# Structured Overlay for Multi-dimensional Range Queries[⋆]


Thorsten Schütt, Florian Schintke, and Alexander Reinefeld

Zuse Institute Berlin



**ostract.** We introduce SONAR, a structured overlay to store and re-
eve objects addressed by multi-dimensional names (*keys*). The overlay
the shape of a multi-dimensional torus, where each node is respon-
le for a contiguous part of the data space. A uniform distribution of
s on the data space is not necessary, because denser areas get assigned
re nodes. To nevertheless support logarithmic routing, SONAR main-
ns, per dimension, fingers to other nodes, that span an exponentially
reasing number of *nodes*. Most other overlays maintain such fingers
the *key-space* instead and therefore require a uniform data distribu-
n. SONAR, in contrast, avoids hashing and is therefore able to per-
m range queries of arbitrary shape in a logarithmic number of routing
ps—independent of the number of system- and query-dimensions.

SONAR needs just one hop for updating an entry in its routing table:
onger finger is calculated by querying the node referred to by the next
orter finger for its shorter finger. This doubles the number of spanned
des and leads to exponentially spaced fingers.


## roduction

ent handling of multi-dimensional range queries in Internet-scale dis-
systems is still an open issue. Several approaches exist, but their lookup
are either expensive (space-filling curves) [2] or use probabilistic ap-
like consistent hashing [10] to build the overlay.

opose a system for storing and retrieving objects with $d$-dimensional
a peer-to-peer network. SONAR (Structured Overlay Network with
Range-queries) directly maps the multi-dimensional data space to a
ional torus. It supports range queries of arbitrary shape, which are use-
ample, in geo-information systems where objects in a given distance of
are sought. SONAR can also be employed in Internet games with mil-
line-players who concurrently interact in a virtual space and need quick
the local surroundings of their avatars. In a broader context, SONAR
mployed as a hierarchical publish/subscribe system, where published
e categorized by several independent attributes. The category of pub-
ents addresses a data point in the $d$-dimensional space and consumers
ng to subareas will receive all events published in their subarea.

aper is organized as follows: First, we discuss related work. Then, in , we introduce SONAR. In Section 4, we present empirical results and n 5 we conclude the paper with a brief summary.

## ated Work

ystems [1] have been proposed that support complex queries with multinal keys and ranges. They can be split into two groups.

*Filling Curves.* These systems [2,9,16] use locality preserving spacerves to map multi-dimensional to one-dimensional keys. They provide ent range queries than the space partitioning schemes described below, a single range query may cover several parts of the curve, which have eried separately (Fig. 5a). Chawathe et al. [7] present performance re- real-world application using Z-curves on top of OpenDHT. The query nce ($\approx 2$ sec. for $\leq 30$ nodes) is rather low due to the layered approach.

*Partitioning.* The schemes using space partitioning split the key-space e nodes. SONAR belongs to this group of systems. The proposed sysnly differ by their routing strategies.

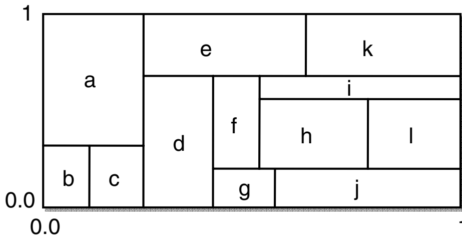
14] was one of the very first DHTs. It hashes the key-space onto a multinal torus. While the topology resembles that of SONAR and MURK w), CAN uses just the neighbors for routing and it does not support eries.

[4] employs a Voronoi-based space partitioning scheme and uses a rld graph overlay with routing tables of size $O(1)$. The overlay is not some regular partitioning scheme (e.g. kd-tree [5]) but uses a sample e to place the fingers.

attribute range queries were also addressed by Mercury [6] which needs umber of replicas per item to achieve logarithmic routing performance. [12] uses super-peers and query-caching to allow multi-attribute range n top of the Bamboo-DHT [15].

an *et al.* [9] proposed two systems for multi-dimensional range queries o-peer systems: SCRAP and MURK. SCRAP uses the traditional ap- mapping multi-dimensional to one-dimensional data with space-filling hich destroys the data locality. Consequently, each single multi-dimenge-query is mapped to several one-dimensional queries. MURK is more our approach, as it divides the data space into hypercuboids with each assigned to one node. In contrast to SONAR, MURK uses a heuristic based on skip graphs [3] to set routing fingers.

## tem Design




**g. 1.** Example two-dimensional overlay with attribute domains $[0, 1]$

R is used to store and retrieve *objects*. It works on a $d$-dimensional torus, *pace*. Objects have a name, the *key*, which is a vector of $d$ components, *utes* of the key. Each *dimension* of the torus is responsible for one *domain*. Figure 1 illustrates a two-dimensional key-space ($[0, 1]^2$). Ar-ocated computers, the *nodes*, are each responsible for a dedicated area oid) in the key-space of the overlay (rectangles in Fig. 1). The *node-* the same extent as the key-space, but is completely filled with nodes. es in the node-space are *adjacent* (or *neighbors*) when their key-space nt. The direct mapping between key-space and node-space guarantees keys to be stored on the same or adjacent nodes, which enables efficient eries across node boundaries by local query propagation.

are dynamically assigned to the key-space such that each node serves he same number of objects. Load-balancing is done by changing the ility of nodes instead of moving around objects in the key-space. That necessary when the number of objects or nodes in the system changes ).

## erlay Topology

ated in Figure 1, the two-dimensional key-space is covered by rectangles, hem containing about the same number of objects. Because the keys ally not uniformly distributed, the rectangles have different sizes and have more than one neighbor per direction. The neighbors are stored *or lists*, one per dimension.

erlay described so far resembles that of CAN [14] except for the hashing which prevents efficient range queries. Consequently, SONAR would also $\sqrt{N}$) network hops if it would just use the neighbors for routing. In the , we introduce routing tables to achieve logarithmic routing performance.

## uting

ng, SONAR uses separate *routing tables*, one per dimension. Each rout-

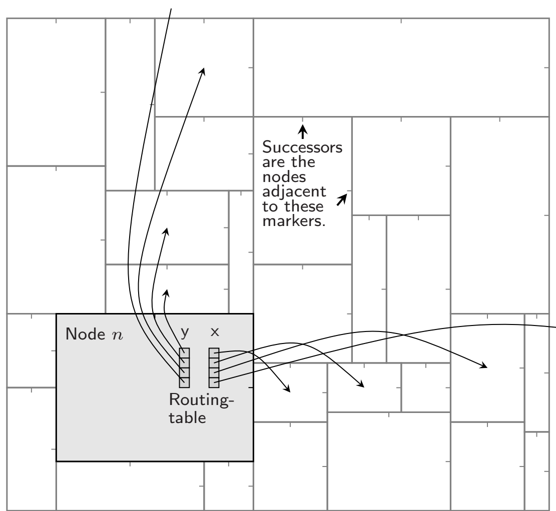



**Fig. 2.** Routing table for the two-dimensional case

culate its $i^{th}$ finger in the routing table, a node looks at its $(i-1)^{th}$
l asks the remote node listed there for the $(i-1)^{th}$ finger. At the lowest
fingers point to the successor.

$$finger_i = \begin{cases} successor & : i = 0 \\ finger_{i-1}.getFinger(i-1) & : i \neq 0 \end{cases}$$

pdate process works in a running system, but also during startup.
all fingers are set to *unknown* except for the finger to the successor.
e second entry will always succeed, because the successor knows its
. Filling further entries may fail (result *unknown*), because the remote
not have determined the corresponding entry yet. But with subsequent
updates, eventually all nodes will get their entries filled. The resulting
is similar to skip lists [13], but the behavior is more deterministic.

*Used for Routing.* A node may have more than one neighbor per
We define the node adjacent to the middle of the respective side to be
*ssor*. Successors are marked by small ticks in Figure 2.
the different box sizes and the calculation of longer fingers from shorter
ers are not necessarily straight in one direction. Slight deviations in $y$-
might occur when following the fingers of the $x$-direction (and vice
shown in Figure 2. Our empirical results indicate, however, that this
affect the logarithmic routing performance (Sect. 4).

*Table Size.* Each node holds approximately $\log N$ fingers in its routing

```
 calculates the entries of a routing table
d updateRoutingTable(int dim) {
nt  i = 1;
ool done = false;

t[dim][0] = this.Successor[dim];

hile (!done) {
 Node candidate = rt[dim][i - 1].getFinger(dim, i - 1);
 if (IsBetween(dim, rt[dim][i - 1].Key, candidate.Key, this.Key)){
   rt[dim][i] = candidate;
   i++;
 } else
   done = true;


 checks whether the resp coordinate of pos lies between start and end
l IsBetween(Dim dim, Key start, Key pos, Key end);
```

**Fig. 3.** Finger calculation for dimension **`dim`**

[6] predicts the system size $N$ by estimating the key density. SONAR
npler, deterministic solution with less overhead.

ch dimension $dim$, SONAR's finger update algorithm (Fig. 3) inserts an
l finger $finger_i$ as long as its position is between that of the last routing
y $finger_{i-1}$ and that of the node itself. Otherwise the new finger circles
he ring and is not inserted.

sults in Section 4 confirm that each node holds indeed $\log N$ fingers.
truction process guarantees—in contrast to Chord [18]—that no two
oint to the same node. Since the fingers in the routing tables span an
ially increasing number of nodes, the routing table of each dimension
al of $\lceil \log D \rceil$ entries on the average, where $D$ is the number of nodes in
tion on the torus.

*Finger Update.* Our periodically running finger update algorithm needs
etwork hop to determine an entry in the routing table. Chord in con-
ds $O(\log n)$ for the same operation, because it performs a DHT lookup
te a finger.

## okup and Range Queries

her DHTs, SONAR uses greedy routing. In each node the finger that
y reduces the Euclidean distance to the target in the key-space is fol-
dependently of the dimension (see Fig. 4).

R supports range queries with multiple attributes. In its most basic
ange query is defined by $d$ intervals for the $d$ attribute domains. The
ery finds all keys whose attributes match the respective intervals and
he corresponding objects. Because of their shape, such range queries
$d$-dimensional *rectangular* range queries.

```
// find the responsible node for a given key
Node find(Point target) {
  Node nextHop = findNextHop(target);
  if (nextHop == this)
    return this;
  else
    return nextHop.Find(target);
}

double getDistance(Node a, Point b);

Node findNextHop(Point target) {
  Node candidate = this;
  double distance = getDistance(this, target);

  if (distance == 0.0)
    // found target
    return this;

  for (int d = 0; d < dimensions; d++) {
    for (int i = 0; i < rt[d].Size; i++) {
      double dist = getDistance(rt[d][i], target);
      if (dist < distance) {
        // new candidate
        candidate = rt[d][i];
        distance = dist;
      }
    }
  }
  // will never happen:
  Assert(candidate != this);
  return candidate;
}
```

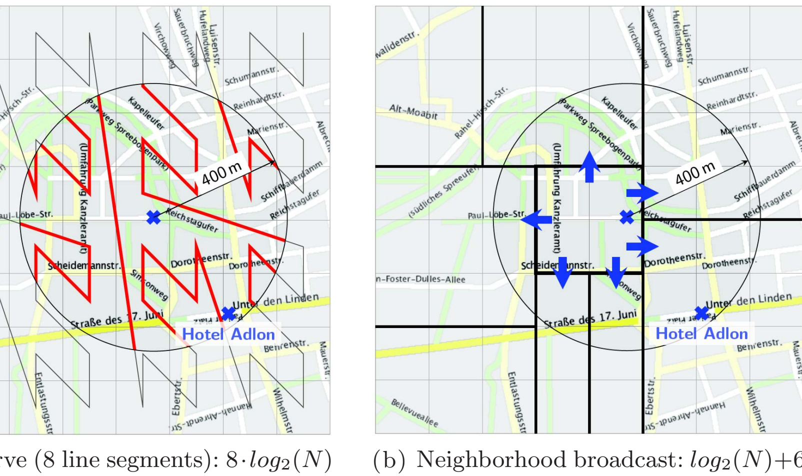
**Fig. 4.** Lookup for a target

er and a radius. Here, we assume a person located in the governmental f Berlin searching for a hotel in 'walking distance' (circle around the The query is first routed to the node responsible for the center of the then forwarded to all neighbors that partially cover the circle (Fig. 5b). y is checked against the local data and the results are returned to the g node. Figure 6 shows the pseudocode of this algorithm. Note that t messages are eliminated. op is an additional check for objects in the rea—in this case for type hotel.

R performs a range query with a single lookup. When the target node hold the complete key range, the query is locally forwarded. Systems e-filling curves, in contrast, usually require more than one lookup for ange query because they map connected areas to multiple independent ents, see Figure 5a.

## pology Maintenance to Handle Churn

*n.* When a node joins the system, the key-space of a participating




ve (8 line segments): $8 \cdot log_2(N)$ (b) Neighborhood broadcast: $log_2(N)+6$

**Fig. 5.** Circular range query

```
// perform a range query
void queryRange(Range r, Operation op) {
  Node center = Find(r.Center);
  center.doRangeQuery(r, op, newId());
}

void doRangeQuery(Range r, Operation op, Id id) {
  // avoid redundant executions
  if (pastQueries.Contains(id))
    return;
  pastQueries.add(id);

  foreach (Node neighbor in this.Neighbors)
    if (r ∩ neighbor.Range != ∅)
      neighbor.doRangeQuery(r \ this.Range, op, id);

  // execute operation locally
  op(this, r);
}
```

**Fig. 6.** Range query algorithm

a random target node: A random position in the key-space is routed d a random walk is started from there. The final target node of this is andidate to be split. The random walk ensures that nodes responsible rger areas of the key-space are not preferred over smaller ones.

the key-space and transfer one part to the new node: Splits are parallel e of the coordinate system axes. The selection of the axis to be split d not strictly favor one dimension over the others because the number des to be contacted for a range query could become disproportionately

*ve.* Handling a leaving node is more difficult, because it is not always
hich node can fill the area of the leaving node. For example, in Figure 1,
f node f cannot be merged with any of its neighbors, because this would
a non-rectangular node-space.
ore the node-space is constructed in such a way that the splitting plane
*d-tree* [5]. KD-trees are used only for topology maintenance, similar as
[9], but not as index structures like in database systems. The space of
node can be taken over by a neighboring node which is also a sibling
-tree. By keeping the tree balanced the probability of having a sibling
hbor increases. Each node must remember its position in the kd-tree, a
describing the path from the root of the tree to the node itself.
eighboring nodes are siblings in the kd-tree, another node must be
fill the gap. Either a neighboring node additionally takes over the re-
ty of the separate area until a free node can be found, or two completely
ent nodes that are siblings in the kd-tree have to be found to merge
thus free a node that takes over the free area. The former concept,
*tual nodes*, is also used for load-balancing in other systems.

*ancing.* Load-balancing can be implemented by either adjusting the
es of the responsibilities locally or freeing nodes in underloaded areas
ng them to overloaded areas. The former has similar issues as a node
e boundaries are interlocked with limited room for adjustments. The
s shown to be converging [11] with predictable performance.
lancing can be based on different metrics for load, like object or query
combination of both. To avoid thrashing effects a threshold for per-
load-balancing round must be introduced.

## pirical Results

g the performance of SONAR we used a traveling salesman data set
4,711 cities[1]. The cities' geographical locations follow a Zipf distribu-
which is also common in other scenarios.
signed the responsibility of nodes by recursively splitting the key-space
ger side, so that each part gets half of the cities until enough rectangles
ed. Figure 7 shows a sample splitting for 256 nodes.
ordinates were mapped onto a doughnut-shaped torus rather than a
cause in a globe all vertical rings meet at the poles. This would not only
outing bottleneck at the poles but would also result in different ring
s for the western and eastern hemisphere (southwards vs. northwards).
8 shows the results for various all-to-all searches in networks of differ-
The routing performance, depicted by the '+' ticks, almost perfectly
he expected $0.5 \log_2 N$ hops. Only in the larger networks the expected

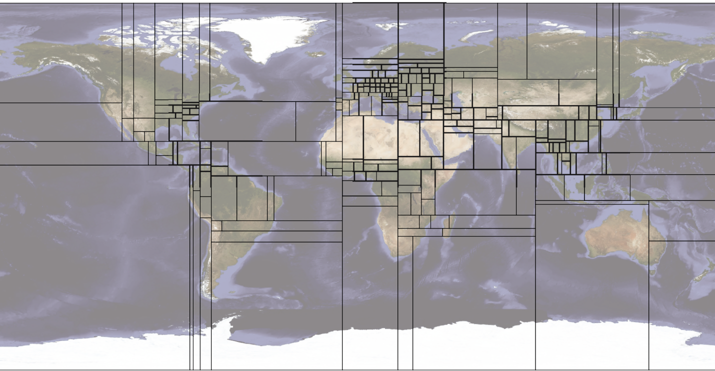


**Fig. 7.** 1,904,711 cities split evenly into 256 rectangular nodes

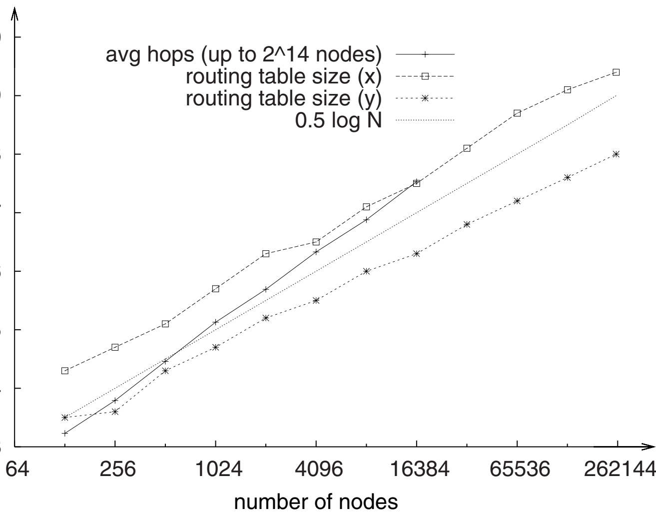



**Fig. 8.** SONAR results for increasing system sizes (2-dimensional)

o checked whether the number of fingers in the routing tables, which are d without global information (Fig. 4), meets our expectations. The '□' the routing table sizes in horizontal direction, and the '∗' ticks represent in vertical direction. As expected, both graphs have the same slope of : One lies consistently above, the other below. This is attributed to the domain sizes of the coordinate system (360 versus 180 degrees) and to en number of splitting planes.

9 gives further insight into the characteristics of SONAR's routing

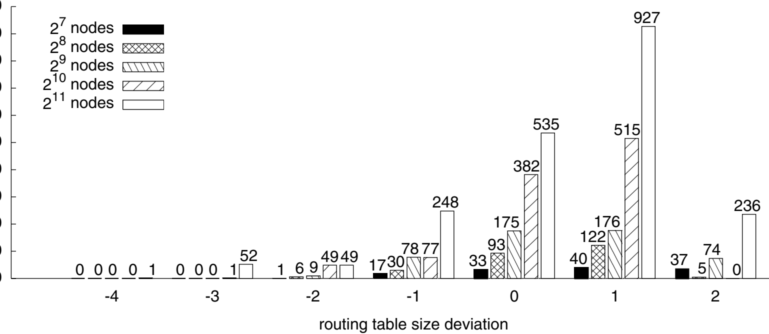



**Fig. 9.** Routing table size deviation from the expected value

out 25% meet the expected size of $\log_2 N$, while there is a decreasing f tables with fewer entries. These deviations are caused by the uneven bution and by SONAR's finger update algorithm which has a tendency n some cases an extra finger that is more than halfway around the ring 'left' of the own node).

## nclusion

efficiently supports range-queries on multi-dimensional data in struc- erlay networks. It needs $O(\log N)$ routing steps for processing range- f arbitrary shapes and an arbitrary number of attribute domains. The culation needs just one hop for updating an entry in the routing table. esented empirical results from a Zipf distributed data set with approxi- vo million keys. The results confirm that SONAR does its routing with a ic number of hops—even in skewed data distributions. Additional tests er practical and uniform distributions (not shown here) gave the same ic routing performance. Furthermore, we observed that the sizes of the ed routing tables are always $O(\log N)$ although they are autonomously ed by the nodes with local information only.


## wledgements

o the anonymous reviewers for their valuable comments. The topo- mages were taken from the 'Blue Marble next generation' project of Earth Observatory. Thanks to Slaven Rezić for the street map of Berlin.


## nces

ejak, A., Xu, Z.: Scalable, efficient range queries for Grid information ser-
In: P2P 2002 (2002)
s, J., Shah, G.: Skip graphs. In: SODA (January 2003)
i-Kashani, F., Shahabi, C.: SWAM: A family of access methods for similarity-
in peer-to-peer data networks. In: CIKM (November 2004)
ey, J.: Multidimensional binary search trees used for associative searching.
unications of the ACM 18(9) (1975)
mbe, A., Agrawal, M., Seshan, S.: Mercury: Supporting scalable multi-
ute range queries. In: ACM SIGCOMM 2004 (August 2004)
athe, Y., Ramabhadran, S., Ratnasamy, S., LaMarca, A., Shenker, S., Heller-
J.: A Case Study in building layered DHT applications. In: SIGCOMM'05
st 2005)
, V., Günther, O.: Multidimensional access methods. ACM Computing Sur-
30(2) (1998)
an, P., Yang, B., Garcia-Molina, H.: One torus to rule them all: Multi-
sional queries in P2P systems. In: WebDB2004 (2004)
r, D., Lehman, E., Leighton, T., Panigrah, R., Levine, M., Lewin, D.: Con-
hashing and random trees: Distributed caching protocols for relieving hot
on the World Wide Web. In: ACM Sympos. Theory of Comp. (May 1997)
r, D., Ruhl, M.: Simple efficient load balancing algorithms for peer-to-peer
ns. In: Voelker, G.M., Shenker, S. (eds.) IPTPS 2004. LNCS, vol. 3279,
ger, Heidelberg (2005)
nheimer, D., Albrecht, J., Patterson, D., Vahdat, A.: Design and implementa-
radeoffs for wide-area resource discovery. In: 14th IEEE Symposium on High
mance Distributed Computing (HPDC-14) (July 2005)
W.: Skip lists: A probabilistic alternative to balanced trees. Communications
ACM (June 1990)
samy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-
ssable network. In: ACM SIGCOMM 2001 (August 2001)
S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a DHT. In:
edings of the USENIX Annual Technical Conference (June 2004)
dt, C., Parashar, M.: Enabling flexible queries with guarantees in P2P sys-
IEEE Internet Computing, 19–26 (May/June 2004)
, T., Schintke, F., Reinefeld, A.: Structured overlay without consistent hash-
mpirical results. In: GP2PC'06 (May 2006)
, I., Morris, R., Kaashoek, M.F., Karger, D., Balakrishnan, H.: Chord: A
le peer-to-peer lookup service for Internet application. In: ACM SIGCOMM
August 2001)
G.: Relative frequency as a determinant of phonetic change. Harvard Studies
ssical Philiology (1929)