NOTICE: this is the author's version of a work that was accepted for publication in Future Generation Computer Systems. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication.
Please cite this article as: T. Scholl, et al., Scalable community-driven data sharing in e-science grids, Future Generation Computer Systems (2008), doi:10.1016/j.future.2008.05.006

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

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 does not cope with the high query load and the 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 minutes), 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 investigated technologies are *community-driven Data Grids* which use decentralized Peer-to-Peer (P2P) technologies in order to provide scalable communication and data management. Community-driven Data Grids are built on the

[⋆] 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.
Email addresses: scholl@in.tum.de (Tobias Scholl), bauerb@in.tum.de (Bernhard Bauer), gufler@in.tum.de (Benjamin Gufler), kuntschk@in.tum.de (Richard Kuntschke), reiser@in.tum.de (Angelika Reiser), kemper@in.tum.de (Alfons Kemper).

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

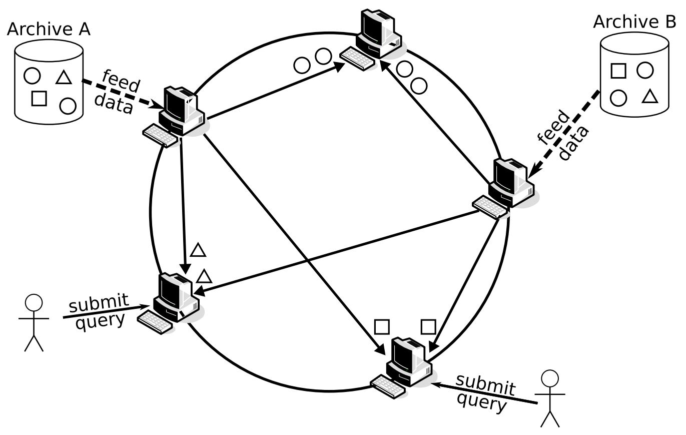

Fig. 1. HiSbase architecture.

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 the 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 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. Figure 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 to reconsider 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 like 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.

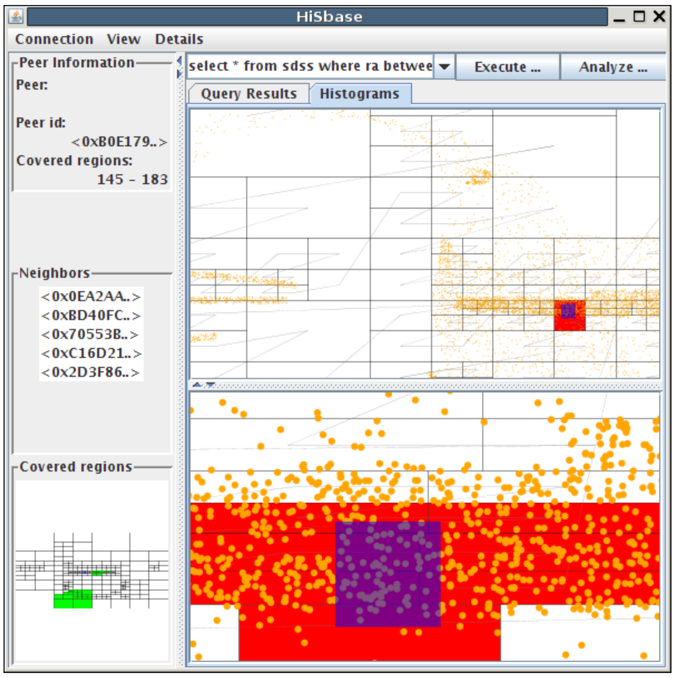

Fig. 2. The HiSbase GUI

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 the PlanetLab test bed in order to demonstrate the functionality of our system. Figure 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.

Catalog	Size	No. of objects	≈ object size
SDSS (DR5)	3.6 TB	215 million	14 KB
TWOMASS	1 TB	471 million	2 KB
USNO-B1.0	0.08 TB	1 000 million	0.9 KB

Table 1
Current astronomical data sets.

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

Table 2
Upcoming e-science data sets.

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 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.

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 Figure 1, objects of the same region in the sky would have the same shape and can be processed locally at the peer which is responsible for the respective region.

After a grace period 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

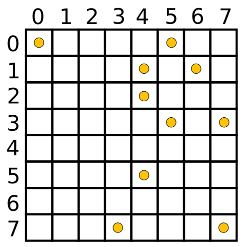

Fig. 3. Sample data space with skewed data distribution.

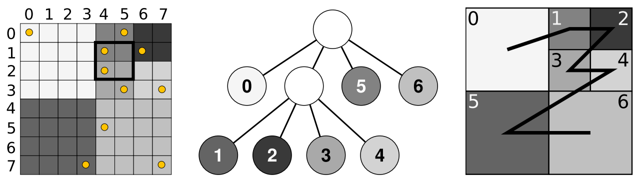

Fig. 4. Left: Z-quadtree regions of our data sample. Middle: Corresponding quadtree. Right: Leaf linearization.

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 kids, 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 (Figure 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].

[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.

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 Figure 4, we show the decomposition of our two-dimensional example data set of Figure 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 (Figure 4, middle). This immediately gives the desired leaf linearization which is shown in Figure 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

```
Input: Z-quadtree $H$ with root node $n_{root}$, query area $A$
Output: Set $S = \{\text{region id } r.id \mid \text{region } r \text{ intersects with } A\}$
  $S \leftarrow \{\}$
  $n \leftarrow n_{root}$
  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.

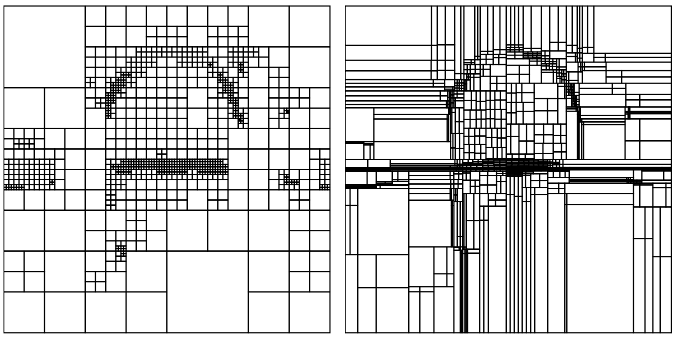
(a) center splitting strategy (b) median splitting strategy

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

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.

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 offers a better data distribution by computing the median in all dimensions independently. Figure 5 contrasts a quadtree with regular decomposition (Figure 5(a)) with a quadtree using our median heuristics (Figure 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

Algorithm 2 Publish data in HiSbase

```
Input: 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

```
Input: 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 $r_c$.
```

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 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 does not add much 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. Figure 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.

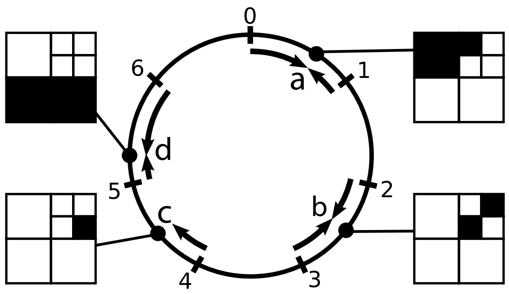

Fig. 6. Mapping of the quadtree of Figure 4 to multiple peers.

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

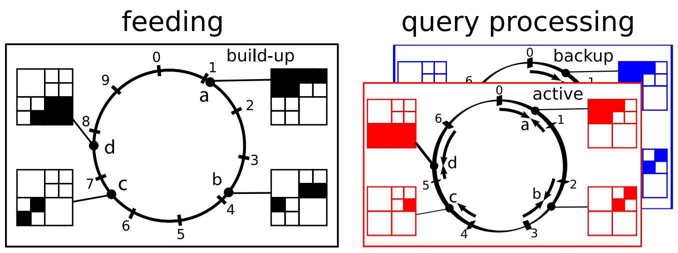

Fig. 7. Histogram evolution.

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.

backup The completed build-up data set is additionally kept on disk as backup for the active data set. This preserves the active data set beyond the lifetime of the current network and can be used if a node is restarted with the same identifier.

Figure 7 illustrates a scenario where the build-up histogram contains additional regions while the active and backup histograms are the same as in Figure 6.

Any of the participating nodes can be used to inject an updated version of a histogram by broadcasting it to the HiSbase network.

4.2.3 *Node Arrival*

When a node joins the HiSbase network, the active histogram will be transmitted to that node and the node needs to receive the data according to its responsibilities. For this purpose, HiSbase reuses the mechanisms of the DHT structure to determine the arrival of new nodes. In Pastry [26], nodes are notified if the leaf set (the nodes which have similar identifiers) changes. Algorithm 4 describes how a notified node determines the data it is no longer responsible for. It then redistributes this data and the newly joined peer can update its database.

4.2.4 *Node Departure*

HiSbase is developed for an environment where the participating servers are quite reliable. High churn is currently not in our focus as distributing the envisioned amounts of data across unreliable peers is not very useful. Nonetheless, some peers might temporarily fail. As mentioned in the introduction, HiSbase does not *replace* but *complement* the "traditional" data centers since these also serve as data sources for distributing the data in HiSbase. A peer that recognizes the departure of a neighboring node and needs to take over parts of the data refetches the data from the according archives.

Algorithm 4 Handling node arrivals

Node p covers a set of regions P. Let P_{new} be the set of regions p is responsible for after a new node has arrived. a_i denotes the area of a region i.

```
if $P_{new} \neq P$ then
    find $P_{move} = P \setminus P_{new}$
    for all $r \in P_{move}$ do
        $a_r = getArea(r)$
        redistribute data from $a_r$ to region $r$
    end for
end if
```

4.3 *Data Distribution (Feeding)*

Connected data centers directly feed data into HiSbase as suggested by Figure 1. Data integration is not in the focus of our work. We assume that the data being fed into HiSbase adheres to a common schema or is already properly transformed. In HiSbase, the histogram is used to determine which peer stores which data. All peers maintain the data objects which are in their histogram buckets, independently from the archive the data comes from. HiSbase abstracts from the specific database system which allows the use and comparison of various traditional as well as main memory database systems.

Data archives which want to publish their data in HiSbase connect to any HiSbase peer, preferably to a peer nearby or to a peer which has a high network bandwidth. Proceeding according to Algorithm 2, the peer uses its histogram to determine which histogram bucket contains a data object by using the *lookupPoint* method. Then it routes the object to the DHT identifier of this region. The message contains the data object and information about the data source. Via the underlying DHT mechanism, the data item arrives at the responsible peer which updates its database.

Distributing each data item individually would introduce a very high overhead. The precomputed histogram allows us to optimize the feeding stage by introducing *bulk feeding*. A peer which feeds the network can buffer several objects for the same region until a threshold is reached. Time-based as well as count-based thresholds are applicable.

Integrating new data sets is achieved by feeding them into the network as described above after the according tables are created at each node. If the new data set is a detailed survey of a sky region that has not yet been covered by any existing archive in the community network, it might be appropriate to create a new histogram in order to improve the data load balancing. In that case, a data sample of the survey is extracted and integrated into the training phase.

4.4 *Query Processing*

Region-based queries are submitted to any peer of the HiSbase network. The peer extracts the multi-dimensional area A from the query predicate. It selects an arbitrary identifier r_c from the set of intersecting regions which is determined by *lookupArea*. The peer p_c which is responsible for region

r_c is the *coordinator*. The coordinator collects intermediate results and performs post-processing tasks (e. g., duplicate elimination).

Let us assume a region-based query was issued at peer d in Figure 6. The area of the query is marked with the thick-lined rectangle in Figure 4. The regions relevant to our example query are the regions 1 and 3. If peer d covers regions relevant to the query, it becomes the coordinator itself. This is not the case in our example. We select region 1 as r_c and thus peer a becomes the coordinator. Peer d forwards a coordination request to peer a. The coordination request contains the query and the relevant regions. After peer a receives the coordination request, it issues the query to its own database (as it covers relevant regions) and sends the query to all other relevant regions. Peer b also participates in the query processing in our example as it covers region 3. It sends its intermediate results back to the coordinator, peer a. After having received all intermediate results, peer a sends the complete result to peer d.

Peers may cover several regions. As region identifiers are used for submitting queries, peers can receive the same query several times. Each peer stores a hash of currently processed queries to avoid multiple evaluations of the same query. Results and error messages are directly transmitted to the coordinator or the submitting node without using the overlay routing algorithm.

4.5 *Query Load-Balancing*

Currently we are looking at several techniques for combining our data load balancing approach with query load balancing techniques to efficiently handle query hot spots. We investigate extensions to our training phase as well as techniques which redistribute load during run-time.

We currently enhance the training phase with query statistics such as earlier workloads. Based on these statistics, the data partitioning can be modified to enable the application of query load balancing techniques such as replication or load migration.

Using two parallel Pastry rings with different histograms increases the data availability within the HiSbase network. By changing the offset (or even the space filling curve) of the mapping process from Section 4.2.1, the second histogram stores the data on different nodes and both copies are available during query processing.

We are also considering to introduce a *master-slave* hierarchy, where idle peers can support overloaded nodes by offering their storage and compute resources. These may be necessary to cope with short-term changes in query load distribution. Whether a peer is overloaded or constitutes a potential slave-node is determined based on workload statistics collected during run-time. These statistics can also augment the training phase for the next histogram evolution (see Section 4.2.2).

5 System Evaluation

We evaluate HiSbase by performing throughput measurements with our Java-based prototype [29] using the *FreePastry*[3] implementation of Pastry. The evaluation data set comprises about 137 million objects from subsets of the ROSAT (25 million objects), SDSS (84 million objects), and TWOMASS (28 million objects) catalogs and has a size of about 50 GB. Figure 8 illustrates the data skew of these data samples.

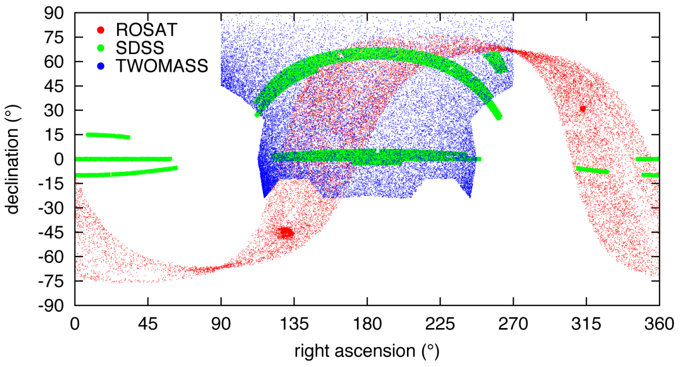

Fig. 8. Evaluation data sets.

5.1 *Throughput Measurements*

We measure throughput for varying *multi-programming levels (MPLs)*, i. e., a varying number of parallel queries in the system, to evaluate at what degree of parallelism a distributed architecture can outperform a centralized solution. Each run has k peers, a batch containing l queries, and an MPL m. MPL=m denotes that *each* peer keeps m parallel queries in the system. At the start of a run, each peer immediately submits m queries. We measure timestamps $s_{p,q}$ and $r_{p,q}$ when peer p has $s$ubmitted its q-th query and has $r$eceived the results, respectively. After receiving an answer, peers submit their next query in order to sustain the multi-programming level.

For measuring the throughput T, we only consider queries processed in the time span when every peer is guaranteed to work on MPL=m parallel queries, the *saturation phase* I_{sat}. I_{sat} is the time interval between the point in time when the last peer has submitted its m-th query and the first peer has submitted its last query, which is expressed formally as:

$$I_{sat} = \left[\max_{1 \le p \le k} (s_{p,m}), \min_{1 \le p \le k} (s_{p,l}) \right] \qquad (1)$$

We shortly illustrate the case of computing I_{sat} for one single HiSbase node p_1. Let MPL=10 and $l = 500$, then the node submits 10 queries to HiSbase in order to reach the desired degree of parallelism. In this scenario, I_{sat} starts at $s_{1,10}$. As soon as a query result is received, a new query is issued to HiSbase. Finally, the saturation phase I_{sat} ends when the node submits its last (500$^{\text{th}}$) query at timestamp $s_{1,500}$. When multiple nodes participate in the HiSbase network, the last $s_{p,10}$ and the first $s_{p,500}$ timestamp determine the saturation phase of the complete network, as defined in Equation 1.

The *throughput* T is based on the number of successfully processed queries during the saturation phase I_{sat}:

$$T = \frac{|\{(p,q) \mid r_{p,q} \in I_{sat}, 1 \le p \le k, 1 \le q \le l\}|}{\overline{I_{sat}}} \qquad (2)$$

We used a body of 730 *cross-match* queries for our evaluation. Cross-match queries determine whether data points from

[3] http://freepastry.org

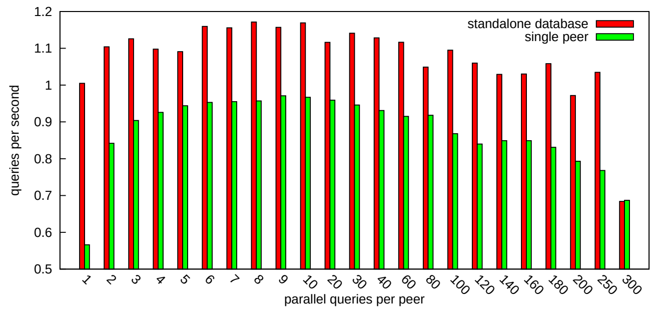

Fig. 9. Query throughput results for the standalone database and single peer configurations.

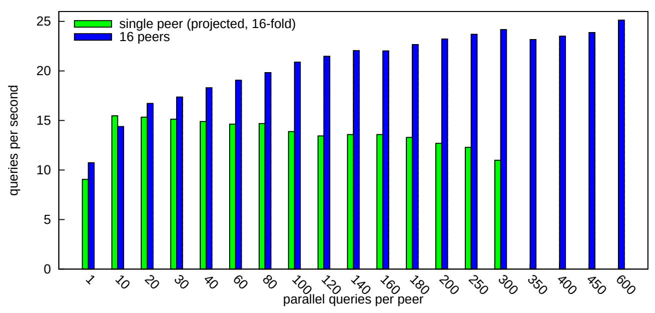

Fig. 10. Throughput comparison of the multi-peer instance with the projected values of the single-peer configuration.

different sources are likely to stem from the same celestial object. The queries were created from 730 random sources of the SDSS catalog, using rectangular regions with an edge length of 0.05°. The size of the query rectangles is based on realistic values and each query covers approximately an area which is $\frac{2}{10^7}$ of the whole sky. Peers submit these queries in random order. To this end, we present results for a histogram based on a Z-quadtree with 256 regions using the center splitting strategy.

5.1.1 *Single Peer Instance*

The first experiment compares the query throughput of a standalone database with the query throughput of the same database used by a single HiSbase peer to measure the overhead introduced by the HiSbase layer. The peer is a Linux server with an Intel Xeon processor at 3.06 GHz, 2 GB RAM, and IBM DB2 V8.1. Queries to the standalone database are submitted via parallel JDBC connections. Figure 9 shows the throughput in queries per second of the standalone database and the single peer HiSbase instance. The throughput increases for both single node setups through higher parallelism until their maximum throughput (sweet spot) is reached. The maximum throughput of both systems is roughly at 10 queries: 1.17 queries per second at MPL=8 for the standalone database and 0.97 for the single peer HiSbase instance at MPL=9. Although the standalone database performed better than the single HiSbase node in our evaluation, HiSbase introduces an acceptable overhead as in practice an instance with multiple (typically hundreds of) peers is used.

Just to give an impression of current throughput figures, the traffic statistics of the SkyServer[4] archive show that in 2007 during an average month about 2 312 queries have been submitted to the SQL interface which corresponds roughly to less than one query per second. However, there are already several occasions where the number of queries per second is significantly higher.

5.1.2 *Multi-Peer Instance*

We tested a multi-peer instance in a local area network (LAN) which measures how HiSbase performs in a setting with low latency and high network bandwidth. The LAN configuration of HiSbase was set up on 16 consumer-class Windows PCs equipped with 1.6 GHz Processors, 512 MB RAM, and again with the IBM DB2 V8.1 database system. Figure 10 contrasts the projected throughput of the single peer configuration described above (by multiplying the previous results with 16) and the 16-peer instance. The 16 peers achieve a stable super-linear throughput compared to the single peer from MPL=20 onwards. Smaller data partitions and especially a higher cache locality constitute this throughput improvement as peers only process similar queries. We did not continue the measurements beyond an MPL=600, which corresponds to 9 600 parallel queries, as expected numbers of parallel users are currently below this degree of parallelism.

5.1.3 *AstroGrid-D and PlanetLab Instance*

In order to verify the scalability of our HiSbase approach, we also conducted benchmarks on resources within AstroGrid-D and D-Grid as well as on the PlanetLab test bed, as PlanetLab is widely used for evaluating globally decentralized applications. In PlanetLab, applications run in so-called *slices* (virtual machines) and in parallel with several other installed applications. Within the AstroGrid-D test bed, the resources are more dedicated, reliable, and have high-bandwidth links. We successfully demonstrated HiSbase using up to 56 resources from our labs, the AstroGrid-D test bed, and on PlanetLab. Performing throughput measurements on such a distributed and heterogeneous environment has many challenges which we want to summarize briefly.

For demonstration purposes, we used the Derby database system which is a pure-Java embedded database developed by Apache. For performing our benchmarks, however, Derby cannot keep up with the performance of full-fledged commercial database systems. Deploying these commercial database systems on all network nodes is not only difficult with regards to licencing issues but also with regard to maintaining the infrastructure. Taming the heterogeneity of the resources is also a non-trivial task as different protocols are needed for transferring data and accessing nodes. Within the LAN of our lab, data either resides on local hard disks or a network attached storage (NAS) and therefore it is easy to administer and to harvest the results. Data transfer between Grid nodes is performed via *GridFTP* and *gsissh*, while in the PlanetLab network *ssh* and *scp* are used. Supporting research communities concerning these practical issues clearly is an important aspect of solving their data management challenges.

[4] http://skyserver.sdss.org/log/en/traffic/

6 Related Work

The HiSbase approach provides several benefits to e-science communities by addressing domain specific data and query characteristics. HiSbase offers a higher throughput via parallelization, higher cache locality, and load-balancing across several sites compared to centralized data management. HiSbase enables scalable sharing of decentralized resources within a community as it uses the DHT mechanism of key-based routing for data distribution and message routing. Using these techniques, new nodes are easily added to the network and heterogeneous database management systems can be integrated with little effort as each HiSbase peer only needs to know its own local database configuration.

Using parallelism and partitioning to increase query throughput is a well-established technique from distributed and parallel databases [18]. Compared to HiSbase, distributed databases run in a more homogeneous setting whereas parallel databases are not designed for world-wide distributed resources. Autonomous database systems [22] also deal with the correlation of several data sources. However, data is not distributed across participating servers (adhering to the nodes' *autonomy*) and thus correlation needs to be done at the client sites which leads to additional data traffic.

DHT architectures such as CAN [25], Chord [33], Pastry [26], and Tapestry [36] overcome the limitations of centralized information systems by storing data in a distributed one-dimensional key space (except for CAN which uses a d-dimensional torus). While these systems achieve load-balancing by randomly hashing data and peers to their key space, they do not support multi-dimensional range queries or preserve spatial locality.

A large variety of systems have been proposed to support (multi-dimensional) range queries [4,12,32,34] or to address data (or execution) load-balancing in P2P environments [2,7,11,23]. These systems are predominantly designed for settings that are very dynamic, i. e., data hot spots and the data itself change very frequently and the systems have a very high churn. This flexibility comes at the price of dealing with each data object (of several hundred million data objects) individually. We exemplify some of these systems below and discuss how they relate to HiSbase.

One approach [4] uses Voronoi diagrams in order to partition the data space and to support queries on multi-dimensional data. Independently, MURK [12] uses k-d trees to realize a similar idea. In these systems, peers covering large data partitions have more neighbors, while in HiSbase the number of neighbors is independent from the number and size of covered regions. SCRAP [12] directly applies a space filling curve to the data and assigns one-dimensional ranges to peers. In HiSbase, the submitting peer exactly determines the histogram regions in the multi-dimensional data space and only these peers are contacted during query processing while SCRAP can only approximate a multi-dimensional query range using multiple one-dimensional ranges.

The distributed quadtree index [34] also supports range queries and objects with multi-dimensional extents. The representatives for the quadtree leaves are randomly placed on the key space of an underlying DHT structure (e. g., Chord [33]). To a certain level (f_{min}) no objects are stored (to avoid the bottleneck of higher-layer nodes) or to avoid too much fragmentation (f_{max}). Each peer caches direct links to the children of the quadtree nodes it is covering. Thus, it takes $O(\log n)$ hops to find an f_{min}-node and then a constant number of steps to reach the relevant leaves. This number of steps also has to be processed with data objects without an extent which are stored at level f_{max}. In HiSbase, no additional routing steps are necessary. HiSbase discovers the relevant region directly and routes to the responsible peer using $O(\log n)$ messages.

Also based on quadtrees, an on-line balancing algorithm for frequent changes in data hot spots has been described [32]. The quadtree leaves are mapped on a skip graph [3] layer using a space filling curve. While the concept is similar to Z-quadtrees, peers need to cover quadtree leaves on the same tree level while in HiSbase there is no such restriction. Accounting for stable data distributions, e-science communities might not benefit as much from such an approach as from techniques increasing query throughput.

How to achieve load balancing in one-dimensional, range-partitioned data is described in [11,2]. The authors of [11] show that load balancing schemes for range-partitioned data in highly dynamic P2P networks either need to adjust the load between neighbors or need to change peer positions within the range. SCRAP is an extension of [11] to multi-dimensional data. In [2], only representative values of the data ranges are maintained in the skip graph. Load balancing between these data ranges is achieved by arranging less-filled (open) buckets close to full (closed) buckets. HotRod [23] addresses query hot spots on one-dimensional data by replicating popular data ranges on additional rings. Their query load-balancing technique could be integrated with our data load-balancing.

P-Ring [7] addresses data skew in an orthogonal manner in comparison to HiSbase. While HiSbase adapts the buckets of the histogram data structure to data skew and distributes these across the cooperating peers, P-Ring has the notion of "helper peers" that support peers which are overloaded by skewed insertions either by data redistribution between neighbors or by merging their data into a neighbor's range. Considering multi-dimensional range queries, P-Ring would need to approximate the query area with multiple one-dimensional intervals. Using the insertion rate of 4 data items per second as in the simulation study of P-Ring, importing 80 million objects would last 33 weeks (20 million seconds) which is inappropriate for e-science communities having terabyte-scale data sets.

Related work in sensor networks (e. g., [1]) illuminates aspects of data distribution and load balancing from a different perspective where data is created within the network. Identifying synergies between our capabilities to directly support data sets on a terabyte-scale with billions of records and the existing approaches to on-line load-balancing is an interesting and challenging task for future research.

7 Conclusions and Future Work

In order to use P2P technologies for data-intensive e-science applications on the Grid, we argue that peers require additional distributed information, such as a histogram data structure. HiSbase allows e-science communities to build up decentralized and cooperative information networks and offers a framework to design histogram data structures for accommodating specific data characteristics and dominant query patterns. The histogram data structure defines a partitioning scheme to benefit from high throughput via parallelism and high cache locality and is also used as routing index for increased flexibility. HiSbase complements existing community-solutions. Given the enormous variety of use cases and applications it is unlikely to find a single best solution.

Future aspects comprise support for the IVOA *Astronomical Data Query Language (ADQL)* and to investigate other data intensive e-science applications such as data mining.

References

[1] M. Aly, K. Pruhs, P. K. Chrysanthis, KDDCS: A Load-Balanced In-Network Data-Centric Storage Scheme for Sensor Networks, in: Proc. of the ACM Intl. Conf. on Information and Knowledge Management, Arlington, VA, USA, 2006.

[2] J. Aspnes, J. Kirsch, A. Krishnamurthy, Load Balancing and Locality in Range-Queriable Data Structures, in: Proc. of ACM Symposium on Principles of Distributed Computing, St. John's, Newfoundland, Canada, 2004.

[3] J. Aspnes, G. Shah, Skip Graphs, in: Proc. of the ACM/SIAM Symposium on Descrete Algorithms, Baltimore, MD, USA, 2003.

[4] F. Banaei-Kashani, C. Shahabi, SWAM: A Family of Access Methods for Similarity-Search in Peer-to-Peer Data Networks, in: Proc. of the ACM Intl. Conf. on Information and Knowledge Management, Washington, DC, USA, 2004.

[5] R. Braumandl, A. Kemper, D. Kossmann, Quality of Service in an Information Economy, ACM Trans. on Internet Technology 3 (4) (2003) 291–333.

[6] A. Carlson, H. Böhringer, T. Scholl, W. Voges, Finding Galaxy Clusters using Grid Computing Technology, in: Proc. of the IEEE Intl. Conf. on e-Science and Grid Computing (demo), Bangalore, India, 2007.

[7] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, J. Shanmugasundaram, P-Ring: An Efficient and Robust P2P Range Index Structure, in: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, Beijing, China, 2007.

[8] H. Enke, M. Steinmetz, T. Radke, A. Reiser, T. Röblitz, M. Högqvist, AstroGrid-D: Enhancing Astronomic Science with Grid Technology, in: Proc. of the German e-Science Conference, Baden-Baden, Germany, 2007.

[9] R. A. Finkel, J. L. Bentley, Quad Trees: A Data Structure for Retrieval on Composite Keys, Acta Informatica 4 (1974) 1–9.

[10] V. Gaede, O. Günther, Multidimensional Access Methods, ACM Computing Surveys 30 (2) (1998) 170–231.

[11] P. Ganesan, M. Bawa, H. Garcia-Molina, Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems, in: Proc. of the Intl. Conf. on Very Large Data Bases, Toronto, Canada, 2004.

[12] P. Ganesan, B. Yang, H. Garcia-Molina, One Torus to Rule them All: Multi-dimensional Queries in P2P Systems, in: Proc. of the Intl. Workshop on the Web and Databases, Maison de la Chimie, Paris, France, 2004.

[13] I. Gargantini, An Effective Way to Represent Quadtrees, Communications of the ACM 25 (12) (1982) 905–910.

[14] J. Gray, M. A. N. Santisteban, A. S. Szalay, The Zones Algorithm for Finding Points-Near-Point or Cross-Matching Spatial Datasets, Tech. Rep. MSR-TR-2006-52, Microsoft Research, Microsoft Cooperation, Redmond, WA, USA (Apr. 2006).

[15] J. Gray, A. Szalay, The World-Wide Telescope, Communications of the ACM 45 (11) (2002) 50–55.

[16] D. Hilbert, Über die stetige Abbildung einer Linie auf ein Flächenstück, Math. Ann. 38 (1891) 459–460.

[17] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, I. Stoica, Querying the Internet with PIER, in: Proc. of the Intl. Conf. on Very Large Data Bases, Berlin, Germany, 2003.

[18] D. Kossmann, The State of the Art in Distributed Query Processing, ACM Computing Surveys 32 (4) (2000) 422–469.

[19] R. Kuntschke, T. Scholl, S. Huber, A. Kemper, A. Reiser, H.-M. Adorf, G. Lemson, W. Voges, Grid-based Data Stream Processing in e-Science, in: Proc. of the IEEE Intl. Conf. on e-Science and Grid Computing, Amsterdam, The Netherlands, 2006.

[20] W. O'Mullane, N. Li, M. Nieto-Santisteban, A. Szalay, A. Thakar, J. Gray, Batch is back: CasJobs, serving multi-TB data on the Web, in: Proc. of the Intl. Conf. on Web Services, Orlando, FL, USA, 2005.

[21] J. Orenstein, T. Merrett, A class of data structures for associative searching, in: Proc. of the ACM SIGACT-SIGMOD Symp. on Principles of Database Sys., Waterloo, Ontario, Canada, 1984.

[22] F. Pentaris, Y. Ioannidis, Query Optimization in distributed Networks of Autonomous Database Systems, ACM Trans. on Database Systems 31 (2) (2006) 537–583.

[23] T. Pitoura, N. Ntarmos, P. Triantafillou, Replication, Load Balancing, and Efficient Range Query Processing in DHT Data Networks, in: Proc. of the Intl. Conf. on Extending Database Technology, Munich, Germany, 2006.

[24] V. Poosala, Y. E. Ioannidis, P. J. Haas, E. J. Shekita, Improved Histograms for Selectivity Estimation of Range Predicates, in: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, Montreal, Quebec, Canada, 1996.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proc. of the ACM SIGCOMM Intl. Conf. on Data Communication, 2001.

[26] A. I. T. Rowstron, P. Druschel, Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems., in: Proc. of the IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001.

[27] H. Samet, The Design and Analysis of Spatial Data Structures, Addison Wesley, 1990.

[28] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann, 2006.

[29] T. Scholl, B. Bauer, B. Gufler, R. Kuntschke, D. Weber, A. Reiser, A. Kemper, HiSbase: Histogram-based P2P Main Memory Data Management, in: Proc. of the Intl. Conf. on Very Large Data Bases (demo), Vienna, Austria, 2007.

[30] T. Scholl, R. Kuntschke, A. Reiser, A. Kemper, Community Training: Partitioning Schemes in Good Shape for Federated Data Grids, in: Proc. of the IEEE Intl. Conf. on e-Science and Grid Computing, Bangalore, India, 2007.

[31] P. Schücker, H. Böhringer, W. Voges, Detection of X-ray Clusters of Galaxies by Matching RASS Photons and SDSS Galaxies within GAVO, Astronomy & Astrophysics 420 (2004) 61–74.

[32] Y. Shu, B. C. Ooi, K.-L. Tan, A. Zhou, Supporting Multi-dimensional Range Queries in Peer-to-Peer Systems, in: Proc. of the IEEE Intl. Conf. on Peer-to-Peer Computing, Konstanz, Germany, 2005.

[33] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, in: Proc. of the ACM SIGCOMM Intl. Conf. on Data Communication, San Diego, CA, USA, 2001.

[34] E. Tanin, A. Harwood, H. Samet, Using a distributed quadtree index in peer-to-peer networks, VLDB Journal 16 (2007) 165–178.

[35] S. Venugopal, R. Buyya, K. Ramamohanarao, A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing, ACM Computing Surveys 38 (1) (2006) 3.

[36] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, J. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, IEEE Journal on Selected Areas in Communications 22 (1) (2004) 41–53.