PePeR: A Distributed Range Addressing Space for Peer-to-Peer Systems*

Antonios Daskos, Shahram Ghandeharizadeh, and Xinghua An

Department of Computer Science, University of Southern California,
Los Angeles, California 90089
{daskos,shahram,anx}@usc.edu

Abstract. This paper describes a **Pe**er-to-**Pe**er **R**ange (PePeR) addressing space to process the "select" relational algebra operator. PePeR consists of several novel design decisions to support both exact-match and range selection predicates. First, it constructs Z ranges per node in order to efficiently route predicates in a decentralized manner. Second, it employs interleaved range declustering to minimize mean time to data loss in the presence of node removal(s). Third, it uses innovative techniques to adjust its addressing space in the presence of node insertion. The insertion of nodes is done in a distributed manner and we present a technique that approximates a uniform distribution of records across the nodes. In addition, we present performance numbers from PePeR and compare it with a distributed hash table (DHT). The obtained results show the following. If the workload of a relation is dominated by range predicates then PePeR is a superior alternative. On the other hand, if the workload of a relation is primarily exact-match retrievals (selection predicates using the equality comparison operator), then a DHT provides better performance. Both PePeR and DHT may co-exist in a peer-to-peer system simultaneously, enabling a database specialist to employ DHT with one relation and PePeR with another.

1 Introduction

A key characteristic of peer-to-peer (P2P) systems is their flexibility to enable a node to join and leave without impacting the overall system. These systems are taxonomized [LCC+02,CS02] into: 1) Centralized systems that employ a central directory server for processing queries, e.g., Napster, 2) Decentralized systems that route queries using their peers. The decentralized systems are further categorized into addressable and ad-hoc addressing. The addressable systems maintain a logical mapping between the location of data and network topology, e.g., OceanStore [KBC+00], CAN [RFH+01], Chord [SMK+01], etc. The ad-hoc addressing systems maintain no mapping and employ flooding of queries to retrieve relevant data, e.g., KaZaA, Gnutella, etc. The focus of this study is on decentralized, addressable P2P systems. To simplify discussion, for the rest of this paper, the term P2P refers to this specific category.

P2P systems may employ a Distributed Hash Table (DHT) [LNS96,PRR97,RFH+01,TJ02,KBC+00] to control placement of data across nodes for efficient retrieval. This is specially true for comparison operators that employ an equality predicate, termed exact-match selection predicates. For example, with a P2P network consisting of thousands of nodes, one may apply a DHT to the name of audio and video clips to assign each to a node. Subsequently, when a user requests a clip by referencing its name, the peers employ the DHT to route the request to the peer containing the referenced clip. This distributed routing is important because it minimizes the amount of state information maintained by a peer, enabling one or more nodes to join and leave without centralized control.

One may employ a DHT with relational databases to control placement of records and processing of exact-match selection predicates. In its simplest form, given a table $S(a_1,a_2, ..., a_n)$, a hash function is applied to the partitioning attribute of each record of S, say $S.a_1$, to map this record to a d dimensional order-preserving Cartesian space. The address space is mapped to the peers in the network, dictating the placement of S's records. An exact-match selection predicate referencing $S.a_1$ such as $S.a_1=C$ (C is a constant) might be initiated at one of the peers, say N_i. N_i applies the hash function to C to compute its hash function value, $V=h(C)$. If V is contained by the range of Cartesian space assigned to N_i then N_i processes the predicate. Otherwise, N_i compares V with the ranges assigned to its neighbors and routes

* This research was supported in part by an unrestricted cash gift from Microsoft Research.

the predicate to the neighbor with the minimum numerical distance to V. This process repeats until the predicate arrives at the node containing the relevant data (distance = 0).

A limitation of DHT is its lack of support for range predicates that reference $S.a_1$. While a system may query each discrete value in a range using DHT, this approach is perceived as infeasible in most cases. Typically, these predicates must be directed to all those nodes containing a fragment of S. This is undesirable for those predicates that retrieve only a few records because many (potentially millions of) nodes will search their repository only to find no relevant data. In addition to wasting resources, this results in a high response time because the underlying overlay network must route this predicate to every single node.

The primary contribution of this paper is PePeR, a distributed addressing space in support of range predicates. PePeR assigns Z ranges of the partitioning attribute of S, $S.a_1$, to each node. It supports distributed routing of both exact-match and range predicates that reference $S.a_1$, along with insertion and removal of nodes in a decentralized manner. In order to enhance availability of data in the presence of node removals, PePeR employs interleaved placement to construct sub-ranges of each range. The placement of these sub-ranges is a natural consequence of how PePeR defines the neighbor relationship between nodes, see Section 3.

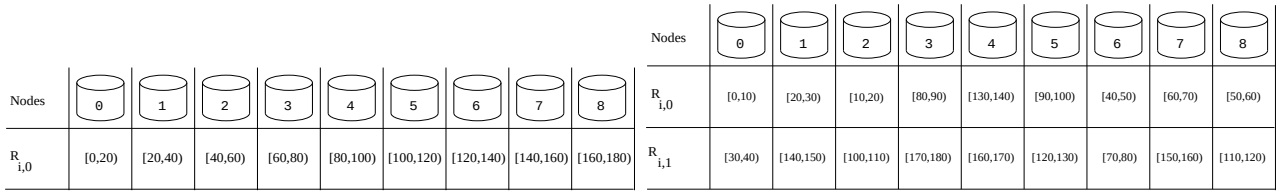
PePeR shares its roots with our prior work on placement of data in multiprocessor DataBase Management Systems (DBMSs), MAGIC [GDQ92,MS98]. PePeR is novel and different because data placement algorithms in multiprocessor DBMSs are centralized in nature. In addition, they lack concepts such as distributed routing of a predicate from one node to another. This concept is fundamental to PePeR (and P2P systems in general).

P2P systems have been the focus of recent study by the database community. For example, processing of complex queries is described in [HHH+02]. Techniques to compute approximate results for complex queries by finding data ranges that are approximate answers are the focus of [GAE03]. Both studies are different because they assume a DHT as their underlying data placement and routing strategy. Hybrid systems, where some functionality is still centralized constitutes the focus of [YGM01]. A similar approach for management of documents is described in [TXKN03].

Most relevant studies are [AS03,AX02,CFCS03]. Skip graphs [AS03] extend an alternative to a balanced tree, skip lists [Pug90], to provide an order-preserving address space on a collection of nodes. When compared with PePeR, skip graphs are different for several reasons. First, PePeR constructs Z ranges per peer (in a distributed manner), in order to facilitate intelligent routing. Skip graphs, on the other hand, construct pointers from one range to several other ranges to accomplish the same. Second, skip graphs construct a hierarchical data structure that might require a node removal to include the participation of many more nodes than its neighboring nodes. With PePeR a node removal impacts $4Z$ nodes. Note that PePeR and skip graphs are complementary and can be combined into a hybrid approach. For example, our interleaved declustering scheme can be used with skip graphs to increase the availability of data. (Availability of data is not considered in [AS03].)

With [AX02], all peers agree on a d-dimensional Hilbert curve that maps a range of $S.a_1$ to CAN's d-dimensional Cartesian coordinate space. Given a query, a peer employs the space filling curve to identify those coordinates of a CAN's Cartesian space containing the relevant data. Next, it employs CAN's routing mechanism to direct the predicate to these nodes. Similarly, MAAN [CFCS03] supports range queries by mapping neumerical attribute values to Chord [SMK+01] DHT. MAAN relies on Chord's SHA1 hashing to assign an identifier consisting of m bits to each participating node. However, when mapping objects using a numerial attribute value, MAAN employs a locality preserving hashing function to assign an object in the m-bit space. PePeR is novel because it is an alternative to a DHT and independent of either CAN or Chord (as such, it is more comparable with skip graphs [AS03]). We intend to quantify the tradeoff associated with these alternatives when compared with PePeR in the near future.

The rest of this paper is organized as follows. Section 2 illustrates the key elements of PePeR using an example. Section 3 presents the terminology required to present PePeR. Section 4 presents a class of decentralized techniques to insert a node into an existing PePeR configuration. In Section 5, we present a performance study of PePeR and its parameters. We also compare PePeR with a flooding environment that directs a predicate to all nodes of P2P system. We conclude with brief remarks in Section 6.



1.a Z=1 1.b Z=2

Fig. 1. PePeR with nine nodes and two different Z values.

2 Overview of PePeR

PePeR constructs a range addressing space for a table $S(a_1, a_2, ..., a_n)$ using one of its attributes, say a_1. This attribute is termed S's partitioning attribute. As an example, Figure 1 shows PePeR with a nine node P2P configuration. This figure assumes $S.a_1$ is an integer attribute with values ranging from 0 to 180. Each node N_i is assigned a range R_i, defined as $[l_i,u_i)$. This notation implies $S.a_1$ value of those rows assigned to node N_i is equal to or greater than l_i and less than u_i. A collection of records corresponding to range R_i is termed a fragment. For example, in Figure 1.a, the fragment assigned to node 1 contains those rows with $S.a_1$ values greater than or equal to 20 and less than 40. With PePeR, a node employs a range to route a predicate and a fragment to process a predicate. We start this overview by describing how the system performs routing in a distributed manner. Next, we explain node removal and data availability. Subsequently, we describe how a new node is inserted into PePeR. This discussion uses terms neighbor, containment, partial containment, and distance by defining them informally. The precise definition of these terms is provided in Section 3.

Distributed predicate routing:

To route a predicate in a distributed manner, PePeR requires a node to maintain the Z ranges assigned to each of its neighbors. The neighbors of a node are defined as those with consecutive ranges (see Section 3 for a formal definition of neighbors). For example, in Figure 1.a, node 4's (N_4) neighbors are N_3 and N_5. N_4 stores the ranges assigned to N_3 and N_5. Figure 2.a shows this neighbor relationship using a double arrow. The two edges of an arrow point to identical values that indicate the consecutiveness of two ranges. (The dashed arrow is an exception and described in the following paragraphs.) The range of a selection predicate P, i.e., $\sigma_{S.a_1 \geq l_P \wedge S.a_1 < u_P}(S)$, is defined as $[l_P, u_P)$. Assume P's range is [10,15) and is initiated at N_4 of Figure 1.a. N_4 routes this predicate towards N_0 by routing it to the neighbor with smallest distance to P, i.e., N_3. N_3 repeats this process by computing the distance between P and its neighbors in order to route P to N_2. This process terminates when P arrives at N_0 because N_0's range contains P. At this point, N_0 processes the predicate using its fragment to produce results.

It is important to distinguish between routing and processing of a predicate P. In our example, P was routed using four different nodes (N_4, N_3, N_2, and N_1) to arrive at N_0 for processing. A node routes P by comparing its distance with the ranges assigned to its neighbors. N_0 processed P by searching its assigned fragment for those records with a value greater than or equal to 10 and less than 15. A node may both process and route a predicate when P partially overlaps its range. For example, if P's range is [15,22) then N_1 both processes P and routes P to N_0 (assuming P was initiated at N_4).

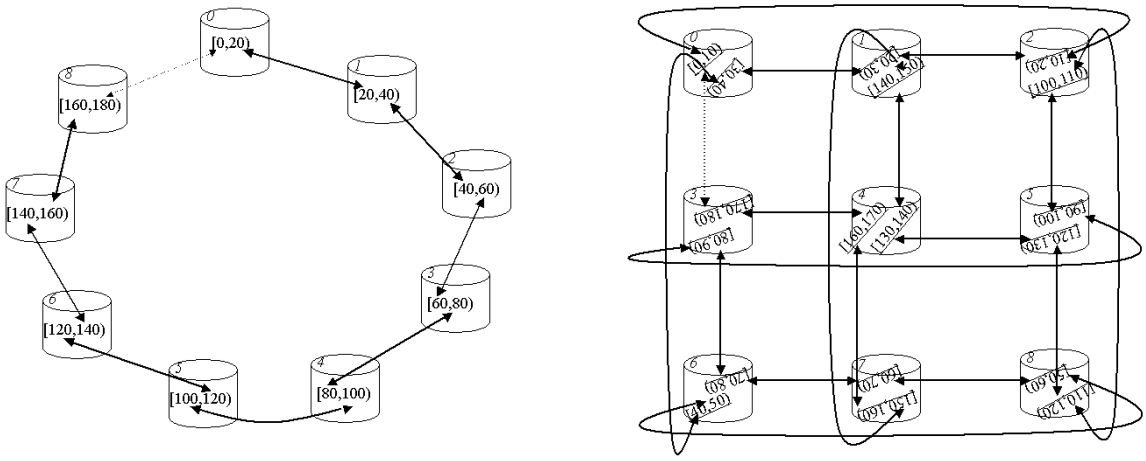
PePeR may assign Z ranges to each node where Z is a positive integer. In Figure 1.a, Z is equal to one. Figure 1.b shows a partitioning of S with 2 ranges assigned to each node, Z=2. We denote the Z ranges assigned to a node i as: $R_{i,0}$, $R_{i,1}$, ..., $R_{i,Z-1}$. For example, N_0 is assigned two ranges: $R_{0,0}$=[0,10), and $R_{0,1}$=[30,40). This results in a Z dimensional space that dictates $2Z$ neighbors for each node, see Figure 2.b. For example, in Figure 2.b, N_4's neighbors are N_1, N_3, N_5, and N_7. The assignment of ranges to nodes is paramount because it impacts the average number of hops. A naive assignment such as round-robin results on an average of $\frac{N}{4}$ hops to process a predicate (independent of Z's value).

Figure 2.b shows the logical neighbor relationship obtained from the assignment of Figure 1.b. Note that consecutive ranges are neighbors. We have intentionally directed each edge of an arrow to a specific value in each node to show this relationship. Once again, consider the routing of the range predicate P [10,15) when initiated at N_4. PePeR's routing mechanism employs distance to guide the predicate towards the peer with relevant data. It employs the distance between P's range and the ranges assigned to N_4's neighbors and

forwards P to the node with closest range. In our example, the range [20,30) assigned to node N_1 is closest and P is routed to this node. N_1 repeats this process to route P to N_2. N_2 processes P because its range [10,20) contains P.

If each node maintains the minimum and maximum values for $S.a_1$ then PePeR realizes a torus (i.e., the dashed line of Figure 2). This might be used for routing a predicate. To illustrate, in Figure 1.a, if the system maintains 180 and 0 as $S.a_1$'s maximum and minimum value, respectively, then it may process a predicate by visiting nodes in the reverse order of range values. For example, if a range predicate P is interested in those rows that satisfy the range [175,177), and is directed to node N_2, then it is routed through N_0 to arrive at node N_3 in two hops. In the absence of $S.a_1$'s minimum and maximum value at each node, P is routed to N_1, followed by N_4 to arrive at N_3. In this example, the presence of torus saved one hop. However, this is not true at all times.

Maintaining the minimum and maximum $S.a_1$ values at each node requires the propagation of these values to all nodes every time they change. If the value of minimum and maximum values becomes inconsistent due to their overlapped propagation while routing predicates, the correctness of routing is **not** impacted. This impacts the number of hops required to route P to the node containing the relevant data.



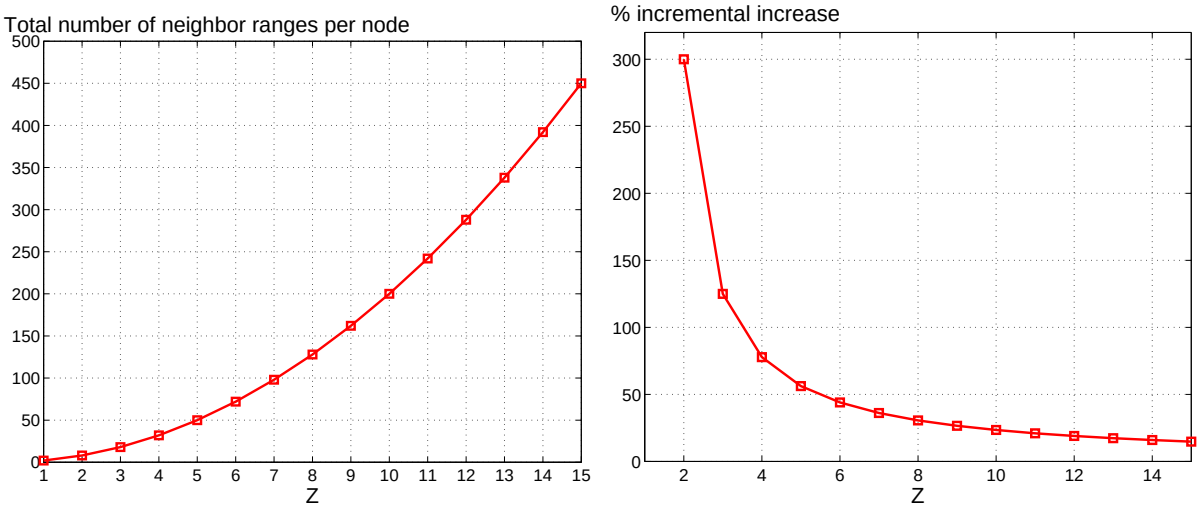
2.a Z=1 2.b Z=2

Fig. 2. Logical neighbor relationship imposed by PePeR with different Z values on the nine nodes using assignment of Figure 1. Each disk represents a node. The dashed arrow corresponds to a logical neighbor relationship that exists if and only if all nodes are provided with the minimum and maximum values for the partitioning attribute.

The distributed nature of PePeR prevents it from computing the shortest path for a predicate P even with a smart assignment and a torus. This is because our routing looks only one step ahead to choose its next hop. This local decision making might route P to more nodes than necessary. As an example, consider the assignment of Figure 1.b and an exact-match predicate P that retrieves $S.a_1$=40. Moreover, assume P is initiated at node N_4. Our routing algorithm must route P to N_6 containing the relevant data. Looking at the logical graph of Figure 2.b, the shortest route consists of two hops: either (1) $N_4 \rightarrow N_3 \rightarrow N_6$, or (1) $N_4 \rightarrow N_7 \rightarrow N_6$. However, using the concept of distance, our greedy routing algorithm is misled with the local optimal distance and routes P from N_4 to N_1 because its distance is closer. This causes P to incur 3 hops: $N_4 \rightarrow N_1 \rightarrow N_0 \rightarrow N_6$.

With PePeR, a node maintains $2Z^2$ ranges assigned to its $2Z$ neighbors. (These ranges are used for routing predicates.) For example, in Figure 2.a, a node maintains the two ranges assigned to its two neighbors. In Figure 2.b, each node maintains the 8 ranges assigned to its four neighbors. As one increases Z's value, the number of ranges known by a node increases (see Figure 3.a), reducing the number of hops incurred by a predicate. The percentage reduction in hops as a function of Z's value levels off with larger Z values. This is because the incremental percentage increase in the number of ranges stored at a node diminishes as a

function of Z, see Figure 3.b. For example, when one increases the value of Z from 1 to 2, there is a 300% increase in the number of neighbor-ranges known by a node. When we increase Z from 2 to 3, the number of ranges known by a node increases by another 125%. When we increase Z from 5 to 6, this percentage increase is only 44%.



3.a Number of neighbor ranges per node

3.b Percentage incremental increase

Fig. 3. While the number of neighbor ranges per node increases dramatically as a function of Z, the percentage incremental increase in the number of ranges per node starts to level off. Percentage incremental increase with Z=i is defined as $\frac{i^2-(i-1)^2}{i^2} * 100$ where $i > 1$.

Node removal:

In order to prevent loss of data in the presence of node removals and/or failures, we employ interleaved declustering. The basic idea here is to construct additional replicas of a fragment and partition it across multiple nodes. This enables PePeR to distribute the workload of a failed node across multiple nodes, enhancing the overall scalability of the framework. Interleaved declustering partitions a range assigned to a node into 2 fragments and assigns it to those 2 neighbors (out of $2Z$ neighbors) that are the logical neighbors of this range. This assignment is performed in a manner that preserves the logical neighbor relationship in presence of node removals. To illustrate, with the logical assignment of Figure 2.b, the range 130-140 assigned to node 4 is range partitioned into [130,135) and [135,140). The first, [130,135), is assigned to N_5 because it is the logical neighbor of the range [120,130) assigned to N_5. The second, [135,140), is assigned to N_1 because it is the logical neighbor of the range [140,150). Similarly, the range [160,170) assigned to N_4 is partitioned into [160,165) and [165,170), and assigned to nodes N_7 and N_3, respectively. Once N_4 is removed, its load is imposed onto four different nodes: N_1, N_3, N_5, and N_7, see Figure 4.b

With PePeR, a node monitors the presence of its neighbors using a heart-beat message. When one or more nodes detect the absence of a neighbor, they enter the process of range-merging. For example, when node N_4 is removed from Figure 1.a, one or more of its neighboring nodes (one of N_5, N_1, N_3, N_7) enters range-merging. During this process, they elevate their backup fragments to a primary status, extending their logical range to cover N_4's range. This in turn changes the logical neighbor relationship to that of Figure 4.b.

Node insertion:

PePeR may insert a node N_{new} into its existing address space in a variety of distributed ways. One approach is for the new node to insert P probes into the system to sample the existing ranges of a table. Each probe visits W nodes by performing random walks. Upon visiting a node, a probe collects the ranges assigned to that node and its neighbors along with other relevant statistics such as the cardinality of a range. N_{new} uses this information to choose Z ranges. Next, N_{new} contacts the owner of each range to break its range into two and assign one of the new ranges and the records belonging to that range to N_{new}. N_{new} may

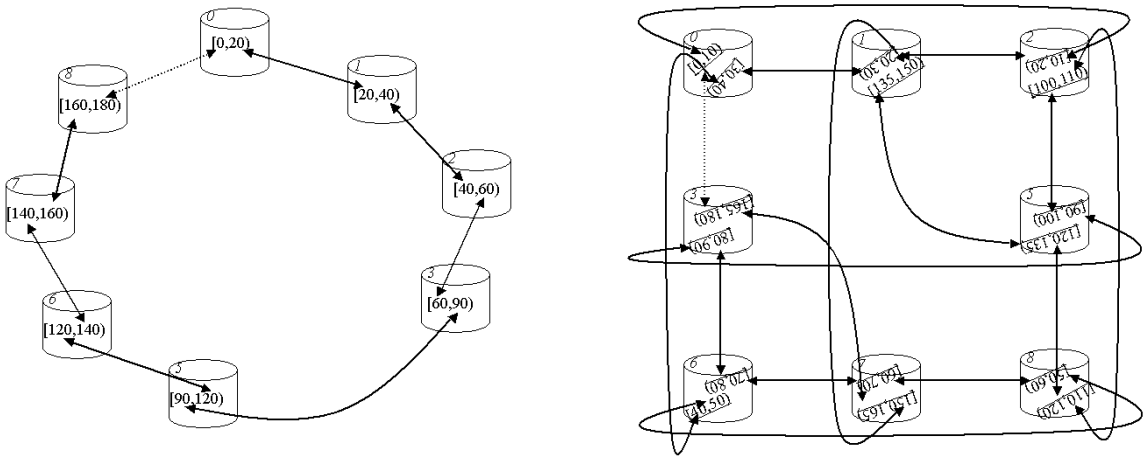


Fig. 4. With PePeR, removal of a node (say N_4) impacts only its neighbors and their logical relationship.

employ different policies when choosing the Z ranges. Section 4 presents three alternative policies and their performance tradeoffs.

3 Definitions and Terminology

This section presents two formal concepts essential to PePeR, namely: (a) neighbor relationship, and (b) the quantification of distance between a node's range and a predicate's range. We use the examples of Section 2 for illustration purposes.

PePeR assumes ranges are consecutive. Thus, given a range $[l_i,u_i)$, there must exist (1) an l_k equal to u_i unless u_i is the maximum value for the partitioning attribute, and (2) a u_k equal to l_i unless l_i is the minimum value for the partitioning attribute. To illustrate, in Figure 1.a, there might exist no records with l_i value equal to 40, however, the system continues to maintain the range [40,60) for node 2 in order to prevent holes and ensure consecutive nature of the ranges. Node 0 (8) satisfies this constraint because its l_0 (u_8) is the minimum (maximum) value for the partitioning attribute.

Two nodes N_i and N_k are neighbors iff one of these four conditions hold true: either (a) $u_i = l_k$, (b) $u_k = l_i$, (c) u_k and l_i are the the maximum and minimum value for the partitioning attribute, respectively, or (d) reverse of (c) where u_i and l_k are the maximum and minimum value for the partitioning attribute, respectively. For example, in Figure 2.a, neighbors of node 2 are nodes 1 and 3 because u_2=l_3 and l_2=u_1. Similarly, neighbors of node 8 are nodes 7 and 0 because u_7=l_8 and u_8 is the maximum partitioning value and l_0 is the minimum partitioning value. In essence, our definition of neighbors forms a torus when the minimum and maximum value of $S.a_1$ is known by all nodes (see the following paragraphs).

Given a range selection predicate P, its range is defined as $[l_P, u_P)$, i.e., $\sigma_{S.a_1 \geq l_P \ \wedge \ S.a_1 < u_P}(S)$. The range of a node N_i **contains** this predicate iff $l_i \leq l_P$ and $u_P \leq u_i$. For example, given a predicate [40,50), N_2's range (see Figure 1.a) contains this predicate. N_i's range **partially-contains** this predicate if either: (a) $l_i < u_P$ and $u_i < u_P$, or (a) $l_P \leq l_i$ and $l_i < u_P$. To illustrate, given a predicate [150,170), the range assigned to node 7 (see Figure 1.a) partially contains this predicate. The reverse of these definitions also apply where a predicate P may either contain or partially-contain the range assigned to a node N_i.

Given a predicate P and node N_i's range $R_{i,j}$, we define their **distance**, denoted $\Delta(P, R_{i,j})$, to be zero if $R_{i,j}$ either contains or partially contains P, or vice versa. Otherwise, the distance between P and $R_{i,j}$ is computed as follows. Assume $S.a_1$'s maximum and minimum values, denoted as $a_{1,min}$ and $a_{1,max}$, respectively, are known by all nodes. First, we compute the length as $Length$= $a_{1,max}$-$a_{1,min}$. Next, we compute the expected median M_P of the range specified by P, M_P=$\frac{l_P+u_P}{2}$. Next, we compute δ_1=M_P-l_j, δ_2=M_P-u_j, δ_3= $Length$ - δ_1, and δ_4= $Length$ - δ_2. Distance of P and R_i is defined as the minimum of

these deltas, $\Delta(P,R_i)=min(\delta_1,\delta_2,\delta_3,\delta_4)$. To illustrate, in Figure 2.a, assuming a range predicate [160,170), distance of M_P (165) relative to N_0's range, R_0=[0,20), is 15. This is because Length=180, resulting in: 1) δ_1=165, 2) δ_2=145, 3) δ_3=15, 4) δ_4=35, and 5) $\Delta(P,R_{1,0})$ = min(165,145,15,35) = 15. The distance of this predicate relative to N_2's range is 55: 1) δ_1=125, 2) δ_2=105, 3) δ_3=55, 4) δ_4=75, 5) $\Delta(P,R_{2,0})$ = min(125,105,55,75) = 55.

If $S.a_1$'s maximum and minimum values are unknown, a node may employ the theoretical positive and negative values of an attribute domain to compute Length in the above methodology. For example, if the age attribute of a table is an unsigned short (16 bit integer), then PePeR may assume 0 and $2^{16}-1$ as the minimum and maximum values of this attribute.

4 Insertion of Nodes

This section describes a decentralized class of techniques, termed probe-based, to insert a new node (say N_{new}) into an existing PePeR configuration. As implied by its name, the primary characteristic of this approach is its use of P probes to identify the Z candidate ranges along with their fragments that will be assigned to N_{new}. A probe performs W random walks from a randomly chosen node, N_{Start}, of the existing configuration. The walks are performed by choosing a random neighbor of the current node, starting with N_{Start}. The termination condition of a probe is satisfied when it either (1) performs W walks, or (2) encounters a node whose neighbors have already been visited by this probe. When a probe encounters one of these termination conditions, it transmits all its accumulated ranges to N_{new}. Once N_{new} has received these ranges, it chooses Z ranges among them. Next, it requests the owner of each chosen range to split its range into two parts and assign one to N_{new}. A simple approach to split a range might be to choose the median that divides that range into two equi-sized fragments.

How N_{new} chooses the Z ranges from the accumulated ranges differentiates the alternative probe-based approaches. Here we describe three alternative policies: Random, Uniform and Elastic. All three strive to minimize the likelihood of N_{new} neighboring the same node multiple times unless it is impossible for the current configuration to satisfy this constraint, e.g., current configuration consists of fewer than Z nodes. In order to achieve this goal, a policy must be aware of the neighbors of each node. Thus, a probe gathers two sets of ranges: set $\{p_1\}$ consisting of the ranges assigned to the visited nodes, and set $\{p_2\}$ consisting of the ranges assigned to the neighbors of visited nodes. If a probe visits W unique nodes, set $\{p_1\}$ will consist of ZW ranges, while set $\{p_2\}$ will consist of a maximum of $2Z^2W-2Z(W-1)$ ranges, depending on the number of distinct nodes that are neighbors to the W visited nodes. This maximum value is realized when there exists W total common neighbors between any two nodes visited by the W walks of a probe. In most cases the cardinality of $\{p_2\}$ will be less than this maximum.

Random, the simplest policy, computes the union of sets $\{p_1\}$ and $\{p_2\}$. Let set $\{p\}$ denote this union, i.e., $\{p\}=\{p_1\}\cup\{p_2\}$. As implied by its name, Random chooses Z ranges from $\{p\}$ randomly. If it selects two or more ranges that belong to the same node then one is kept and the remaining ones are discarded. The discarded ones are replaced by re-applying random to the remaining elements of $\{p\}$. This process may repeat several times until all Z chosen ranges belong to different nodes. If $\{p\}$ becomes empty then new probes are generated to accumulate additional ranges. The new probes might be tagged with the identity of the already known nodes in order to prevent a probe from visiting them again. If this fails ω attempts, a technique proceeds to choose the remaining ranges from the available list (causing N_{new} to neighbor a single node multiple times).

Uniform, the second policy, strives to construct equi-sized fragments. It requires the original probe to gather the cardinality of a fragment for each range in $\{p_1\}$ and $\{p_2\}$. Uniform computes the union of sets $\{p_1\}$ and $\{p_2\}$, $\{p\}=\{p_1\}\cup\{p_2\}$. It sorts $\{p\}$ in descending order based on the cardinality of each range. Next, it selects the first Z ranges with highest cardinality that belong to different nodes. If this fails, Uniform selects the first Z ranges from set $\{p\}$ (those with highest cardinality), causing N_{new} to neighbor the same node multiple times. An obvious variation of Uniform might generate new probes in case of failures to accumulate new ranges (similar to Random). An investigation of this variations is a future research direction.

Elastic is an adaptable technique with tunable parameters that controls both the fragment size and the numerical distance between the Z ranges assigned to a node. It is designed to strike a compromise between the following two objectives: (1) construction of equi-sized fragments that have at least $\lceil\frac{RS}{2}\rceil$ records, and (2) minimization of the number of hops when routing a predicate. Consider each objective in turn. The first

is similar to Uniform with two differences: First is the parameter RS that requires Elastic to choose Z ranges each with more than RS records. Second, Elastic chooses Z ranges only from set $\{p_1\}$.

To realize the second objective, Elastic uses γ to control the variation in the distance covered by the Z chosen ranges. Its details are as follows. First, from set $\{p_1\}$, it identifies the range with the highest number of records, R_h. With multiple candidates, it chooses one randomly. Next, it identifies each range R_i by its mean, m_i=$\frac{l_i+u_i}{2}$, and computes its distance d_i from R_h, $d_i = |m_i - m_h|$. Let A denote the maximum d_i. Elastic chooses z distances from N_h, each denoted as D_i such that:

$$D_i = \begin{cases} 0, \ if \ i = 0 \\ D_{i-1} + \gamma^{a_i} d, \ 1 \leq i \leq z-1, \ 0 \leq a_i \leq z-2, \ a_i \neq a_j \ for \ i \neq j \end{cases} \quad (1)$$

where $d = A\frac{\gamma-1}{\gamma^{z-1}-1}$. This ensures $D_{z-1} = A$. At the same time, Elastic requires the Z identified ranges are chosen such that N_{new} does not neighbor the same node multiple times. It employs both sets $\{p_1\}$ and $\{p_2\}$ for this purpose. To achieve this, the algorithm evaluates all possible combinations of Z ranges from Z different nodes and choose the best one. An obvious variation of Elastic is to choose the first Z elements from the union of $\{p_1\}$ and $\{p_2\}$. An investigation of this alternative is a future research direction.

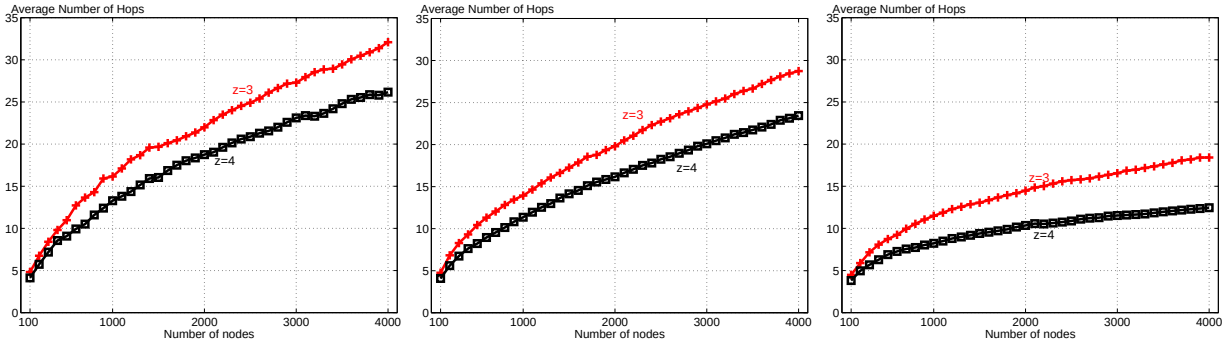
We illustrate Elastic with an example. Consider a tenth node joining the PePeR network of Figure 2.b. Assume its probe visits nodes N_4, N_5 and N_8, constructing set $\{p_1\}$ with six elements: $p_{1,0}$=$R_{8,0}$=[50,60), $p_{1,1}$=$R_{5,0}$=[90,100), $p_{1,2}$=$R_{8,1}$=[110,120), $p_{1,3}$=$R_{5,1}$=[120,130), $p_{1,4}$=$R_{4,0}$=[130,140) and $p_{1,5}$=$R_{4,1}$=[160,170). Elastic picks the range with the highest number of records, say $p_{1,4}$, i.e., R_h=$p_{1,4}$=[130,140). The mean of this range is 135, m_h=135. Next, it computes the mean for each range in set $\{p_1\}$ and their distances d_i from m_h, resulting in the following: $\{m_0$=55, d_0=80$\}$, $\{m_1$=95, d_1=40$\}$, $\{m_2$=115, d_2=20$\}$, $\{m_3$=125, d_3=10$\}$, $\{m_4$=135, d_4=0$\}$, $\{m_5$=165, d_5=30$\}$. A equals the maximum d_i value, in this case, d_0=80. Assuming the value of γ is 10, Elastic computes D_i as follows: $d = 80\frac{10-1}{10^{2-1}-1}$=80, $a_0 = 0$, D_0=0 and $D_1 = 80$. This leads the new node to target ranges $R_{4,0}$=[130,140) and $R_{8,0}$=[50,60) because $|d_4 - D_0| + |d_0 - D_1|$ is smaller than any other choice of ranges and they belong to different nodes. Next, the new node sends a request to nodes 4 and 8 to: (a) split each identified range into two, and (b) assign each new range and its appropriate fragment to the new node.

In this example, with $Z = 2$, the computation of D_0 and D_1 is independent of γ because it is raised to power of 0 in Equation 1. For $Z \geq 3$ the value of γ impacts the choice of ranges assigned to the new node. Consider what happens when Z=3 and the probe gathers the same ranges as before. Now, $A = d_{max} = 80$, resulting in $d = 7.273$ and $D_0 = 0$, D_1=7.273 and D_2= 80, assuming a_0=0 and a_1=1. We can see that D_2=A, thus ensuring that the last of the chosen ranges is as far as possible from the first one, which is either R_h or one close to R_h. The γ value will guide the algorithm into choosing the middle (second) range. Depending on its value, that choice will be closer to either the first or the third one. Based on the three calculated values, the chosen ranges would then be $R_{4,0}$, $R_{5,1}$ and $R_{8,0}$ because they minimize $|d_4 - D_0| + |d_3 - D_1| + |d_0 - D_2|$ and belong to different nodes.

4.1 A Comparison

We compared these alternative insertion techniques in a simulation study. Our implementation is flexible enough to support different numerical data types. Other data types such as string can be supported by mapping alphabets to a numerical data type. In our experiments, the partitioning attribute of a table is a real number uniformly distributed between 0.0 and 1.0. We start by placing this table on one node. Next, we increase the number of nodes from 1 to 4000 by inserting nodes one at a time. In all experiments the number of probes (P) equals Z, and W equals 10. Every time the number of nodes in our configuration becomes a multiple of hundred, we measure the average number of hops required to route 100,000 randomly generated range predicates.

Figure 5 shows the number of hops with Random, Uniform, and Elastic with two different Z parameter values, 3 and 4. This figure shows the average number of hops as a function of the number of nodes for a table consisting of one million records. Elastic was configured with γ=15. These results show Uniform and Random perform approximately the same number of hops. Elastic results in fewer hops. This is because Elastic controls the distance between the ranges assigned to a newly inserted node. With smaller γ values (less than 5), Elastic results in the same average number of hops as Uniform and Random.



5.a Random 5.b Uniform 5.c Elastic

Fig. 5. Average number of hops with the alternative insertion strategies.

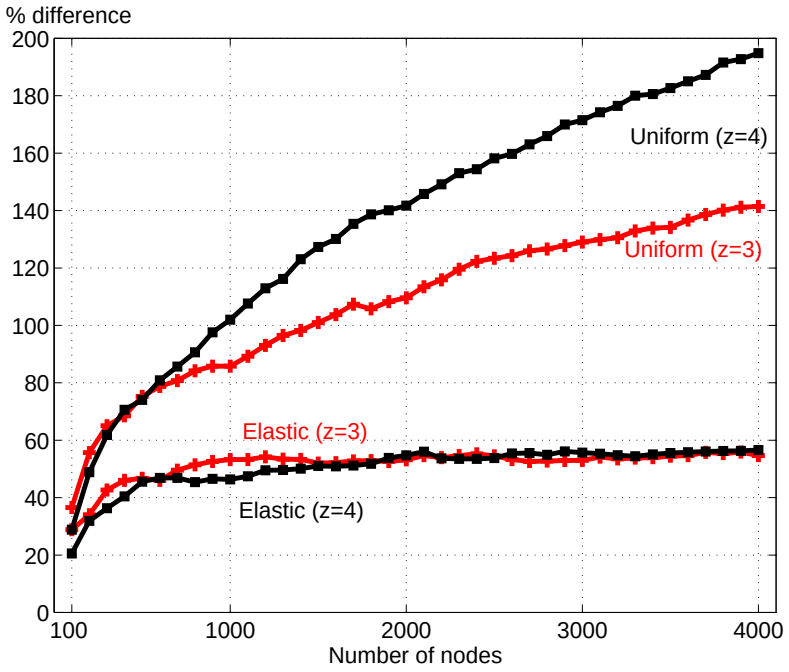
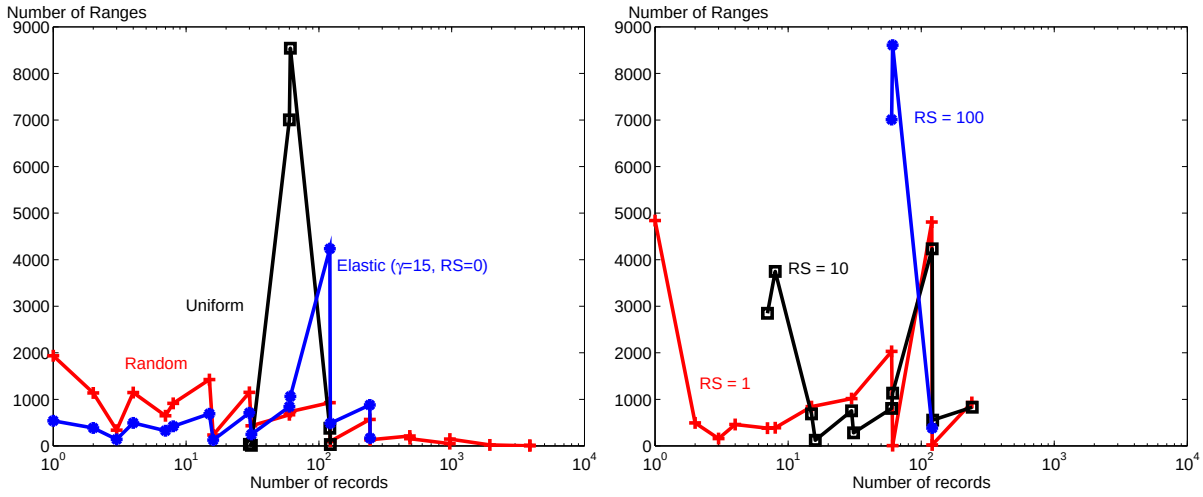


Fig. 6. Percentage difference in number of hops relative to the theoretical lower bound. If h_e and h_t denote the number of hops with Elastic and the theoretical lower bound, respectively, then percentage difference is defined as $100 \times \frac{h_e - h_t}{h_t}$.

The theoretical lower bound on the number of hops with PePeR is $\frac{d}{4}\sqrt[d]{\mathcal{N}}$ (similar to CAN [RFH+01]). Figure 6 shows the percentage difference between the number of hops observed with Uniform and Elastic relative to this theoretical lower bound. The results show that Elastic remains a fixed distance away from the theoretical lower bound with an increasing number of nodes. This stable characteristic of Elastic is a desirable advantage.



7.a Alternative insertion techniques

7.b Elastic with different RS values

Fig. 7. Distribution of records across the ranges. This is shown as the number of ranges for each unique range cardinality.

Figure 7.a shows the number of records for the partition corresponding to each range with Z=4. (The obtained results for Z=3 is similar and eliminated from this presentation.) Uniform constructs ranges with six different sizes: 30, 31, 60, 61, 121, and 122 records. More than 95% of the ranges contain either 60 or 61 records. Random results in non-equi sized ranges that vary from 0 to 3875 records. Elastic reduces this variability by limiting the cardinality of the largest range to 243 records. (Elastic continues to construct ranges with zero records.)

With Elastic, the variability in the range sizes can be controlled by manipulating the value of RS. Figure 7.b shows the distribution of records across the ranges with different RS values. Note that with RS=100, Elastic constructs ranges that behave the same as Uniform. With RS=1, Elastic avoids ranges with zero records. However, it does construct many ranges with one record. Different RS values have marginal impact on the average number of hops observed with Elastic. Figure 8 shows the percentage difference between RS=0 and three different RS values: 1, 10, and 100. The percentage difference is almost always lower than 10%.

5 A Comparison with DHT

When compared with a DHT, the primary benefit of PePeR is its ability to direct range predicates to the node(s) containing relevant data. A DHT, on the other hand, must direct these predicates to all nodes containing a fragment of a relation. For those predicates that retrieve only a few rows, this forces many nodes to search their fragments only to find no relevant data, reducing the overall processing capability of the nodes. Figure 9 shows the response time (RT) as a function of arrival rate for an open simulation model consisting of 256 nodes. It shows PePeR with two different Z values, 2 and 4. It also includes the scenario where a predicate is routed to all nodes when CAN is configured with $d = 4$. Note that CAN is not designed to do flooding; we performed the flooding to simulate a CAN when processing a range predicate. We term this configuration All-Nodes. In this model, the network time required to route a message is 5 milli-seconds

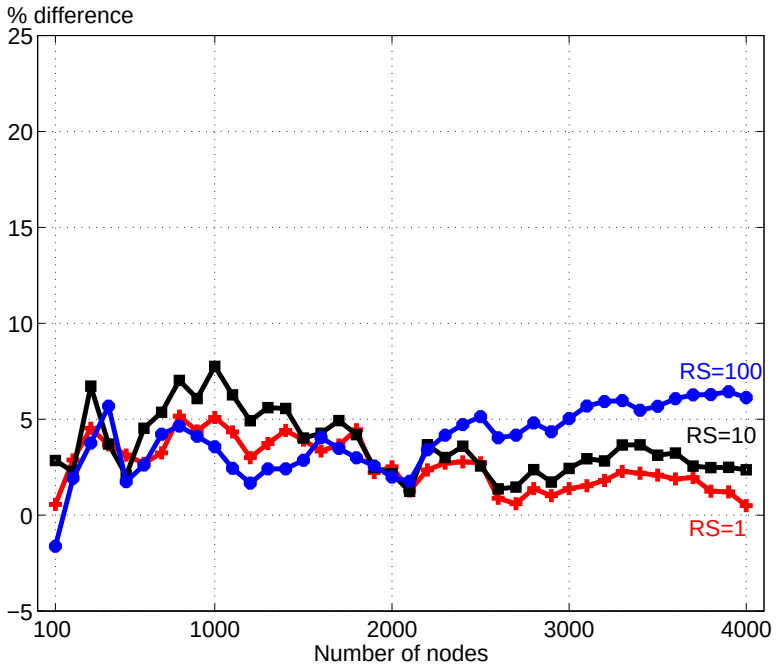


Fig. 8. Percentage difference in the number of hops with Elastic and RS=0 when compared with three different RS values 1, 10 and 100. Assuming h_i denotes the number of hops with a given RS=i value, percentage difference between RS=0 and RS=100 is defined as $100 \times \frac{h_0 - h_{100}}{h_0}$.

(msec). The time required by a peer to process a predicate is 20 msec. With a small arrival rate (λ=2), the average response time when the query is routed to all nodes is 221 msec. With PePeR when Z=2 (Z=4), the average response time is 257 (140) msec. The response time with All-Nodes is better than PePeR with Z=2 because PePeR must perform 9 hops as compare to a maximum of 8 hops with All-Nodes. (PePeR performs 9 hops because of the local optimal routing decision discussed in Section 2.) When Z=4, PePeR performs 4.6 hops, outperforming All-Nodes.

With a high arrival rate, All-Nodes becomes fully utilized with an arrival rate of 40 predicates per millisecond because all 256 nodes must process each predicate. With PePeR only the nodes containing relevant data search for data, freeing other nodes to perform useful work. With Z=2, the system starts to observe hot spots and bottlenecks with arrival rates between 440 and 460 predicates per millisecond. With Z=4, the number of hops is smaller, relieving additional nodes from the overhead of routing to process predicates. This enables the system to process additional predicates to support a higher arrival rate ranging 800 and 850. These results make a convincing case for the use of PePeR with range predicates.

6 Conclusion and Future Research Directions

This paper describes PePeR in support of range queries in a decentralized, addressable P2P system. We outlined Random, Uniform and Elastic as decentralized techniques to enable a new node to join an existing PePeR address space. The obtained results demonstrate both the flexibility and superiority of Elastic. Results from an open simulation model show the superiority of PePeR over a DHT for a workload dominated by range predicates.

Our short term research directions are three folds. First, we intend to study techniques to publish a relation with an existing PePeR configuration. Second, we plan to qualitatively evaluate whether minimizing the number of hops is the right criterion for database applications. For example, when inserting nodes, our design objective is to maintain a reasonable number of hops. It is not clear whether this objective would enhance response time (or throughput) of a P2P system. An alternative objective might be to distribute the workload of a relation more evenly across the nodes based on resource heterogeneity [ZG00,VBW98,GGGK03]. We intend

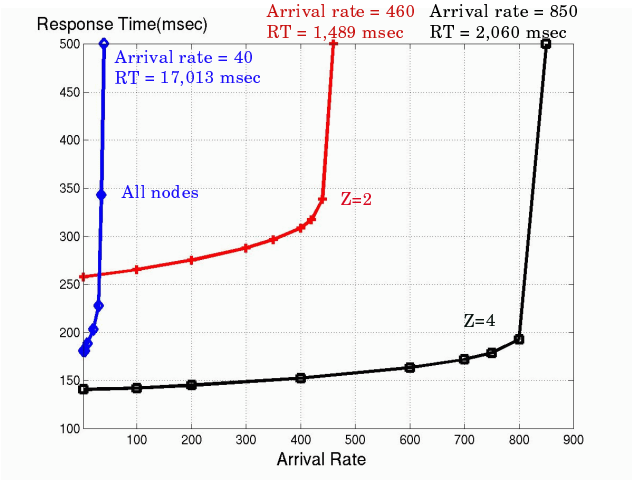


Fig. 9. Response time as a function of arrival rate for a 256 node configuration.

to study extensions of PePeR with this alternative. Third, we hypothesize PePeR and Skip Graphs [AS03] to be members of a more general framework. We intend to investigate this framework and its hybrid instances of PePeR and Skip Graphs.

References

[AS03] J. Aspnes and G. Shah. Skip Graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 2003.

[AX02] A. Andrzejak and Z. Xu. Scalable, Efficient Range Queries for Grid Information Services. In *Proceedings of the Second IEEE International Conference on Peer-to-Peer Computing (P2P2002)*. Linkping University, Sweden, September 2002.

[CFCS03] M. Cai, M. Frank, J. Chen, and P. Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. In *Proceedings of the Fourth International Workshop on Grid Computing (Grid 2003)*. Phoenix, Arizona, November 2003.

[CS02] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of the ACM SIGCOMM*, August 2002.

[GAE03] A. Gupta, D. Agrawal, and A. El Abbadi. Approximate Range Selection Queries In Peer-to-Peer Systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research*, Asilomar, California, United States, January 2003.

[GDQ92] S. Ghandeharizadeh, D. DeWitt, and W. Qureshi. A Performance Analysis of Alternative Multi-Attribute Declustering Strategies. In *Proceedings of the 1992 ACM-SIGMOD Conference*, May 1992.

[GGGK03] S. Ghandeharizadeh, S. Gao, C. Gahagan, and R. Krauss. High Performance Parallel Database Management Systems. In J. Blazewicz, W. Kubiak, T. Morzy, and M. Rusinkiewicz, editors, *Handbook on Data Management and Information Systems*. Springer, 2003.

[HHH+02] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *Proceedings of First International Workshop on Peer-to-Peer Systems*, March 2002.

[KBC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[LCC+02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the* 16^th *Annual ACM International Conference on Supercomputing*, 2002.

[LNS96] W. Litwin, M.-A. Neimat, and D. A. Schneider. LH* - A Scalable, Distributed Data Structure. *TODS*, 21(4):480–525, 1996.

[MS98] B. Moon and J. Saltz. Scalability Analysis of Declustering Methods for Multidimensional Range Queries. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):310–327, March 1998.

[PRR97] C. Plaxton, R. Rajaram, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, June 1997.

[Pug90] W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *ACM SIGCOMM*, 2001.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, 2001.

[TJ02] M. Theimer and M. Jones. Overlook: Scalable Name Service on an Overlay Network. In *The 22nd International Conference on Distributed Computing Systems*, July 2002.

[TXKN03] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and Nikolaos Ntarmos. Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research*, Asilomar, California, United States, January 2003.

[VBW98] R. Vingralek, Y. Breitbart, and G. Weikum. Snowball: Scalable Storage on Networks of Workstations with Balanced Load. *Distributed and Parallel Databases*, 6(2):117–156, 1998.

[YGM01] B. Yang and H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. In *The VLDB Journal*, pages 561–570, sep 2001.

[ZG00] R. Zimmermann and S. Ghandeharizadeh. HERA: Heterogeneous Extension of RAID. In *In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, June 2000.