Future Generation Computer Systems 25 (2009) 290–300

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Scalable community-driven data sharing in e-science grids☆

Tobias Scholl*, Bernhard Bauer, Benjamin Gufler, Richard Kuntschke, Angelika Reiser, Alfons Kemper

Institut für Informatik, Technische Universität München, 85748 Garching bei München, Germany

ARTICLE INFO

Article history:
Received 30 November 2007
Received in revised form
14 April 2008
Accepted 14 May 2008
Available online 20 May 2008

Keywords:
Data sharing (H.3.5)
Distributed databases (H.2.4)
Scientific databases (H.2.8)

ABSTRACT

E-science projects of various disciplines face a fundamental challenge: thousands of users want to obtain new scientific results by application-specific and dynamic correlation of data from globally distributed sources. Considering the involved enormous and exponentially growing data volumes, centralized data management reaches its limits. Since scientific data are often highly skewed and exploration tasks exhibit a large degree of spatial locality, we propose the locality-aware allocation of data objects onto a distributed network of interoperating databases. HiSbase is an approach to data management in scientific *federated Data Grids* that addresses the scalability issue by combining established techniques of database research in the field of spatial data structures (*quadtrees*), *histograms*, and *parallel databases* with the scalable resource sharing and load balancing capabilities of decentralized Peer-to-Peer (P2P) networks. The proposed combination constitutes a complementary e-science infrastructure enabling load balancing and increased query throughput.

1. Introduction

E-science communities such as climatology, astrophysics, medicine, and the geosciences face the fundamental challenge of managing data volumes generated by upcoming applications with expected data rates of several terabytes a day and petabytes a year. The anticipated continuous growth at an exponential rate further increases the need for scalable information management. Collaborating researchers from all over the world access these distributed data sources [15] in order to find new scientific results.

1.1. Challenges

Future e-science communities require the efficient processing of data volumes that centralized data processing or a data warehouse approach cannot sufficiently scale up to. Centralized data processing, where researchers ship data on demand from the distributed sources to a processing site – most often their own computer – has the deficiency of high transmission cost. On the other hand, a data warehouse cannot cope with the high query load and demanding throughput requirements. In astronomy, for example, most often the individual projects provide interfaces to their own data set for interactive or service-based data retrieval. These service interfaces are standardized by the *International Virtual Observatory Alliance (IVOA)*[1] in order to ensure interoperability between the various interfaces. User queries can consume only a limited amount of CPU resources (e.g., 10 min), have a result size limit (e.g., 100 000 rows), and the number of parallel queries per user is restricted in order to allow fair use and to avoid overloading the servers. Batch systems (such as CasJobs [20]) offer less restrictive access to the data sources and sometimes even a private database for later processing or sharing the results with colleagues. However, some queries might suffer from long queuing times.

Furthermore, we observe that in many e-science communities, data sets are highly skewed and scientific data analysis tasks exhibit a large degree of spatial locality. Dealing with *data skew* while *preserving spatial locality* is fundamental to realize a scalable information infrastructure for these communities. A more detailed scenario from the astrophysics domain exhibiting these characteristics is given in Section 2.

To avert the scalability issues of their current systems, communities investigate different technologies. The adaption to domain-specific data and query characteristics is fundamental for these approaches to result in benefits for the researchers. These characteristics can include properties such as data skew and complex multi-dimensional range queries. Among the technologies investigated are *community-driven Data Grids* which use decentralized Peer-to-Peer (P2P) technologies in order to provide scalable communication and data management. Community-driven

☆ This work is part of the AstroGrid-D project and of D-Grid and is funded by the German Federal Ministry of Education and Research (BMBF) under contract 01AK804F and by Microsoft Research Cambridge (MSRC) under contract 2005-041.

* Corresponding author. Tel.: +49 89 2 89 17276; fax: +49 89 2 89 17263.
E-mail addresses: scholl@in.tum.de (T. Scholl), bauerb@in.tum.de (B. Bauer), gufler@in.tum.de (B. Gufler), kuntschk@in.tum.de (R. Kuntschke), reiser@in.tum.de (A. Reiser), kemper@in.tum.de (A. Kemper).

[1] http://www.ivoa.net.

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.05.006

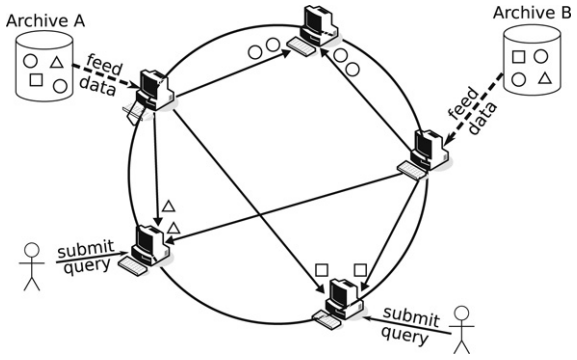


Fig. 1. HiSbase architecture.

Data Grids are built on the data sharing approach of *federated Data Grids* [35] and extend it by relaxing the *data autonomy* requirement for achieving better data load balancing and improving query throughput. *Distributed hash tables* (DHT) allow the seamless integration of new peers and resources. The symmetry of these networks, i.e., the fact that peers act as servers (providing data) and as clients (issuing queries), offers increased fault-tolerance and robustness. In a DHT system, peers automatically detect node failures and fix the overlay communication.

1.2. HiSbase Architecture

In this paper, we describe HiSbase, a distributed information infrastructure that allows sharing of CPU resources and storage across scientific communities to build a community-driven Data Grid. We distribute data across (e.g., hundreds of) peers according to predominant query patterns to achieve a higher throughput for data analysis tasks. Therefore, most processing tasks can be performed locally, achieving high cache locality as peers mainly process queries on logically related data hosted by themselves. Fig. 1 illustrates this approach on an abstract level. In the figure, logically related data originating from (possibly) different distributed sources are denoted by the same geometric shapes. HiSbase partitions and allocates data fed into the system by means of community-specific distribution functions, called *histograms*. Thereby, related data objects of various sources are mapped to the same peers. In Section 3, we discuss several candidate data structures that preserve spatial locality and adapt to the data distribution.

HiSbase, as described in Section 4, incorporates multi-dimensional data and histograms as follows:

- We precompute the histogram of the actual data space in a preparatory *training phase* based on a training set and pass it to the initial HiSbase peer during startup (Section 4.1).
- Additional peers subsequently joining the network receive their own local copy of the histogram from a neighboring HiSbase peer.
- HiSbase allocates data at peers according to the precomputed histogram (Section 4.2) and uses the histogram as a routing index. Data archives feed data into HiSbase by sending their data to any HiSbase peer which routes the data to the responsible peer (Section 4.3).
- Every HiSbase peer accepts queries and routes them to a coordinator peer which owns (some of) the data needed to process the query. If the coordinator does not cover all the data relevant to the query, it guides cooperative query processing among all peers contributing to the query result (Section 4.4).

In Section 5, we discuss the performance of a single peer and a multi-peer HiSbase instance within a local area network in comparison to a centralized database server with regard to query throughput. We further outline the projected experiments within the AstroGrid-D testbed.

1.3. Contributions

Scalable data sharing for e-science grids.

HiSbase realizes a scalable information economy [5] for e-science Data Grids by building on advances in proven DHT-based P2P systems such as Chord [33] and Pastry [26], as well as on achievements in P2P-based query processing [17]. HiSbase combines these techniques with histograms for preserving data locality, spatial data structures such as the quadtree [27] for efficient access to histogram buckets, and space filling curves [21] for mapping histogram buckets to the DHT key space. There have been inspiring contributions extending DHTs to support multi-dimensional range queries [4,12,32,34] and describing load-balancing schemes for data and execution skew [2,7,11,23] in the face of a varying data population and high network churn. However, these systems currently treat the data items individually which results in prohibitive costs in an e-science environment, e.g., it requires several months to distribute data of several million objects to the participating sites.

Preserving locality and handling data skew through domain specific partitioning.

We suggest reconsidering static partitioning schemes as an application domain specific hash function to allow scalable information management in e-science communities. Occasionally, this hash function is updated to accommodate better load-balancing, just as database systems regularly update query optimizer statistics. HiSbase targets collaborative communities having vast data volumes with fairly stable data distributions. Long-term distribution changes can also be leveled by reorganizing the histogram.

Increased query throughput.

We investigate the potential offered by P2P networks for increasing query throughput in data-intensive e-science applications. Achieving sufficient query throughput constitutes one of the main deficiencies of centralized data management.

2. Sample application domain: Astrophysics

The abstract scenario above is applicable to many e-science domains including climatology, geophysics, and medicine. We employ data and use cases from the astrophysics domain for further illustrations, since we are currently developing a distributed information management platform for the German astrophysics community (AstroGrid-D) [8] within *D-Grid*, the German e-science and Grid Computing initiative. This platform facilitates collaborations with national as well as international partners.

In e-science, results of different investigations (experiments, surveys, observations, etc.) are compared or combined to gain further insight or to obtain the complete picture of a particular phenomenon. Astrophysical example use cases comprise the creation of probability maps for galaxy clusters [6,31] and the combination of observational data from several archives covering, for example, various wavelength ranges in order to classify spectral energy distributions [19].

In previous work we focused on the practical aspects of developing a HiSbase instance for the astrophysics community [29]. Implementing a prototype in realistic scenarios closely cooperating with a community is fundamental in our view to ensure the applicability of our approach. Using solely simulation studies often does not correctly represent the challenges of distributed systems, e.g., if the simulation model does not capture all relevant parameters. We deployed a HiSbase instance with up to 56 nodes on the resources of the AstroGrid-D test bed, D-Grid resources, and on nodes within

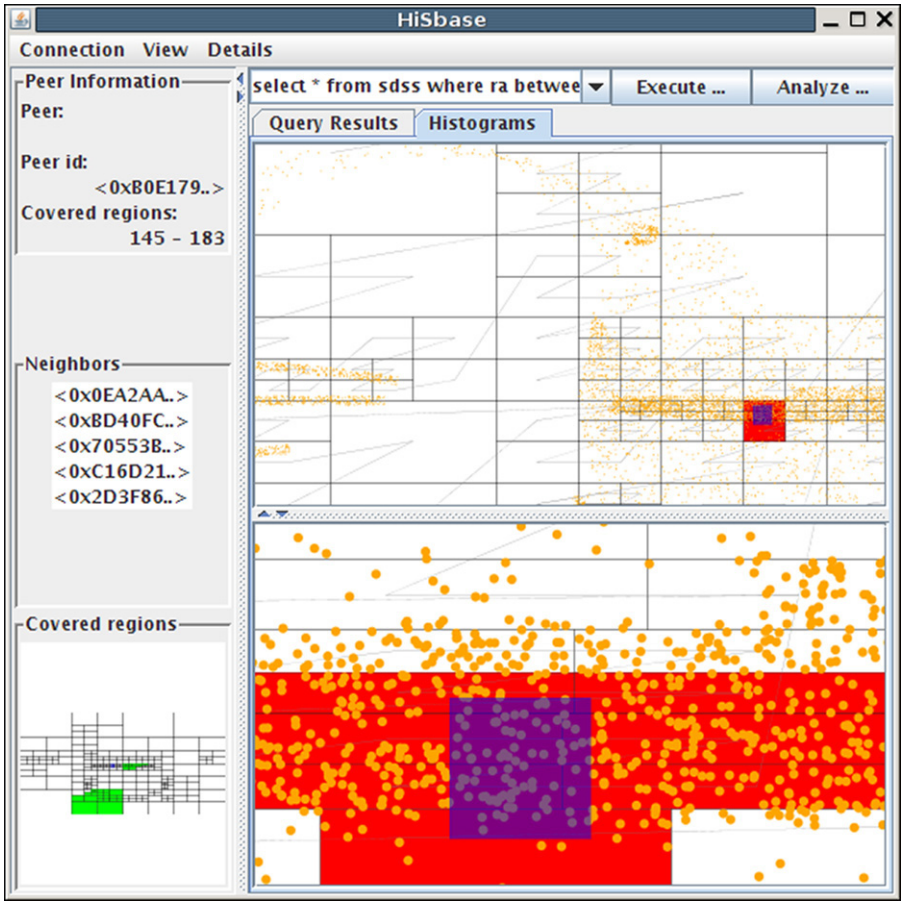


Fig. 2. The HiSbase GUI.

the PlanetLab test bed in order to demonstrate the functionality of our system. Fig. 2 illustrates some aspects of HiSbase such as submitting queries, comparing different histogram data structures, and providing status information on the connected HiSbase nodes. Our prototype uses the relational data model and SQL as the current specification for the IVOA *Astronomical Data Query Language (ADQL)* is also SQL-based.

Furthermore, in earlier work we proposed a framework for comparing various *a priori* calculated histogram data structures and gave several different measures to evaluate the effectiveness of the data structures [30]. This framework allows communities to experiment with different data structures before deciding which suits their needs best. This is a necessity for efficiently distributing and processing data sets at a large scale and distinguishes HiSbase from other proposals in the literature.

To give an idea of the future scalability challenges, Table 1 summarizes the size, the number of objects, and the approximate size of an individual object for three of the major current astrophysical catalogs SDSS (http://www.sdss.org/dr5/), TWOMASS (http://www.ipac.caltech.edu/2mass/), and USNO-B1.0 (http://www.nofs.navy.mil/data/fchpix/cfra.html). Assuming a HiSbase network for astrophysics with one thousand dedicated Data Grid nodes, the catalogs of Table 1 could be kept almost completely in the main memory, each node covering about 5 GB of data.

These data sets still could be managed at a single site, although with restrictions such as high transmission costs or limited resource availability. Upcoming e-science projects (see Table 2) in astrophysics and high energy physics face a data deluge which will be distributed across several sites. Examples for such upcoming projects are the Panoramic Survey Telescope and Rapid Response System (Pan-STARRS, http://pan-starrs.ifa.hawaii.edu/public/), the Large Synoptic Survey Telescope (LSST, http://lsst.org/), and the Low Frequency Array (LOFAR, http://www.lofar.org/) in astrophysics, as well as the Large Hadron Collider (LHC, http://lhc.web.cern.ch/lhc/) in high energy physics.

Table 1
Current astronomical data sets

Catalog	Size (TB)	No. of objects (million)	≈Object size (kB)
SDSS (DR5)	3.6	215	14
TWOMASS	1	471	2
USNO-B1.0	0.08	1000	0.9

Table 2
Upcoming e-science data sets

Project	Daily data rate (TB)	Yearly rate (PB)
Pan-STARRS	10	4
LSST	18	7
LOFAR	33	12
LHC	42	15

Researchers usually access and analyze logically related subsets of these data volumes. The restrictions of such subsets are mostly based on specific data characteristics. Typical access patterns over astrophysical data sets are point-near-point queries, point-in-region queries, and nearest-neighbor-searches. Such queries are usually *region-based*, i.e., they process data within certain regions of the sky. These regions are specified by the two-dimensional celestial coordinates *right ascension* and *declination*. Region-based queries can, of course, also contain predicates on attributes other than the celestial coordinates. In case of celestial objects, other attributes might comprise detection time, catalog-identifier, temperature, or energy level. In Fig. 1, objects of the same region in the sky would have the same shape and can be

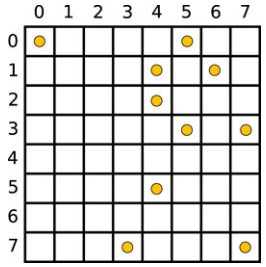


Fig. 3. Sample data space with skewed data distribution.

processed locally at the peer which is responsible for the respective region.

After a period of grace of about one year, basically all outcomes of astrophysical projects supported by public funding become publicly available. An increasing share of scientific research is performed by looking "at databases" rather than by looking directly at the sky. In order to ensure reproducibility, published data sets are not changed. Instead, new additional versions are made available.

Through the outreach of astronomy projects, many amateurs, school children, and university students do their research on these data sets. Basically, everyone who is able to surf the web can access astrophysics data. Therefore, suitable information systems need to support many users.

Traditionally, *federated Data Grids* retain data autonomy, i.e., the participating institutes keep full control over their data and deploy security policies that only allow users with appropriate credentials to access shared data resources. In situations such as the ones described above, where institutes are keen on making already published data sets available to a large audience, community-driven Data Grids constitute an interesting approach to distributed data management.

3. Locality preservation

To allow efficient query processing on logically related data sets we need to *preserve the locality of data*. Data locality is especially important for the performance of data analysis tasks in astrophysics. Distributing data objects randomly across a global information network severely impairs the performance of astrophysical query patterns.

3.1. Data skew

Many application domains have highly skewed data sets. This skew originates from data spaces with a mix of densely and sparsely populated regions. The differences in data density may arise from the original data distribution or from the fact that some regions have been investigated more extensively than others, i.e., more data has been collected and is available. In astrophysics, celestial objects are not distributed uniformly over the sky, e.g., considering high data density in the galactic plane or a supernova. We use an abstract skewed data sample (Fig. 3) for illustration.

In HiSbase, we preserve spatial proximity to efficiently process region-based queries (Section 4.1) while addressing the imbalance of the data distribution. HiSbase achieves this goal by calculating a histogram that equips the Data Grid with a community-specific data distribution. Among others, we describe the Z-quadtree histogram data structure that we designed to preserve spatial locality for astrophysics data sets. Z-quadtrees are *quadtrees* whose leaves correspond to histogram buckets and are linearized on the DHT key space using a space filling curve. These trees provide efficient access to histogram buckets (regions) while balancing the data load across data nodes.[2] The extension of histogram data structures to additionally consider query skew is part of ongoing work and we outline some of our ideas in Section 4.5.

3.2. Histogram data structures

HiSbase enables communities to design data structures for distributing their data across several nodes and to adapt to data and query characteristics of that particular community. We call these data structures *histograms* for their similarities to standard histograms. Histograms are, for example, commonly used in relational database management systems as means for selectivity estimations [24].

Within HiSbase, histograms H are used in order to look up multi-dimensional areas A and points p.

lookupArea(H,A) : S This method plays a central part during query processing. Given a multi-dimensional data area A, *lookupArea* returns the set S of region identifiers of histogram H which intersect with A.

lookupPoint(H,p) : r Mainly used during data distribution, *lookupPoint* returns the region identifier r of histogram H which contains a multi-dimensional data point p.

Most of the following histogram data structures are inspired by the intensive research conducted by the computer science community on locality-aware data structures developed for accessing and efficiently storing multi-dimensional data [10,28]. The individual community is free to choose any data structures implementing the interface required by HiSbase and, therefore, we are strengthening the histogram-aspect rather than the aspect of indexing multi-dimensional data.

3.2.1. Z-quadtree: A histogram based on quadtrees

The shape of data partitions defined by candidate data structures should be simple (e.g., squares). This allows simple (SQL) queries to retrieve data during the process of integrating new peers (see Section 4.2).

In the following, we describe the *Z-quadtree* as our preferred data structure which is inspired by *quadtrees* [27].

A Z-quadtree partitions the data space according to the principle of recursive decomposition. For a d-dimensional data space, a Z-quadtree node either is a leaf with a d-dimensional data region or an inner node with 2^d children. The leaves of the quadtree correspond to the histogram buckets. After the Z-quadtree buckets are calculated they are linearized using the Z-order space filling curve [21].

The *linearization* is then used to map the buckets on the DHT key space. We use a space filling curve instead of a random mapping as the curve preserves spatial proximity if one peer covers several buckets. If buckets are adjacent, they are likely to be managed by the same peer.

Starting with a single leaf covering the entire data space, we sequentially insert the training set into the tree (Section 4.1). If the number of objects in the area of a leaf exceeds a predefined threshold, its capacity, the leaf is split into 2^d subareas according to the quadtree splitting strategy. Inner nodes forward the objects to the corresponding child. On the left in Fig. 4, we show the decomposition of our two-dimensional example data set of Fig. 3 using a leaf capacity of two objects. After the complete training set is inserted, each leaf is assigned a *region identifier* using a depth-first search (Fig. 4, middle). This immediately gives the desired leaf

[2] In the following, we use the terms *regions* and *histogram buckets* interchangeably. The *leaves* of a Z-quadtree represent the histogram buckets for that particular histogram data structure.

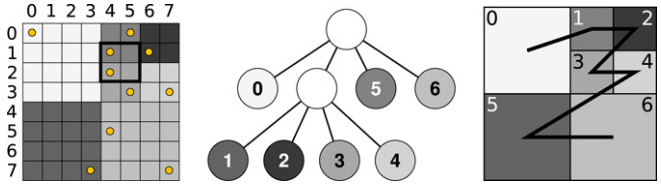


Fig. 4. Left: Z-quadtree regions of our data sample. Middle: Corresponding quadtree. Right: Leaf linearization.

linearization which is shown in Fig. 4 on the right. While using the Z-order is the canonical leaf linearization, other space filling curves such as the Hilbert curve [16] are also applicable. Without the region linearization, queries intersecting multiple regions would most likely introduce additional traffic as the regions would be located at multiple nodes. Yet from our experience, most queries intersect one region only.

Algorithm 1 describes how the set S of region identifiers that intersect with a query area A is retrieved in a Z-quadtree H. Starting at the root node, *lookupArea* is executed recursively. If the region r_n of a leaf n intersects with query area A, its region identifier $r_n.id$ is added to the result set S. Intersecting inner nodes invoke *lookupArea* on every subtree. The method to find the region which contains a data point, *lookupPoint*, can be realized similarly.

Algorithm 1 *lookupArea*(H, A) for Z-quadtrees

Require: Z-quadtree H with root node n_{root}, query area A
Ensure: Set $S = \{$region id $r.id \mid$ region r intersects with $A\}$
  $S \leftarrow \{\}$
  $n \leftarrow n_{root}$
  if region r_n of n intersects with A **then**
    if n is leaf **then**
      $S \leftarrow S \cup \{r_n.id\}$
    else {n is inner node}
      for all subtrees H_{child} of n **do**
        $S \leftarrow S \cup lookupArea(H_{child}, A)$
      end for
    end if
  end if

Z-quadtrees use the same concept as *linear quadtrees* [13], a data structure used in image encoding. Using a lower resolution for sparsely populated data subspaces in Z-quadtrees corresponds to compressing the representation for common subpixels of the linear quadtrees.

In contrast to the original quadtree, which is a spatial index structure, the Z-quadtree is used for data dissemination, as a routing index, and during query processing. The actual training data used to create a histogram is not stored in the data structure distributed to all peers.

Lookups performed during data feeding and query processing benefit from the regular structure of the quadtree leaves. The *center splitting strategy* divides the region of a node into equally sized subregions and is the default strategy for quadtrees. Therefore the (final) quadtree is insensitive to the insertion order of the data. Furthermore, the tree can be stored and communicated in a very compressed form as region boundaries can be derived by recursively dividing the complete data space until the position of the region is reached.

As the capacity of quadtree leaves is only an upper bound, not all quadtree leaves will be fully filled. Rare pathological cases, e.g., a high data concentration in a very small area of the data space, might result in a degenerated tree having many empty regions. While we define the Z-quadtree top-down, we actually build the Z-quadtree bottom up. We prefer bottom-up construction over building the tree top-down because the latter requires to determine the capacity for the quadtree leaves before starting the training phase as splitting a leaf is triggered if its capacity is exceeded. Furthermore, building Z-quadtrees bottom-up is a requirement for other splitting strategies to work correctly such as the median splitting strategy introduced in the following section.

Algorithm 2 Publish data in HiSbase

Require: Histogram H, multi-dimensional data point p
  Region id $r \leftarrow lookupPoint(H, p)$
  Send $newPointMessage(p)$ to r.

Algorithm 3 Query data in HiSbase

Require: Histogram H, multi-dimensional query area A.
  Set of relevant region ids $S_R \leftarrow lookupArea(H, A)$
  Select *coordinator* r_c from S_R
  Send $newQueryMessage(A, S_R)$ to r_c.

3.2.2. Related histogram data structures

In [30], we report on experiments with an additional quadtree-based histogram which uses a *median splitting strategy* in order to address the issue of empty leaves. It uses *median-based heuristics* for splitting a leaf at the median instead of at the center. For our astronomical example, the heuristics determine the split point (m_{ra}, m_{dec}) by computing the median for *ra*-coordinates and *dec*-coordinates independently. Our heuristics are similar to the technique used by *optimized point Quadtrees* [9], which only compute the median in the first dimension and thus guarantee that no leaf contains more than half the data of the original leaf. In the average case, our heuristics offer a better data distribution by computing the median in all dimensions independently. Fig. 5 contrasts a quadtree with regular decomposition (Fig. 5(a)) with a quadtree using our median heuristics (Fig. 5(b)), respectively. We furthermore discuss how application-specific data structures, such as the *zones* index [14], can be applied as histogram data structure offering various trade-offs. Further interesting spatial or multi-dimensional data structures can be found in the survey by Gaede and Günther [10] and the book by Samet [28].

4. Architectural design

The architectural design of HiSbase offers researchers a framework for data and resource sharing within their community. Algorithms 2 and 3 formally define the interface for data publication and access within HiSbase.

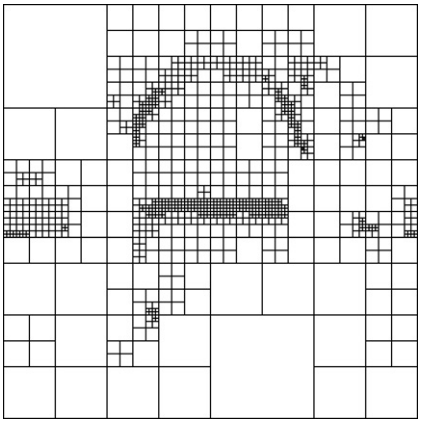
In this section, we outline the creation of histograms during the training phase and the information maintained at HiSbase nodes. Finally, we describe data publication and node collaboration during query processing.

4.1. Training phase (Histogram build-up)

The Training phase comprises three steps:

(i) Extracting the training samples,
(ii) defining the partitioning of the data space,
(iii) and distributing the partitions to the data nodes.

For constructing the histogram, data from each data source is taken into account. We can either use the entire data archive or a representative subsample. However, transmitting the entire data archive for histogram extraction is presumably prohibitive. For example, the subset could be extracted using a random sample. We achieved good histograms using 10 percent data samples in our *a priori* analysis. Such an *a priori* analysis is applicable as the



(a) Center splitting strategy. (b) Median splitting strategy.

Fig. 5. A Z-quadtree employing different splitting strategies.

data distribution does not change significantly very often (e.g., on a yearly basis) in many scientific domains.

After the training set is inserted into the histogram, the histogram is serialized for distribution within the network. We note that only the histogram structure is serialized. The training data is discarded.

The resulting histogram is passed to the initial peer in the HiSbase network. Peers subsequently joining the network receive the histogram from any other peer in the network. So each peer keeps a copy of the histogram.

The number of histogram regions is determined beforehand. In our experiments, we used histograms with up to ten times more regions than the anticipated number of peers. This offers a good trade-off between allowing more peers than initially estimated, histogram size, and complexity of finding the relevant regions during query processing. The size of the histogram is small in comparison to the amount of data transmitted during query processing. As peers presumably get their histogram from a physical neighbor, histogram distribution adds little overhead to the setup phase of the HiSbase network.

4.2. HiSbase network

While the overall design of HiSbase abstracts from the underlying DHT implementation, we use the distributed hash table (DHT) infrastructure *Pastry* [26] to manage peers and route messages in HiSbase. Like Chord [33], Pastry maps data and peers to a one-dimensional key ring. In contrast to Chord, Pastry optimizes the initial phase of routing by preferring physical neighbors to speed up communication within the overlay network.

4.2.1. Mapping nodes to regions

The histogram regions are uniformly mapped onto the DHT ring identifiers. Remember, the skew is accounted for by varying the size of the regions. In the case of the Z-quadtree, the histogram regions correspond to the leaves. Due to this uniform distribution, all regions are mapped to a peer with equal probability regardless of their individual size. The size of regions might vary due to the adaption to data skew. The peers get a random identifier and are responsible for regions close to their identifier. Fig. 6 illustrates the evenly distributed regions (0–6) and their mapping to randomly distributed peers (a, b, c, d) on the DHT key space. We use the routing of the underlying DHT system to automatically assign regions to peers. To ensure that messages destined for a specific region are received by the appropriate peer, we use the region identifiers for message routing.

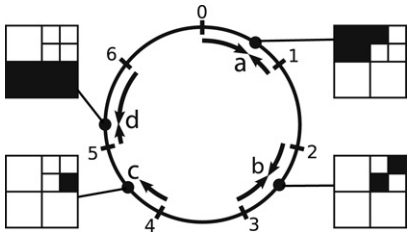


Fig. 6. Mapping of the quadtree of Fig. 4 to multiple peers.

We prefer to use the key-based routing functionality of the underlying DHT infrastructure over using a direct mapping of histogram buckets on peers or using a centralized directory for the histogram in combination with a histogram cache at the individual peers. A direct mapping would require every peer to maintain the complete list of participating peers and also the mapping of the individual histogram buckets to the peers. Using the key-based routing, each peer stores only $O(\log n)$ neighbors and the mapping is done automatically by the underlying fabric. Updating a histogram via a distributed broadcast is not more expensive than distributing an updated histogram from a central site. We can reuse functionality already implemented by the P2P substrate and leverage the increased flexibility and the automatic handling of node failures.

4.2.2. Evolving the histogram

The histogram serves HiSbase as a partitioning function, defining the data set a node is responsible for. To either achieve a better load-balancing or level long-term data distribution changes, HiSbase nodes maintain three histograms and their accompanying data sets. Each pair of histogram and data set can evolve during the run-time of HiSbase and has one of the following three functionalities: the *build-up*, *active*, and *backup* functionality.

build-up The currently running *feeding* process, which is described in the following section, distributes data according to the build-up histogram. After a new histogram has been distributed among the peers, HiSbase prepares this build-up data set and stores it on disk.

active Once the build-up phase is completed, the active histogram and data set are used during *query processing* and nodes keep them completely (or at least the relevant parts) in main memory. The active histogram is further used for messaging.