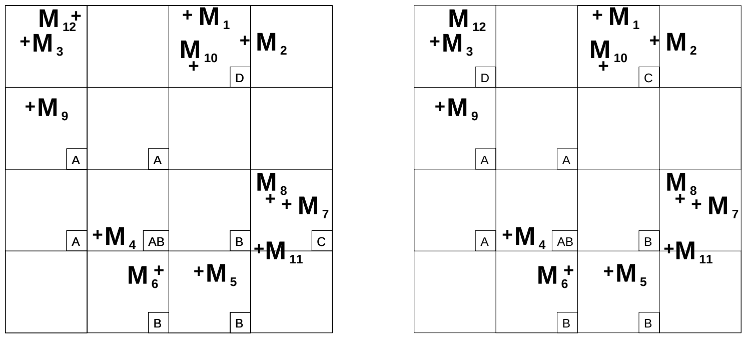

**Fig. 6.** The output from stage 2: coarse-grained placement of free clusters with two simulated annealing runs. Each of the two coarse-grained placements shows the blocks A, B, C, D from within which machines for those four clusters will be selected.

Stage 3, step 1: Call `singleServerNear/Far` for detailed placement

| | |
|---|---|
| A: $S_1 \rightarrow [M_4,\ M_9]$   B: $S_2 \rightarrow [M_6,\ M_4,\ M_5]$ | 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: $S_5 \rightarrow [M_2,\ M_1,\ M_{10}]$ | C: $S_3 \rightarrow [M_2,\ M_1,\ M_{10}]$ D: $S_5 \rightarrow [M_3,\ M_{12}]$ |
| $S_4 \rightarrow [M_7,\ M_8,\ M_{11}]$ | $S_4 \rightarrow [M_2,\ M_1,\ M_{10}]$ |

Stage 3, step 2: Form Server Clusters

A: $\begin{bmatrix} S_1 \rightarrow M_4 \\ S_1 \rightarrow M_9 \end{bmatrix}$ B: $\begin{bmatrix} S_2 \rightarrow M_6 \\ S_2 \rightarrow M_4 \\ S_2 \rightarrow M_5 \end{bmatrix}$ | 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: $\begin{bmatrix} S_5 \rightarrow M_2 \\ S_5 \rightarrow M_1 \\ S_5 \rightarrow M_{10} \end{bmatrix}$ | 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 3, step 3: Form Search Results

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

**Fig. 7.** The output from stage 3: detailed Server placement, using the two coarse-grained placements from Figure 6

total of $totalTimes$ of graph assignments spread across the different sub-queries to obtain several (hopefully very different) graph assignments. The output, for our example, of stage 2 is shown in Figure 6.

## 3.5 Third Stage

The third stage of the algorithm is concerned with mapping the servers to machines in the index. This is done one cluster at a time (in decreasing number of fixed servers in the cluster) and initially involves one call to `singleServerNear` or `singleServer-Far` for the whole cluster. We restrict the blocks searched to the ones assigned to that cluster by coarse-grained placement. As well as drastically cutting the running time by never forming the cross product of the server-to-machine assignments, this design returns a wide variety of alternatives.

We use `singleServerNear` for constrained clusters (supplying the co-ordinates of the fixed locations) and `singleServerFar` for free clusters (supplying the centers of all the other clusters with a far relation to the current one). If there are no results for a particular server then we hope that a different graph assignment will give a result for free clusters, for constrained clusters a nearest-neighbor search can be used.

For each cluster we then make maps from servers to machines. The maps are simply made by making a map of the highest ranked machine, then the second highest and so on. These are then ranked on total intra-near set distance, making sure we have no duplicate machines in a map.

Finally, these lists are combined in the same way over all the clusters and ranked using the global cost function, returning the required number of maps. The third algorithm stage is demonstrated in Figure 7.

## 4 Results

This section presents some initial results from this algorithm, using the simplified quad-tree data-representation described in Section 3.1. The data-set used for realistic distribution of servers is the GNP [21] data-set holding the location of 869 internet hosts calculated using the GNP network location algorithm using 19 tracers, with up to 18 dimensions.

We compare our algorithm to two other algorithms: the *brute-force* algorithm, tries every possible assignment, using the “Ranking Function” described below and so returns the best possible assignments. The *single-server* algorithm attempts to emulate how a user would best possibly use the `singleServerNear/Far` calls and so compares our server co-allocation algorithm to a server-at-a-time approach.

For test purposes we generate a series of example queries, parameterised by a number of servers to include and a “width” (number of top level near/far terms) of the query. At the top-level of the query is either a conjunction or a disjunction with two conjunctions below. The number of terms in this logical expression, either near terms or far terms, is equal to width. Far terms have either a fixed term and one or two free servers or near terms or just two or three free servers or near terms. Near terms are similar but only contain fixed terms and servers. A fixed term is a disjunction of one or two fixed locations. Where we do not give a single query type, the generator picks uniformly parameters in the range of widths 2 to 4 and servers 1 to 3.

We also define a ranking function, used to compare the algorithms, which given an assignment of servers to machines and the original query (specified in the first form given in Section 2 so as to also test the pre-processing stage), gives the total distance scaled to the range $[0,\ 1)$. The function works over the structure of the original query, conjunctions becoming the minimum operator, disjunction the sum operator, near the

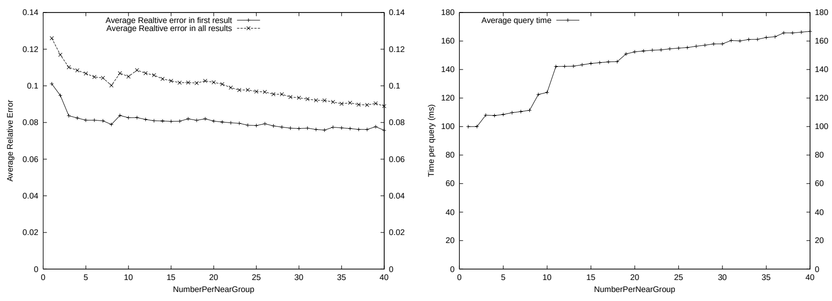

**Fig. 8.** How changing the numberPerNearGroup parameter affects the accuracy and running time

sum plus the distance between each pair of branches' average position and far the sum minus the distance between each pair of branches' average position.

There are many environments and parameters we may care to test our algorithm with, these include the actual number of machines in the index (for example the number of known XenoServers), the dimensionality of the index space, the form of the queries; particularly the number of servers (free variables) in the query and its exact shape and the parameters of the algorithm; especially `numberPerNearGroup` and `totalTimes`. Each of these we seek to experiment in these results.

Unless otherwise stated we use an index with 200 items in it and average the results over 1000 different queries generated by the query generator (mixed queries as described above). The dimensionality of the index space is three dimensions, `numberPerNearGroup` is set to 30 and `totalTimes` to 10.

## 4.1 Algorithm Parameters

Figure 8 shows how the performance and running time of the heuristic algorithm varies with the `numberPerNearGroup` parameter. Using the distance function above we compare both the distance for the highest-ranked result ("first") and the average over all of the returned results ("all") to the same query using the brute-force algorithm. The results suggest that increasing the `numberPerNearGroup` parameter reduces the gap between the heuristic and ideal algorithm, with diminishing results, at the expense of the processing time taken.

We also looked at how changing the `totalTimes` parameter affects algorithm performance. As expected there is a direct linear relationship between it and processing time and increasing `totalTimes` gives small drop in in the relative error.

## 4.2 Query Type

The next area we investigated was how the type of query affected the results. To achieve this we tried queries with one to three servers and with query widths from one to five. For each of these we processed 100 queries in an index of 100 items, this is different than in our other experiments as we needed to cut them down to run in an acceptable time . These parameters had to be chosen carefully due to the $O(N^s)$ (s = number of (free) servers in the query) nature of the brute-force algorithm, that we use to compare to.

The graphs in Figure 9 present the results, the first graph demonstrates that the “first” results are largely constant in their relative error, whereas the “all” results are widely differing.

Where there are a low number of servers the “all” results are better than the “first” results. This is because we are searching for few servers, we only have one far or near. The placement of this will be quite simple, so each of the `totalTimes` attempts will choose the same assignment of near groups. This means that the algorithm will work much more like the brute-force algorithm. Normally the heuristic algorithm tries to produce many varying solutions, whereas the brute-force always picks the most optimal, but these tend to differ only slightly. In this case the assignment of near groups to the same place means that we will have the same behaviour in both, so we expect the “all” results to be good.

For high number of servers and low width we have very constrained queries, with widths of one and two with three servers. These are inherently hard to process, but at the same time will often allow many varied solutions of similar distance, so in this case we have high average relative error in “all”. Finally for high number of servers and high width we have queries of medium constraint and so we have results that on average only vary by around 10% from the ideal results.

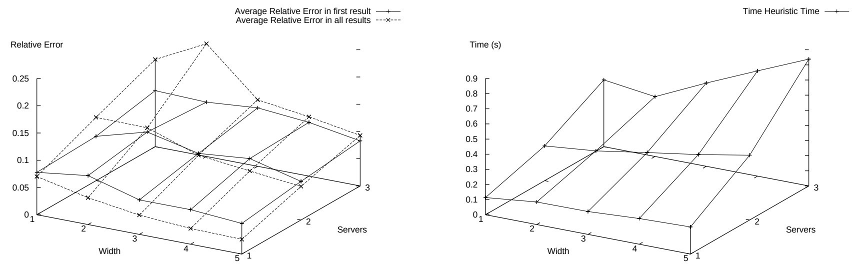

**Fig. 9.** How the error and time per query are affected by the query type

The second graph in Figure 9 looks at how the running time of the new algorithm is affected by the query type. The heuristic algorithm has linear complexity in both the width of query and the number of servers. The value for three servers in a single width query is anomalous – these are very hard queries which take longer as it is hard for the simulated annealing part of the algorithm to find suitable swaps. We also looked at how the brute-force algorithm running time scales, this was as expected a linear increase in running time as the width of the query increases and exponential for the number of servers, the dominating factor is the $O(N^s)$ term. Further the processing time is generally hundred of times slower than the heuristic algorithm.

## 4.3 Number of Dimensions

Figure 10 shows results of how the number of dimensions affects the running time and the average relative error. While the brute-force algorithm suffers a linear time increase with dimension, the heuristic algorithm suffers an exponential increase, but the time is still two orders of magnitude less than for the brute-force algorithm. It is worth noting that when we optimise the algorithm for one particular number of dimensions we get at least ten times faster execution time.

Figure 10 also shows that there is a linear improvement in the relative error to the ideal solution as the number of dimensions increases. There is a lot of variance in this data, but as the experiments take days it is impractical to obtain more detailed results. The improvement in results are probably due to the grid cells that are used in the heuristic algorithm containing fewer items when the number of dimensions increases, meaning greater choice in the assignment of the near groups and so the far relations can be better serviced.

It would seem sensible to use as large a number of dimensions as possible but this would mean a corresponding exponential increase in the index size and time for the heuristic method. In reality the dimensionality of the location co-ordinates may be set externally in the location system or through other considerations.

## 4.4 Number of Data Items

Two experiments were performed to investigate how the performance of the algorithms varies with respect to the number of items in the index. The first compared the heuristic, brute-force and single server algorithms using the GNP data. In this experiment we varied the numbers of items in the index up to 700 and averaged over 25 queries – once again to keep within a reasonable time.

The second experiment used a data-set derived from the Skitter measurements[1] from CAIDA. The Skitter data-set is a much larger data-set with around 20 tracers measuring route and RTT information for between 100,000 and over 800,000 hosts. Hosts are probed once per day and we take the minimum RTT over 20 days in October 2003 giving 34254 internet host locations in three dimensions. With this data we could obtain results for up to 34000 items in the index, taking results only for heuristic and single server, as the brute-force would take many hours per query. the results are averaged over 100 queries (once again due to time considerations).

Unlike the previous sections we compare the average value of the ranking function, rather than the average relative error in both these experiments.

In the first experiment we observed that the brute-force algorithm produces the best results, followed by the heuristic then the single server. The differences remain fairly constant, except in the first few results, so we do not show the graph. The difference in the first item is about 9% of the range of scaled distances between brute-force and heuristic and another 6%. to the single server. Therefore our algorithm is firmly placed between the ideal algorithm and the best a user can do. An interesting point is that on average the single server algorithm, for low number of items, has a higher “all”

[1] http://www.caida.org/tools/measurement/skitter/

than “first”. This is an artifact that the heuristic and single server algorithms rank their results using a different “distance” function to the one used to evaluate the results in these experiments.

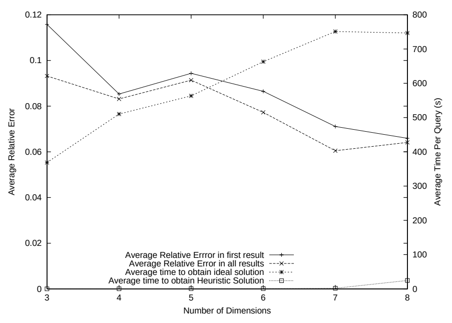

**Fig. 10.** How the running time and error are affected by the number of dimensions

Figure 11 shows the results from the second experiments into how the number of items affects running time and performance. With more items in their databases the algorithms become further apart in terms of the distances of the results produced, the heuristic algorithm being on average up to 13% better in the first result and 9% better for all the results. Therefore the heuristic algorithm becomes even more favourable over the single server algorithm when we have tens of thousands of items.

It is important to note that there will be a fair error in the approximation to the positions of the hosts. In our location based co-ordinate scheme [30] we see average relative errors of about 35%. Which numerically dwarfs the errors presented here. These location errors though do not necessarily produce a 35% error in the results of these algorithms, as the location systems may still produce coordinates that will produce a correct ranking despite the error. The exact way these errors contribute to the higher-level algorithm error is a matter of future research. Further the algorithm presented here has wider applicability than this one area and in these areas the differences we see here will be important.

We also compared the running time: both the algorithms seem to have a running time of $O(N^2)$. With few items the single server is much faster, but with over twenty thousand items (the sort of numbers we expect to deal with) the heuristic algorithms outperform the single server algorithm. So the heuristic algorithm seems to be at the expected point in between the single server (fast but higher error) and the brute-force algorithm (ideal solution, but impractical running time).

## 5 Related Work

There are two main areas of work related to our algorithm, Spatial Databases and Wide Area Resource Discovery.

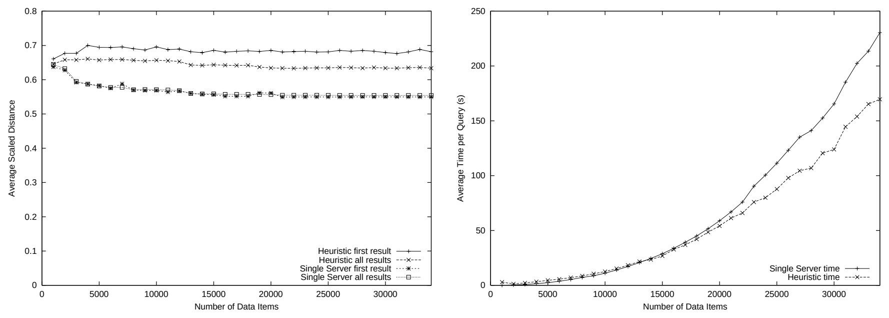

**Fig. 11.** How the running time and error is affected by the number of items for heuristic and single server algorithms (Skitter data)

### 5.1 Spatial Databases

There are many different metric space and multidimensional data indexing schemes in the area of databases. These include those based on trees like the MVPTree [4], UB tree [25] PR/PMR trees [16] and the R-tree [5]. Further methods are based on a grid-like partitioning like the Grid file [22] and the BANG file [10]. The VA file [36] is very similar to our scheme, approximating the multidimensional index by splitting into $2^b$ cells, assigning each a unique bit-string and then creating a list of these approximations. For a wider range of multi-dimensional indexes refer to a survey such as [3, 11]

A number of spatial databases have been designed using these methods for example SAND [8] and a number of further papers have tried to provide a theoretical basis for spatial database queries [**?**,23].

Lately there have been a number of efforts to distribute databases, in a Peer-to-Peer fashion [1, 32] forsaking some of the strict database invariants to allow efficient wide-area distributed databases.

### 5.2 Wide Area Resource Discovery

Historically Wide Area Resource Discovery has been built around searches for a single object as shown in the work on SLP [34], which is a centralized object location protocol, aimed at intranets. With server discovery it was based on the query “find the server closest to me” as implemented in hierarchical resource discovery systems like SSDS [17].

Many distributed approaches for single server resource discovery have been compared by Guyton and Schwartz [13]. JXTA [35], an all-in peer-to-peer system, includes a search mechanism for routing XML queries between “search consumers” and “search information providers” on a publish/subscribe model with ‘hubs’ acting as search engines. There is initially no method of linking these hubs together.

In their work on Astrolabe, van Renesse *et al.* use Bloom filters combined hierarchically to summarize interest in publish/subscribe groups [33]. Harren *et al.* investigated, in simulation, how to support an expressive set of queries over CAN [15]. However, their focus was on a database-style of interface, for instance building ‘join’ operators by the temporary creation of new DHT name-spaces.

Systems with the required search capabilities have traditionally not been distributed, but there is a number of distributed search systems and recently much research has started in this area. Pier [18] is a distributed query processing engine, an attempt to combine database and peer to peer technology. Recently work done at Rutgers [28] has looked at using a Hilbert Space Map over a Chord [31] network to perform Multi-Dimensional Searches, although the searches proposed are not as flexible as the sort proposed here. This work was similar to our previous work [29] on providing complex queries over Pastry [26]. Further on-going developments include Iris [7] which performs wide-area querying for sensor networks and Sophia[2] which is a distributed Prolog engine in which powerful distributed systems can be built.

Work at UCSB [12, 27] has centered on providing caches for range based database searches, this assumes reasonably static and non-distributed data, whereas our scheme looks to serve a situation in which the data is dynamic – although this first version assumes static data.

We envision our resource discovery system to be considerably different to work in Grid resource discovery, with wider types of Resource Discovery expected. Iamnichi and Foster [19] look at Resource Discovery in the context of Grid environments. An unstructured approach is taken, but this is still significantly different to the proposal for topological routing. Condor’s Matchmaking system [6], has been widely used for resource discovery in both cluster and Grid computing. It has a *ClassAd* based system, representing job and server advertisements with a central matcher executing a policy independent matching algorithm. This is the closest model to ours, although jobs are matched with whole machines, rather than execution environments. Further we also tackle a different area of the co-location problem than the extended Gangmatcher [24] as we have location as our motivating factor.

Distributed Constraint Satisfaction [9, 37], is also a relevant area, looking to solve the theoretical problem of Constraint Satisfaction in the wide-area, this is a similar problem to our server resource discovery.

## 6 Conclusion

We presented an algorithm which allows complex resource requests from users to be quickly serviced while at the same time not compromising on the quality of the results that are returned. It was shown that the algorithm outperforms in time per query the brute-force algorithm and is comparable to the single-server algorithm and even quicker than it for large indexes. Further the heuristic algorithm produces results of a quality that lies in between the heuristic and single-server algorithm. This means that in using this algorithm over the underlying simple searches a user will be able to produce results that are significantly better than by doing the searches himself.

[2] see `http://www.cs.princeton.edu/~mhw/sophia`

### 6.1 Future Work

We are currently working on a distributed implementation of the resource co-allocation algorithm version based on our work on XenoSearch [29]. This will support dynamic data, using a Hilbert Space index to map server locations to a single common index and the idea of aggregation points to summarize regular ranges of the index space. Bloom filters held at aggregation points will allow the search method from XenoSearch to be used, although these are not obtainable from the map. Parts of the map can also sensibly be widely replicated, e.g. by gossiping.

Once we have a distributed version of the algorithm we wish to perform a large scale deployment as part of the XenoServer [2, 14] testbed roll-out.

## 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.

## References

1. M. Annamalai. *Designing an Efficient Distributed Digital Library Database for Image Data*. PhD thesis, Purdue University, December 1997.
2. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *the Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.
3. Jon Louis Bentley and Jerome H. Friedman. Data structures for range searching. *ACM Computing Surveys (CSUR)*, 11(4):397–409, 1979.
4. Tolga Bozkaya and Meral Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, 1999.
5. King Lum Cheung and Ada Wai-Chee Fu. Enhanced nearest neighbour search on the r-tree. *ACM SIGMOD Record*, 27(3):16–21, 1998.
6. Nicholas Coleman, Rajesh Raman, Miron Livny, and Marvin Solomon. Distributed policy management and comprehension with classified advertisements. Technical Report TR 1481, University of Wisconsin, Computer Science Department, April 2003.
7. Amol Deshpande, Suman Nath, Phillip B. Gibbons, and Srinivasan Seshan. Cache-and-query for wide area sensor databases. In *SIGMOD*, 2003.
8. C. Esperanca and H. Samet. Spatial database programming using SAND. In *Proceedings of the Seventh International Symposium on Spatial Data Handling*, pages A29–A42, September 1996.
9. Marko Fabiunke. Parallel distributed constraint satisfaction. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 1585–1591, June 1999.
10. Michael Freeston. The bang file: A new kind of grid file. In *Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 260–269. ACM Press, 1987.

11. Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
12. Abhishek Gupta, Divyakant Agrawal, and Amr 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.
13. James D. Guyton and Michael F. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *proceedings of ACM SIGCOMM*, pages 288–298, August 1995.
14. Steven Hand, Tim Harris, Evangelos Kotsovinos, and Ian Pratt. Controlling the XenoServer Open Platform. In *Proceedings of the 6th International Conference on Open Architectures and Network Programming (OPENARCH)*, April 2003.
15. Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, and Ion Stoica. Complex queries in DHT-based peer-to-peer networks. *Lecture Notes in Computer Science*, 2429:242–250, March 2002.
16. 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.
17. Todd D. Hodes, Steven E. Czerwinski, Ben Y. Zhao, Anthony D. Joseph, and Randy H. Katz. An Architecture for Secure Service Discovery Service. *Wireless Networks*, 8(2–3):213–230, March 2002.
18. Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the internet with PIER. In *Proceedings of VLDB*, 2003.
19. A. Iamnitchi and I. Foster. On Fully Decentralized Resource Discovery in Grid Environments. In *Proceedings of the International Workshop on Grid Computing, Denver, Colorado*, November 2001.
20. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.
21. E. Ng and H. Zhang. Predicting Internet network distance with coordiantes-based approaches. In *INFOCOM'02, New York, USA*, 2002.
22. J. Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems (TODS)*, 9(1):38–71, 1984.
23. Jan Paredaens, Jan Van den Bussche, and Dirk Van Gucht. Towards a theory of spatial database queries (extended abstract). In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 279–288. ACM Press, 1994.
24. Rajesh Raman, Miron Livny, and Marvin Solomon. Policy driven hetrogeneous resource co-allocation with gangmatching. In *Proceedings of the Twelfth International Symposium on High-Performance Distributed Computing (HPDC-12)*, pages 80–89, June 2003.
25. Frank Ramsaka, Volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhardt, and Rudolf Bayer. Integrating the ub-tree into a database system kernel. In *VLDB 200*.
26. Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of Middleware 2001 IFIP/ACM international conference on distributed systems platforms*, November 2001.
27. 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, University of California at Santa Barbara, October 2002.
28. Cristina Schmidt and Manish Parashar. Flexible information discovery in decentralized distributed systems. In *Proceedings of the 12th IEEE Symposium on High Performance Distributed Computing*, June 2003.

29. David Spence and Tim Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *Proceedings of the 12th IEEE symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.
30. David R. Spence. An implementation of a coordinate based location system. Technical Report UCAM-CL-TR-576, University of Cambridge, Computer Laboratory, November 2003.
31. Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In Roch Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, New York, August 2001. ACM Press.
32. Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. *VLDB Journal: Very Large Data Bases*, 5(1):48–63, 1996.
33. Robbert van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.
34. J. Veizades, E. Guttman, C. Perkins, and S. Kaplin. Service location protocal, June 1997. Request for Comments RFC 2165.
35. Steve Waterhouse. JXTA search: distributed search for distributed networks, May 2001. Sun Microsystems.
36. Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 194–205, August 1998.
37. Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.