XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform

David Spence and Tim Harris

University of Cambridge Computer Laboratory
J J Thomson Avenue, Cambridge, UK, CB3 0FD
E-mail: {firstname.lastname}@cl.cam.ac.uk

Abstract

We describe the XenoSearch system for performing expressive resource discovery searches in a distributed environment. We represent server meta-data, such as their locations and facilities, as points in a multi-dimensional space and then express queries as predicates over these points. Each XenoSearch node holds a portion of this space and the key goal of XenoSearch is to direct queries to those nodes containing the meta-data of matching XenoServers. Communication between these XenoSearch nodes is based on the self-organizing Pastry peer-to-peer routing substrate. Our initial performance evaluation on a wide-area prototype shows that queries are only a factor of 3-5 times longer than basic Pastry routing, while supporting multi-dimensional searches of arbitrary shapes.

1 Introduction

In the XenoServer project [9] we are building a public infrastructure for wide-area distributed computing, creating a world in which XenoServer execution platforms are scattered across the globe and available for any member of the public. This allows users to run programs at points throughout the network to reduce communication latency, avoid network bottlenecks and minimize long-haul network charges. XenoServers can be used to deploy large-scale experimental services, and to provide a network presence for transiently-connected mobile devices.

This naturally prompts the question "How does a client find a suitable XenoServer to use"? We anticipate a wide range of job sizes and requirements, not just the large-scale parallel and scientific computation that typifies Grid systems [1, 7, 8]. Furthermore, we view network location as the key driver in selecting XenoServers; perhaps to find one near a particular client, perhaps near a remote machine with which the deployed task will interact, or perhaps within the network to minimize the maximum round trip time between users of a collaborative tool or immersive game. Different jobs will have different needs.

As a consequence of this diversity, the resource discovery system that we are building has a number of notable requirements:

1. Server selection is not just case of "find the server closest to me", but often "find the server best placed for this arbitrary [set of] host[s]". These hosts may not themselves be part of the XenoServer platform; they may well be uncooperative, perhaps being opponents in a multi-player game.

2. As well as network locations, other criteria need to be addressed. These include discrete properties ("the server must provide a Linux execution environment") and predicates on continuous ones ("the server must have a bandwidth available greater than 300Kb/s"). Societal factors such as operating jurisdiction may also feature.

3. Queries will be complex and span a combination of these different criteria, for instance searching for servers that closely match a series of terms combined using boolean operations.

4. The clients initiating queries will be heterogeneous and quite frequently resource-constrained (e.g. on slow connections).

In this paper we introduce the XenoSearch system that we have developed to manage resource discovery queries of this kind. In Section 2 we provide a high-level overview of the system, the key aspects of its architecture and how they relate to the XenoServer platform. Section 3 then introduces the design of XenoSearch and Section 4 describes our current implementation. We evaluate its performance in Section 5, running in a wide-area setting over the PlanetLab testbed [22]. Finally, Section 6 discusses related work and Section 7 concludes.

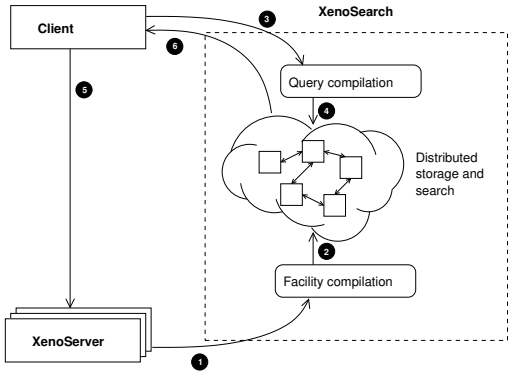

Figure 1. XenoSearch overview.

2 Architectural overview

Figure 1 provides an overview of XenoSearch. In outline, the system operates at two different levels; a high-level which is specific to the XenoServer platform and a low-level distributed search algorithm which can be re-targeted for a variety of settings. Use of XenoSearch proceeds in six steps, as indicated on the figure:

1. XenoServers periodically deposit high-level information about the facilities that they have and the current load on their resources.
2. These are distilled to the low-level formats used by the distributed search algorithm.
3. Clients issue high-level queries specifying their job requirements.
4. These are also mapped to low-level queries and issued to the distributed search algorithm, ensuring that the resulting query is at least as inclusive as its high-level counterpart.
5. The resulting set of possible matching XenoServers is returned to the client.
6. The client directly queries matching servers to confirm their current status and suitability.

The focus in this paper is on the general-purpose low-level distributed search algorithm. The companion platform architecture paper introduces the broader aspects of resource discovery and relates them to the XenoServer platform as a whole [14]. In outline, however, these high-level formats take the form of XML fragments such as that sketched in Figure 2. This gives information in a form which is meaningful to XenoServer operators and to clients defining job requirements.

```
<?xmlversion="1.0" encoding="UTF-8"?>
<EnvironmentalRequirements>
  <Basic>
     <EnvName>MyEnvironment</EnvName>
     <EnvType>Linux</EnvType>
     <EnvKernel>2.4.18</EnvKernel>
     ...
  </Basic>
  <Network>
     <EnvIPAddresses>1</EnvIPAddresses>
     ...
  </Network>
  <QoS>
     <CPUmsps>150</CPUmsps>
     <NetMbps>5</NetMbps>
     ....
  </QoS>
  <Position>
     <NearHost>www.cam.ac.uk</NearHost>
     <NearOneOf>
        <Host>www.bbc.com</Host>
        <Host>www.cnn.com</Host>
     </NearOneOf>
  </Position>
</EnvironmentalRequirements>
```

Figure 2. Example XML Query

The low-level format we use for describing job requirements and server facilities is based on a mapping to a multi-dimensional space. Different dimensions of this space can correspond to different resources (such as the amount of free CPU time on a machine), to different location metrics (such as a strictly-geographic location, or one based on network measurements) and also to different discrete properties present in some servers and absent in others.

Clusters of XenoServers at the same site can be thought of as a single multi-processor machine within XenoSearch. To aggregate the site's XenoServers to a single resource some values will be considered identical across the cluster; like topological location, while others will be represented by the total over the cluster; like CPU time available. Each XenoServer, or cluster of XenoServers, can therefore be considered as a point within this space, moving through it as the server's characteristics change.

Similarly, a query can be considered as a predicate over these points, selecting those which provide appropriate facilities for the task that is being deployed. Simple queries may take the form of k-nearest-neighbour searches around some "optimal" position. Some server characteristics are variable, such as available CPU time, and so the data held by XenoSearch may not be up-to-date. This is why clients are ultimately responsible for selecting a server from the list returned by XenoSearch (Step 5 on Figure 1.

Simple nearest-neighbour searches can attach different weights to the various dimensions in order to reflect the im-

portance of the quantities represented. We can then use a standard L_2-norm. Other queries may be more complex, for instance to find the servers in a non-connected volume. In this paper we consider queries that can be expressed as range-based searches on a number of different dimensions, combined through boolean AND and OR operators.

2.1 Deployment in the XenoServer Open Platform

We distinguish between a number of entities involved in the XenoServer platform. XenoServers are responsible for running tasks or hosting mobile code on behalf of clients, and are operated by a range of potentially competing organisations. Their role is the same as merchants in a commercial setting. Control-plane aspects of the system are managed by one or more *XenoCorps*. These are the analogue of credit card companies such as VISA and MasterCard – existing as trusted third parties between merchants and clients and providing 'value added' services to compete for business.

XenoSearch allows us to explore a range of different configurations, ranging from a centralised system, operating as a single node, to a configuration in which each registered XenoServer runs a XenoSearch node. In practice we envisage an intermediate position with a number of XenoSearch nodes distributed geographically. As with other services that form part of the XenoServer platform, we expect that these search nodes may well be hosted on XenoServers, using them as a convenient deployment platform and as a mechanism for controlling and accounting resource usage.

As we shall see in Section 3, a property of XenoSearch is that the software infrastructure necessary to support each different search dimension can be operated separately and partitioned between one or more XenoSearch nodes. This has a number of consequences. Firstly, it allows the *XenoCorps* which act as trusted-third-parties between clients and servers to provide certain important search dimensions themselves. For instance, these could include basic location properties which server operators might not be trusted to provide accurately. Furthermore, it allows independent organisations to provide their own additional search-dimensions and to populate these with information about servers. A wide variety of models are possible, including subscription services in which servers pay to be included and also "consumer guide" style services in which one organisation makes available its own subjective recommendations for others to consult or to ignore. The XML query language allows such "ad-hoc" dimensions to be incorporated alongside existing ones.

3 Distributed multi-dimensional searching

The key facility provided by XenoSearch is to allow such queries to be distributed between a number of nodes. These XenoSearch nodes form a self-organising system between which information about server facilities is partitioned. Queries can then be directed to only those nodes at which solutions may be found. If parts of the state-space are replicated by several XenoSearch nodes, queries can be directed to the most appropriate replica. Within each node we can exploit existing algorithms for multi-dimensional searching [4].

Before turning to the design and implementation of our distributed search algorithm, we will define the interface that it exports:

Distributed search interface

```
void addpoint(handle s, point p)
{handle} search(query q)
```

An `addpoint` operation associates handle `s` with a point `p` in the search space. For instance, handles could identify individual XenoServers. Each handle can only be associated with a single point and so, as a server's status changes, it will perform a succession of `addpoint` operations. Step 2 on Figure 1 is an invocation of `addpoint`.

A `search` operation returns a set of handles that might match a query `q` constructed from *simple queries* by boolean AND and OR operators. A *simple query* is a range-based predicate on one of the dimensions of the space in which points are defined. Step 4 on Figure 1 is an invocation of `search`. Note that we expect clients of `search` to make their own investigations of the results that they receive. In the case of XenoSearch this is necessary because a server's resource availability may have changed and so up-to-date information is needed before deploying a job on the server. A key goal in designing a good implementation is to improve the precision with which searches operate and to develop an acceptable balance between this, implementation complexity, and run-time costs.

After introducing the routing substrate over which we build our work (Section 3.1), our design follows two stages. Firstly, in Section 3.2 we construct a mechanism for performing range-based searches within a single dimension. Secondly, in Section 3.3 we perform searches separately for each of the simple queries from which a search is constructed and then combine the results.

3.1 Routing substrate

Our design is built over the API provided by the Pastry [26] routing substrate, developed at Microsoft Research and Rice University. In outline, this provides a simple mechanism concerned with routing messages between a

self-organising collection of nodes. In Pastry, each node is assigned a unique 128-bit `nodeId`. This ID space is usually depicted as a ring with the minimal and maximal values adjacent. Aside from initialisation, there is a single invocation that may be made on the Pastry module:

Pastry routing substrate

```
void route(msg m, key k)
```

This requests that the message `m` be sent to the Pastry node whose ID is closest to the 128-bit key `k`. Normally, `nodeId`s are selected uniformly at random, giving probabilistic load-balancing assuming that `route` messages are similarly distributed. An application built over Pastry receives call-backs in a number of situations:

Pastry call-backs

```
void deliver(msg m, key k)
key forward(msg m, key k, key next)
void leafSetChanged(leaf_set l)
```

The first of these, `deliver`, is invoked to deliver `m` on the node whose ID is closest to `k`. The second, `forward`, is invoked at each intermediate node through which `m` passes, providing an opportunity to modify `m` or to return an alternative destination key. Finally, `leafSetChanged` is invoked when there is a change to the node's *leaf-set*, which is the set of closest Pastry nodes in the key space.

Applications are built over Pastry by introducing an application-specific interpretation of the key space. For instance, a simple distributed hashtable (DHT) could generate keys from the hashed items, with each node holding the portion of the hashtable whose keys are closest to its own `nodeId`. Alternatively, the Scribe [27] application-layer multicast system uses keys to identify multicast groups, with the owning Pastry node holding their membership.

Pastry routes messages by splitting the destination key into groups of `b` bits and sending it, hop by hop, to a node which has at least one more matching group at the start of its `nodeId`. For instance, as shown in Figure 3, a message sent from 23333333 to 02000000 could first go to **0**3321212 (matching the first group), then **02**321221 (matching the first two) and so on. It is easy to see that the performance is going to be bounded by $O(\log n)$: this is proved in [26] and experimentally it performs slightly better. If the node is near the final destination (by `NodeId`) it will be contained within the leaf-set and delivered directly.

Although Pastry therefore provides a message-based API, we shall proceed in this section to describe the operation of XenoSearch in terms of a procedural interface, before looking at how this is implemented in terms of messages in Section 4.

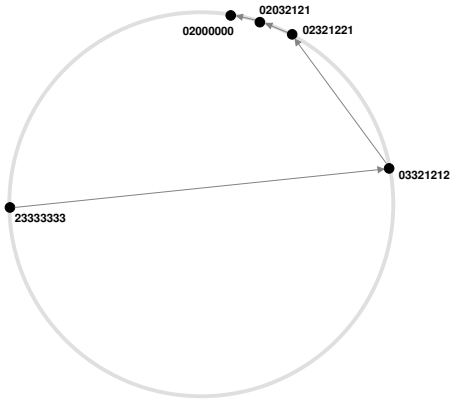

Figure 3. Example Pastry ring routing to 02000000 from 23333333 (with 16-bit nodeId expressed as eight 2-bit values)

3.2 Range-based searching

The first part of our design is to build an application over Pastry which supports operations to return sets of servers whose keys lie within some range in a single dimension. Servers themselves are identified by globally unique *handles*.

Range-based searches

```
void addpoint_1d(handle s, key k)
handle_summary search_1d(key k1, key k2)
```

A `handle_summary` represents a set of possible results from a search request. It should include all of the matching handles but may include additional ones, either because of approximations in the implementation of the initial search, or because of approximations made in the representation of the summary.

A separate Pastry ring operates for each dimension with XenoSearch nodes registering separately in each ring. A XenoServer invokes `addpoint_1d` for each dimension deriving a key from its co-ordinate in that dimension – while the `key` is therefore likely to be different in each ring, the `handle` that it uses to identify itself remains common.

In each dimension, information is conceptually held in the form of a tree with the leaves being the individual XenoServers and interior nodes being *aggregation points* (AP) which summarise the membership of ranges of nodes below them. These APs are themselves identified by locations in the key space which can be determined algorithmically by forming keys with successively longer suffices of the form $10\ldots0$. For instance, the AP directly above 10233102 is 10233101, then 10233110, then 10233100 and so on:

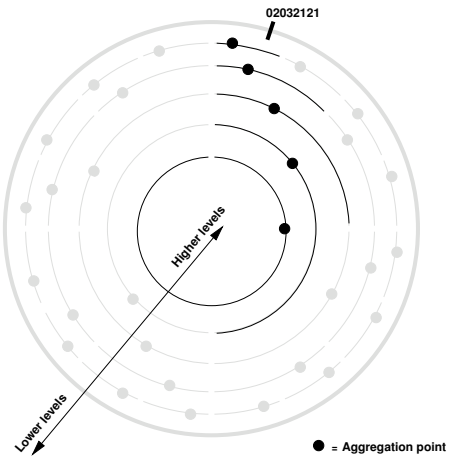

Figure 4. How aggregation points cover Pastry rings. The outer edge represents the key space, inner arcs regions owned by each AP. Darker arcs show how 02032121 is covered.

Aggregation Point	Range of Nodes
10233101	10233100...10233103
10233110	10233100...10233133
10233100	10233000...10233333
10231000	10230000...10233333
10210000	10200000...10233333
10100000	10000000...10333333
11000000	10000000...13333333
10000000	00000000...33333333

The XenoSearch node closest in the key space to an AP is responsible for managing this information and for dealing with messages it receives as a consequence This locality is provided by the Pastry substrate as its basic routing is to the node closest to the destination key. Each AP can directly answer `search_1d` queries corresponding to the full key range that it covers. This is shown graphically in Figure 4.

3.3 Multi-dimensional searching

Multi-dimensional searching is performed by making a series of `search_1d` invocations for each of the separate dimensions involved in the query and forming their intersection. Of course, in addition to simple volume-based queries, we can in fact support more general queries based on union and intersection. In either case, the node performing the search is left with a single `handle_summary` which indicates the servers which may match the original query. Queries can be initiated by any node in the overlay.

The final step is to translate this summary back to an explicit set of handles to be returned to the client, using a single additional operation:

Multi-dimensional search

```
{handle} expand_summary(key k,
                        handle_summary s)
```

An invocation of `expand_summary` routed to the key `k` of an 'below' `k` which are contained in the summary set `s`. Effectively it proceeds as an application-layer multicast down the aggregation point tree, ending in those nodes. The multicast is filtered at all levels to include only those that may occur in the `handle_summary`.

4 Implementation

We now turn to a number of aspects of how the operations defined in Section 3 can be implemented over Sun's Java SDK version 1.4.1 and FreePastry[1]. In Section 4.1 we describe the implementation and management of the `handle_summary` sets returned by range searches. In Section 4.2 we describe the information held at aggregation points and how this is updated as XenoSearch nodes join and leave. Section 4.3 describes how these are used to implement single-dimensional range searches. Finally Section 4.4 explains how query summaries are translated to a set of handles.

4.1 Summary sets

We summarise sets of handles by *Bloom filters* [3]. Each filter provides a constant-space approximate representation of the contents of a set. Errors between the actual set and the Bloom filter representation are always of the form of false positives to inclusion tests. Concretely, a Bloom filter is a bit vector and items are inserted by setting the bits corresponding to a number of independent hash functions. Set-wise operations can be performed directly by AND (for intersection) and OR (for union) between these vectors. Inclusion tests are performed by checking that all of the bits given by the hash functions are set – therefore, as the number of items in the set increases, so does the likelihood of false positives. Figure 5 illustrates this scheme.

4.2 Aggregation Point Structure

The XenoSearch node owning each aggregation point (AP) holds a summary set for each of its children, along with the approximate counts of the membership of those sets and a timestamp indicating when this information was last refreshed by the children. These children may either be lower APs, or may be individual entries that have been defined in the range that the AP covers. Each AP constructs its

[1] `http://www.cs.rice.edu/CS/Systems/Pastry/`

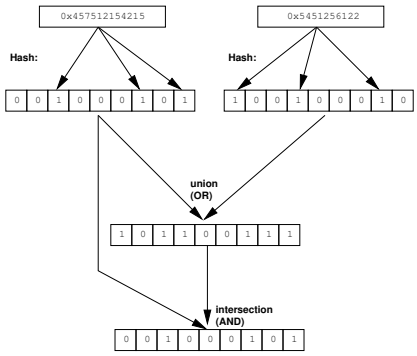

Figure 5. Bloom filters constructed for two singleton sets and the operations of union and intersection.

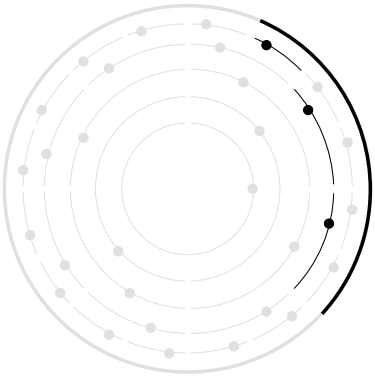

Figure 6. Aggregation points queried to return handle summaries for the indicated region of the key space.

own filter from these components and reconstructs it whenever any of them changes.

The information held at APs is treated as soft-state in that XenoServers continue to periodically invoke `add_point_1d` to reflect their current status and nodes within the tree periodically update the summaries held at their parents. This balances the load across APs, so long as nodes within the tree have a similar number of children to the number of XenoServers managed by leaf APs. This soft-state system can be used because we assume that further checking of the returned results will be performed, XenoSearch returns an approximate result set and so the AP structure does not need to always totally up to date.

This uses a combination of Pastry messages, which can be directed toward specific key values in each ring, and a custom UDP-based protocol to avoid the overhead of a Pastry message send when the XenoSearch node hosting an AP has not changed. The Pastry `leafSetChanged` call-back is used to trigger a change from one UDP endpoint to another, causing the previous owner of the AP to inform its children of the new owner's location.

4.3 Implementing `search_1d`

The `search_1d` operation is implemented by sending Pastry messages to the minimal number of APs which cover the requested range. For instance, the heavy lines in Figure 6 indicate the three indicated section of the key space. Each AP is contacted in parallel using asynchronous I/O operations to collect the results.

4.4 Implementing `expand_summary`

The `expand_summary` operation is implemented by introducing a `FilteredMulticast` message which carries the original query to the nodes present in the summary set of results. It is sent directly to each of the IP addresses (by IP unicast) of the APs which might contain results, again as indicated in Figure 6. These APs then forward this on to their children (using the Pastry ring, within which their children will tend to be members of the leaf-set, if not on the same node). The multicast can be pruned if an AP has no children which could match the summary set.

However, note that it does not have to be implemented in this top-down manner: APs can be contacted directly without needing to go through their parents. This may be useful to avoid placing a high load on APs towards the top of the tree if searches cover very large regions of the key space. The disadvantage is that it requires more network round-trips to the (possibly poorly-connected) search client.

5 Results

In this section we present a number of results from our prototype system which can run over the XenoServer platform, over PlanetLab, or as a standalone application.

5.1 Initial simulation

We initially developed a simple simulation to inform a number of the implementation decisions – in particular the size of the Bloom filters to use and the branching factor at each aggregation point. The system did not involve distribution and so our main performance metric was the *precision* that XenoSearch delivered: that is, the ratio of the number of servers returned by `search` to the number of genuine matches. We assumed that each server was an individual XenoSearch node as the most extreme case.

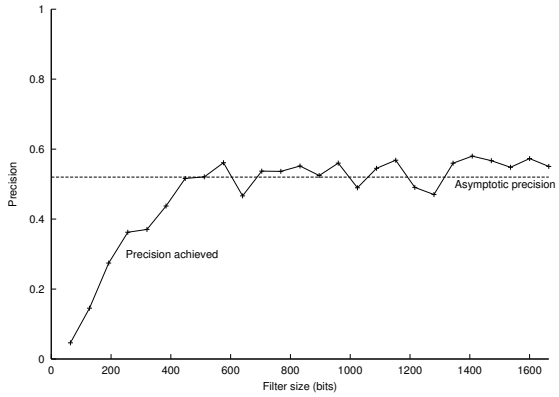

Figure 7. The effect of the Bloom filter size on precision.

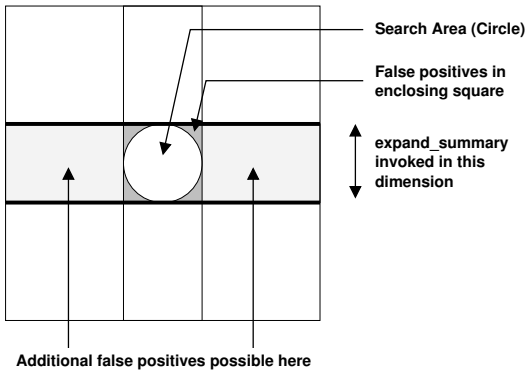

Figure 8. Possible sources of imprecision (2D).

We considered simple 3-dimensional searches, finding points within a sphere. Figure 7 shows how the size of the Bloom filters impacts the precision. While there is a clear trade-off between the filter size and the resulting precision there are diminishing returns beyond 448-bit filters. Beyond that point, appreciable numbers of false positives are not being generated by the Bloom filters. However, note that false positives in the filter will only lead to false positives in the results if they lie below an aggregation point targeted by `expand_summary`, as indicated in Figure 8 – false positives in other sections are benign because they do not receive that `expand_summary` request as we filter the multicast on the search range in that dimension.

In this setting, the best precision we could hope for is $\pi/6 = 0.52$ as we are approximating a sphere with a cube. Of course, the "curse of dimensionality" will have an impact with higher numbers of dimensions [34], but in the context of server discovery we expect only moderate numbers of search dimensions such as 4–7. More will be available but we expect most users will only wish to use a few. In practical settings we expect better results: searches will either be looking for "square" ranges, or we can approximate a sphere more closely by searching in additional dimensions defined as rotations of the existing ones.

5.2 Simple searches

We deployed our prototype XenoSearch system over 18 PlanetLab nodes (13 in North America, 4 in Europe and 1 in Australasia). Each machine ran a single XenoSearch node holding, on average, $\frac{1}{18}$ of the information for each dimension.

We discarded start-up transients, these are caused by the JVM optimising various parts of the code and only affect the first 10 results. As XenoSearch will be run as a service, these will not affect normal operation.

We performed 100 timed remote searches – i.e. searches which could not be serviced locally – and compared the time taken against that taken for a basic Pastry message delivery. This gives a search time roughly independent of distance travelled in the network. Figure 9 shows the results for searching in 1 to 5 dimensions.

From the graph it can be seen that the vast majority of searches take 3-3.5 times longer than routing a basic message over Pastry. This follows from the design of the system. Firstly a series of Pastry messages are sent to request handle summaries. Secondly, the results are returned to the requester. Thirdly, a filtered multicast is sent directly to the aggregation points within the query and then onward over Pastry down the tree. Due to the small number of nodes, the APs we send to will be on the same node that the multicast message destination, in the majority of cases. Therefore the filtered multicast step will normally be a single UDP delay. Furthermore, with this moderate number of nodes, routing over Pastry will generally be satisfied by the leaf-sets. It is therefore consistent to expect a latency ratio of just over 3.

As we increase the number of nodes, we would expect the Pastry message cost to increase, as $\log_{2^b} n$. This will mean the saving of using UDP-based messages directly in XenoSearch will become important. At the same time we will, of course, begin to see more situations involving additional work in the filtered multicast stage (again approximately $\log_{2^b} n$). Although searches will always be bounded by $O(\log n)$, it remains to see the exact effect that these two factors will have when combined.

As we increase the number of dimensions, we increase the response-latency only marginally. This is because the network part of the searching algorithm happens in parallel and vastly dominates the CPU-bound computation on each node. This implies that the system is scalable with respect

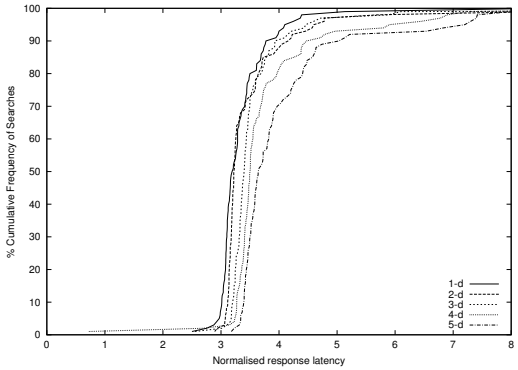

Figure 9. Cumulative frequency graph of latency for hyper-sphere searches in successively greater numbers of dimensions, normalised so Pastry messages have latency 1.

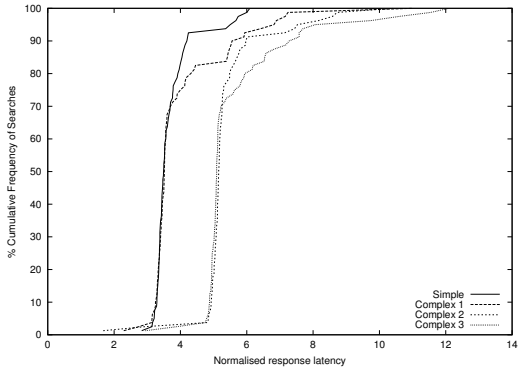

Figure 10. Cumulative frequency graph of latency for a number of successively more complex searches, normalised so Pastry messages have latency 1.

to dimensions. Previous experiments showed the local CPU bound time to scale linearly with the number of dimensions, for our local machines it was 9.5ms for 1 dimension, rising by about 3.6ms per dimension. This is much less than the network bound time which can be up to 200ms for planetlab and much more than the sub 1ms for remote CPU time.

We also investigated the volume of network traffic generated by searches. Excluding aggregation point (AP) updates, a 2-d range-based search generated approximately 2 Kilobytes of Pastry messages and 2.7 Kilobytes of other UDP traffic. Each additional dimension adds around 800 bytes to each. The network traffic generated by AP updates can be reduced arbitrarily by controlling the refresh cycles and timeouts involved.

5.3 Complex searches

Our second set of results show how the system performed given more complex queries, such as those that will be used in the XenoServer Open Platform. Figure 10 shows the results of the experiment, as before we used the same 18 servers, this time we obtained 80 results, comparing the routing stretch over Pastry for various types of searches.

The first of these, "Simple" is a hyper-sphere search in all 5 dimensions, as used in the section above. The remaining, "Complex-1", "Complex-2" and "Complex-3" perform more realistic searches. In these cases we imagined the 5 dimensional space to be split up as 3 dimensions for position and the fourth and fifth dimensions for continuous QoS attributes such as CPU speed and link bandwidth. In the position dimensions we performed one, two or three hyper-sphere searches respectively combining the results by union. This corresponds to a query where we would like a server that is close to one of three hosts. In the fourth and fifth dimensions we performed range searches, corresponding to queries for a range of CPU speeds or link bandwidths. The individual results were then intersected to find the servers which fulfilled the criteria.

The simple search and one with one hyper-sphere perform similarly. This is not surprising; they are both performing a hyper-sphere/cube search. The searches which add additional hyper-sphere regions introduce 2 additional Pastry-delays. This is explained by the fact that only in these searches do we visit dimensions twice, meaning that the second request (made up of a set of messages) to that same dimension gets queued behind the first set as they are routed through Pastry. This delay is only felt once in the 3 hyper-sphere search, therefore we expect all searches of more complexity to take similar time.

6 Related Work

As we saw in the introduction, resource discovery in XenoServers is not like the traditional closest server discovery, as performed by hierarchical resource discovery systems like SSDS [18]. These require searches to be started at the host we wish to find a server for, which may be impossible or impractical as we often wish to return a set of servers, or to search for those close to some remote location.

Centralised directory systems like SLP [32] are mainly useful for intranets and are not really useful for wide-area distributed use. Guyton and Schwartz [13] compare many of the various methods of location. JXTA [33], 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.

Various kinds of distributed search have been developed in the peer-to-peer community. These are often just the mechanisms by which naming is performed in those systems, rather than being directly intended for searching. Examples include simple file sharing applications [5] and also distributed hashtable abstractions (DHTs) [17, 23, 24, 26, 29] as well as skip graphs [2] and distributed tries [10]. In all of these cases, the search facilities are too limited for direct use in the XenoServer platform, allowing only filename based searches in the former systems, or nearest-key based searches in the latter. Andrzejak and Xu [1] extend CAN for range queries, although in the one-dimensional context.

As we have seen in building our work over Pastry [26], these distributed systems form useful substrates for developing applications. Other authors have built search mechanisms over these systems. Many are distributed implementations of existing Information Retrieval (IR) techniques, for instance PlanetP [6] using a "Name dropper" flood algorithm [15] and pSearch [30] and Sedar [21] using CAN. More traditionally, efficient keyword searches have been researched, which distribute an inverted index [11, 25].

In their work on Astrolabe, van Renesse *et al.* use Bloom filters combined hierarchically to summarise interest in publish/subscribe groups [31]. Harren *et al.* investigated, in simulation, how to support an expressive set of queries over CAN [16]. However, their focus was on a database-style of interface, for instance building 'join' operators by the temporary creation of new DHT name-spaces.

Work at UCSB [12, 28] has centred on providing caches for range based database searches, this assumes reasonably static and non-distributed data, whereas our scheme looks at dynamic distributed searches. Work by Iamnichi and Foster [19] looks at Resource Discovery in the context of Grid environments. An unstructured approach is taken, while ours is a structured approach, allowing us to bound the hops to distant nodes. This is important because we believe the commonest type of search in XenoSearch will be to locate a topological distant XenoServer.

We did consider basing our system on CAN instead of Pastry. CAN would seem to be a natural choice because it itself locates nodes within a multi-dimensional space rather than Pastry's 1-d keys. Although this would enable a simple "find node containing point `p`" style of search, it does not lend itself readily to the more general style of search which we propose. We would have to route a request to the centre of the search area then perform a flood search of the surrounding area to find the nodes that are contained within the search. It would be hard to perform searches that are not limited to simple connected regions. It would also prevent the ad-hoc development of additional search metrics in the way we described in Section 2.1. Such decentralisation is key to the success of the XenoServer platform as an infrastructure service open to and accepted by the public.

7 Conclusion

In this paper we have introduced the self-organising XenoSearch system, providing a mechanism through which information can be partitioned between a number of distributed nodes, and queries routed to subsets of those nodes on which results may be found. Providing a system which can service arbitrarily complex volume searches in multi-dimensional vector spaces, while only being a factor of 3-5 slower than plain Pastry routing.

This was presented with the aim to develop a resource discovery system for the XenoServer Open Platform, supporting complex queries over server resource availability or location. In keeping with the XenoServer architecture, our system allows third parties to readily introduce new 'dimensions' on which searches can be performed without centralised control.

Our evaluation of XenoSearch presented here is only preliminary and we look forward to and are already testing the system in less synthetic situations and workloads within PlanetLab and the XenoServer Open Platform.

In future work, we plan to develop XenoSearch in a number of directions. Firstly, it is still clearly possible to identify kinds of query which cannot be made over the current system. For instance, a client may seek a number of machines with inter-dependent properties (such as having different owners or dispersed geographic locations). Such extensions raise interesting questions both in terms of the query language and in terms of the implementation of searches. Secondly, we see this current work as being an interesting step from original peer-to-peer applications toward more controlled self-organising distributed systems. We are investigating what common building blocks are generally useful in such cases, and whether lightweight implementations exist. Finally, we intend to evaluate the extent to which space-filling curves can be used in this context to map multi-dimensional data onto a single-dimensional space [20].

We are continuing this work as part of the FutureGRID UK e-science project in collaboration with Microsoft Research.

8 Acknoledgments

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

References

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

[2] J. Aspnes and G. Shah. Skip graphs. In *proceedings of 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA*, pages 384–393, January 2003.

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

[4] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[5] J. Crowcroft and I. Pratt. Peer-to-Peer: Peering into the Future. In *Proceedings of the IFIP-TC6 Networks 2002 Conference, Pisa, Italy.*, May 2002.

[6] F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. Technical Report DCS-TR-483, Rutgers University, 2002.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *10th IEEE Symposium On High Performance Distributed Computing*, August 2001.

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

[9] K. Fraser, S. Hand, T. Harris, I. Leslie, and I. Pratt. The XenoServer Computing Infrastructure. Technical Report UCAM-CL-TR-552, University of Cambridge, Computer Laboratory, January 2003.

[10] M. J. Freedman and R. Vingralek. Efficient Peer-to-Peer Lookup Based on a Distributed Trie. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.

[11] O. D. Gnawali. A Keyword-Set Search System for Peer-to-Peer Networks. Master's thesis, Electrical Engineering and Compter Science, MIT, June 2002.

[12] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate range selection queries in peer-to-peer systems. Technical report, Department of Computer Science, University of California at Santa Barbara, October 2002.

[13] J. D. Guyton and M. F. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *proceedings of ACM SIGCOMM*, pages 288–298, August 1995.

[14] S. Hand, T. Harris, E. Kotsovinos, and I. Pratt. Controlling the XenoServer Open Platform. In *Proceedings of the 6th International Conference on Open Architectures and Network Programming (OPENARCH)*, April 2003.

[15] M. Harchol-Balter, F. T. Leighton, and D. Lewin. Resource Discovery in Distributed Networks. In *proceedings of the 18th annual ACM Symposium on Principles of Distributed Computing*, pages 229–237, May 1999.

[16] M. Harren, J. M. 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, Mar. 2002.

[17] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed Object Location in a Dynamic Network. In *Proceedings of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures*, pages 41–52, August 2002.

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

[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] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Record*, 30(1):19–24, 2001.

[21] M. Mahalingam, C. Tang, and Z. Xu. Towards a Semantic, Deep Archival File System. Technical Report HPL-2002-199, Hewlett-Packard Research Labs, July 2002.

[22] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *proceedings of the first ACM HotNets Workshop*, October 2002.

[23] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, June 1997.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *proceedings of ACM SIGCOMM*, pages 161–172, August 2001.

[25] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. http://issg.cs.duke.edu/search/.

[26] A. Rowstron and P. 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] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[28] O. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi. Query processing over peer-to-peer data sharing systems. Technical report, Department of Computer Science, University of California at Santa Barbara, October 2002.

[29] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In R. 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.

[30] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information Retrieval in Structured Overlays. In *First Workshop on Hot Topics in Networking*, October 2002.

[31] R. van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining, 2001. ACM TOCS (Conditionally accepted November 2001, revised September 2002).

[32] J. Veizades, E. Guttman, C. Perkins, and S. Kaplin. Service location protocal, June 1997.

[33] S. Waterhouse. JXTA search: distributed search for distributed networks, May 2001. Sun Microsystems.

[34] 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 Int. Conf. Very Large Data Bases, VLDB*, pages 194–205, August 1998.