An overlay network for resource discovery in Grids*

Manfred Hauswirth, Roman Schmidt
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract

As Grids try to achieve optimal and balanced utilization of unused resources in a distributed system, fast and efficient discovery of resource states is a key requirement. For small to medium scale Grids, solutions such as the approach in Globus work fine. However, for large, up to global-scale Grids, this approach is not efficient and does not scale. Additionally, even for smaller Grids, a centralized solution will always be a performance bottleneck and a single point of failure. In this paper we investigate the applicability of a structured peer-to-peer system (overlay network) for the discovery of Grid resources. Each node in the Grid becomes a peer in the overlay network, which provides a distributed directory service that allows the participants to discover resources and maintain resource states. Overlay networks implicitly balance load, scale well to very large numbers in terms of nodes and data, and meet the partial failure property of distributed systems, i.e., the system remains operational despite partial failures. We will outline a proof-of-concept implementation based on our P-Grid overlay network, present experimental results from a large-scale deployment on PlanetLab and discuss the pros and cons of overlay networks in the context of Grids.

1. Introduction

Fast discovery of available resources and efficient maintenance of resource states are key requirements for Grids to achieve optimal utilization of the system and to balance load among the participating computers. Current systems such as Globus or Condor use server-based solutions to assign computation requests (jobs) to available resources. Resources publish their current state at servers which can then be matched with job requirements to enable automatic job scheduling. This approach works fine for small to medium scale Grids, e.g., server clusters. For large-scale Grids this approach does not scale well and will be a performance bottleneck and a single point of failure, even if replicated servers (clusters) are used, because though the computational load may be shared among the cluster nodes they still would use a single network connection which may break down and render the whole system dysfunctional. Only distributed approaches can remedy this problem, which, however comes at certain costs, as always when a centralized system is being distributed. The question we want to investigate in this paper is, whether overlay networks are a viable solution to address these problems and to what extent they can help to minimize the costs of the distribution in respect to discovery.

Similar to Grids, P2P systems build on the principles of cooperation and sharing of resources. P2P systems such as Gnutella, KaZaA, and eMule have proven their applicability for global-scale resource discovery and file-sharing. Currently, research focuses on *structured* overlay networks to optimize performance. Examples of structured overlays are Chord, Tapestry, and P-Grid. The underlying idea in these systems is to build a distributed index of the resources available at the nodes. The index is constructed such that the search space (key space) is partitioned and each key space partition is assigned to a peer. For being able to answer queries on all the resources available in the system, each peer builds up a routing table that enables it to forward queries which it cannot answer locally, to peers that are "closer" to the result. The construction and routing strategies for these indexes differ a lot in terms of robustness and flexibility. However, their common goal is to minimize the forwarding steps required for successfully resolving queries in the presence of arbitrary peer and connection failures, as network communication is the major bottleneck in distributed systems. At the moment the best practical solutions offer logarithmic search complexity, i.e., in a directory consisting of n nodes the maximum number of communication steps among the nodes to resolve a query is $O(\log n)$. To achieve robustness, multiple peers (replicas) are assigned to each key space partition and the system has to keep them

*The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and was supported by the Swiss National Funding Agency OFES as part of the European project DIP (Data, Information, and Process Integration with Semantic Web Services) No 507483.

in sync. The goal is to have at least one peer per partition available to answer/forward queries at any time. The degree of fault tolerance is tunable to system dependent requirements by the replication factor and the number of entries in routing tables. A comprehensive overview of current structured overlay networks and the applied techniques is given in [11].

In the following we will present a completely decentralized approach to resource discovery. We use P-Grid, our implementation of a structured overlay network, as the base technology for the distributed directory assuming that the software is installed on all Grid nodes. Without constraining general applicability, we use the directory to manage resource requests for jobs which require CPU cycles. A generalization to multiple resources is straight-forward as will become clear when we present the approach. The basic idea is that requesters publish their resource requirements (CPU cycles) in the directory, resource providers query the directory for matching requests, and can then decide which requester to contact and offer the resource to. To enable this efficiently, the directory (overlay network) has to support efficient range queries and updates which P-Grid offers and we will describe in the following. As an alternative strategy, many systems index the offered resources and allow requesters to search for resources matching their job requirements. While basically both strategies can offer the same level of functionality, this strategy puts much higher requirements on the overlay network in respect to update frequencies as we assume job descriptions to change less frequently. Indexing job requirements enables the providers to actively decide when to offer resources and to whom, i.e., as soon as computers have idle resource (CPU cycles in our experiments) they can look for jobs to process, whereas with the other strategy users have to poll the system for suitable resources until they become available.

2. Related Work

The latest Monitoring and Discovery Service (MDS) of Globus is based around WSRF (Web Services Resource Framework) standards providing an index service, i.e., a registry similar to UDDI, to maintain the set of registered Grid resources in virtual organizations. MDS provides query and subscription interfaces to arbitrarily detailed resource data and a trigger interface that can be configured to take action when pre-configured conditions are met. Indexes can be combined in a hierarchical fashion to aggregate data at different levels. Though MDS works well for small to medium scales, it remains unclear how it could support global-scale grids as the hierarchical organization and query routing has hot-spots and single-points-of failure.

Condor [10] targets primarily optimal CPU utilization. It uses a centralized matchmaker to match resource requests with offers. The centralized architecture is efficient for small grids in LANs for which Condor was designed initially but does not scale up to larger scales.

Iamnitchi et al. [8] propose resource discovery based on an unstructured network similar to Gnutella combined with more sophisticated query forwarding strategies taken from the Freenet overlay network. Requests are forwarded to one neighbor only based on experiences obtained from previous requests, thus trying to reduce network traffic and the number of requests per peer compared to simple query flooding as used by Gnutella. The approach suffers from higher numbers of required hops to resolve a query compared to our approach and provides no lookup guarantees, i.e., an unsuccessful lookup does not necessarily mean that no resource meeting the requirements is available because a suitable peer was simply not reached.

The most similar approach to ours is presented in [7] and uses a range-query-enhanced version of CAN [12]. Resource descriptions and jobs are stored in the overlay network enabling users to find suitable resources and computers to find suitable jobs. The fundamental problem of this approach lies in the overlay network because the ranges themselves are hashed, and hence, simple key search operations are not supported or are highly inefficient. Since both key and range queries are needed, it is desirable to have one mechanism supporting both, instead of maintaining separate hash table for keys, and separate hash tables for ranges, because such a strategy fails to reuse the resources of the peers. In the context of Grids the system may become inefficient because resource informations such as available memory are highly dynamic and have to be up-to-date. In case a resource changes its state a different peer would become responsible for this resource leading to frequent deletions and inserts (or updates) for the overlay network which CAN does not tolerate well. Moreover, since they use CAN as the underlying network, the search efficiency guarantees hold only for uniform partitioning of the space, which conflicts with storage load which is arbitrarily distributed, as will be the case for caching range queries, more so because queries will also be non-uniformly distributed.

3. The P-Grid overlay network

We use the P-Grid overlay network [1, 3] as the base technology for our distributed resource discovery service. P-Grid is a structured overlay network based on the so-called distributed hash table (DHT) approach. In DHTs peer identifications and resource keys are hashed into one key space. By this mapping responsibilities for partitions of the key space can be assigned to peers, i.e., which peer is responsible for answering queries for what partition. To ensure that each partition of the key space is reachable from any peer, each peer maintains a routing table. The routing table of a peer is constructed such that it holds peers with exponentially increasing distance in the key space from its own position in the key space. This technique basically builds a small-world graph [9], which enables search in $O(\log n)$ steps. Basically all systems referred to as DHTs are based on variants of this approach and only differ in respect to fixed (P-Grid, Pastry) vs. variable key space partitioning (Chord), the topology of the key space (ring, interval, torus, etc.), and how routing information is maintained (redundant entries, dealing with network dynamics and failures, etc.).

Without constraining general applicability we use binary keys in P-Grid. This is not a fundamental limitation as a generalization of the P-Grid system to k-ary structures is natural, and exists. P-Grid peers refer to a common underlying binary trie structure in order to organize their routing tables as opposed to other topologies, such as rings (Chord), multi-dimensional spaces (CAN), or hypercubes (HyperCuP). Tries are a generalization of trees. A trie is a tree for storing strings in which there is one node for every common prefix. The strings are stored in extra leaf nodes. In the following we will use the terms trie and tree conterminously.

In P-Grid each peer $p \in P$ is associated with a leaf of the binary tree. Each leaf corresponds to a binary string $\pi \in \Pi$, also called the *key space partition*. Thus each peer p is associated with a path $\pi(p)$. For search, the peer stores for each prefix $\pi(p, l)$ of $\pi(p)$ of length l a set of references $\rho(p, l)$ to peers q with property $\overline{\pi(p, l)} = \pi(q, l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level which enables the implementation of prefix routing for efficient search. The cost for storing the references and the associated maintenance cost scale as they are bounded by the depth of the underlying binary tree.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ the binary key $key(d)$ is calculated using an order-preserving hash function, i.e., $\forall s_1, s_2 : s_1 < s_2 \Rightarrow h(s_1) < h(s_2)$, which is pre-requisite for efficient range querying as information is being clustered. $key(d)$ has $\pi(p)$ as prefix but it is not excluded that temporarily also other data items are stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. Moreover, for fault-tolerance, query load-balancing and hot-spot handling, multiple peers are associated with the same key-space partition (structural replication), and peers additionally also maintain references $\sigma(p)$ to peers with the same path, i.e., their replicas, and use epidemic algorithms to maintain replica consistency. Figure 1 shows a simple example of a P-Grid tree. Note that, while the network uses a tree/trie abstraction, the system is in fact hierarchy-less, and all peers reside at the leaf nodes. This avoids hot-spots and single-points-of-failures.

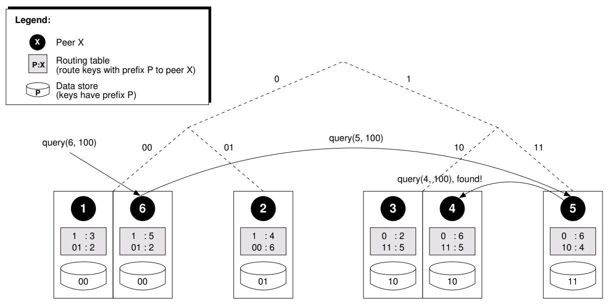

Figure 1. P-Grid overlay network

P-Grid supports a set of basic operations: *Retrieve(key)* for searching a certain key and retrieving the associated data item, *Insert(key, value)* for storing new data items, *Update(key, value)* for updating a data item, and *Delete(key)* for deleting a data item. Since P-Grid uses a binary tree, *Retrieve(key)* is of complexity $O(\log |\Pi|)$, measured in messages required for resolving a search request, in a balanced tree, i.e., all paths associated with peers are of equal length. Skewed data distributions may imbalance the tree, so that it may seem that search cost may become non-logarithmic in the number of messages. However, in [2] it is shown that due to the randomized choice of routing references from the complimentary subtree, the expected search cost remains logarithmic ($0.5 \log n$), independently of how the P-Grid is structured. The intuition why this works is that in search operations keys are not resolved bit-wise but in larger blocks thus the search costs remain logarithmic in terms of messages. This is important as P-Grid's order-preserving hashing may lead to non-uniform key distributions.

The basic search algorithm is shown in Algorithm 1.

Algorithm 1 Search in P-Grid: Retrieve(key, p)

```
1: if $\pi(p) \subseteq key$ then
2:     return($d \in \delta(p) | key(d) = key$);
3: else
4:     determine $l$ such that $\pi(key, l) = \overline{\pi(p, l)}$;
5:     r = randomly selected element from $\rho(p, l)$;
6:     Retrieve(key, r);
7: end if
```

p in the algorithm denotes the peer that currently processes the request. The algorithm always terminates successfully, if the P-Grid is complete (ensured by the construction algorithm) and at least one peer in each partition is reachable (ensured through redundant routing table entries and replication). Due to the definition of ρ and $Retrieve(key, p)$ it will always find the location of a peer at which the search can continue (use of completeness). With each invocation of $Retrieve(key, p)$ the length of the common prefix of $\pi(p)$ and key increases at least by one and therefore the algorithm always terminates.

Insert(key, value) and *Delete(key)* are based on P-Grid's more general update functionality [5], *Update(key, value)*, which provides probabilistic guarantees for consistency and is efficient even in highly unreliable, replicated environments, i.e., $O(\log |\Pi| + replication\ factor)$. An insert operation is executed in two logical phases: First an arbitrary peer responsible for the key-space to which the key belongs is located (*Retrieve(key)*) and then the found peer notifies its replicas about the inserted *key* using a light-weight hybrid push-and-pull gossiping mechanism. Deleting and updating a data item works alike.

Besides search for exact keys, P-Grid also supports substring search and range queries of arbitrary granularity, which are specifically important in the context of discovering Grid resources. In fact, range queries come quite natural with P-Grid's underlying trie abstraction, which is a common data structure in databases to enable good clustering and efficient query processing. Given a query for a range $[lb, ub]$ we can use a sequential (*min-max*) or a parallel strategy (*shower*) to resolve the query. In the sequential case we just have to locate a peer which is responsible for $h(lb)$ using *Retrieve(key, p)*, and then step through the leave nodes of the trie until we hit a peer responsible for $h(ub)$. The complexity in terms of messages for this approach would be $O(\log n)$ for locating the first peer plus one message for each peer in the interval. The parallel strategy would first determine the longest common prefix of $h(lb)$ and $h(ub)$. When this prefix has been resolved in the routing process, the range query has reached the sub-trie that holds the range and thus the query can be forwarded to the peers in the sub-trie in parallel. This requires a bit more messages but reduces latency. Both algorithms are efficient in terms of message complexity, can answer queries of arbitrary granularity, and are independent of the size of the queried range and only depend on the size of the result set. Detailed analyses of the message complexity of both approaches along with an experimental evaluation of the implementation on PlanetLab are given in [6].

4. Our Approach

Our approach can be based on an overlay network constructed and maintained by peers participating in a Grid or peers devoted explicitly for resource management. It may be sufficient to use Grid nodes to manage the resources but dedicated nodes may improve the overall performance because they are independent of the Grid's utilization and load. If dedicated nodes are used, they should still be distributed among institutions and across administrative boarders to avoid a single point of failure regarding network connectivity. The number of peers would depend on the number of nodes in the Grid and the number of users and jobs which can be very high for P-Grid. With overlay networks it is as well possible to start with a smaller setup and add peers as needed without requiring reconfiguration. Temporary peer failures due to hardware maintenance or defects are compensated by the built-in replication and failure resilience of P-Grid (replication of data and routing information). Job requests and queries to find suitable jobs may be issued from any peer in the overlay, i.e., all peers can act as entry points providing equal performance characteristics. However, to avoid overloading a single peer, requests should be issued to peers uniformly and randomly which is easy to achieve.

4.1. Job advertisements

In our setup users define their jobs by the number of required CPU cycles and optional resource requirements such as disk space or free memory, i.e., the number of required CPU cycles is used as the key for indexing. The number of requirements attached to the key (CPU cycles) is not limited and can be extended but everyone must use the same vocabulary to describe them. A possible job advertisement could then be given as `CPU_cycles=3500,disk=50MB,mem=1024MB, advertiser=http://need.cpu.com/job42`. Job advertisements are stored in the overlay network and resource providers can look for matching jobs they are willing to process. Assuming that CPU cycle requirements are distributed over a certain range, different key distributions must be handled. To show the applicability of P-Grid, we compared a uniform distribution of CPU cycle requirements which typically is easy to handle, with a highly skewed Pareto or Zipf distribution, for example, if most job advertisement are at the maximum of possible values and then sharply decrease. We chose these distributions to show the insensitivity of our approach towards standard distributions that can occur frequently in practice (P-Grid's performance remains logarithmic even for such skewed distributions as described in Section 3). Due to space limitations, however, we can only present these two extreme cases.

4.2. Job matching

Computers having idle CPU cycles use the overlay network to find the next suitable job(s) they can process. A job is suitable for a computer if it not exceeds the number of required CPU cycles and the computer fulfills optional job requirements such as disk space, free memory, etc. However, the concrete process to decide this can be handled by each node individually. To find a suitable job, a computer could use any thinkable strategy. For example, if a computer would only want to service jobs inside a certain range to optimize resource usage and simplify scheduling, then this would result in a simple range query. If a computer would accept any job up to a given maximum, then again this would be a range query with 0 as the lower bound and telling the overlay to only return a certain number of hits to not overload the network (this can be enforced very easily). Job advertisements could also include priorities to enable a computer to select the most relevant one. Once a matching job has been discovered by any strategy, the computer willing to process it, would simply contact the advertising node, who then can remove the advertisement from the overlay and send the job to the node which contacted it.

5. Experimental evaluation

From the above description of our approach it is clear that it most critically depends on the efficient implementation of range queries. In this section we present experimental results for range query efficiency from a large-scale deployment of P-Grid on the PlanetLab [4] infrastructure. PalnetLab offers a planet-wide distributed testbed for practical testing of large-scale distributed systems. We show results for both types of range query algorithms (min-max, shower) for two extreme key distributions. The implementation used in our experiments is available from `http://www.p-grid.org/`.

In the experiments we used a network of 250 peers each running on a dedicated physical PlanetLab node. We inserted 2500 unique data items into the overlay, i.e., each peer should be responsible for 75 data items on average given an average replication factor of 5. Due to this replication of data which is necessary to compensate for node and network failures we had a total of 12500 data items in the system. To show that the algorithms work for any data distribution, we used two different data sets, one uniformly distributed and one Pareto distributed (with a probability density function of $\frac{a\ k^a}{x^{1+a}}$ and parameters $k = 1$ and $a = 2.0$) as shown in Figure 2. Pareto is a typical long-tail distribution which occurs frequently in query distributions. We will see in the experiments that P-Grid is insensible to such distributions due to the efficiency of the underlying load-balancing algorithm which balances both storage and replication load. We can thus safely infer that if the results are good for a Pareto distribution, the system will perform equally well for other frequent long-tail distributions, e.g., Zipf.

Each peer selected randomly 10 data items of a global set according to one of these distributions. The peers then constructed a P-Grid which had an average height of $5 * \log_2 \frac{2500}{10*5} = 5.6$. Then range queries which affected data from all partitions of the data sets were issued. The queries were started from random peers with random lower range bounds, and were constructed in a way, such that they would return 50, 100, 150, 200, 400, and 800 data items. For each

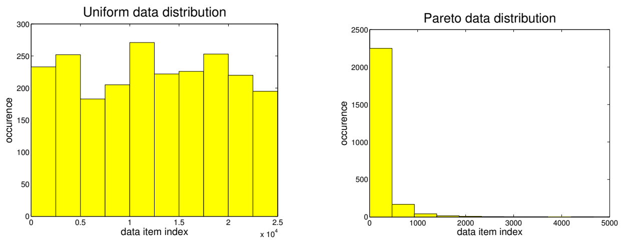

Figure 2. Data set distributions

of the six answer set sizes, each of the two distributions, and each of the two algorithms, one query was issued by each of the 250 peers, i.e., a total of $6*2*2*250 = 6000$ queries resulting in 250 values per data point in the figures below. The main objective of our experiments were to demonstrate the cost/latency trade-off of the algorithms, and to show, that because of the use of a trie-structured overlay network, the cost of range queries is independent of the data distribution and the size of the range, but only dependent on the used algorithm and the size of the answer set.

Figure 3 shows the costs incurred by range queries in terms of message latency (hops), i.e., the maximum number of messages required to hit each sub-partition of the range, i.e., one peer in each sub-partition. The error bars in the plot represent the standard deviation.

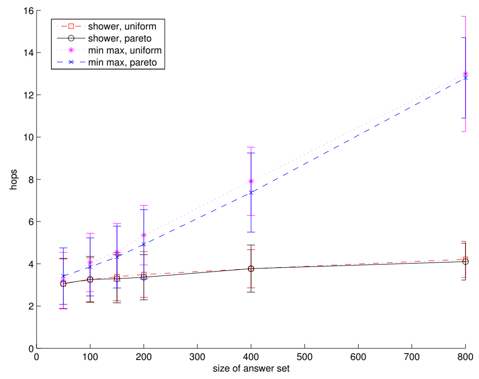

Figure 3. Message latency (hops)

On average we need 3 hops to reach a responsible peer for both types of algorithms, but the min-max algorithm then suffers a bit from the sequential traversal of the range to reach all sub-partitions after reaching the lower bound. This leads to increasing hop counts with increasing range sizes, whereas for the shower algorithm the number of hops remains constant, i.e., it is rather insensitive to the size of the answer set. However, this benefit comes at the cost of an increase in the overall messages as shown in Figure 4.

The shower algorithm requires a slightly higher number

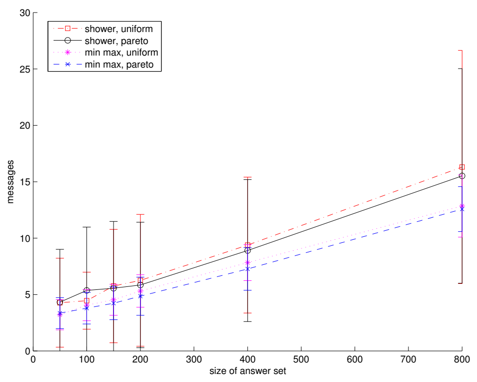

Figure 4. Message cost

of messages but improves latency as it sends them to the responsible peers in parallel. Therefore all peers responsible for a range are reached after 3 hops (in the experiment's setup) independent of the range size. Range queries with an answer set size of 50 are answered mostly by one peer because peers on average are responsible for 50 to 100 data items. It can further be seen that both algorithms perform equally well for both data distributions and scale well. An increase of the answer set size by a multiplicative factor of the average peer storage size yields an additional message on average which is the best possible result achievable with limited storage available at the peers.

6. Conclusions

We have presented an approach of using the P-Grid structured overlay network for resource discovery in Grids based on advertising jobs in the overlay. To demonstrate the applicability and efficiency of this approach, we presented experimental results from a large-scale PlanetLab deployment of our implementation. From the experimental results we can observe that the approach scales well, the cost and latencies are low, and the system offers an efficient decentralized discovery service in a real-world networking scenario.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems*, 2001.

[2] K. Aberer. Scalable Data Access in P2P Systems Using Unbalanced Search Trees. In *Proceedings of the 4th Workshop on Distributed Data and Structures (WDAS'2002)*, 2002.

[3] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: A Self-organizing Structured P2P System. *ACM SIGMOD Record*, 32(3), 2003.

[4] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3), July 2003.

[5] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *International Conference on Distributed Computing Systems (ICDCS)*, 2003.

[6] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer. Range queries in trie-structured overlays. Technical Report IC/2004/111, Ecole Polytechnique Fédérale de Lausanne (EPFL), 2004.

[7] A. Gupta, D. Agrawal, and A. E. Abbadi. Distributed Resource Discovery in Large Scale Computing Systems. In *SAINT*, pages 320–326, 2005.

[8] A. Iamnitchi and I. T. Foster. On fully decentralized resource discovery in grid environments. In *GRID '01: Proceedings of the Second International Workshop on Grid Computing*, pages 51–62, London, UK, 2001. Springer-Verlag.

[9] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.

[10] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*, pages 104–111, June 1988.

[11] J. Risson and T. Moors. Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. Technical Report UNSW-EE-P2P-1-1, University of New South Wales, Sydney, Australia, Sep. 2004. `http://www.ee.unsw.edu.au/~timm/pubs/robust_p2p/submitted.pdf`.

[12] O. D. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi. A Peer-to-peer Framework for Caching Range Queries. In *ICDE*, pages 165–176, 2004.