Workload-Aware Data Partitioning in Community-Driven Data Grids*

Tobias Scholl   Bernhard Bauer   Jessica Müller   Benjamin Gufler
Angelika Reiser   Alfons Kemper

Department of Computer Science
Technische Universität München
Munich, Germany
{scholl, bauerb, muelleje, gufler, reiser, kemper}@in.tum.de

ABSTRACT

Collaborative research in various scientific disciplines requires support for scalable data management enabling the efficient correlation of globally distributed data sources. Motivated by the expected data rates of upcoming projects and a growing number of users, communities explore new data management techniques for achieving high throughput. *Community-driven data grids* deliver such high-throughput data distribution for scientific federations by partitioning data according to application-specific data and query characteristics. *Query hot spots* are an important and challenging problem in this environment. Existing approaches to load-balancing from Peer-to-Peer (P2P) data management and sensor networks do not directly meet the requirements of a data-intensive e-science environment. In this paper, our contributions are partitioning schemes based on multi-dimensional index structures enabling communities to trade off data load balancing and handling query hot spots via *splitting* and *replication*. We evaluate the partitioning schemes with two typical kinds of data sets from the astrophysics domain and workloads extracted from Sloan Digital Sky Survey (SDSS) query traces and perform throughput measurements in real and simulated networks. The experiments demonstrate the improved workload distribution capabilities and give promising directions for the development of future community grids.

1. INTRODUCTION

Many e-science communities such as astrophysics, climatology, or biology face huge data volumes from current experiments. Due to the expected data rates of upcoming projects, e. g., the *Panoramic Survey Telescope and Rapid Response System* (Pan-STARRS), producing several terabytes a day, current centralized data management approaches offer only limited scalability.

Correlating and combining information from various observations or experiments are key for finding new scientific insights. Additionally, published data sets are not changed in order to ensure the reproducibility of experiments. Instead, new additional versions are made available. In order to deal with the sheer data volumes, the communities join forces in *Virtual Organizations* and build infrastructures for their scientific federations, so-called *data grids* [27]. These data grids interconnect dedicated resources using high-bandwidth networks and enable researchers to share their data sets within the community. Mostly, data sets are currently provided to the whole community by the institutions conducting the experiments, hosting the data on their own servers. This approach of autonomous data management is not well-suited for the application scenario just described as each data source needs to be queried individually and (probably large) intermediate results need to be shipped across the network. Moreover, an *increasing popularity* within the user community puts high demands on the various architectural design choices, such as providing high query throughput. Another challenging aspect are *skewed data distributions* in the data sets as well as *query hot spots*.

Together with our partners from the AstroGrid-D [7] community project within the D-Grid initiative, we construct a grid environment that supports users to bring their everyday science to the grid. Our main focus are applications that access scientific databases from the grid or use grid-based data stream management [12]. It is therefore a natural choice for us to use data and workloads from the astrophysics community, though our techniques presented in this paper are also applicable to other domains.

1.1 Problem Statement

Enabling e-science communities to address the two major issues, *high-throughput data management* and *correlation of distributed data sources*, new data sharing infrastructures are required to 1) directly deal with several terabytes or even petabytes of data, 2) integrate the existing high-bandwidth networks with several hundred nodes within the community, and 3) offer high throughput to cope with a steadily growing user community.

For the scope of this paper, we focus on *how to incorporate workload-awareness into a data sharing infrastructure for e-science communities?* The fact that query workloads for scientific data sets are not uniformly distributed is substantiated by an analysis [11] of a query trace from the *Sloan Digital Sky Survey* (SDSS) [26].

When some areas of the data space are very popular within the community, data nodes covering these areas, so-called *query hot spots*, tend to become the bottleneck during query processing. In general, query load balancing can be achieved via *splitting* (parti-

tioning) and *replication* (Section 2.2).

Several inspiring proposals on load balancing within P2P data management systems [4, 9, 16] or data-centric storages [3] exist in related work. However, these proposals do not meet the requirements for data-intensive applications from e-science communities as they only apply data partitioning or do not deal with skewed data distributions. Other data structures [6] that are designed for large scale data sets currently do not address query hot spots (Section 5).

We propose *community-driven data grids* (Section 2) as a decentralized and scalable approach to scientific data management using the existing available capacities—both CPU and main memory—of the community network resources. So far, only data load balancing aspects have been considered during data partitioning within HiSbase [20, 21], our prototypical implementation of a community-driven data grid.

1.2 Contributions

In this paper, we generalize the data-driven partitioning schemes of community-driven data grids to a cost-based partitioning in order to address two important challenges in scientific federations: data and query load balancing.

For this cost model, we define several *weight functions* (Section 3). Besides data partitionings which consider only data load, we propose several weight functions that allow advanced and *workload-aware* weighting schemes. One weight function, for example, combines the weight for points and queries to compute the *heat* of regions as the product of the associated data points and queries. Finally, we describe a weight function that decides by using the *extent of queries* whether replication is better than splitting a region.

In Section 4, we evaluate our approach using quadtree-based partitioning schemes introduced in previous work [22]. We use a sample of one million queries from an SDSS query trace on three observational data sets and a synthetic workload on a uniform data sample from the Millennium[1] simulation. We further perform several throughput measurements on our local resources of the AstroGrid-D testbed as well as in a simulated network with the various partitioning schemes. The evaluation results assess the effectiveness and applicability of our load balancing techniques as presented in this paper. We conclude in Section 6 and give an outlook of ongoing work and future research issues.

2. COMMUNITY-DRIVEN DATA GRIDS

In e-science communities such as astrophysics, geosciences, and climatology, the combination of various globally distributed data sources is fundamental in order to obtain new scientific results. Therefore, most projects publish data sets from their experiments, observations, or simulations online and create *scientific federations*. In order to perform correlation queries in such a federated environment, data needs to be transmitted to one site and correlated there. Data might be replicated using mirrors, but basically each institute is responsible for providing scalable data access to its own archive (and for ensuring increased fault tolerance via replication). If communities are not depending on data autonomy, they could create a centralized data site which would offer access to all available data. While correlation now can be performed directly on-site and no network traffic beyond the results of queries is necessary, query processing needs to be handled carefully due to the centralized approach. For example, several job queues could be used in order to differentiate between short running (one minute) and long running (several hours) queries [14].
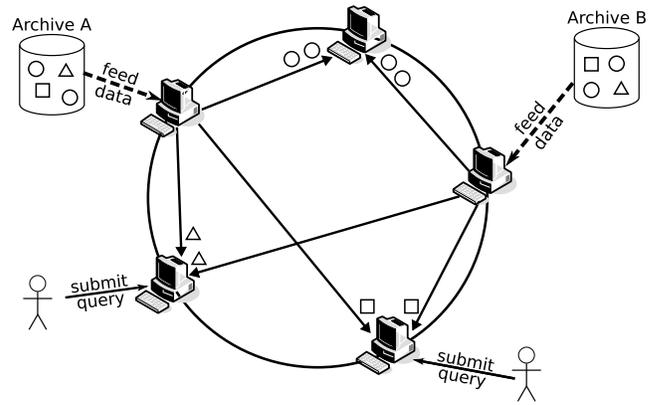
Figure 1: Architecture for community-driven data grids.

These data access services, regardless whether federated or centralized, experience increased popularity within the research community and the public domain. Thus, these data services are used by more and more people. Due to the increasing interest, scalability issues with the current approaches arise. For example, users interacting with the public SDSS archive are only allowed to run their queries for a limited amount of time and the result size is restricted in order to avoid overloading the server.

2.1 HiSbase Architecture

In the following we give an overview of HiSbase [20, 21], our prototype implementation for community-driven data grids. Community-driven data grids enable collaborating researchers to share their data sets in a common infrastructure. Technologies developed for decentralized Peer-to-Peer (P2P) architectures provide scalable communication and data management to overcome the deficiencies of centralized approaches. Based on distributed hash tables (DHT), new nodes and their resources are integrated seamlessly. Built on top of a P2P overlay network infrastructure, such as Chord [25] or Pastry [17], HiSbase provides a framework for data publication and efficient data access which adapts to the data and query characteristics of their specific domain.

Thereby, community-driven data grids achieve high throughput in query processing as data is distributed across numerous (e. g., hundreds of) nodes according to predominant query patterns. As a consequence, most processing tasks can be performed locally, achieving high cache locality as nodes mainly process queries on logically related data. 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. The data grid allocates data fed into the system by means of community-specific distribution functions. Thereby, related data objects of various sources are mapped to identical nodes.

We will use a data grid for astrophysics as running example where data from the same area of the sky is mapped to the identical node. This is appropriate for typical access patterns to astrophysical data sets like point-near-point and point-in region queries. 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*.

The data partitions distributed to nodes have a specific capacity and thus can also be seen as data buckets. As data sampling and training drive the data placement in these buckets, we also denote partitioning schemes as *histograms*.

HiSbase partitions multi-dimensional e-science data across an

(i) Splitting (Partitioning)   (ii) Replication
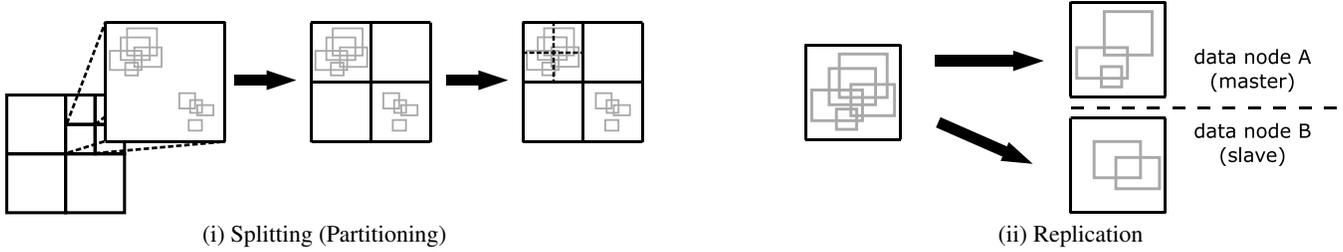
Figure 2: Query load balancing (gray query rectangles) via splitting and replication

initial set of nodes for data load-balancing. Later on, of course, additional resources (data and nodes) can be added to the community network, which is constructed 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 node during startup. Additional nodes subsequently joining the network receive their own local copy of the histogram from a neighboring node. HiSbase allocates data according to the precomputed histogram and uses the histogram as a routing index. By initially sending their data to any HiSbase node, data archives feed data into HiSbase.

In the following we describe the training phase, the partitioning schemes used, and the mapping process of HiSbase.

We use a training phase to construct the partitioning scheme which describes how the data is to be partitioned across the data nodes. Our weight functions of Section 3 will be integrated into this phase. By means of our weight functions, the partitioning process becomes also workload-aware, i. e., it uses information about data *and* queries and the queries' extent. First, we select representative training samples from the data sets. In case of workload-aware partitioning schemes, we additionally use a training workload, e. g., a query trace from an existing archive. Starting with a single partition, the partitioning process is continued until the cost of the histogram reaches a predefined limit. Until this limit is reached, the partition contributing the highest cost (i. e., *weight*) is selected and divided. Exemplary limits are the histogram size or a maximum size for all partitions. The HiSbase framework then provides several tools to support the research community in comparing various partitioning data structures and strategies. Eventually, a partitioning scheme is selected that shows the best load-balancing capabilities and other required characteristics. One such characteristic is a regular shape of partitions (squares or rectangles), as these are preferable in order to limit the complexity of query processing.

Quadtrees [8, 18] exhibit this characteristic and therefore we use quadtree-based partitioning schemes throughout this paper. For a d-dimensional data space, a quadtree is recursively defined to be either a leaf with a d-dimensional hypercube data region or an inner node with 2^d child trees. Each leaf is capable of storing a certain amount of objects and is split if the capacity is exceeded. Additionally, quadtrees adapt to the data density which is useful for communities with skewed data sets. They split densely populated areas more often, resulting in approximately even data distribution across all leaves. In previous work [22], we describe the evaluation of quadtree-based partitioning schemes using our training framework in more detail.

As most of the prevalent P2P infrastructures such as Chord [25] and Pastry [17] use a one-dimensional key space, we use a space filling curve to linearize the partitions of the partitioning scheme, e. g., Z-Order [15, 13] or Hilbert curve [10]. We assign *region ids* to the partitions with the Z-Order space filling curve, as it is easy to compute and corresponds naturally to a depth-first ordering of the quadtree leaves. Furthermore, the mapping process benefits from

the preservation of the spatial locality between neighboring partitions by the space filling curve. Once a HiSbase network is set up, every HiSbase node accepts queries and routes them to the relevant regions. These regions are identified via the histogram data structure. The key-based routing mechanism of the underlying P2P infrastructure identifies the responsible nodes and only those nodes cooperatively contribute to the query result and merge intermediate results if necessary. Previous work [20, 21] provides additional material on the query processing and data distribution within HiSbase.

2.2 Load Balancing Techniques

Query load balancing is a challenging task in distributed query processing. When dealing with popular ("hot") data, two strategies are generally applied in order to reduce the heat at the node predominantly responsible for the data:

Splitting (Partitioning) By further dividing the partition, parts of the query load can be moved to a different region. If that region[2] is covered by another node, load is balanced between these nodes. If hot areas are distributed among multiple nodes, good load balancing is achieved.

Replication Sometimes migration is not possible (e. g., one single data object is "hot") or desirable (e. g., the query processing would result in more communication overhead). In such cases, load balancing only is achieved by making multiple copies of the hot region at several locations. If all replicas participate equally during query processing, our design again achieves good load balancing.

Figure 2 shows the increased flexibility of load balancing techniques that apply both partitioning and replication. Initially, partitioning succeeds in dividing the two hot spot areas denoted by several gray query rectangles. The second partitioning step in Figure 2(i), however, would introduce additional communication overhead. In such situations, we can mitigate query hot spots better by replicating the original data area. Thus, multiple copies are available during query processing (Figure 2(ii)).

For deciding whether we gain more from replicating a region instead of splitting it, the following information is considered in our heuristics:

amount of data we still prefer to split those regions first that contain a considerable amount of data due to the importance of data load balancing,

number of queries regions with many queries should be handled before those regions whose workload is considerably low for query load balancing reasons,

---

[2]In the following, we use the terms *regions* and *data partitions* interchangeably.
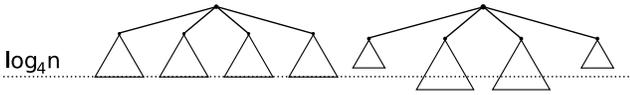
Figure 3: Impact of skew on the height of the leaves.
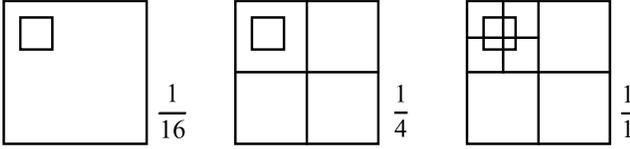
Figure 4: Impact of splitting a leaf on its ratio to a query area.

extent of regions and queries in order to balance query load, we rather replicate regions whose workload predominantly consists of queries having a large *extent*, i. e., they cover a large area compared to the area of the region itself.

Figure 3 depicts the basic idea, why it is important to incorporate the relationship between the extents of regions and queries in a replication-aware weighting scheme. If all data and queries were uniformly distributed, a quadtree-based partitioning scheme with n regions has a maximum height of $\log_{2^d} n$ in the general case and $\log_4 n$ for our running example. If either data or queries are skewed, the regions with less "load" are leaves with height $h < \log_4 n$ (with a larger area) and those in a hot spot area have $h > log_4 n$ (with a smaller area). A query area which covers about $\frac{1}{16}$ of a region, will cover $\frac{1}{4}$ on the next level and eventually will have the same size as a region after another split, as shown in Figure 4. Thus, the query is very likely to span multiple regions and to produce additional communication overhead.

In order to take the aspects just described into account, a weight function needs to adhere to the following heuristics. The weight function distinguishes regions with many data points from regions with only a few data points. Similarly, it regards workloads with many and few queries separately and moreover pays attention to whether queries cover a small or a big area of the region.

If a region contains *little data* and only a *few queries*, it should neither be split nor replicated as it does not contribute significantly to the overall load of the system.

In case of regions that contain *little data* which is interesting to *many queries*, we replicate those regions that have many big queries. These data partitions can be replicated at several nodes and then all replica are available for query processing. Partitions with many small queries are further split, as the resulting partitions possibly will fall into the category with few data points and few queries.

If a partition contains *many data points* but is only relevant to *few queries*, we prefer to split the partition. The performance of the small queries might increase as they run on smaller data sets which will even out the additional communication overhead for the big queries for the overall performance.

The crucial class for the weighting scheme is the forth category of regions which contain *much data as well as many queries*. This category requires a good choice between splitting a region if small queries mainly constitute the workload and replicating the region if the ratio of big queries is higher.

Table 1 summarizes the options to either split or replicate regions based on their associated number of data points and queries.

After having described the intuition about our criteria for query load balancing, we now present our approach to creating workload-aware data partitionings using weight functions.

Data Points	Few Queries		Many Queries	
	Small	Big	Small	Big
Few	–		SPLIT	REPLICATE
Many	SPLIT		SPLIT	REPLICATE

Table 1: Categorization of regions for the replication-aware weight function

3. REGION WEIGHT FUNCTIONS

In the following, we define a set of three weight functions for points, queries, and regions. This enables communities to create partitioning schemes for their data grids with a higher flexibility.

During the course of discussion, we will use several variables for points, queries, and regions which we will define in this paragraph. During the creation of our partitioning scheme, dubbed the *training phase*, we use a representative *training data set P* and a *training workload Q*. The variable p denotes a data point from P and q is a query from Q. Note that scientific data sets often comprise many dimensions. For simplicity and ease of presentation, we only consider the projection to the most predominant attributes from the query patterns (such as the two celestial coordinates in our astrophysics example) during the training phase. When distributing the data partitions, the complete data is distributed.

The hyperrectangle A_q describes the boundaries of query q within the data space of the partitioning scheme. Likewise, we define the area A_r covered by the region r. These areas are important building blocks for our weight functions.

We consider data points as *relevant* for query q if they reside within A_q. We further denote the queries for which p is relevant as Q_p and define P_q as the set of points which are relevant for query q.

$$Q_p = \{q \in Q \mid p \in A_q\} \qquad (1)$$
$$P_q = \{p \in P \mid p \in A_q\} \qquad (2)$$

The set P_r of data points within a region r and the set Q_r of queries which intersect region r is defined in a similar fashion.

$$P_r = \{p \in P \mid p \in A_r\} \qquad (3)$$
$$Q_r = \{q \in Q \mid A_q \cap A_r \neq \emptyset\} \qquad (4)$$

3.1 Point Weight

If a partitioning scheme is targeted at balancing data skew, the weight of a data point is relevant. Data skew can originate 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 general, we can define the weight $\text{w}(p)$ of a point p as a function of its default weight σ and the queries Q_p it is relevant for:

$$\text{w}(p) = \sigma + f(Q_p) \qquad (5)$$

When weighting data points, each point has a default weight σ, e. g., $\sigma = 1$. Now we also want to consider queries for which a point p is relevant. For example, if a point is relevant for 10 queries, it will have an additional weight of 10. Note that if we set default weight $\sigma = 0$, only data points which are relevant to *any* query are considered during the training phase.

Example 1: Cardinality Function. In the introductory example from above, we used the function $f : Q_p \mapsto |Q_p|$. It is a reasonable candidate function: easy to understand, strictly monotonically increasing, and easy to compute.

Points	Queries	Regions	Example	Load Balancing
1	–	$h(P_r)$	w_p	data
–	1	$i(Q_r)$	w_q	queries
$f(Q_p)$	–	$h(P_r)$	w_{Q_p}	data and queries
–	$g(P_q)$	$i(Q_r)$	w_{P_q}	data and queries
1	1	$h(P_r) \cdot i(Q_r)$	w_{pq}	data and queries

Table 2: Overview of region weight functions in Section 3.3

Example 2: Scaled Weight Function for Point Data.
The extent of the actual query hot spot(s) is unknown during the training phase. While we can locate the positions of query hot spots with our representative training workload Q, we can only approximate the extent of the area of the data space that will be subject to high query workload. Situations where only a limited number of queries is available during training make such estimates more difficult. As a consequence, we try to approximate the actual hot spots by increasing the query area A_q for all queries q by a *scaling factor* $\phi \geq 1$ in every dimension. We denote this area as $A_{q,\phi}$ in the following.[3] Also, we introduce a new parameter λ in the weight function of data points in order to scale the importance of Q_p in relation to the default weight σ. Thus, Equation 5 is extended to:

$$w_{\text{scaled},\lambda,\phi}(p) \quad = \quad \sigma + \lambda \cdot |\{q \in Q \mid p \in A_{q,\phi}\}| \qquad (6)$$

Tuning the parameters ϕ and λ for w_{scaled} can be quite difficult. Choosing the wrong scaling factor can yield counterproductive partitioning schemes, which was confirmed by our experimental results (Section 4.1).

3.2 Query Weight

The weight for queries is defined in a similar fashion. We assign a default weight γ to each query, which represents the default processing cost for any query. Depending on the set P_q, we add an additional query weight $g(P_q)$. In the following, we use $g(P_q) = |P_q|$.

$$w(q) \quad = \quad \gamma + g(P_q) \qquad (7)$$

3.3 Combining Data and Query Weights

The weight functions for data points (Equations 5 and 6) and queries (Equation 7) constitute the basic building blocks for defining the weight of a region. Based on the weight of the individual partitions, we always split the partition with the highest weight next. The weight of a region r depends on the data points P_r it contains and the queries Q_r which intersect with its area.

$$w(r) \quad = \quad h(P_r) \otimes i(Q_r), \quad \text{where} \otimes \in \{+, \cdot\} \qquad (8)$$

For the rest of the paper, we will only discuss the multiplication of both weights ($\cdot$ is used for $\otimes$).

Composing the weight functions for points and queries to the weight of a region will result in partitioning schemes that are optimized for various load balancing goals. Depending on the combination, a partitioning scheme can achieve load balancing for data, for queries, or for both. In Table 2, we summarize five general patterns of how the weight functions of points and queries are combined for the weight of a region. We associate the examples discussed in the following with their corresponding pattern and state their load balancing capabilities.

Weight functions for the first two approaches consider either only the data points or only the queries for computing the weight of a region. The first (w_p) just counts the data points within a region, the second (w_q) only considers the number of queries intersecting with each partition.

---
[3]Equation 1 and 2 are still valid, as $A_{q,1} = A_q$.

All remaining three alternatives consider both data and queries for weighting the regions. In the third and fourth approach (w_{Q_p} and w_{P_q}), the weight of a data region only depends on one of the building blocks—either points or queries—however the weight of the chosen building block is influenced by the other, e. g., we weight each data point according to its relevance for queries. The last weight function w_{pq} computes the *heat* of a region by multiplying the number of objects within a region and the number of queries intersecting the region. This weight function implicitly scales all queries until they cover the entire area of the region(s) they intersect and assigns more weight to regions that contain lots of data and receive many queries. Thus, this weight function provides a notion of the overall load based both on data and on queries. If the region contains no data, its weight is 0 and therefore it is unlikely to be split. If regions receive no queries, we prefer to split those regions that contain more data. We therefore use $(|Q_r|+1)$ in case of queries. Thus, if a region receives no queries, it still has the same weight as when using w_p.

After we have introduced the five weight functions intuitively, the Equations 9–13 give their formal definition.

$$w_p(r) \quad = \quad |P_r| \qquad (9)$$
$$w_q(r) \quad = \quad |Q_r| \qquad (10)$$
$$w_{Q_p}(r) \quad = \quad \sum_{p \in P_r} w(p) \qquad (11)$$
$$w_{P_q}(r) \quad = \quad \sum_{q \in Q_r} w(q) \qquad (12)$$
$$w_{pq}(r) \quad = \quad |P_r| \cdot (|Q_r|+1) \qquad (13)$$

The relevance of data points for a particular query q can also be described using the indicator function $\mathbf{1}_{A_q} : P \rightarrow \{0,1\}$:

$$\mathbf{1}_{A_q}(p) \quad = \quad \begin{cases} 0 & \text{if } p \notin A_q, \\ 1 & \text{if } p \in A_q. \end{cases} \qquad (14)$$

Equation 15 shows that w_{Q_p} and w_{P_q} define the same weight function if we set the default weights to $\sigma = \gamma = 0$ in Equations 5 and 7 and $f(Q_p) = |Q_p|$ and $g(P_q) = |P_q|$, respectively. Intuitively, counting points weighted by the queries they are relevant for is equivalent to counting queries weighted by the points that are relevant for them.

$$w_{Q_p}(r) = \sum_{p \in P_r} w(p) = \sum_{p \in P_r} \sum_{q \in Q_r} \mathbf{1}_{A_q}(p) = \sum_{q \in Q_r} w(q) = w_{P_q}(r) \qquad (15)$$

For the sake of simplicity and the discussion in this paper we use the weight-factors σ, γ, ϕ, and λ as constants. Other scenarios, where σ, for example, is a function that returns the average size of a data point p depending on the catalog it originates from, show promising results but are beyond the scope of this paper.

3.4 Adding Query Extents

The pure heat-based weight function w_{pq} captivates with its simplicity. However, if there is a small hot-spot area, heat-based partitioning may split that area multiple times as it tries to reduce the query load imbalance. This can lead to communication-thrashing, i. e., too much communication between nodes covering neighboring partitions is necessary to retrieve the complete result.

For example in our two-dimensional quadtree-based partitioning schemes for astrophysics data, a query area A_q containing the centroid of the region area A_r, would be split into four subqueries. In the worst case, four different nodes are responsible for these regions. This would result in four-times overhead, as intermediate results need to be transmitted and the query uses CPU resources on

four nodes. Under such circumstances, we prefer to keep the region as a whole and rather replicate it with our master-slave approach as described in Section 3.6.

Our replication-aware weight function w_{A_q} incorporates the extents of queries and regions by classifying the queries according to the fraction of the area A_q of query q and the area A_r of region r. Thus, weight function w_{A_q} realizes the behavior from Table 1 in Section 2.2. For $0 < \alpha \le \beta < 1$, the sets of small (big) queries Q_r^{small} (Q_r^{big}) are defined in Equations 16 and 17, respectively.

$$Q_r^{small} = \{q \in Q_r \quad | \quad A_q < \alpha \cdot A_r\} \qquad (16)$$

$$Q_r^{big} = \{q \in Q_r \quad | \quad A_q > \beta \cdot A_r\} \qquad (17)$$

Based on the classification for Q_r, we define the *splitting gain* for region r, $\text{gain}_s(r)$, as the number of small queries in r (Equation 18). Analogously, we define $\text{gain}_r(r)$, the *replication gain* for region r with the number of big queries intersecting r (Equation 19). For the same reason as in w_{pq}, we add one to both cardinalities to deal with regions that receive no queries or whose query sets Q^{small} or Q^{big} are empty.

$$\text{gain}_s(r) = |Q_r^{small}| + 1 \qquad (18)$$

$$\text{gain}_r(r) = |Q_r^{big}| + 1 \qquad (19)$$

The replication-aware cost function w_{A_q} compares $\text{gain}_s(r)$ and $\text{gain}_r(r)$ to determine whether a region should be split or not. As long as splitting a region is considered beneficial, only the value of $\text{gain}_s(r)$ is used. As soon as the "big" queries outnumber the "small" queries, we reduce the weight of a region considerably, by multiplying the size of $|P_r|$ with the fraction of the small queries and big queries. In some application domains it might be desirable to additionally specify the preference τ for either splitting or replication. This is formalized by Equation 20.

$$w_{A_q, \alpha, \beta, \tau}(r) = \begin{cases} |P_r| \cdot \dfrac{\text{gain}_s(r)}{\text{gain}_r(r)} \,, \text{if } \tau \cdot \text{gain}_s(r) < \text{gain}_r(r)\,, \\ |P_r| \cdot \text{gain}_s(r) \,\,, \text{otherwise}\,. \end{cases}$$
$$(20)$$

The values for α and β strongly depend on workload characteristics of the application domain, as we realized during our evaluation. At first thought, values like $\alpha = \frac{1}{4}$, $\beta = \frac{3}{4}$ or $\alpha = \frac{1}{10}$, $\beta = \frac{9}{10}$ seem reasonable. Remarkably, those combinations have a fairly large "blind angle", i.e., they ignore queries which have area extents between both thresholds. Especially, the decision in favor of splitting a region is sensitive to this gap. After having made the decision to split a region, those "hidden" queries will probably intersect multiple regions causing high overhead. Thus, we suggest to use the same values for α and β.

3.5 Cost Analysis

The complexity of the weight functions described in the previous sections strongly depends on the choice of functions f, g, h, and i as well as on the data structures used for storing the training set P and the training workload Q. A naive approach iterating over all queries in the workload in order to acquire the weight for all data points would lead to an overall complexity of $O(|P| \cdot |Q|)$.

The complexity and overhead of maintaining the data points and queries as well as the complexity of performing the weighting can be reduced via appropriate data structures. We use hierarchical, tree-like data structures, e.g., quadtrees [8, 18] for creating our partitioning schemes and for storing data points and queries. The leaves of the quadtree correspond to the individual regions. Trees offer a good pruning capability, i.e., one can decide quickly whether a point or a query is relevant. We store both, queries and points, in

the same index structure. This allows us to reduce both, the number of data points and queries which need to be considered for computing the weight of a region.

We decided to redundantly store queries which span multiple regions at the leaf-level, i.e., at every region, instead of storing them at inner nodes of the tree, e.g., the nodes that fully contain the bounding box of the hyperrectangle. This further simplifies computing Q_p and Q_r because we do not inspect query sets at inner nodes on the path from the root to the leaf-level.[4] For computing weight functions such as w_{Q_p} or w_{P_q}, containment queries are necessary to decide which query areas contain a data point. These queries can become quite complex, especially if large query workloads are used. Computing the heat of a region by using the weight function $w_{pq}(r)$, however, is compellingly simple. We only need to multiply the sizes of the two sets P_r and Q_r in order to compute the weight of a region and avoid the cost for comparing each data point of P_r with each query in Q_r. These adaptions further reduce the complexity to compute the weight of a region to $O(|P| + |Q|)$. Only when a region is split, we need to reorganize the sets P_r and Q_r. To summarize, by using a hierarchical data structure for creating the data partitions, we integrate most of the weighting cost into the tree maintenance and only need two lookups in order to compute w_{pq}. For the replication-aware weight function w_{A_q}, we also use the maintenance methods of the tree. When splitting a region, its queries are immediately classified for the newly created leaves into the corresponding sets Q^{small} and Q^{big} by at most two comparisons. Thus, only two additional counters for storing the values of $\text{gain}_s(r)$ and $\text{gain}_r(r)$ are necessary.

3.6 Replication during Runtime

In this section we have provided several weight functions for the partitioning process of data which combine weight functions for data points and queries to weight functions for regions. Starting from pure data-based weight functions, we proposed several weight functions that additionally take the workload into account.

During runtime, nodes within the HiSbase network need to decide which of their regions need replication. During the training process, we can identify those replication candidates. The regions r whose $\text{gain}_r(r)$ is higher than $\text{gain}_s(r)$, are perfect candidates for replication and can be annotated as such. Nodes then can explicitly prefer those regions for replication.

In order to support overloaded nodes during runtime with additional resources, HiSbase builds a *master-slave* hierarchy. Lightly loaded nodes, i.e., nodes having less data or less queries to process, offer some of their capacity as slaves to overloaded peers. Master nodes then send some of their *hot* data subsets to such slave-nodes. During query processing all replica are available for query load-balancing purposes. Please note that the master-slave relationship is defined with regard to a single region. Thus, a node can be master for one its own regions as well as slave for other regions in parallel.

Due to space limitations, we cannot discuss design alternatives such as stopping the partitioning process if every leaf should be replicated, or the complete process of selecting replica.

4. EVALUATION

In the following, we will present our analysis of the various workload-aware partitioning schemes introduced in the previous section. Two aspects were important for our evaluation settings: the analytical properties of the partitioning schemes and their impact on the overall throughput in a real deployment. In our opinion,

---

[4]This is basically the same trade-off as between MX-CIF and extended MX-CIF quadtrees [19].

Parameter	Value	Description
P	P_{obs}, P_{mil}	Data Sets used for training sample extraction.
Q	Q_{obs}, Q_{mil}	Workloads used for workload-aware training.
s	0.1%, 1%, 10%	Size of the training set (of P).
n	$4^2, 4^3, 4^4, 4^5,$ $4^6, 4^7, 4^8, 4^9$	Size of the partitioning schemes.

Table 3: General parameters for the evaluation setup.

Weight Function	Parameter and Value
$w_{Q_p}, w_{\text{scaled}, \lambda, \phi}$	$\lambda = 0.01, \phi = 10, 20, 40, 80$ (for P_{obs}) $\lambda = 1, \phi = 10, 50, 100, 200, 400$ (for P_{mil}) $\sigma = 1$ (both data sets)
w_{A_q}	$\alpha, \beta \in \{ \frac{1}{4096} (\approx 0.0002), \frac{1}{256} (\approx 0.004),$ $\frac{1}{16} (= 0.0625), \frac{1}{4} (= 0.25) \}, \tau = 1$

Table 4: Weight function specific parameters.

complementing results obtained from statistical analysis or simulations with experiments of a actually deployed system is fundamental for assessing distributed architectures.

4.1 Partitioning Scheme Properties

In order to evaluate the analytical properties of the partitioning schemes created with the *data-based* (w_p), *heat-based* (w_{pq}), and *extent-based* (w_{A_q}) weight functions outlined in this paper, we conducted several experiments. We give a detailed description of the parameters, data sets, and query workloads used during our evaluation. We present and discuss results with respect to our goal of achieving a workload-aware data partitioning.

For the evaluation, we implemented a Java-based prototype which created partitioning schemes according to our weight functions, based on a training data sample P and a representative query workload Q. First, we evaluated the performance of our technique on data samples from three astrophysical catalogs using a query trace from the SDSS catalog. Afterwards, we assessed the effect of our workload-aware training on a data sample from astrophysical simulations using a synthetic workload. This gave valuable additional information, as the simulation data is quite uniformly distributed and so the impact of some parameters was visible more clearly.

We constructed quadtree-based partitioning schemes using the standard splitting strategy as well as the median-based heuristics from [22]. Here, we only discuss results with standard quadtrees. For each approach, we varied the number of partitions to be all powers of four between 16 (4^2) and 262 144 (4^9) as these can be generated exactly by quadtrees.

For both data sets used during the evaluation, we drew several training samples of different sizes (0.1%, 1%, and 10%) to benchmark the quality of results produced from small data sets. We extracted the random samples with functionality provided by relational database systems. We report on the results obtained from quadtree-based partitioning schemes using the standard splitting strategy based on the 0.1% and 1% samples, each containing about 150 000 and 1 500 000 data points, respectively. Table 3 summarizes the general parameters used during the evaluation.

From the weight functions defined in Section 3, we used the uniform point (*data-based*) weight function w_p as a baseline for our comparisons, and the *query-based* point weight function w_{Q_p} with various values for the default weight σ, the importance λ of Q_p, and the scaling factor ϕ. Furthermore, we used the *heat-based* weight function w_{pq} and the query *extent-based* weight function

(i) Data set P_{obs}.

(ii) Query set Q_{obs}.
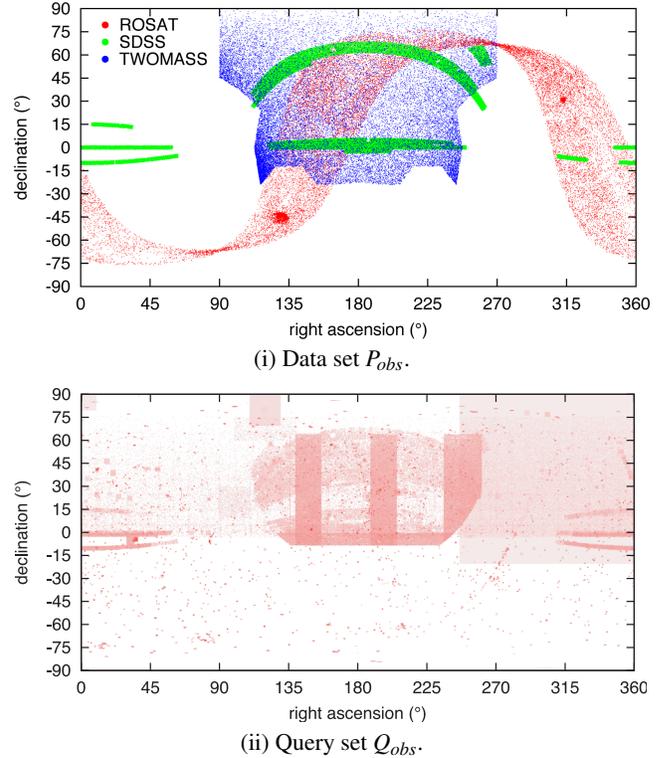
Figure 5: The observational data and workload.

w_{A_q} with $\tau = 1$ and with varying α and β thresholds. Table 4 gives a summary of the used weight function parameters.

4.1.1 Results from the Observational Data Set

The first data set P_{obs} (see Figure 5(i)) are 137 million objects drawn from subsets of the ROSAT (25 million objects), SDSS (84 million objects), and TWOMASS (28 million objects) catalogs. We clearly see that the data sets exhibit a high data skew.

For the observational data set P_{obs}, the corresponding query set Q_{obs} was constructed from real queries issued to the web interface[5] of the SDSS catalog in August 2006. Because the queries used radial searches, the query areas were mapped to square areas with the same midpoint and an edge length corresponding to the diameter of the circular search area. Queries with the default search parameters for the web interface were removed from the query set, as this particular query alone made up 12% of the entire query log. The remaining 1 100 000 queries were used during our evaluation. In Figure 5(ii), we clearly see that the workload is non-uniform and exhibits many query hot spots.

Parameters for Extent-based Partitioning Schemes. The values for the parameters of w_{A_q} are motivated by the observation in Section 2.2 that the fraction between a query and a region increases four-fold with every split. Analyzing the workload Q_{obs}, we found that the median of the used search radii in Q_{obs} is at 0.2 arc minutes and 75% of the queries have a radius smaller than 0.4 arc minutes. This is extremely small, as queries with an 0.2 arc minutes radius cover only $\frac{1}{10^9}$ of the whole sky. If the queries are at such a small scale, we need to adapt β accordingly. For example, our smallest value of β, 0.0002, corresponds to $\frac{1}{4^6}$ and classifies those queries as big that will increase the network load with high probability, when their region is split an additional six times.

---

[5] http://cas.sdss.org/astrodr6/en/tools/search/
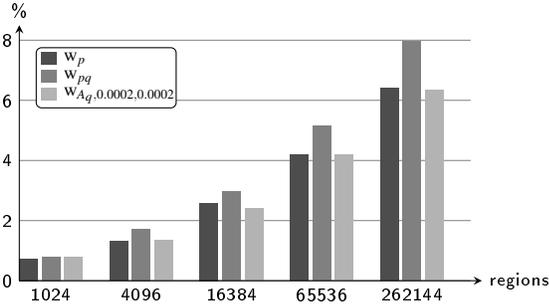
Figure 6: Percentage of queries in Q_{obs} that are answered with consulting more than one partition

(i) w_p (ii) w_{pq}
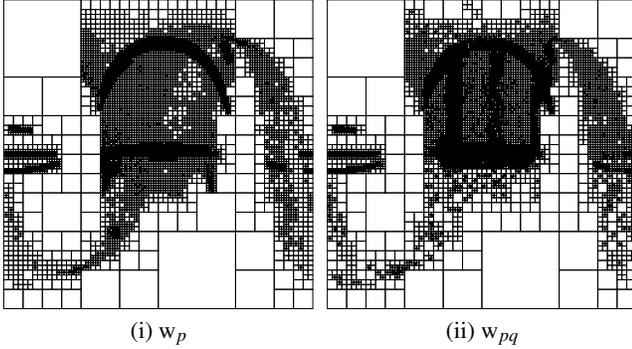
Figure 7: Quadtree-based partitioning schemes of P_{obs} with 16 384 regions.

Spatial Locality and Small Partitioning Schemes. During the course of our evaluation, we made several observations. First of all, the example workload Q_{obs} from the SDSS archive, shows the expected high spatial locality, as can be seen from Figure 6. With all tested partitioning schemes, for less than 10% of the queries multiple partitions contain relevant information. Communities with such workload characteristics greatly benefit from the high degree of parallelism within the system. For up to 1 024 histogram regions, the number of one-region queries is identical and the partitioning schemes only have minor differences. Therefore, partitioning data—even with a partitioning scheme which is only based on the data—can migrate load to different partitions which work in parallel. For uniform query loads and communities at the very beginning of building their grid infrastructure, data-based partitioning is completely sufficient.

Adaption to Query Workloads. When we compare the partitioning schemes of P_{obs} in Figure 7 with the original data and query set from Figure 5, we can observe the similarity between the data distribution and the data-based weight function w_p and also the heat-based partitioning w_{pq} and the query load Q_{obs}. Thus, we can see that our weight functions are able to create workload-aware data partitionings.

Load Balancing Capabilities. Figure 8 shows that the heat-based weight function w_{pq} distributes the overall load significantly better across multiple nodes for quadtree-based partitioning schemes on the observational data set using the SDSS workload than the weight function w_p which focuses on data load only. We quantitatively evaluated the total query load by calculating the sum of the individual query loads for each region, and the uniformness of the load distribution using the Gini coefficient as in [16]. The Gini coefficient is defined as the area between the Lorenz curve for the distribution and the diagonal.
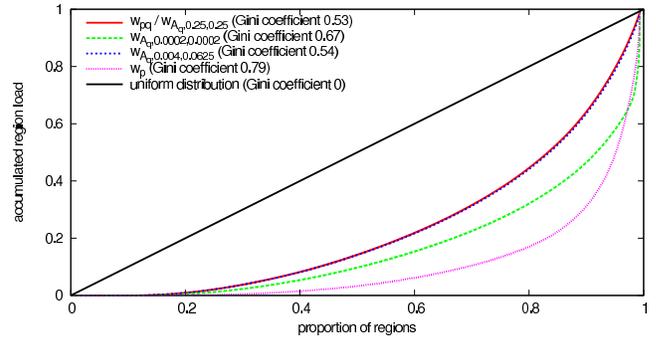
Figure 8: Lorenz curves for P_{obs} for partitioning schemes with 4 096 regions and weight functions w_p and w_{pq}
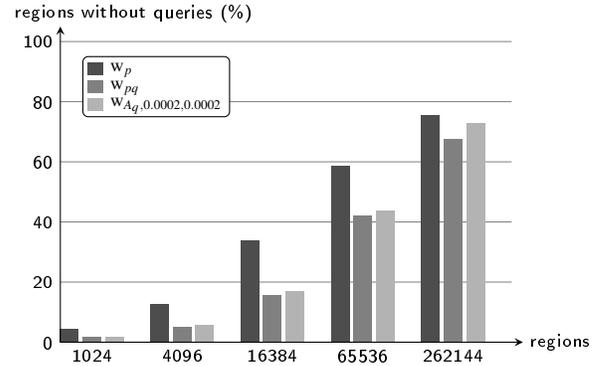
Figure 9: Comparison of the percentage of regions that receive no queries from Q_{obs}.

While w_p only achieves a Gini coefficient of 0.79, w_{pq} has a coefficient of 0.53, which is considered a fair load distribution in distributed systems [16]. The partitioning scheme of $\text{w}_{A_q,0.25,0.25}$ achieves the same load distribution and $\text{w}_{A_q,0.004,0.0625}$ differs only marginally from the heat-based partitioning. With 0.67, the Gini coefficient of $\text{w}_{A_q,0.0002,0.0002}$ lies between w_{pq} and w_p.

In the w_p approach for data load balancing, 20% of the regions receive 83% of the overall system load. For our workload-aware technique w_{pq}, these 20% handle less than 60% of the overall load. When using the extent-based $\text{w}_{A_q,0.0002,0.0002}$, 20% of the regions process 68% of the load, as the weight function recognizes some candidate regions for replication. Thus, the query load is less balanced as in the w_{pq} partitioning scheme. We will see in the following that with regards to other characteristics the extent-based approach is preferable to the heat-based technique.

Regions without Queries. When analyzing the weight functions with regard to regions that do not take part in query processing, the less such regions, the better the weight function distributes the load to several partitions. w_{pq} always achieves the best result. Up to 65 536 regions, both w_{pq} and w_{A_q} are at a comparable level and between 15% and 50% better than pure data load balancing.

In Figure 9, w_{pq} always has the lowest number of regions without queries. The following analysis, however, shows that w_{pq} is too eager in splitting regions further and further and therefore introduces significant communication overhead.

Reduced Traffic by Workload-Awareness. In order to investigate the communication overhead, we compared our partitioning schemes to a scheme where every query could be answered by a single region. All regions that need to be contacted additionally,
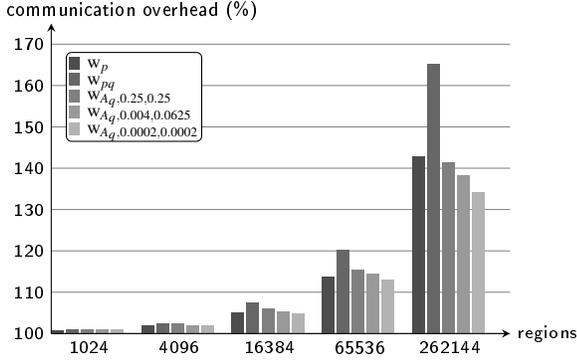
Figure 10: Communication overhead for partitioning schemes of P_{obs} in comparison to a centralized setting.

increase the communication overhead of the specific weight function. Based on our observation that typical workloads exhibit a high degree of spatial locality, we prefer queries that intersect a single region which is guaranteed to be mapped to one peer. Formally, we compute the communication overhead as

$$\frac{\sum_{q \in Q} |\{r \in R \mid A_q \cap A_r \neq \emptyset\}|}{|Q|} \quad (21)$$

Figure 10 shows the relative traffic overhead for w_p, w_{pq}, and some variations of w_{A_q}. We clearly see that w_{A_q} produces lower traffic than w_{pq}. Moreover, with a reasonable choice of α and β, w_{A_q} produces not more traffic than w_p.

4.1.2 Results from the Millennium Data Set

Finally, we shortly discuss how the different weight functions performed on the *Millennium* data set P_{mil} with the query workload Q_{mil}. P_{mil} (Figure 11(i)) comprises about 160 million objects from the Millennium simulation [24] which are distributed on the area $[-45°, 45°] \times [-45°, 45°]$.

The query areas for Q_{mil} were artificially generated with their midpoints (p_x, p_y) following a two-dimensional Gaussian distribution with mean $(0,0)$ and variance chosen in such a way that 90% of the midpoints fall into the square area in the center taking 10% of the space. The actual query areas were then constructed around the midpoints from $(p_x - r, p_y - r)$ to $(p_x + r, p_y + r)$ with the query "radius" r chosen randomly from $\{0.025, 0.1, 0.2, 0.25, 0.5\}$ arc minutes, which correspond to the 5 most frequent query radii from the query workload Q_{obs}, introduced above. In this way, 11 000 queries were generated for training and testing the resulting partitioning schemes.

In Figure 11, some of the partitioning schemes of P_{mil} with 1 024 partitions are shown. Each region is colored with its heat-based weight (the w_{pq}-value) on a scale from cold (white) to hot (red), which is normalized over the compared partitioning schemes. The w_p-partitioning scheme, in Figure 11(ii), is a completely balanced quadtree with 1 024 same-sized partitions as the data distribution is almost uniform. The hot spot in the center of the data space is also clearly visible as the query load is not considered. Figure 11(iii) shows how w_{Q_p} splits the hot regions first and the partitioning scheme adapts to the hot spot. As the number of regions is fixed, regions at the border of the data space are not split further and now contain more data. From Figure 11(iv) it becomes obvious why: regions at the border of the hot spot contain 16-times the data than in the w_p partitioning but also receive many queries and thus their load will be too high. Finally, we see in Figure 11(v) that the heat-based weighting scheme w_{pq} approximates the extent of the hot spot very good but does not lose sight of data load balanc-

ing. Actually, the extent-based weight function w_{A_q} produced the same partitioning scheme. Note the four different sizes of regions in our workload-aware partitionings: regions are 16-times smaller (in the very center of the hot spot), 4-times smaller, unchanged, and 4-times larger (the cold regions at the border of the data space) than the regions of the w_p partitioning scheme.

4.2 Throughput Evaluation

The previous analysis of the histograms was based on the training data and both the training and testing query sets. The following throughput measurements are conducted on a distributed or simulated HiSbase instances. We intentionally do not use the master-slave approach during runtime as described in Section 3.6 for replication in order to emphasize the throughput variations purely based on the choice of the weight function.

As in previous evaluations [20], we measure throughput for varying *multi-programming levels (MPLs)*, i. e., a varying number of parallel queries in the system, to evaluate to what extent the use of our workload-aware histogram data structures improve the system throughput. 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 *submitted* its q-th query and has *received* 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 during the *saturation phase* I_{sat}. I_{sat} is the time span when every peer has issued MPL=m parallel queries, i. e., 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:

$$I_{sat} = \left[\max_{1 \leq p \leq k} (s_{p,m}), \min_{1 \leq p \leq k} (s_{p,l}) \right] \quad (22)$$

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

$$T = \frac{|\{(p,q) \mid r_{p,q} \in I_{sat}, 1 \leq p \leq k, 1 \leq q \leq l\}|}{I_{sat}} \quad (23)$$

During the throughput evaluation, we used partitioning schemes created with the *data-based* (w_p), *heat-based* w_{pq}, and *extent-based* $w_{A_q, 0.0002, 0.0002}$ weight functions, respectively. The size of the histogram—4 096 for real and 262 144 for simulated networks—is chosen to be large enough to ensure that all peers are responsible for data partitions. The results shown are the averages built over three evaluation runs in both (the real and simulated) cases.

Each single HiSbase node was configured to allow ten parallel queries on its local database, as suggested in [20]. The following evaluation shows that especially higher multi-programming levels for the HiSbase nodes increase the overall throughput; we report on MPLs selected from $\{10, 50, 100, 300, 500\}$.

4.2.1 Results from the Observational Workload

For our throughput experiments, we chose a subset Q_{eval} of about 22 000 queries from the query trace Q_{obs}. We selected queries which were among the top 20% fasted queries when executed on a single DB2 database containing all data. Those queries—with running times between 1 and 4 ms—would be penalized most severely from being submitted to a queuing system. Each node randomly selected 5 000 queries from Q_{eval} for its query batch.

Our 16 computer lab nodes are from consumer-class Linux PCs equipped with 1.6 GHz processors, 512 MB RAM and running DB2
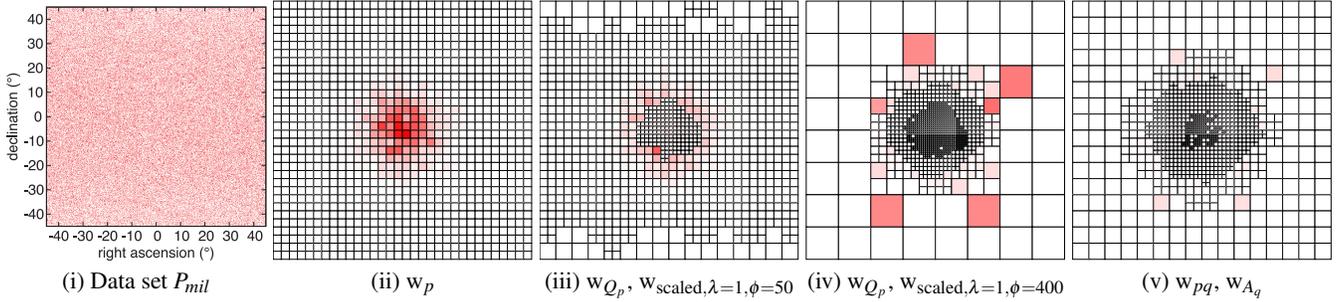
Figure 11: Partitioning with 1 024 regions for P_{mil}.

(i) Data set P_{mil} (ii) w_p (iii) w_{Q_p}, $w_{scaled,\lambda=1,\phi=50}$ (iv) w_{Q_p}, $w_{scaled,\lambda=1,\phi=400}$ (v) w_{pq}, w_{A_q}
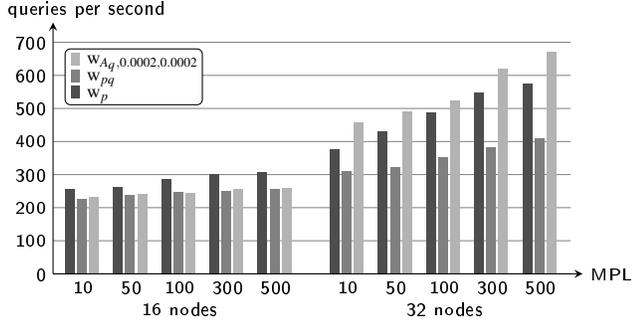
Figure 12: Throughput of deployments with 16 and 32 nodes

V9.1.4. For the measurements with 32 nodes, we additionally used 16 nodes from our local AstroGrid-D resources having between 1 and 4 GB main memory and 2.8 GHz Intel Xeon CPUs. We used the H2 database[6] for performing the local query processing. After assigning each node a random identifier, we distributed the data according to the partitioning scheme.

Figure 12 shows the throughput achieved by a HiSbase network using a partitioning scheme with 4 096 partitions which are distributed among 16 and 32 nodes, respectively. For 16 peers, we see that the w_p partitioning scheme achieves surprisingly the highest throughput, while for the setup with 32 nodes, the extent-based technique w_{A_q} outperforms both w_p and w_{pq}. Due to several query hot spots in Q_{eval}, only a fraction of the additional nodes can significantly participate during query processing. While w_{A_q} increases the throughput in a near-linear fashion, the gain of w_p and w_{pq} is only sub-linear.

We have successfully demonstrated HiSbase [21] on PlanetLab and we see great value of PlanetLab for evaluating the algorithmic properties (like messaging overhead) of distributed architectures in volatile environments. However, our initial throughput measurements on 100 PlanetLab nodes showed that the PlanetLab framework is not suitable for performing throughput evaluations of data-intensive grid applications. Issues like bandwidth-limited links and limited main memory access (below 160 MB) do not reflect the anticipated infrastructure for community grids. We therefore abandoned using PlanetLab and evaluated the throughput trend of larger deployments with FreePastry's discrete-event simulator instead.

During our simulations, we evaluated three different partitioning schemes (w_p, w_{pq}, w_{A_q}) with 262 144 partitions using the MPLs from above on HiSbase networks with 100, 300, and 1 000 nodes, respectively.[7] Database accesses were simulated by returning the result set after a delay which corresponded to the time obtained

---

[6] http://h2database.com

[7] We did not measure the throughput of 1 000 nodes with MPL=500 as current e-science scenarios not yet require such high parallelism.

during the selection process for the Q_{eval} queries. The simulation engine does not model running time improvements due to caching effects or smaller databases at each nodes. Likewise, we assumed that parallel running queries do not interfere and the measured running time is also valid for ten parallel queries. Runs with the same setups like in the distributed scenarios with 16 and 32 nodes verified that the results of the simulator are realistic. The ratios between the simulated measurements and the real results were at a reasonable level between 1.07 and 2.25.

The results from the simulations showed a similar trend as in the real deployments. With all tested partitioning strategies, the extreme query hot spots diminished the load balancing effect of adding new nodes. Figure 13 depicts that workload-aware partitioning schemes perform better for high MPL levels (MPL=300 and MPL=500) and large networks with 300 and 1 000 nodes than the pure data-load approach.

4.2.2 Results from the Region-Uniform Workload

Analyzing query workload Q_{eval} for the partitioning schemes with 262 144 regions revealed that 3% of the regions receive any query (as opposed to the 27% in Figure 9 based on the complete query trace Q_{obs}). In order to evaluate the scalability under a uniform query workload, we generated workloads where 92% of the regions intersect with queries. The 600 000 generated queries were uniformly distributed among the regions and the query areas were constructed in a similar fashion as for the millenium data set (see Section 4.1.2). According to a uniform distribution, we picked a region identifier and a center point for the query area within that region. As all histograms achieved similiar results, Figure 14 depicts only how the extent-based histogram balances the query load uniformly in a near-linear and even super-linear throughput (stressed by the trend line for MPL=300), especially when comparing the 300-nodes and 1000-nodes networks.

4.3 Summary

In summary, the evaluation corroborates that query hot spots are an important issue in scientific data management. The analytical evaluation showed that our partitioning techniques adapt to the query workload and that our extent-based technique (w_{A_q}) has the same message overhead as a data-based distribution while also offering query load balancing. The throughput experiments showed that the extent-based partitioning is best in taking advantage of additional nodes, especially for highly parallel workloads. For all partitioning schemes, throughput significantly improves when all nodes participate during query processing. In the presence of query skew, however, this can only achieved by replication during runtime. Therefore, employing load balancing during runtime with techniques such as a master-slave approach and replicating "hot" data based on monitoring statistics are the next important steps towards fully workload-aware community-driven data grids.
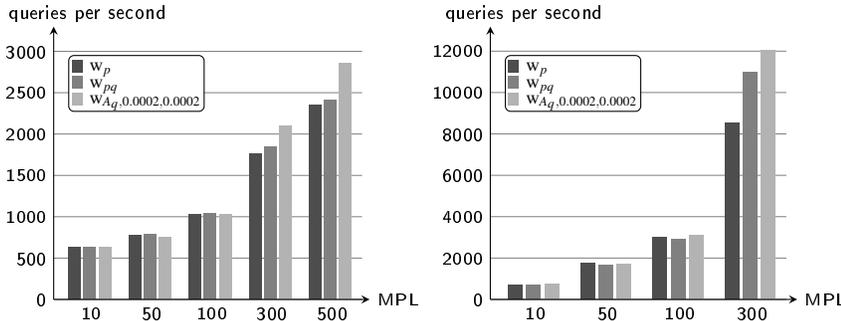
Figure 13: Throughput on the observational query workload with simulated networks with 300 (left) and 1 000 (right) nodes

Figure 14: Throughput for the region-uniform query workload

5. RELATED WORK

Federated data grids and many of the various design options for data grids are summarized by a recent survey [27]. They are discussed with regards to organizational structure, data transport, data replication, and scheduling by defining a comprehensive taxonomy for data grids and mapping existing data grids to that topology. Throughout this paper, we enhanced the data-driven partitioning techniques for *community-driven data grids* [20] to a cost-based model and described partitioning techniques, which allow data load balancing and are aware of query hot spots. Many data sets of publicly funded projects become (and remain) available to the public after a period of grace of one year and can be accessed by the community. Community-driven data grids have a main focus on this multitude of data repositories, where institutions are no longer interested in keeping them "private" and do not enforce *data autonomy* but *cooperatively* share data with colleagues. Previous work [22] has described a framework for comparing different partitioning techniques and discussed several measures for evaluating the partitioning quality with regard to data load balancing. It also discusses two quadtree-based partitioning schemes, without and with a median-based heuristic, which have been extended to support our weight functions and have been used in our evaluation experiments. Our work is reminiscent of the achievements in the context of parallel databases (e. g., [1, 5]) which addressed many similar issues yet in a far more stable and homogeneous setting.

Several other interesting proposals for accessing and correlating scientific data sets have been proposed in the literature. GIME [29], for example, takes a different approach to geotechnical information management in federated data grids. While preserving the data autonomy of the participating institutions, the system uses a replicated index (based on quadtrees or R-trees) for managing the bounding boxes of participating archives. Thus, it reduces the number of messages by submitting the query only to those archives whose minimum bounding box actually intersects the query area. The system does not address data load balancing or query load balancing issues. Load imbalance can arise for data archives covering a large area. These archives have to process more queries than small archives and several data sets cannot be combined directly on-site. Having a similar goal, i. e., reducing the network traffic generated by correlating distributed data sources, the authors of [28] describe various techniques for finding optimal join orders within scientific data federations.

The fact that workloads on astrophysical data sets are mostly spatial queries (selections on the celestial coordinates or corresponding stored procedures) and that these workloads exhibit a high query skew is supported by an extensive analysis of the traffic for the SDSS SkyServer [23] and an experience report on migrating the SkyServer on MonetDB [11]. The query load used on our skewed
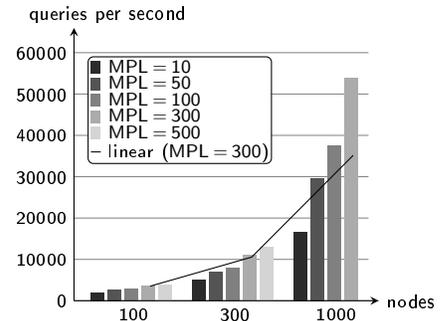
data set is the same as used by the latter.

In the following, we discuss related work [9, 4, 16] investigating techniques of load balancing for P2P networks. The major difference to these techniques compared to our solution is that load balancing in P2P networks is designed for networks with highly dynamic data and mostly deals with either skewed data distributions or skewed query load, but not both. The flexibility needed in such a fast changing environment comes at the price of dealing with each data object individually, which can result in prohibitive costs for disseminating vast amounts of data to multiple nodes. The authors of [9] 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. HotRod [16] addresses query hot spots on one-dimensional data by replicating popular data ranges on additional rings but does not deal with skewed data distributions. P-Ring [4] addresses data skew in an orthogonal manner in comparison to the partitioning-based approach, but does not consider query hot spots. While our partitioning schemes adapt the regions to data skew and query skew distributing 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. In P-Ring it is required that there are less data partitions than peers. If P-Ring would be extended in order to support these large data sets by supporting our notion of regions, that requirement would lead to larger partitions which are not as easily distributed as the regions created with our workload-aware partitioning schemes. Furthermore, P-Ring does not perform data replication.

SD-Rtree [6] is a *Scalable Distributed Data Structure (SDDS)* which targets large data sets of spatial objects. An SDDS has the following characteristics: 1) It has no central data index, 2) servers are dynamically added to the system when needed, and 3) the clients access the SDSS which is potentially outdated. SD-Rtrees perform data load-balancing by data partitioning and reorganization similar to AVL trees. The histogram in HiSbase differs from the SD-Rtree index in that it is used also for query load balancing and it uses the multi-dimensional index structure to determine candidates regions for replication.

Related work in sensor networks (e. g., [2, 3]) illuminates aspects of data distribution and load balancing from a different perspective where data is created within the network and the predominant goal is to increase quality of data and reduce the power consumption in order to increase the lifetime of a sensor network. As these solutions also deal with individual data objects, it is currently unclear whether they can be directly applied to petabyte-scale data sets of e-science communities. However, it is an interesting question for future investigations.

6. ONGOING WORK AND CONCLUSION

Supporting the efforts for building global-scale data management solutions within many e-science communities such as biology or astrophysics is a challenging task. In this paper, we have described several weight functions to create cost-based partitioning schemes for community-driven data grids that address data skew, query hot spots—each on its own or in combination—and finally, a weight function that only splits data regions if the gain of doing so is higher than the gain of replicating that region.

We evaluated our weight functions on a data set from astrophysical observations and data from an astrophysical simulation with actual application workloads. For small communities, surprisingly simple partitioning schemes already achieved good load balancing results. With increasing number of partitions, the extent-based weight function outperforms the other schemes with regards to reduced communication overhead and load balancing. Based on our throughput evaluation, workload-aware partitioning alone is not sufficient to completely level out query hot spots. As a consequence, we currently incorporate load balancing techniques such as a master-slave hierarchy in our data grid infrastructure.

Further interesting open research issues are adaption to heterogeneous nodes with different capacities or whether the complementary approach of merging *cold* regions can be included in our training phase and how communities can benefit from that. Finally, we are also looking into different data-intensive applications such as distributed data-mining in scientific communities.

Given the diversity of goals, resources, and applications among various scientific communities, it is clear that there is no single best data management solution. Thus, it is an important as well as an interesting challenge for the database community to explore possible alternatives for building scalable data management solutions for tomorrow's scientific federations.

Acknowledgements

7. REFERENCES

[1] M. Abdelguerfi and K.-F. Wong. *Parallel Database Techniques*. Wiley-IEEE Computer Society Press, 1998.

[2] M. Aly, N. Morsillo, P. K. Chrysanthis, and K. Pruhs. Zone Sharing: A Hot-Spots Decomposition Scheme for Data-Centric Storage in Sensor Networks. In *DMSN*, pages 21–26, Trondheim, Norway, Aug. 2005.

[3] M. Aly, K. Pruhs, and P. K. Chrysanthis. KDDCS: A Load-Balanced In-Network Data-Centric Storage Scheme for Sensor Networks. In *CIKM*, pages 317–326, Arlington, VA, USA, Nov. 2006.

[4] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. P-Ring: An Efficient and Robust P2P Range Index Structure. In *SIGMOD*, pages 223–234, Beijing, China, June 2007.

[5] D. J. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, 35(6):85–98, 1992.

[6] C. du Mouza, W. Litwin, and P. Rigaux. SD-Rtree: A Scalable Distributed Rtree. In *ICDE*, pages 296–305, Istanbul, Turkey, Apr. 2007.

[7] H. Enke, M. Steinmetz, T. Radke, A. Reiser, T. Röblitz, and M. Högqvist. AstroGrid-D: Enhancing Astronomic Science with Grid Technology. In *German e-Science Conference*, Baden-Baden, Germany, May 2007.

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

[9] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *VLDB*, pages 444–455, Toronto, Canada, Sept. 2004.

[10] D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Math. Ann.*, 38:459–460, 1891.

[11] M. Ivanova, N. Nes, R. Goncalves, and M. Kersten. MonetDB/SQL Meets SkyServer: the Challenges of a Scientific Database. In *SSDBM*, page 13, Banff, Canada, July 2007.

[12] R. Kuntschke, T. Scholl, S. Huber, A. Kemper, A. Reiser, H.-M. Adorf, G. Lemson, and W. Voges. Grid-based Data Stream Processing in e-Science. In *e-Science*, page 30, Amsterdam, The Netherlands, Dec. 2006.

[13] V. Markl and R. Bayer. Processing Relational OLAP Queries with UB-Trees and Multidimensional Hierarchical Clustering. In *DMDW*, page 1, Stockholm, Sweden, June 2000.

[14] W. O'Mullane, N. Li, M. Nieto-Santisteban, A. Szalay, A. Thakar, and J. Gray. Batch is back: CasJobs, serving multi-TB data on the Web. In *ICWS*, pages 33–40, Orlando, FL, USA, July 2005.

[15] J. Orenstein and T. Merrett. A class of data structures for associative searching. In *PODS*, pages 181–190, Waterloo, Ontario, Canada, Apr. 1984.

[16] T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, Load Balancing, and Efficient Range Query Processing in DHT Data Networks. In *EDBT*, pages 131–148, Munich, Germany, Mar. 2006.

[17] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware*, pages 329–350, Heidelberg, Germany, Nov. 2001.

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

[19] H. Samet. Hierarchical Representations of Collections of Small Rectangles. *ACM Comput. Surv.*, 20(4):271–309, Dec. 1998.

[20] T. Scholl, B. Bauer, B. Gufler, R. Kuntschke, A. Reiser, and A. Kemper. Scalable community-driven data sharing in e-science grids. *FGCS*, 2008. doi: 10.1016/j.future.2008.05.006.

[21] T. Scholl, B. Bauer, B. Gufler, R. Kuntschke, D. Weber, A. Reiser, and A. Kemper. HiSbase: Histogram-based P2P Main Memory Data Management. In *VLDB (demo)*, pages 1394–1397, Vienna, Austria, Sept. 2007.

[22] T. Scholl, R. Kuntschke, A. Reiser, and A. Kemper. Community Training: Partitioning Schemes in Good Shape for Federated Data Grids. In *e-Science*, pages 195–203, Bangalore, India, Dec. 2007.

[23] V. Singh, J. Gray, A. Thakar, A. Szalay, J. Raddick, B. Boroski, S. Lebedeva, and B. Yanny. SkyServer Traffic Report – The First Five Years. Technical Report MS-TR-2006-190, Microsoft Research, Microsoft Cooperation, Redmond, WA, USA, Dec. 2006.

[24] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce. Simulating the joint evolution of quasars, galaxies and their large-scale distribution. *Nature*, 435:629–636, June 2005.

[25] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, pages 149–160, San Diego, CA, USA, Aug. 2001.

[26] A. S. Szalay, J. Gray, A. R. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg. The SDSS skyserver: public access to the sloan digital sky server data. In *SIGMOD*, pages 570–581, Madison, WI, USA, 2002.

[27] S. Venugopal, R. Buyya, and K. Ramamohanarao. A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing. *ACM Comput. Surv.*, 38(1):3, Mar. 2006.

[28] X. Wang, R. Burns, A. Terzis, and A. Deshpande. Network-Aware Join Processing in Global-Scale Database Federations. In *ICDE*, pages 586–595, Cancun, Mexico, Apr. 2008.

[29] R. Zimmermann, W.-S. Ku, H. Wang, A. Zand, and J.-P. Bardet. A Distributed Geotechnical Information Management and Exchange Architecture. *IEEE Internet Computing*, 10(5):26–33, 2006.