SkiPeR: A Family of Distributed Range Addressing Spaces for Peer-to-Peer Systems [*]

Antonios Daskos, Shahram Ghandeharizadeh, Ramin Shahriari
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
{antonios.daskos, shahram, rshahriari}@usc.edu

Abstract

A relational database management system based on a peer-to-peer network must support both exact-match and range selection predicates efficiently. One approach to process range predicates is to design and implement a distributed range addressing space in support of decentralized routing of predicates, insertion and removal of peers. To route predicates in O(log N) with a N peer system, two alternative approaches have been proposed. The first, Skip Graphs [1, 16], employs a hierarchical order-preserving structure on the peers and their assigned ranges. The second, PePeR [8], constructs multiple ranges per peer and controls their assignment across the peers intelligently. The primary contribution of this paper is SkiPeR, a general purpose technique that unifies Skip Graphs and PePeR. SkiPeR consists of two key parameters whose extreme settings correspond to Skip Graphs and PePeR. There are in-between parameter settings that are a tradeoff between the amount of state information maintained by each peer, continued routing of predicates in the presence of peer removal, and the number of hops required to route a predicate. A secondary contribution of this study is to quantify these tradeoffs. This enables a system designer to choose parameters settings that fit the requirements of an application and the characteristics of the underlying peer-to-peer network.

1 Introduction

A diverse range of applications may utilize peer-to-peer (P2P) systems as a large scale distributed storage system. A key component of P2P systems is its addressing space that stores (value, item) pairs and routes predicates to retrieve relevant items. Ideally, an addressing space should route both exact-match and range predicates in O(log N) where N is the number of nodes. To illustrate, consider a digital library application with an interface that enables users to retrieve manuscripts based on author's last-name, affiliations, topic, key-terms, published-forum, and its number of citations. Each criterion is a value with a manuscript as its key. An example equality search predicate is to retrieve all those manuscripts that appeared in SEDE 2006, "published forum = SEDE 2006". An example range query is to retrieve all those papers with more than one hundred citations, "number of citations > 100". There might also be conjunctive or disjunctive predicates that are a combination of range and equality clauses such as retrieve all those manuscripts by Ghandeharizadeh with more than ten citations, "last-name = Ghandeharizadeh and number of citations > 10".

One may taxonomize P2P techniques to process both equality and range predicates along two dimensions. Its first dimension corresponds to the addressing space and separates those techniques that map data to a Distributed Hash Table (DHT) [20, 22, 24, 27, 18], termed DHT-based, from those that employ an order-preserving addressing space, termed interval-based. The second dimension describes applications and whether they can tolerate not finding relevant data sets when they are present in the peer-to-peer network. If so, these applications are termed Approximate (Approx). Otherwise, they are termed Precise. Thus, there are four categories: DHT-based/Approx [10, 25], DHT-based/Precise [5, 2, 28], Interval-based/Approx[1], and Interval-based/Precise [1, 16, 8]. SkiPeR unifies the alternative techniques belonging to the Interval-based/Precise category. In the following, we provide a

*This research was supported in part by NSF grant IIS-0307908, Annenberg fellowship, and unrestricted cash gifts from NCR and Microsoft Research. A shorter version of this paper appeared in *ISCA 15th International Conference on Software Engineering and Data Engineering*, (SEDE-2006), LA, CA, July 6-8, 2006.

[1]We are not aware of techniques specifically designed for Interval-based/Approx category.

brief survey of the remaining three alternatives.

With the DHT-based/Approx techniques, the results of prior range predicates are stored in a P2P network for subsequent retrieval by future predicates. A predicate may not find its results even though a prior range predicate contains it and its results exists in the P2P. In this case, the predicate might be directed to another system containing the referenced table. Both Chord and CAN have been investigated by [10] and [25], respectively, for approximate range query processing. The concept of forwarding is introduced in [25] to increase the percentage of queries that find their data in the P2P network. SkiPeR focuses on Precise applications, rendering these techniques outside the scope of this paper.

The DHT-based/Precise category of techniques either maps a range to a zone of CAN DHT using a Hilbert Curve technique [2] or employs Chord DHT with an order-preserving hash function [5, 28] to control the placement of records across the nodes. With these techniques, advance knowledge of the domain of partitioning attribute (its lower and upper bound) enhances the efficiency of routing. With SkiPeR, this advance knowledge makes no difference.

The Interval-based/Precise category is the focus of this study. It consists of Skip Graphs [1, 16], PePeR [8], and P-Ring [7, 19]. Skip Graphs extend and adapt Skip Lists [23] to realize a redundant hierarchical structure on the ranges assigned to the N nodes of a P2P network. The characteristics of this structure are controlled using a parameter termed alphabet size, denoted α. PePeR constructs Z ranges for each node. It assigns these ranges to the nodes intelligently in order to facilitate efficient routing of selection predicates. P-Ring [19] is similar to PePeR because it assigns multiple ranges to each node. At the same time, it strives to ensure the number of items stored in each peer is between sf and $2.sf$ where sf is the some storage factor in order to balance storage between peers.

The primary contribution of this paper is SkiPeR, a family of distributed range addressing spaces. SkiPeR unifies Skip Graphs and PePeR into one technique with two different parameters, α and Z. We do not try to incorporate P-Ring because, in our assumed environment, it is difficult if not impossible for each peer to know the value of sf. The extreme setting of SkiPeR's parameters realize Skip Graphs (Z=1) and PePeR (α=∞). Other parameter settings ($Z \geq 2$ and $\alpha < \infty$) realize a hybrid of these two techniques. Routing of predicates with SkiPeR is different than the routing technique employed by either Skip Graphs or PePeR, see Section 2. Another contribution of this study is a host of availability techniques that enable the network to minimize data loss in the presence of peer removals. We consider both graceful and ungraceful node removals. With the first, a departing peer informs its neighboring nodes of its intention to leave the network and collaborates with these nodes to leave gracefully, leaving the networking addressing space and content in a consistent state. With ungraceful node removals, a peer leaves the network abruptly with no advance notice due to factors such as power failures and loss of network connectivity [2]. We introduce analytical models to quantify data loss as a function of Mean Time Node Removal (MTNR). Section 3 quantifies the tradeoff associated with SkiPeR's alternative parameter settings. It demonstrates a larger number of neighbors reduces the average number of hops to process a predicate, and increases the resiliency of the network to peer removals. SkiPeR is flexible enough to support many different configurations (richer than both Skip Graphs and PePeR), providing an application and its target environment with the flexibility to choose the right parameter settings to accomplish a pre-specified requirement.

The rest of this paper is organized as follows. Section 2 provides an overview of SkiPeR. Section 3 quantifies the performance tradeoffs associated with alternative parameter settings of SkiPeR. Brief conclusions and future research directions are contained in Section 4.

2 SkiPeR

SkiPeR's objective is to maintain an order preserving address space on a P2P network of devices to facilitates efficient processing of range and exact-match predicates, insertion and removal of nodes. By efficient, we mean a decentralized technique with either no or minimal overhead as a function of the number of nodes in the P2P network. This objective is accomplished in two ways. First, SkiPeR employs ranges to facilitate routing of a selection predicate to the node containing relevant data, termed target node. Second, SkiPeR maintains a logical, order-preserving relationship between ranges to facilitate O(log N) routing of a predicate to its target node(s). The later is accomplished using either the concept of neighbors, a divide-and-conquer approach, or both. These concepts can be designed and implemented in many ways. Alternative design decisions are a tradeoff in number of hops to route a predicate, incurred overhead when either inserting or removing a node, and percentage of failed searches when a fraction of nodes are simultaneously removed from the P2P network. In the following, we present a design and quantify its tradeoffs in Section 3.

[2] Graceful node removals are considered by [19]. It does not consider ungraceful node removals.

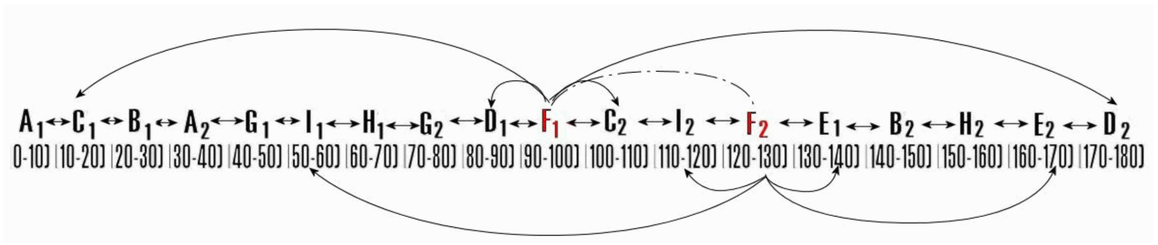


Figure 1: Assignment of 2 ranges per node for a nine peer configuration. Total number of ranges is 18.

Given a relation $S(a_1, a_2, ..., a_n)$, one of its attributes must be specified as its partitioning attribute, say a_1. This attribute is used to horizontally range partition records of S across the newly inserted nodes of a peer-to-peer network. Here, we simplify the discussion by assuming records are assigned to a newly inserted node immediately. In practice, the system may delay this partitioning (and assign either some metadata pertaining to ranges) until the credentials of the newly inserted node is established. This decision does no impact SkiPeR's design.

A range of $S.a_1$ contains a lower and an upper bound, denoted as $[l_i,u_i)$. The tuples corresponding to this range are termed a fragment of S. SkiPeR employs a divide-and-conquer approach to efficiently route a predicate by constructing more ranges than nodes. It may assign Z ranges to a node joining the P2P network [8]. The value of Z might be determined dynamically based on the capabilities of this node. Without loss of generality and in order to simplify discussion, assume Z is statically defined for the P2P network. Figure 1 shows 18 ranges of $S.a_1$ across a nine node P2P network with Z=2. In this figure, the nine nodes are numbered A, B, C, D, etc. A node appears twice, indicating the assignment of two different ranges to that node. For example, node A is assigned two different ranges: [0,10) and [30,40). Node F is assigned ranges [90,100) and [120,130). Two consecutive ranges are neighbors. A node must store its neighbor information (i.e., ranges) in order to route a predicate to its target node. Figure 1, shows node F and its reachability in the range address space. In addition to being able to reach ranges immediately adjacent to its assigned ranges (i.e., those ranges assigned to nodes D, C, I, and E), F may traverse the entire address space of the partitioning attribute value, e.g., it may jump to the either [160,170) or [50,60). This is because these ranges are assigned to its neighboring nodes (D, C, I, and E). Intermediate nodes collaborate to route a predicate towards a target node. (The routing mechanism is detailed in the following paragraphs and Figure 3.) As an example, if a predicate referencing rows in the range [42,45) is initiated at node F then F routes this predicate to node I because [50,60] is the closest interval. Subsequently, I forwards this predicate to node G. This results in two hops with node G processing this predicate.

SkiPeR constructs a hierarchical structure on the nodes to further reduce the number of hops to route a predicate. This hierarchy is also a divide-and-conquer approach to facilitate jumps in the decentralized range addressing space when routing a predicate, similar to traversal of a tree structure such as a B$^+$-tree [3] or a Skip list [23]. The traditional hierarchical data structures must be adapted to a P2P environment for two reasons. First, memory is not shared amongst nodes. Second, insertion and removal of a node should impact a few nodes in order to restore the integrity of the hierarchical structure. Similar to Skip Graphs, SkiPeR employs a variant of Skip Lists adapted to P2P networks. Each node generates random word by using a finite alphabet consisting of α letters. For example, a finite alphabet consisting of two letters (α=2) is $\{0, 1\}$. The ranges assigned to a node inherit this word. Figure 2 shows the hierarchical structure constructed on the ranges of Figure 1 with $\alpha = 2$. The word chosen by each node is shown below each range. The connectivity between the ranges at level 0 is order preserving and independent of a node's chosen word. Starting with level i where $i \geq 1$, ranges with identical first i letters constitute a list. For example, there are two lists at level 1, one starting with A_1 whose first letter is "0" and the other starting with B_1 whose first letter is "1". As another example, nodes A, D, and E belong to the same list at level 2 because the first two letters of their words are "00". As the number of levels increases (larger i values where $i \geq 1$), two trends are observed: First, the number of lists increases with a maximum of α^i lists at level i. Second, lists at level i are subsets of lists appearing at level $i - 1$.

Typically, members of a list at a higher level traverse a larger fraction of the address space. For example, at level 3, A may traverse from its assigned range of [30,40) to [130,140) assigned to node E. At level

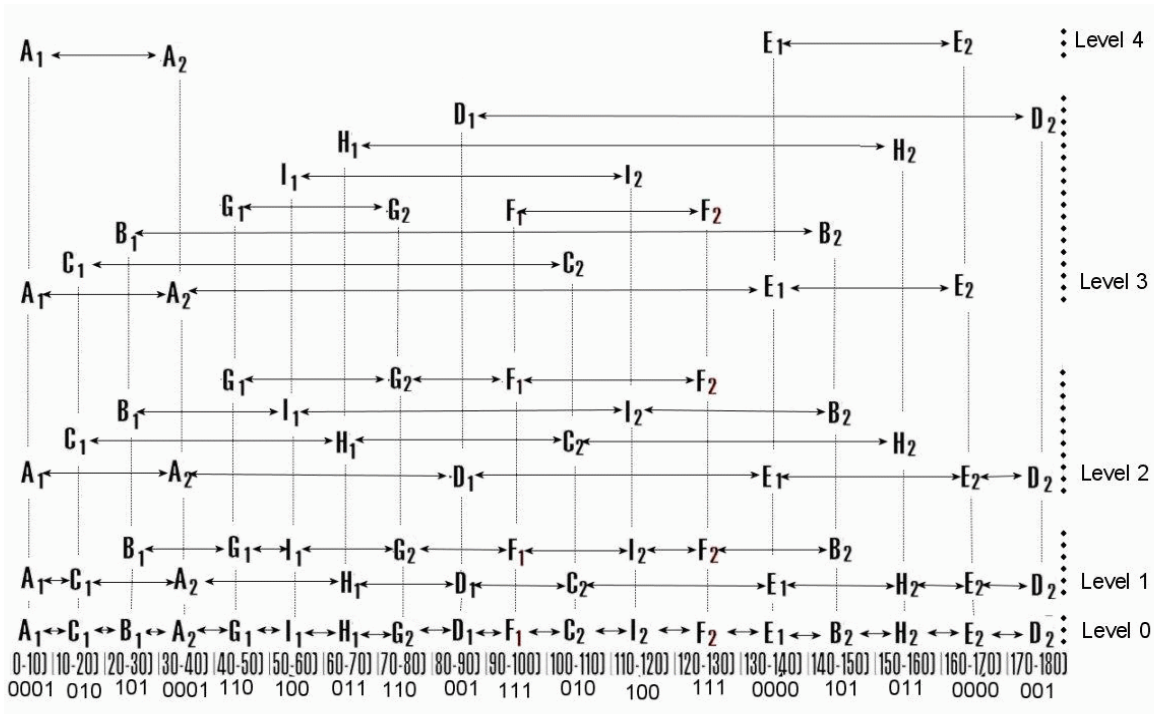


Figure 2: SkiPeR constructs a hierarchical structure across the ranges assigned to nodes.

2, A may traverse from [30,40) to [80,90) assigned to D. At level 1, A traverses from [30,40) to [60,70) assigned to node H. Finally, at level 0, A traverses to [40,50) assigned to node G. Note that all ranges of a node will belong to the same list independent of level id. This is because they share the same word assigned to a node. A future research direction is to explore alternative word assignments to a range.

This process of generating levels and lists is performed by each node. A node stops this process once a list at its highest level contains all ranges assigned to itself. In Figure 2, this causes nodes A and E to construct lists up to level 4, while other nodes stop at level 3. This termination condition holds true when the words assigned across nodes are unique. Its violation is detected when two or more nodes are repeatedly neighbors across different levels. At some level, a node may alter its chosen bit to produce unique words across nodes.

From Figure 2, note that a node may neighbor itself multiple times. For example, at level 2, node A neighbors itself (A_1 and A_2 appear in the same list). A physical implementation does not need to represent this repetition because node A is aware of its two assigned ranges. By increasing alphabet size, α, the number of lists at each level is increased, reducing the number of distinct neighbors per node. This reduction has two disadvantages. First, it increases the number of hops required to process a predicate. Second, it reduces resiliency of SkiPeR to node removals, by reducing the likelihood of the network to route a predicate to its target node even though it is alive and present in the network. At the same time, this reduction has the following advantage: it reduces the overhead of removing and inserting nodes by minimizing the number of neighbors that must be notified when a new node is either inserted or removed.

When a predicate is routed to a randomly chosen node (termed entry node, N_e), this node checks to see if one of its ranges contains the predicate. If so, it proceeds to process the predicate, producing results for the client. In addition, it may flood the predicate to the neighbors whose assigned ranges overlap the predicate. We do not discuss flooding mechanisms and refer the interested reader to [2] for a description and evaluation of three alternative techniques. If none of the ranges assigned to N_e qualifies, this node routes the predicate using the pseudo-code of SkiPeR, shown in Figure 3. This is achieved by N_e computing the distance between the predicate and the ranges assigned to its neighbors, see distance and ClosestRange in Figure 3. At each hop, the predicate is routed towards the target node containing the relevant data. Note that the pseudo-code of Figure 3 is flexible enough to detect when a predicate cannot find its target node due to either its removal or partitioning of the network.

We considered four different techniques when designing SkiPeR's routing technique. Appendix 1 details these and presents their experimental comparison. The obtained results show SkiPeR's routing technique is superior to other alternatives. In particular, this routing technique outperforms Skip Graphs' routing technique by more than 40% when Z=1. Skip Graphs did not develop a routing technique similar to the one employed by SkiPeR because they were designed for management of resources with grid computing (and not for processing of range predicates with a table). SkiPeR outperforms PePeR's routing algorithm because PePeR did not consider a multi-level hierachy.

Peer Removal: A peer might be removed from SkiPeR either gracefully or un-gracefully. The later might be due to power failures, shut downs, lost network connections, etc. We describe SkiPeR assuming an environment that supports both scenarios. With un-graceful removals, peers may loose data and the logical address space might become inconsistent. One may assume elaborate environments consisting of one or more centralized servers that are known by peers and referenced when un-graceful neighbor removals are detected. These are beyond the focus of this study. In the following, we start with a description of un-graceful peer removal because: 1) its design decisions impact a graceful peer removal, and 2) SkiPeR is almost always configured for an un-graceful peer removal.

With un-graceful peer removals, we address two key challenges. First, a class of replication techniques to restore the SkiPeR network. Second, a class of replication techniques to prevent the likelihood of data loss. Consider each in turn. A peer removal results in removal of Z ranges from the SkiPeR address space. For each range, two peers containing its neighboring range detect its removal (unless, it is a range containing either the lower or upper limit of the partitioning attribute value, in which case, one peer detects its removal). Each peer initiates the discovery of the other. The duration of this process is termed $T_{Discovery}$. Assuming the maximum number of levels is $\chi = Log_{\alpha}N$, a simple discovery strategy is as follows. When a peer detects removal of its right neighbor at level i, it generates a probe for each level 0 to χ to navigate one node to its right neighbor that is alive (level i is exempt). Next, each probe navigates to the left to identify the appropriate logical neighbor. This approach fails to recover the network when all the probes fail to identify the logical neighbor. In the example of Figure 2, when node F is removed, nodes C, D, I and E initiate the discovery process. Probes generated by C and D are designed to bridge the gap left open by F_1. The

```
distance(range $R$, key $\kappa$){
    $R$ is represented as $[l_R, u_R)$
    If ($l_R \leq \kappa \leq u_R$) return 0;
    Else if ($u_R \leq \kappa$) return $\kappa - u_R$;
    Else return $l_R - \kappa$;
}

ClosestRange (Node $N_i$, key $\kappa$){
    List all ranges $R_i$ owned by $N_i$ in ascending order,
    same as I-Level;
    For each $R_i$ compute $\delta_i$=distance($R_i$, $\kappa$);
    Let $m$ = min($\delta_i$);
    Discard all ranges with $\delta_i$ greater than $m$;
    If $u_i$ is the same for remaining ranges then
        return the largest (last) element;
    Else if $l_i$ is the same for remaining ranges then
        return the smallest (first) element;
    Else return the largest element where $u_i \leq \kappa$;
}

Route (predicate $P$, node $N_i$){
  Assume $P$ is of the form $[l_P, u_P)$;
  $\kappa = \frac{l_P + u_P}{2}$;
  Assume all ranges have a fixed order
  identified by their order in I-Level;
  $R_c$= ClosestRange($N_i$, $\kappa$); this computes $N_i$'s range
  closest to $\kappa$;
  CurrentDistance = distance ($R_c$,$\kappa$);
  {$R_n$} = Enumerate all ranges known by those neighbors
  of $N_i$ that are alive;
  For each element $e_i$ of {$R_n$}
    $\delta_i$ = distance($e_i$,$\kappa$);
  Let BestDistance = min($\delta_i$);
  If (BestDistance < CurrentDistance)
    Route $P$ to the neighbor containing the range that
    corresponds to the BestDistance;
    (with multiple candidates, choose one randomly);
  Else if (BestDistance > CurrentDistance)
    Routing has failed because of node removal;
    return SearchFailure;
  Else if (BestDistance = CurrentDistance) {
    If ($\kappa \geq u_{R_c}$) {
      Enumerate ranges belonging to the same list as $R_c$
      and positioned to its immediate **right**;
      Route $P$ to the node containing the range that appears
      at the highest level with distance from $\kappa$
        equal to BestDistance and $u \leq \kappa$;
    }
    Else if ($\kappa < l_{R_c}$) {
      Enumerate ranges belonging to the same list as $R_c$
      and positioned to its immediate **left**;
      Route $P$ to the node containing the range that
      appears at the highest level with distance from
        $\kappa$ equal to BestDistance and $l > \kappa$;
    }
  }
}
```

Figure 3: Pseudo-code for SkiPeR's routing technique.

same applies to probes generated by nodes I and E, recovering the gap attributed to F_2. Consider the probe generated by D only. This node generates probes to C at level 1, and E at level 2. These probes are searching for a peer with a range neighboring F_1. It may use the routing mechanism by searching for the value 90, the lower limit of the range assinged to F_1. The first probe succeeds immediately, restoring the neighbor relationship between D_1 and C_2 immediately. The other probe blocks at node E assuming the gap left open by F_2 has not been bridged. Once this gap is bridged, the probe may continue its search to identify C as the new peer of D. Otherwise, after a certain amount of time, it informs D of its current pending status. If all probes generated by C and D suffer from the same faith then SkiPeR has encountered a Logical Network Failure, LNF. Experimental results of Section 3 quantify LNF for a variety of configurations. With LNF, a peer may request its peers to search for its logical neighbor by repeating the above process. Ultimately, if this fails after a certain number of attempts, a peer may discard its data and rejoin one of the existing live nodes. This expensive operation should be prevented.

One approach to prevent $T_{Discovery}$ is to replicate the neighbor information transitively. This technique requires a peer at level i to maintain the identity of its right neighbor one hop away. The choice of either right or left is arbitrary and might be implemented using the ascending (or decending) order of ranges. When a peer detects removal of a right neighbor, it uses its transitive neighbor information to restore its logical neighbor relationship one hop away, repairing SkiPeR's network. With this replication technique, the discovery process is initiated when two adjacent ranges are removed at level i of SkiPeR's address space. This technique incurs extra overheads when nodes are inserted because an inserted node at level i must (a) identify its own transitive right neighbor and (b) update the node to the left that must point to it as its transitive neighbor. To illustrate this technique, D_1 might maintain the fact that C_2 is its transitive neighbor (by one hop) to the right at level 0. When D_1 detects removal of F_1, it contacts C_2 to re-establish its logical neighbor relationship at level 0. (Detection of a removed node is detailed in the following paragraphs.) This process repeats at each level. When both F and C (containing two adjacent ranges F_1 and C_2) are removed, the transitive information contained by D_1 becomes useless, preventing it from repairing the network at level 0. In this case, node D must inititiate the discovery process.

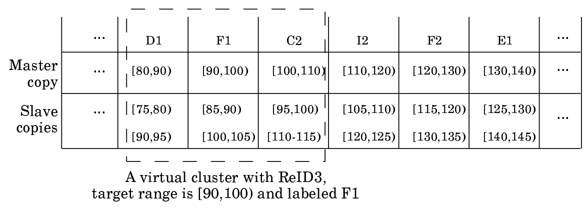
This approach might replicate neighbor metadata at only level 0. In this case, it is termed 1Cell replication. An alternative is to require this replication for each level, termed 1Slice replication. With one peer removal, 1Cell eliminates the need for the discovery process. It increases the number of messages required to insert a node by two extra messages. With two adjacent range removals, 1Slice minimizes the likelihood of the discovery process failing because by repairing the higher levels, it enhances the chances of a probe repairing the network. 1Slice increases the number of messages required to insert a node from $2Zlog_{\alpha}N$ to $4Zlog_{\alpha}N$. The transmission of messages are performed in parallel when inserting a peer.

Detection of a removed peer might be performed in either a lazy or an eager manner. With a lazy approach, a peer detects a missing peer when routing a predicate. The pseudo-code for routing a predicate performs this detection, see Figure 3. An eager alternative would require each peer to monitor its neighbors using a heart-beat message [24] to detect a node removal and determine if data is lost.

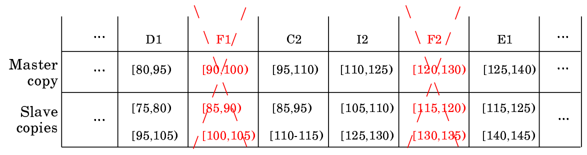
In order to minimize data loss, SkiPeR replicates a relation. A taxonomy of possible approaches is as follows: 1) adapt a variation of Mirroring [17, 4, 13], Interleaved declustering [6], Chained-declustering [14] to construct secondary copies of a table, 2) construct η copies of a table and prevent a peer from being assigned overlapping ranges for two or more copies of the table, and 3) a hybrid of these two techniques. In the following, we describe each approach in detail. Next, we derive analytical models to estimate the Mean Time to Data Loss (MTDL).

SkiPeR may construct backup replicas of data by borrowing concepts from parallel database management systems. For a survey of these techniques see [6, 15]. Here, we describe a Restricted form of Interleaved Declustering (ReID) with SkiPeR which enables each peer to construct virtual clusters in a decentralized manner. Each cluster consists of ω ranges, $1 < \omega \leq 3$. We use the cluster size to term the different variations of ReID: ReID3 denotes the version where a cluster consists of 3 ranges and ReID2 for a cluster consisting of 2 ranges. ReID3 is similar to Interleaved declustering while ReID2 resembles Chained declustering. With ReID3, a virtual cluster corresponds to one range, termed target range, and its logical right and left neighbors. For example, in Figure 2, the target range F_1 has two neighbors D_1 and C_2. In each cluster, ReID3 partitions the target fragment into two sub-ranges and assigns each sub-range to a neighboring fragment. In our example with Figure 2, the target range [90,100) is partitioned into two sub-ranges: [90,95) and [95,100). The first is assigned to D_1 because they are neighbors, and the second is assigned to C_2, see Figure 4.

When a peer, say C, detects failure of a neighbor, say F, it initiates SkiPeR's data recovery technique. During this process, the peers neighboring the

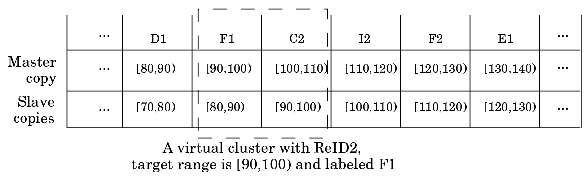


4.a Normal mode of operation

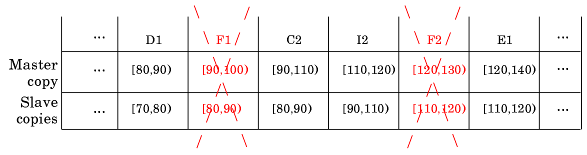


4.b After recovery from F's removal

Figure 4: Placement of ranges with ReID3.



5.a Normal mode of operation



5.b After recovery from F's removal

Figure 5: Placement of ranges with ReID2.

removed node: (a) promote the slave sub-ranges corresponding to the removed node to be masters, and (b) construct backup copies of these promoted sub-ranges and any slave sub-ranges that were assigned to the removed peer. The duration of this step is denoted T_{Copy}. To illustrate, Figure 4 shows un-graceful removal of F. After $T_{Discovery}$, D_1 and C_2 (I_2 and E_1) re-establish their neighbor relationship. Next, D promotes [90,95) to master status. The same is performed by C for [95,100), I for [120,135), and E for [125,130). These promotions are performed independently because all nodes neighbor F and have detected its removal. Subsequently, each node constructs a copy of the missing slave sub-ranges to prevent loss of data with future peers removals: 1) D constructs a slave copy of [85,95) on C, 2) C constructs a slave copy of [95,105) on D, 3) I constructs a slave copy of [115,125) on E, and 4) E constructs a slave copy of [125,135) on I. These can be conceptualized as two independent logical events: $D \leftrightarrow C$, and $I \leftrightarrow E$. While Steps 1 and 2 are most likely sequential, note that Steps 1 and 3 can be performed in parallel. The same observation applies to Steps 2 and 4. See Figure 4.b for the final placement of master and slave sub-ranges.

With ReID2, a cluster consists of two ranges: a target range and its right[3] neighbor. It also replicates the meta-data corresponding to the neighbors of a range with the slave copy. For example, ReID2 constructs a cluster consisting of F_1 and C_1, see Figure 5. F would assign a copy of the fragment corresponding to [90-100) to C. Moreover, C contains a copy of F_1's logical neighbor relationships at level 0. Similar to ReID3, ReID2 results in data loss when two adjacent nodes are removed. During recovery phase, ReID2 must construct slave copies of the newly promoted master and lost slaves. For example, if F is removed, node C must construct a copy of its promoted master range [90,100) on I, and copy the lost slave range [80,90) from peer D. For F_2, E must construct a copy of [120,130) on B and copy the lost slave range [110,120) from I. This can be conceptualize as two independent logical events: $D \rightarrow C \rightarrow I$, and $I \rightarrow E \rightarrow B$. The presence of I in both logical events is coincidental with this example because C_2 and I_2 are neighbors. With a larger number of nodes, the likelihood of this is minimized.

Duration of T_{copy} is a function of the network characteristics, load on the nodes participating in the construction of backup copies, and the size of fragment that must be transferred to different nodes. When one increases the value of Z with SkiPeR, a larger number of nodes participate in the recovery phase, preventing one node from processing a large amount of data. This distributes the workload of recovery due to a removed node more evenly across the available peers, preventing formation of hot spots and bottlenecks. Assuming an under utilized system, T_{copy} is determined by the fragment with the longest copying time. This is partially dependent on the size of the fragment and the RTT between the participating nodes. This can be computed with ReID2 and ReID3 for a specific instance of SkiPeR using the available analytical models for TCP [26, 21] which have been independently verified by [12]. One such a model is detailed in Section 3.

When compared with one another, both ReID2 and ReID3 transfer the same amount of data during T_{Copy}. Both techniques require the same number of messages during $T_{Discovery}$. During node insertion, when slave copies are originally constructed, ReID2 employs 3 peers for each range assigned to a peer. However, ReID3 employs 4 peers. This expedites the

[3]Either right or left neighbor, this choice is arbitrary.

insertion process when peers provide similar performance. When peers are heterogeneous, ReID3 might suffer from a longer node insertion time because it must wait for the slowest peer to complete its task. Finally, ReID3 results in a higher probability of data loss when compared with ReID2, see Section 3.

With the second approach, SkiPeR constructs η replicas of a table and satisfies the following constraint when inserting a new peer: The new peer may not contain overlapping ranges of two or more copies of the table. An overhead of maintaining multiple copies is to propagate updates to all copies. To ensure consistency, a copy might be designated as master and the remaining $\eta - 1$ copies as slaves [11]. All updates are directed to the master copy which then propagates them to the slave copies. When a search fails using the master copy, it is re-issued using a slave copy. In order to minimize response time, the system may issue searches against all copies in parallel. SkiPeR may reconstruct the lost fragment of the master copy as follows. A peer detects the missing neighbor and probes the network to find the neighbor at level 0 that it must re-connect with. For example, in Figure 2, if node F fails, two pairs of nodes, C and D along with I and E, must discover that they are neighbors at level 0. Each pair would enter a negotiation phase to either assume responsibility for the entire range or a subrange. For example, C and D may negotiate to decide that C should be assigned [90,95) and D should be assigned [95,100). Next, each node searches one of the $\eta - 1$ copies to retrieve the relevant records to restore the lost data. For example, the new range [90,95) assigned to node D is populated from a secondary copy that has the relevant records. In the worst case scenario, SkiPeR may not find the missing data from η-1 copies due to other node removals. In this case, the data is lost.

Mean Time to Data Loss (MTDL) is defined as the average time between loss of data. We derive a formula to compute MTDL by assuming the duration of recovery phase, denoted T_{rec}, is known. Next, we derive a formula to estimate T_{rec}.

The probability of any particular node being removed during T_{rec} is $p_{rem}=\frac{T_{rec}}{MTNR}$ where MTNR denotes the average time between the removal of a node. The probability of a particular node not being removed during T_{rec} is 1-p_{rem}. The probability of not incurring a second node removal within the same cluster of ReID is $(1-p_{rem})^{\omega-1}$ where ω is 3 and 2 with ReID3 and ReID2, respectively. Thus, $MTDL = \frac{MTNR}{1-(1-p_{rem})^{(\omega-1)}}$ when $\eta = 1$. With η copies of a relation and no ReID, all η copies of a range must fail in order for that range to be lost. Hence, $MTDL = \frac{MTNR}{\frac{p_{rem}}{\eta-1}}$. With both η copies and ReID, $MTDL = \frac{MTNR}{\frac{p_{rem}}{\eta-1}} + \frac{MTNR}{1-(1-\frac{p_{rem}}{\eta})^{(\omega-1)}}$ as long as SkiPeR assigns master and slave replicas of η copies of overlapping ranges to different nodes. Note that we assume MTNR is independent of the number of nodes in a SkiPeR network. Extension of our proposed framework with models where MTNR is dependent on N is trivial because it would be similar to those proposed for an array of magnetic disk drives, e.g., see Section 3.2 of [6].

Graceful peer removal is similar to un-graceful peer removal with one key difference: The discovery process is eliminated. Thus, $T_{Graceful} = T_{Bridge} + T_{Copy}$ where T_{Bridge} is the time required to repair SkiPeR's logical neighbor relationship at each level due to gaps attributed to the removed node. This assumes SkiPeR is deployed with either ReID2 or ReID3. Extensions of this discussion to configurations deployed without ReID and $\eta > 1$ is trivial and eliminated from this presentation.

3 Performance Evaluation

We employed a four dimensional experimental design to quantify SkiPeR's tradeoffs. It controls (1) the number of peers, (2) distribution of the partitioning attribute value, (3) the predicate and its selectivity factor, and (4) techniques employed for un-graceful peer removal, MTNR, and models that estimate network time. We conducted different experiments with different values for these dimensions. We analyzed configurations starting with one peer and increasing to 100, 1000, and 10,000 peers, observing 99, 999, and 9999 peer insertions, respectively. Once a configuration reaches its target size, we perform 100,000 random predicate lookups and measure the average number of hops to process these predicates. We also gather other system parameters such as the average number of neighbors maintained by a peer and the average number of ranges containing records relevant to a predicate.

We performed this evaluation using different table sizes with three alternative distributions of the partitioning attribute value: Uniform, Zipfian-Value, and Zipfian-Range. In the following, we focus on a table with one million records. A record is 10.24 Megabytes in size consisting of both traditional and multimedia attribute types, e.g., integers, text, audio and video clips, etc. Thus, this table is 9.77 Terabytes in size. With Uniform, the table consists of one million unique partitioning attribute values. With a Zipfian-Value, there are 1000 unique partitioning attribute values and the occurrence of each value is skewed. A Zipfian function with a mean of 1 was used to generate this distribution, causing a few values to be repeated thousands of times while some values are repeated a few hun-

		Uniform						Zipfian-Value		
		Avg Hops	# of Neighbors	Data Recovery		Network Repair		Avg Hops	# of Neighbors	Qualifying ranges
				$T_{Graceful}$ (Minutes)	MTDL (Hours)	# of Gaps Fixed	% of Failures			
Z=1 (Skip Graph)	α=2	8.07	13.13	15.94	3.77	24.3	0.1	5	13.17	292.07
	α=4	9.29	10.25	15.94	3.77	12.3	0.13	5.79	10.22	291.75
	α=5	9.74	9.55	15.94	3.77	10.7	0.1	6.09	9.52	283.58
	α=6	10.14	8.99	15.94	3.77	9.6	0.08	6.35	8.99	283.45
	α=8	10.74	8.27	15.94	3.77	8.3	0.09	6.71	8.27	290.34
	α=10	11.25	7.77	15.94	3.77	7.5	0.1	7.08	7.75	286.41
α=2	Z=2	6.35	24.54	7.98	7.53	53.3	0.05	2.32	24.83	997.58
	Z=4	5.41	40.89	5.33	11.26	62.15	34.93	1.65	43.41	937.64
	Z=6	4.85	48.62	4.48	13.41	62.51	48.33	1.63	54.21	1195.65
	Z=8	4.56	52.94	4.03	14.87	68.51	52.38	1.57	63.9	1841.86
	Z=10	4.37	56.67	3.75	15.99	74.26	55.30	1.57	72.5	2964.11
α=∞ (PePeR)	Z=2	92.2	3.99	8.08	7.43	1.45	51.73	43.58	3.99	461.36
	Z=3	32.88	5.94	6.22	9.65	3.26	34.55	10.84	5.96	711.52
	Z=4	7.49	7.93	5.35	4.73	5.18	25.88	6.24	7.96	791.23
	Z=6	12.52	11.69	4.49	13.38	9.06	17.45	3.57	11.87	1232.77
	Z=8	9.3	15.15	4.03	14.88	12.91	13.68	3.01	15.68	1391.98
	Z=10	7.8	18.48	3.74	16.06	16.74	11.62	2.78	19.43	1791.45

Table 1: A performance analysis of SkiPeR with 10,000 peers, MTNR is one peer removal every hour. All numbers are averages. SkiPeR with Z=1 and different alphabet sizes is Skip Graph. SkiPeR with $\alpha = \infty$ and different Z values is PePeR.

dred times. With a Zipfian-Range, the table consists of one million unique values distributed in a skewed manner across 1000 consecutive ranges. SkiPeR behaves almost identically with Zipfian-Range and Uniform. Thus, Zipfian-Range is eliminated from further discussion. Moreover, the obtained trends are almost identical for the three different configuration sizes, motivating us to present the results from the 10,000 peer P2P network.

We also analyzed different kinds of predicates. In the following, we focus on a predicate of the following structure: $l < S.a_1 \leq u$. In all experiments, we randomly pick a range from the available peers and ensure l and u are contained in this range. With Uniform, the number of qualifying records is dependent on the values of l and u. However, a predicate is always directed towards at least one range. With Zipfian-Value, many ranges may qualify a predicate because the randomly chosen range might overlap many others. In all experiments, a predicate is initiated at a randomly chosen peer.

In the reported results, we assume MTNR is 1 hour. In order to estimate T_{Copy} with peer removals, we employed an anlytical model for network transmission time with TCP, see [26, 21, 12]. These models estimate transmission time as function of network loss rate, round-trip time (RTT), TCP Reno's parameter settings, and amount of data to be transmitted. Assuming a zero loss-rate for the network, the Network Transmission Time of data consisting of P packets is:

$$T(P) = \begin{cases} RTT \times \lfloor log_2 P \rfloor & if\ P \leq P_{exp} \\ RTT \times (\lfloor log_2 W_{max} \rfloor + \lceil \frac{P-P_{exp}}{W_{max}} \rceil + 2) & else \end{cases} \quad (1)$$

where P_{exp} is the number of packets transmitted during slow start phase, P_{exp}=$2^{\lceil log_2 W_{max} \rceil} + W_{max} - 1$. W_{max} is the maximum window size. With TCP Reno [9] and the Linux operating system, W_{max} is typically set to 45. We used this parameter setting for our experimental purposes. We assumed RTT is 90 Milliseconds, resembling the typical round-trip times from the east to the west coast of United States.

As discussed in Section 2, T_{Copy} requires participation of multiple peers with ReID. Our peer insertion algorithm distributes the records almost uniformly across the peers. We assume a total of 1 Gigabyte of data is migrated by the neighbors of a removed peer. Moreover, we assume the time required to migrate this data by N participants scales as a function of $log_2 N$ (and not N because we consider it to be more realistic than a linear speedup). This arbitrary decision has an identical impact on both ReID2 and ReID3.

Table 1 quantifies the characteristics of SkiPeR with different parameter settings for a Uniform distribution of the partitioning attribute value. It shows a subset of the obtained results with Zipfian-Value because its observed trends with peer removal is almost identical to that with Uniform. This table shows the extreme parameter settings of SkiPeR corresponding to Skip Graphs (Z=1) and PePeR (α=∞). In the following, we discuss number of hops with Uniform and Zipfian-Value, data recovery, and network repair in turn.

With Uniform, as one increases the value of Z with α=∞, SkiPeR performs fewer hops to process a predicate. This is because the number of neighbor ranges known by a peer increases as a function of Z, increasing the efficiency of routing. Only one peer is used to process the predicate because the predicate is very selective. With Zipfian-Value, a higher Z value causes many peers to share the same range corresponding to the value with a high repetition count. While this reduces the number of hops necessary to route a predicate, the average number of ranges containing a predicate increases. This is because we require a predicate to randomly choose a range that exists in the P2P network. Every time it chooses one of the frequently repeated values, many ranges qualify because they overlap one another with these values. With Z=10, almost 18% of the ranges participate in predicate processing.

With Z=1 (Skip Graphs), the number of qualifying ranges is independent of the alphabet size, α. This is because the number of ranges at level 0 is dictated by the number of peers and independent of the α value. As we increase the value of α, the likelihood of a peer appearing in a list at levels one and higher decreases, reducing the number of neighbor information per peer. This reduces the efficiency of routing, resulting in additional hops.

With a moderate parameter setting (α=2 and $Z \geq 2$), the number of *distinct* neighbors is sensitive to the distribution of the partitioning attribute values. The same does not hold true with the extreme α and Z parameter settings (Skip Graphs and PePeR). To explain this difference, note that with α=∞, the number of neighbors should be $2Z$. The observed values are lower because at times a peer neighbors itself. (Recall that Table 1 reports the average number of distinct neighbors.) With Z=1 and $\alpha \geq 2$, a peer cannot neighbor itself because a range assigned to each peer is unique. With $Z \geq 2$ and $\alpha \geq 2$, a peer may neighbor itself multiple times in lists at different levels. The results demonstrate a larger number of distinct neighbors per peer with a skewed distribution of partitioning attribute value. Appendix 3 presents further quantifies the behavior of SkiPeR with different Z and α values.

$T_{Graceful}$ is a constant when $Z = 1$ because the average number of peers for a range at level zero remains fixed at two and is independent of α. This means a fixed number of peers participate in data recovery with both ReID2 and ReID3. With $\alpha = 2$, as we increase Z, additional neighbors participate in the construction of backup copies, reducing the amount of time required for a peer to remove itself gracefully. Note that this time does not scale linearly because we assumed a $Log_2 N$ decrease in the amount of required time as a function of N peers. By reducing $T_{Graceful}$, the MTDL is enhanced. The MTDL reported in Table 1 corresponds to ReID2. The MTDL observed with ReID3 is 25% to 32% lower because a cluster consists of 3 ranges instead of 2. Note that SkiPeR improves upon Skip Graph to support configurations that result in different MTDL values.

We also analyzed the 1Slice algorithm and the percentage of time our simple discovery technique fails when 2 adjacent ranges are removed at the same time. In these experiments, we analyze all possible ways to remove two adjacent ranges for a 10,000 node configuration. Under the "Network Repair" heading of Table 1, we show the average number of gaps and the percentage of time the discovery process fails. A gap may consist of either one or more missing adjacent ranges. When Z=1 (Skip Graph), removal of two adjacent ranges causes only two peers to be removed at the same time, resulting in a maximum of $2log_\alpha N$ gaps. As we increase the value of α, the network levels becomes sparse, reducing the number of gaps. The likelihood of the discovery process failing (0.1%) is small because only two adjacent nodes at level 0 are removed. With $Z > 1$, removal of two adjacent ranges might result in removal of more ranges at level zero. This is because a range removal simulates the removal of a peer that is assigned Z ranges. For example, with $Z = 2$, the removal of two adjacent ranges implies removal of two nodes each containing two ranges, resulting in removal of 4 ranges from level 0 of SkiPeR. When $\alpha = \infty$, increasing the value of Z results in a higher number of gaps. At first glance, it is suprising to see a lower percentage of failures. This is because the neighbor relationship between peers increases as a function of Z, enhancing the likelihood of the discovery process re-connecting the network. With $\alpha = 2$ and $Z \geq 2$, two adjacent range removals is similar to slicing the skip graph into $2Z$ segments at each level. This results in a large number of gaps that must be fixed, increasing the percentage of time the discovery process fails.

Figure 6 shows the percentage of successful predicate lookups in the presence of peer removals with different SkiPeR parameter settings. In these experiments, peers are not allowed to repair SkiPeR's logical network when they detect a peer removal. Our objective is to measure how sensitive the network is during the repair process (T_{Bridge} and $T_{Discovery}$). If the network is partitioned, we continue to choose a randomly chosen live peer (which may belong to any one of the partitions) to initiate the routing of a predicate. We choose the predicate such that its referenced range is contained by one of the live peers. This quantifies how often a search is successfully directed towards the peer containing the relevant data. Figure 6 shows obtained results for Uniform. The y-axis of this figure is the percentage of failed predicate lookups while the x-axis is the percentage of removed peers. A smaller alphabet size (α value) improves the percentage of lookups. While increasing Z provides improvements, its percentage improvement with $\alpha = \infty$ is marginal. Figure 6 shows SkiPeR with its moderate parameter settings ($\alpha = 2$ and a Z value of either 3 or 4) results in best resiliency to node removal. These settings also produces competetive MTDL values.

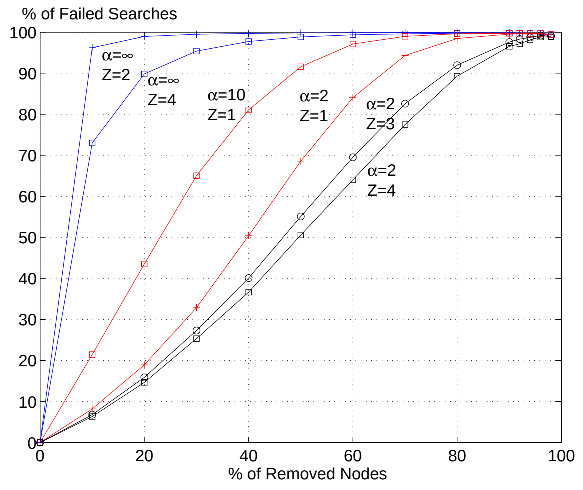


Figure 6: Percentage of failed searches as a function of node removals with 10,000 nodes using different SkiPeR parameter settings.

4 Conclusion and Future Research Directions

SkiPeR is a generalization of two decentralized range addressing techniques, namely, Skip Graphs and PePeR. SkiPeR supports hybrid configurations that cannot be supported by either Skip Graphs or PePeR. At the same time, SkiPeR may realize all possible configurations supported by both Skip Graphs and PePeR.

An immediate research direction is to investigate the design and implementation of a multi-attribute SkiPeR. Another is to compare SkiPeR with alternative DHT-based/Precise [5, 2, 28] approaches to process range predicates.

References

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

[2] 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.

[3] R. Bayer. Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms. *Acta Informatica*, 1:290–306, 1972.

[4] D. Bitton and J. Gray. Disk Shadowing. In *Very Large Databases*, August 1988.

[5] 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.

[6] G. Copeland and T. Keller. A Comparison of High-Availability Media Recovery Techniques. In *Proceedings of ACM SIGMOD*, pages 98–109, 1989.

[7] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. An Indexing Framework for Peer-to-Peer Systems. In *Sigmod*, pages 939–940, March 2004.

[8] A. Daskos, S. Ghandeharizadeh, and X. An. PePeR: A Distributed Range Addressing Space for P2P Systems. In *Proceedings of the International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, Sept. 2003.

[9] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, 26(3):5–21, July 1996.

[10] 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.

[11] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proceedings of ACM SIGMOD*, pages 173–182, 1996.

[12] S. Ghandeharizadeh, C. Papadopoulos, P. Pol, and R. Zhou. NAM: A Network Adaptable Middleware to Enhance Response Time of Web Services. In *MASCOTS*, October 2003.

[13] Tandem Database Group. Nonstop SQL: A Distributed High-Performance, High-Reliability Implementation of SQL. In *Workshop on High Performance Transaction Systems*, 1987.

[14] H. Hsiao and D. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *Proceedings of 6th International Data Engineering Conference*, pages 456–465, 1990.

[15] Hui-I Hsiao and David DeWitt. A Performance Study of Three High Availability Data Replication Strategies. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 18–28, 1991.

[16] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS'03)*, March 2003.

[17] J. A. Katzman. A Fault-Tolerant Computing System. In *Proceedings of the Eleventh Hawaii Conference on System Sciences*, January 1978.

[18] 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.

[19] P. Linga, A. Crainiceanu, J. Gehrke, and J. Shanmugasundaram. Guaranteeing Correctness and Availability in P2P Range Indices. In *Sigmod*, June 2005.

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

[21] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, Vancouver, CA, 1998.

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

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

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

[25] O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. Query Processing Over Peer-To-Peer Data Sharing Systems. In *ICDE*, 2004.

[26] B. Sikdar, S. Kalyanaraman, and K. S. Vastola. An Integrated Model for the Latency and Steady-State Throughput of TCP Connections. *Performance Evaluation*, 46(2-3):139–154, 2001.

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

[28] P. Triantafillou and T. Pitoura. Towards a Unifying Framework for Complex Query Processing Over Structured Peer-to-Peer Networks. In *Proceedings of the International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, Sept. 2003.

APPENDIX 1: Four routing techniques

We considered four different techniques when designing SkiPeR's routing technique. We describe these in turn and briefly compare them with one another. The first corresponds to the routing technique employed by Skip Graphs and described in [1]. The fourth one is the one presented in Figure 3.

Given a predicate P referencing a range of the form $[l_P,u_P)$, all routing techniques compute $\kappa = \frac{l_P+u_P}{2}$. The first routing technique, termed Stairwell, keeps track of levels as it routes a predicate from one node to another, always considering a neighbor range either at the same or a lower level. At each step, there exists a Current Range (CR) with $[l_{CR},u_{CR})$ to guide the routing process. Stairwell chooses range B such that it is either at the same or a lower level than CR. Moreover, B must lie between CR and κ in the address space while moving down the levels, i.e., either $0 \leq \frac{l_B-\kappa}{l_{CR}-\kappa} \leq 1$ if $\kappa < l_{CR}$ or $0 \leq \frac{l_B-\kappa}{u_{CR}-\kappa} \leq 1$ if $\kappa \geq u_{CR}$. This means routing of κ may not either overshoot or undershoot B. To illustrate, recall Figure 2, and consider routing of P when its κ=112 and is initiated at node A. Stairwell routes P from A_2 to range D_1 because its distance is smallest. Next, P is routed to range C_2, and finally to I_2. Node I

	Z=1,α=2	Z=2,α=2	Z=4,α=2
SkiPeR	8.07	6.35	5.41
Spiral	8.01	7.98	7.98
Battlement	10.24	10.56	10.50
Stairwell	11.41	11.79	12.34

Table 2: Average number of hops with four alternative routing techniques with 10,000 nodes and a table with one million records. The partitioning attribute values are unique.

processes P because its range contains 112. In this example, Stairwell performs 3 hops to process P.

The second routing technique, termed Battlement, improves upon Stairwell in one way: it allows the routing mechanism to reset CR to the highest level of Current Range known by a node. This facilitates higher jumps in the address space with no overshooting, i.e., the picked range is between CR and κ. To illustrate, consider the structure of Figure 2 where C_2 does not exist at level 1. To route κ=112 initiated at node A, this algorithm visits the following ranges in sequence: A_2, D_1, F_1, and I_2. This results in 3 hops. One would encounter four hops with Stairwell with the following sequence: A_2, D_1, F_1, C_2, and I_2 (assuming C_2 does not exist at level 1 of Figure 2).

The third routing technique, termed Spiral, improves upon Battlement by allowing the routing algorithm to overshoot a neighbor's range as long as it minimizes distance. More formally, either $-1 < \frac{l_B-\kappa}{l_{CR}-\kappa} \leq 1$ if $\kappa < l_{CR}$ or $-1 < \frac{l_B-\kappa}{u_{CR}-\kappa} \leq 1$ if $\kappa \geq u_{CR}$. In our example, this overshooting routes P by visiting the following sequence of ranges starting with A: A_2, E_1, C_2, and I_2. While this results in the same number of hops as Stairwell, note that Spiral made a larger jump at level 3.

The final routing technique, shown in Figure 3 and employed in the evaluation of Section 3, considers all ranges known by the owner of each neighbor range to compute the node with the closest range to κ. This algorithm employs the extra range information a node has about its neighbors. This produces an unintuitive path, with range B seemingly more distant to κ than CR, while in reality B is owned by a node whose other ranges include at least one range closer to κ than CR. With our example, this algorithm would use all nodes known by A (B, C, D, E, G, H) when routing P. It routes P to C because C_2 has a smaller distance to 112. From C, P is routed directly to I_2, incurring 2 hops to process P.

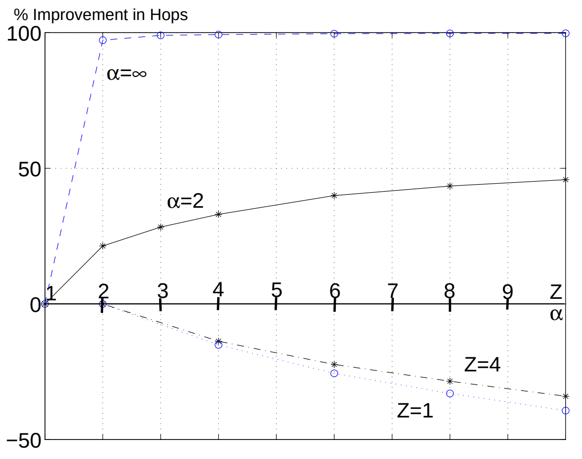
Table 2 shows the average number of hops for routing of 100,000 randomly generated predicates with the four alternative routing techniques and a one million records table using a 10,000 peer SkiPeR network. The different columns of this table show alternative SkiPeR configurations with different Z values. With Z=1, the resulting structure is identical to Skip Graphs. The results show SkiPeR's routing technique is superior to other alternatives. SkiPeR outperforms Stairwell (Skip Graphs' routing technique) and Battlements by more than 40% when Z=1. With values of $Z > 1$, SkiPeR outperforms these techniques by a wider margin. Note that Spiral is comparable to SkiPeR when Z=1. This is because a neighbor provides the same amount of information to both techniques with Z=1. With $Z > 1$, SkiPeR is provided with additional information, enabling it to outperform Spiral.

APPENDIX 2: Node insertion

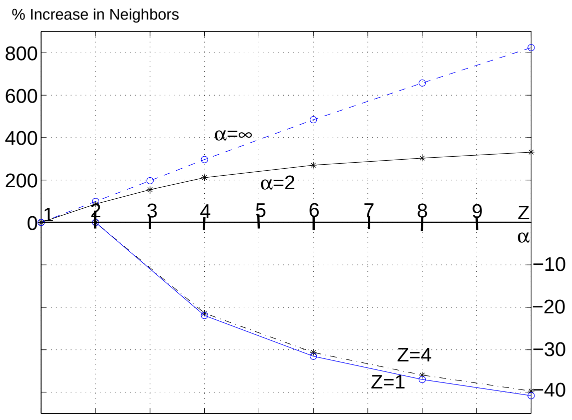
SkiPeR inserts a new node N_{new} into its addressing space in two steps. First, it employs Elastic of PePeR [8] to assign Z ranges from the available nodes to N_{new}. Second, it extends the hierarchy by generating a new word for N_{new} and inserting it at the right position in the appropriate list at different levels [1, 16]. Below, we provide an overview of these two steps and refer the interested reader to [8, 1, 16] for details. When N_{new} joins the network, Elastic employs probes to sample the available ranges in the network. Next, it picks Z of these sampled ranges with the objective to facilitate both short and long jumps within the address space. N_{new} contacts the owner of these ranges to split the chosen range into half and assign these ranges along with their data to N_{new}. In [8], it is shown that Elastic is stable as a function of the number of nodes and constructs an assignment that yields an average hops which is 60% higher than a theoretical lower-bound independent of the number of nodes. The maintenance of the hierarchy are discussed extensively by both [1] and [16]. SkiPeR employs similar techniques to facilitate decentralized insertion of a node once Elastic assigns it Z ranges.

Novel extensions of SkiPeR to this prior work is two folds. First, SkiPeR requires a new peer to implement either the 1Cell or 1Slice replication technique in order to minimize the possibility of discovery process failing, see the discussion of node removal in Section 2. Second, a new peer must construct the slave copy of its assigned range using either ReID2 or ReID3. There are important differences between ReID2 and ReID3 because they require the participation of a different number of peers. An analysis of this tradeoff is a future research direction and eliminated from this presentation.

APPENDIX 3: Sensitivity of SkiPeR's routing to α and Z



7.a Number of hops



7.b Number of neighbors

Figure 7: Percentage change in hops and number of neighbors with 10,000 peers (Uniform distribution).

Figures 7.a and 7.b characterize SkiPeR's behavior with different α and Z values. In these figures, the x-axis corresponds to α and Z for the negative and positive y-axis values, respectively. When we fix the value of α and increase the value of Z on the x-axis, we see a substantial enhancement in the number of hops because there is a significant increase in the number of neighbors; see the positive regions of Figures 7.a and 7.b. On the other hand, when Z is fixed and we increase the value of α, the average number of peers per node decreases, resulting in a negative percentage improvement in hops and neighbors. We elaborate on these in turn.

With α=∞, as one increases Z, the percentage improvement in hops relative to Z=1 is dramatic: Average number of hops is enhanced from 333 to 7.8 hops. This is due to a significant increase in number of neighbors as a function of Z; several hundred percentage increase in Figure 7.b. With α=2, the percentage improvement as a function of Z is approximately 40% with Uniform and 65% with Zipfian-Value. This difference is due to a larger number of neighbors per peer realized with the skewed distribution, see Figure 7.b. Also, note that the percentage improvement in hops as a function of Z is marginal beyond Z=4.

Focusing on the negative y-axis values of Figure 7.a, with $Z = 1$, when we increase α from 2 to 4, the number of hops increases, showing a negative percentage improvement. This is due to a percentage decrease in number of neighbors, see Figure 7.b. The observed trends are almost identical for different Z values, 1 and 4.