Technische Universität München
Fakultät für Informatik

Diplomarbeit in Informatik

Data and query load balancing in grid-based data management

Bernhard K. Bauer

Technische Universität München
Fakultät für Informatik

Diplomarbeit in Informatik

Data and query load balancing in grid-based data management

Daten- und Anfragelastbalancierung für Grid-basiertes Datenmanagement

Bernhard K. Bauer

Supervisor:	Prof. Alfons Kemper, Ph.D.
Advisor:	Dipl.-Inf. Tobias Scholl
Submission Date:	15. Juni 2008

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this diploma thesis only supported by declared resources.

Garching, ...
15. Juni 2008 Bernhard K. Bauer

Abstract

Centralized access to data sets in E-Science communities like astrophysics is often encumbered by performance bottlenecks. While distributed databases try to adress this problem, they are facing challenges presented by query "hot spots", necessitating load balancing mechanisms.

In this work, we incorporate a mechanism for balancing data load as well as query load into HiSbase, a distributed data management system for spatial data. The approach tries to extend the design principles behind HiSbase by considering samples of not only the data but also the expected query load in the training phase and adapting the data dissemination schema to it, while still offering flexibility for the deployment to grid nodes.

We define query-aware weight functions for the training phase as well as different schemes for replication and query processing, experimentally ascertaining the adaptation of the training phase to the query workload.

Contents

1 Introduction

The globalization of data-based sciences. E-Science communities like astrophysics are facing various challenges associated with the ever-growing data sets. Big astronomical catalogs like the *Sloan Digital Sky Survey* (SDSS), *ROSAT* or the *Two Micron All Sky Survey* (2MASS) map hundreds of millions of astronomical objects [27, 28, 25]. Upcoming projects like the *Panoramic Survey Telescope and Rapid Response System* (Pan-STARRS), which creates images consisting of 1.4 gigapixels, covering an area equivalent to the entire sky four times a month [11], are expected to produce several terabytes of data per day.

Global repositories for spatial data are becoming increasingly mainstream, that is, they are getting attention from outside the academic world. Famous examples are Google Earth[1] for satellite pictures and maps and Google Sky or the WorldWideTelescope[2] by Microsoft Research for astronomical data. Both are aimed at a broad audience including amateurs[3], gathering data from various sources as well as allowing the addition of user-generated content in the case of Google Earth.

For many publicly-funded projects in the U.S., the data sets gathered have to be published after a grace period of one year. Publishing data has other advantages as well: It enables cooperation with previously unknown researchers and draws public attention to the conducted research.

However, providing centralized access can present a performance bottleneck, which is why many institutions are imposing service restrictions, for example on the number of results or the run time of the query, or using job queues like [14].

Another challenge is enabling exchange of data and correlation of data from different sources like *crossmatches* [9] of catalogs observing different wavebands for astronomical data. While organizations like the *International Virtual Observatory Alliance* (IVOA) are defining standards for exchanging astronomical data between instituions, the technical problem of having to copy huge data sets from one location to another to perform crossmatches still remains.

Various approaches to distributed data processing are being developed; while we are trying to touch upon some of them in chapter 5, in the focus of this work are *community-driven data grids* like HiSbase.

[1] `http://earth.google.com`

[2] `http://www.worldwidetelescope.org`

[3] The author estimates that he learned more about geography using Google Earth than he ever did in school.

1.1 Application scenario

The idea behind the HiSbase system is to create a single but *distributed* repository for spatial data, avoiding the bottleneck and single point of failure of a centralized system. HiSbase presents a flexible solution for this, in that it can be deployed as a global network distributing data from various archives or as a local grid acting like a caching proxy for back-end data archives. There is also ongoing research to extend HiSbase to more complicated data mining tasks than spatial cross-matches, profiting from the local availability of data. While the examples here are taken from the domain of astrophysics, HiSbase could also be used for other application domains dealing with spatial data, like geology or metereology.

1.2 Goals

The central task of this thesis, incorporating mechanisms for dealing with hot spots into HiSbase, presents some new objectives, while the old objectives still have to be kept in mind.

1.2.1 Load balancing

The crucial point in a distributed database system is of course the proper distribution of resource load to the available resources, namely space and processing time, in order not to overload any of the components.

Data. Astronomical data sets typically exhibit a large data skew as in figure 1.1, in part because the footprint area of many astronomical catalogs does not comprise the whole sky. A distributed database system has to be able to cope with this data skew, not only because of limited storage on each node, but also because storing a smaller amount of data can improve cache locality, adding to the responsiveness of the system.

In order to enable reproducability, published data sets are not changed; instead, new versions (*releases*) are published in regular intervals like every year. The absence of insertions and deletions in the data set are what makes having a training phase feasible.

Queries. Analyses of queries on the SDSS catalog like [10, 24] have established that – in addition to the data skew – the queries also exhibit a high skew, leading to *query hot spots* as visible in figure 1.2. In general, the query distribution may not exactly follow the data distribution. We are focusing on a common type of queries for astronomical data, *range queries*, so strictly speaking, the query distribution is higher-dimensional (because a rectangle has twice as many degrees of freedom as a point in the space), but as single query areas normally cover only a small portion of the entire data space, usually the query density can be assumed to be of the same dimensionality as the data density.

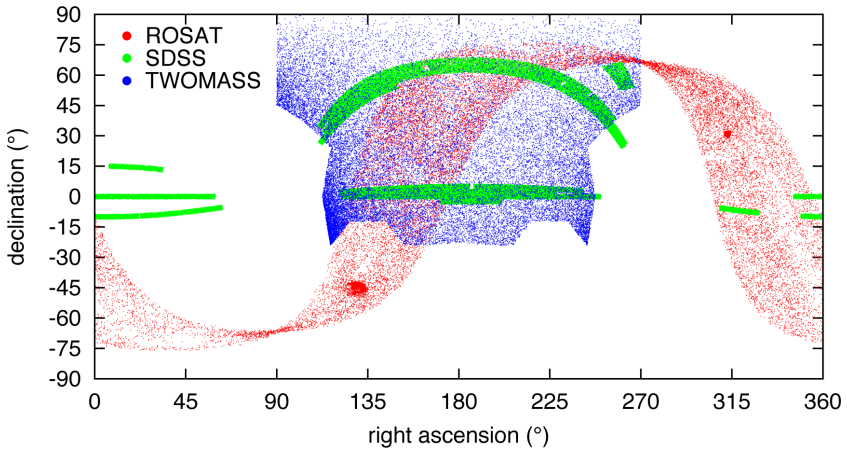


Figure 1.1: A highly skewed astronomical data set from three different catalogs.

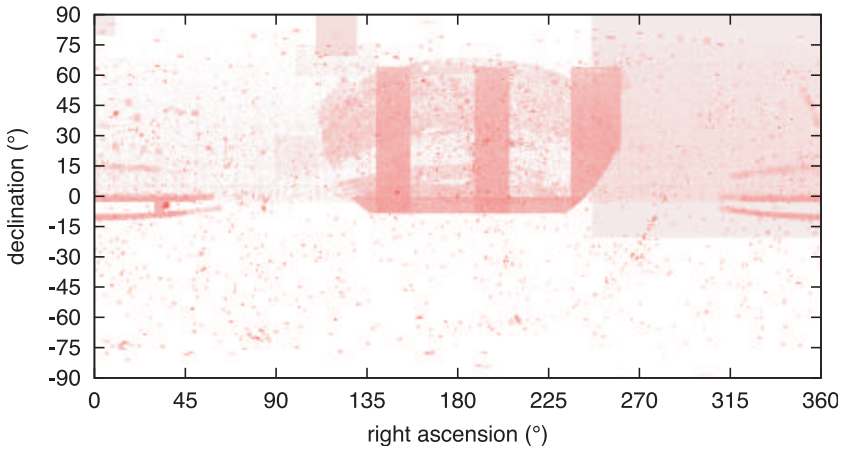


Figure 1.2: Queries submitted to the SkyServer web interface in August 2006.

1.2.2 Replication

Depending on the query patterns, replication of data may be a mandatory requirement for load balancing. Normally, replication is also incorporated for redundancy reasons; however, this is not our main target here, as the data archives can still function as backups in the case of data loss.

1.2.3 Overhead reduction

In order to ensure responsiveness and keep the strain on the nodes within a limit, the number of contacted nodes to process a query should be kept minimal. This means reducing data fragmentation by having each node store a spatially limited region.

2 Background

In this chapter, in order to get a feel for the context of the task at hand, we will first informally define the terms "hot-spot" and "heat", particularly with regard to our astronomical application domain. Then we will take a deeper look at the HiSbase system, trying to identify the points where modifications are necessary to incorporate load balancing capabilities into the system.

2.1 Hot spots

Hot spots in query processing can arise at different levels: At the data level, the query level or the region level. We will look at each of these in turn.

2.1.1 Data level

In a spatial data distribution, there are areas that are of bigger interest to users than others. In the astronomical domain, this may be due to interesting phenomena captured in these areas, the availability of data from different catalogs, giving rise to possible correlations, or simply the availability of more data.[1] In section 3.2.1, we try to capture this notion with point weight functions.

2.1.2 Query level

A *query-level* hot spot consists of a query that is issued repeatedly with the exact same parameters. An example for this is the default query in the SkyServer web interface[2]. However, in general, query-level hot spots are more common for exact matches as opposed to the range queries HiSbase was designed for. As such, they could be handled with other methods, which are outside the scope of this work.

2.1.3 Region level

A hot spot at region level is characterized by a high query processing load ("*heat*") at the site of the node storing it. Quantitatively, heat is a function of the number (and complexity) of the queries processed by the node as well as the amount of data relevante to the queries.

[1] Conversely, areas outside the footprint of any catalog are likely to be of low interest, a "cold spot" of sorts.

[2] `http://cas.sdss.org/astrodr6/en/tools/search/`

For now, we will have to make do with a semi-formal definition (Later on, we will use a similar but more formal definition for heat-based weight functions as well as for the evaluation of the modified training phase):

$$\text{Heat} := \text{Queries} \times \text{Data} \tag{2.1}$$

From this "equation", we can derive several observations: First, if we are distributing the heat equally, it means that nodes with many queries should have little data, and vice versa.

Second, reducing heat can be achieved in two different ways: by reducing the amount of data (splitting the region into smaller regions as in figure 2.1) or the number of queries (using replication and load balancing as in figure 2.2).[3]

As a final observation, we note that mutual replication between two nodes with equal heat does not change it. This becomes clear when we see that while the data held by each node doubles, the amount of queries is cut in half because of query load balancing.

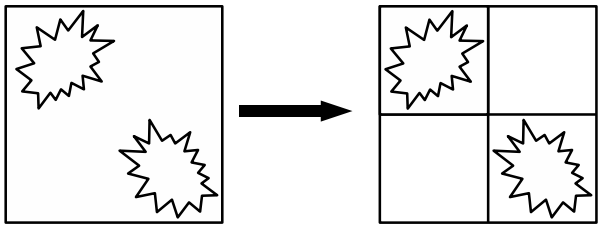


Figure 2.1: Splitting a region with a hot spot into smaller regions.

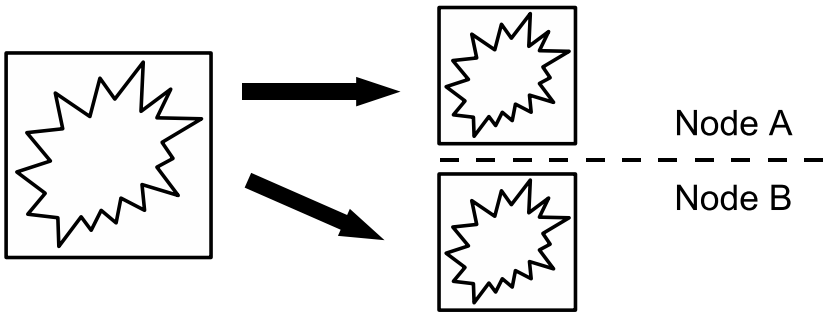


Figure 2.2: Replicating a hot spot across two nodes.

[3]Figures 2.1 and 2.2 are from [19].

2.2 HiSbase

HiSbase [30, 21, 20] is a distributed data management system for spatial data built on top of a *distributed hash table* (DHT), namely FreePastry[4], which implements the Pastry interface described in [17]. It can be structured into three phases running in succession: Training, data dissemination and query processing. In the next sections, we will take a closer look at each of these phases in turn.

2.2.1 Training

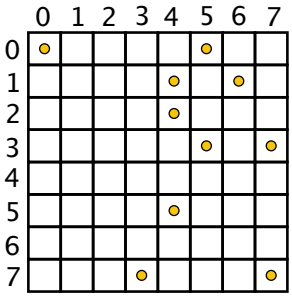


Figure 2.3: An exemplary data sample used for training on the data space $[0, 7] \times [0, 7]$.

The training phase uses a recursive decomposition of the data space based on quadtrees [18, 7] to create (hyper-)rectangular regions containing approximately the same amount of data, corresponding to leaves of the quadtree. To do this, we create a random sample of the whole data set, retaining only the coordinates of the data space (right ascension and declination in the astronomical domain) from the sample points.

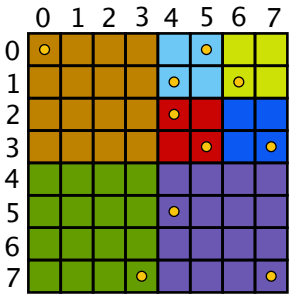


Figure 2.4: Quadtree with a maximum of two points per leaf.

Starting with a degenerate quadtree consisting of only one leaf, which contains the complete training set, we repeatedly split up the leaf with the highest number of points,

[4] `http://www.freepastry.org`

until the desired number of leaves is reached. There are various strategies for choosing the split point described in [22]; among them, the most common are picking the middle point of the region or the median in all dimensions of the points.

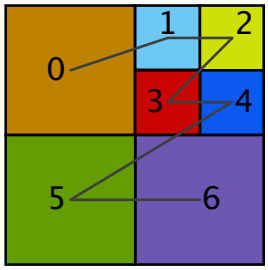


Figure 2.5: Linearization of the Quadtree using the Z-order.

The quadtree is linearized with a Z-curve [15] by traversing the tree in a depth-first manner and assigning a sequential region ID in turn to each leaf visited, yielding a linear ordering of the regions. An example for the training phase from [20] can be seen in figures 2.3 to 2.5.

It should be noted that the training phase is independent of the actual set of nodes; it just creates a histogram of the data distribution which will be used later on.

2.2.2 Data dissemination

Compared to a regular hash table, the actual assignment of the data to the peers in a DHT works indirectly by applying a hash function to the data items, mapping them onto the abstract key space of the DHT. As this would destroy the spatial locality properties of the data, we map the regions regularly-spaced on the key space, which is a ring in the case of Pastry. Similarly, each peer is assigned a *random* identifier on the key space. The DHT takes over routing a message with a given identifier to the peer nearest to it (in the sense of the proximity metric, which is the euclidean distance on the ring for Pastry).

Every peer is now responsible for the set of those regions (or identifiers) that are routed to it, comprising an interval on the key space. Continuing the example from the previous section, figure 2.6 shows the resulting mapping from regions to peers.

2.2.3 Query processing

To process a query, we use the histogram, which is distributed to all peers in advance, to look up all regions intersecting the query area. By simply addressing a region on the key space, we can use the DHT to route queries to the peer responsible for the region. In the case of a query area spanning more than one region, we determine a *coordinator* from the set of regions. The coordinator sends out requests to the other

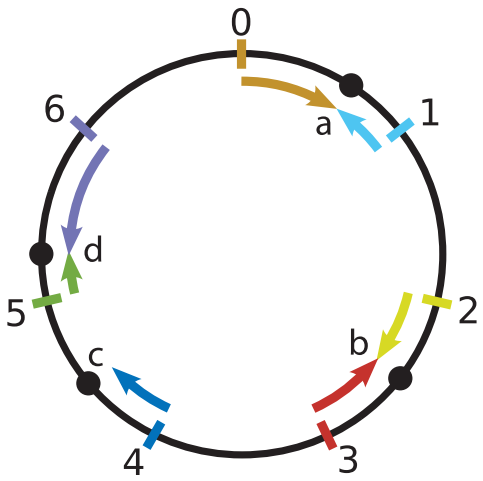
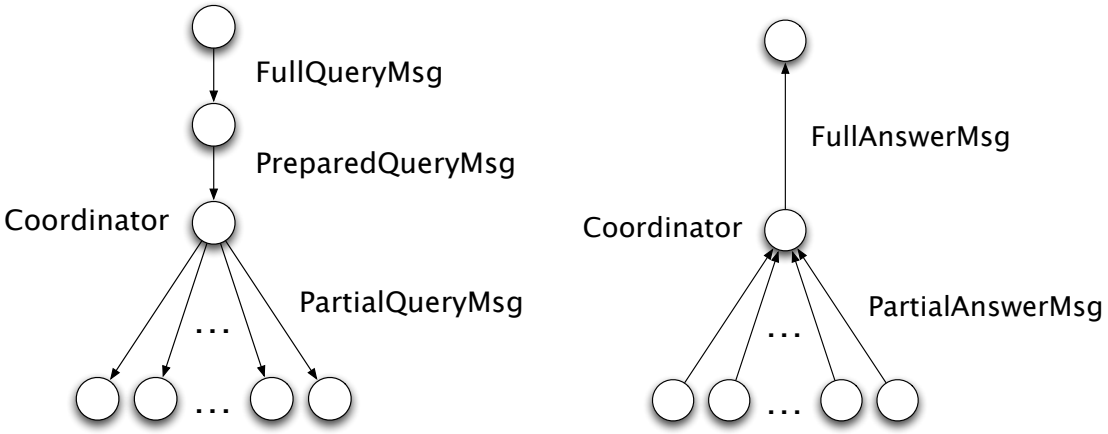


Figure 2.6: Mapping regions (numbers) to peers (letters) on the key space.

regions, collects their responses and returns them to the original requestor, as shown in figure 2.7. Since a peer can store more than one region (and in turn receive a query more than once, because requests are sent out to each *region* individually), the peers keep track of recently processed queries, using a unique query identifier to ignore duplicate queries.



(a) Sending out queries

(b) Collecting the responses

Figure 2.7: Parallel query processing.

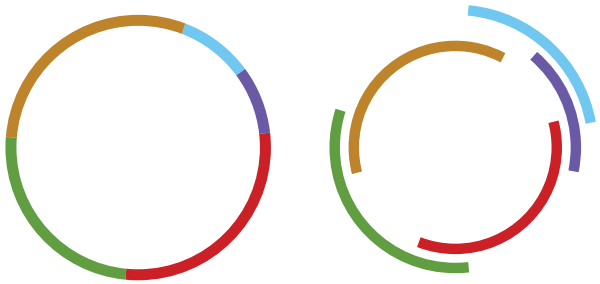
3 Histogram-centered load balancing

In this chapter, we explore a histogram-centered approach to load balancing, that is, an approach where we incorporate the query workload into the training process and modify the data dissemination and query processing phases to adapt to the histogram obtained in the training phase. We try to extend the original design, which is also peer-agnostic, meaning that the training phase does not take into account the actual set of peers. Dynamic approaches which measure the actual query load at runtime and try to balance the load based thereon are also explored, albeit more shortly.

After presenting the main ideas this approach is based on, the following sections respectively explain the modifications to the training, data dissemination and query processing phases needed to put the approach into effect.

3.1 Overview

First, we note that achieving a good load balancing for queries without using replication can be – depending on the actual query load – difficult, if not impossible. This is of course due to hot-spots that cannot be further decomposed, for example because the queries encompassing them are bigger than the regions.



(a) Non-overlapping (b) Overlapping

Figure 3.1: Decomposition of the key space into non-overlapping (primary) and overlapping intervals (comprising replicated as well as primary data).

If we want to incorporate replication into our data dissemination scheme, we first look at the scheme without replication and try to generalize from there. In the DHT system, each peer covers an interval of ID's in the key space, which together partition the complete key space. The obvious way to generalize this is to have the peers covering

overlapping intervals on the key space, as in figure 3.1. The length of the intervals here can be chosen in such a way that every peer has approximately the same amount of data, thus also evening out the data distribution.

In order to achieve the desired degree of replication, the peers have to be distributed more densely in regions with a high quantity of queries, compared to the approach only focused on data distribution. We do this by using the anticipated query workload in the training phase to split regions with a higher number of expected queries further than in the normal training. This way, areas with high query density are split into more regions, which later on provides for more peers being mapped into the area, because of the uniform distribution of regions and peers on the key space. The query workload can be gathered for example from query logs during a first, query-unaware deployment. Considering the data and query densities over all regions, we to approximate the *inverse* of the query density with the data density in order to achieve an equal heat distribution, in accordance with equation 2.1.

It should be noted that by still covering a whole interval of identifiers (and therefore regions), the actual area covered by a peer is expected to retain its spatial locality, even when "over-splitting" regions with a high query load. Thus, while the number of *regions* affected by a query may go up, the actual number of *peers* needed to answer the query should stay low.

3.2 Training

The normal training process chooses the leaf to split based on the number of points, so we generalize this to a weight function, splitting the leaf with the biggest *weight*. This weight function can also take into account the expected query workload. While the weight function used for this purpose operates on leaves, we first examine weight functions for points and queries, which serve as building blocks for the leaf weight functions.

3.2.1 Weighting points

Modeling the query distribution. Weight functions for data points are based on the assumption that some areas of the data space are more "interesting" than others to clients of the system and therefore attract more queries. Hence, we construct a distribution over the data space and define the weight of a data item as the density at this point. The corresponding generative model for the queries would pick a random sample from this query distribution and construct a query area around this point.[1]

Some aspects of this distribution could be predicted from the data distribution; for example, because researchers are interested in correlations between different data sets for the same area of space, an area which is covered by many catalogs could be expected to be of higher interest and therefore the target of more queries. In general, though, the "interestingness" is difficult to measure in any way but empirically.

[1] Which is how we constructed a synthetic workload for a uniformly distributed data set in chapter 4.

Weight functions. Calculation of the value of the weight function for a point p is based on the corresponding query density at this point. If we model the query distribution via a set of training queries Q, the weight function w_Q is defined over the number of queries, whose query area A_q contains the point:

$$\mathrm{w}_Q(p) := f(|\{q \mid p \in A_q\}|) \tag{3.1}$$

where the *counting function* f is an affine function of the form

$$f(x) := \lambda \cdot x + \sigma \tag{3.2}$$

As a simple example for f, we choose $f(x) = x + 1$. The reason for using an affine function over a linear function is to assign a positive weight to points which fall outside of any query area. In the complete absence of training queries, i.e. $Q = \emptyset$, this weight function has a value of 1 for every point, and thus is equivalent to the standard query-unaware weight. In general, the slope λ of the counting function serves as a *weight factor* for the number of queries, with the offset σ signifying the *default point weight*. The value of λ can be chosen depending on the catalog the point originated from, so as to also take into account the actual space needed to store the complete point, for an even more accurate data load balancing.

Because of the small size of the sample query areas, a high number of queries would be needed in order to completely cover the data space, particularly in the astronomical application domain. For example, the most frequent area for a query on the SkyServer web interface has a diameter of 0.025 arc minutes, which means that at least 156 million queries would be necessary in order to cover every point in the whole $[0, 360] \times [-90, 90]$ square degree space. For a random and more skewed query distribution (which we are interested in), the number of queries needed to accurately cover the whole data space would be even higher.

The effect of using a smaller number of training queries becomes clear on points outside of the areas of the training queries: Such a point is assigned the default, minimum point weight, although it may well be lying in an area with a high query density, possibly leading to unwanted hot spots when handling a "real" query workload. This problem of unexpected lower performance on previously unseen data sets is similar to the problem of *overfitting* in density estimation, where the assumed model is adapted too strongly to the training data instead of the actual target density.

To overcome this problem, instead of the original query areas, "blown up" versions $\hat{A}_q$ are used, which are scaled around their middle point by a constant factor ϕ. To compensate, the weight factor λ is reduced by approximately the same factor. This corresponds to a simple variant of blurring the distribution.

The new weight function is the following:

$$\mathrm{w}_Q(p) := f(|\{q \mid p \in \hat{A}_q\}|) \tag{3.3}$$

Even with this additional smoothing, getting the parameters ϕ and λ right can be quite difficult and may often require multiple iterations of training and testing. Another

approach would be to move away from a non-parametric model to a simpler one that uses different resolutions for sampling the query density, which eventually gives rise to the heat-based weight function defined in section 3.2.3.

3.2.2 Weighting queries

Weighting queries by a set of points P works along the lines of weighting points by a set of queries as defined in the previous section. The point-based weight function for a query q is defined as follows:

$$\mathrm{w}_P(q) := f(|\{p \mid p \in A_q\}|) \tag{3.4}$$

The duality between weighting points and queries. Using a *linear* counting function f and the indicator function $\mathbf{1}[\varphi]$, which is 1 if the formula φ is true and 0 otherwise, we can write the weight functions in the following way:

$$\begin{aligned} \mathrm{w}_P(q) &= \sum_{p \in P} \lambda \cdot \mathbf{1}[p \in A_q] \\ \mathrm{w}_Q(p) &= \sum_{q \in Q} \lambda \cdot \mathbf{1}[p \in A_q] \end{aligned}$$

for a set Q of queries and a set P of points. Therefore, summing over the weight functions, we get:

$$\sum_{q \in Q} \mathrm{w}_P(q) = \sum_{q \in Q} \sum_{p \in P} \lambda \cdot \mathbf{1}[p \in A_q] = \sum_{p \in P} \mathrm{w}_Q(p) \tag{3.5}$$

This equation could also be extended to affine counting functions.

3.2.3 Weighting leaves

Weighting whole leaves contains the central idea of the modified training process, as the training process now tries to even out the weight distribution over all regions rather than simply the number of points.

When weighting a leaf l, we associate with it as usual the set P_l of all points contained in its area, and also the set Q_l of all queries *intersecting* it. The easiest weight function sums over all point weights:

$$\mathrm{w}_{Q_p}(l) = \sum_{p \in P_l} \mathrm{w}_{Q_l}(p)$$

or over all query weights:

$$\mathrm{w}_{P_q}(l) = \sum_{q \in Q_l} \mathrm{w}_{P_l}(q)$$

which is roughly equivalent according to eq. 3.5.

A heat-based weight function. Another possible weight function, w_{pq}, combines queries and points into a measure of the heat, as defined in equation 2.1:

$$\mathrm{w}_{pq}(l) := f(|Q_l|) \cdot |P_l| \tag{3.6}$$

where f is a counting function like in section 3.2.1. An offset for the number of points is not necessary, and the scaling factor for the number of points could be incorporated into f. When using a counting function with an offset of 1, this weight function is again equivalent to the default method in the absence of any training queries.

Compared to the approach using the query-based point weight function, this weight function corresponds to blowing the query areas up to the full extent of the leaf area, which is a coarse method of blurring the query distribution, adapted to the data distribution.

3.3 Data dissemination

The data dissemination phase is modified by incorporating an extra replication phase, usually after the normal data staging is completed and every peer has its own primary data. When replicating a region from peer B, peer A first sends a request to peer B. Peer B then returns the data and also stores the identity of peer A, which means that a peer knows the locations of all replicas of its own regions.

The data for replicated regions is stored alongside with the primary data. For the purpose of query processing, replicated and original regions are treated identically.

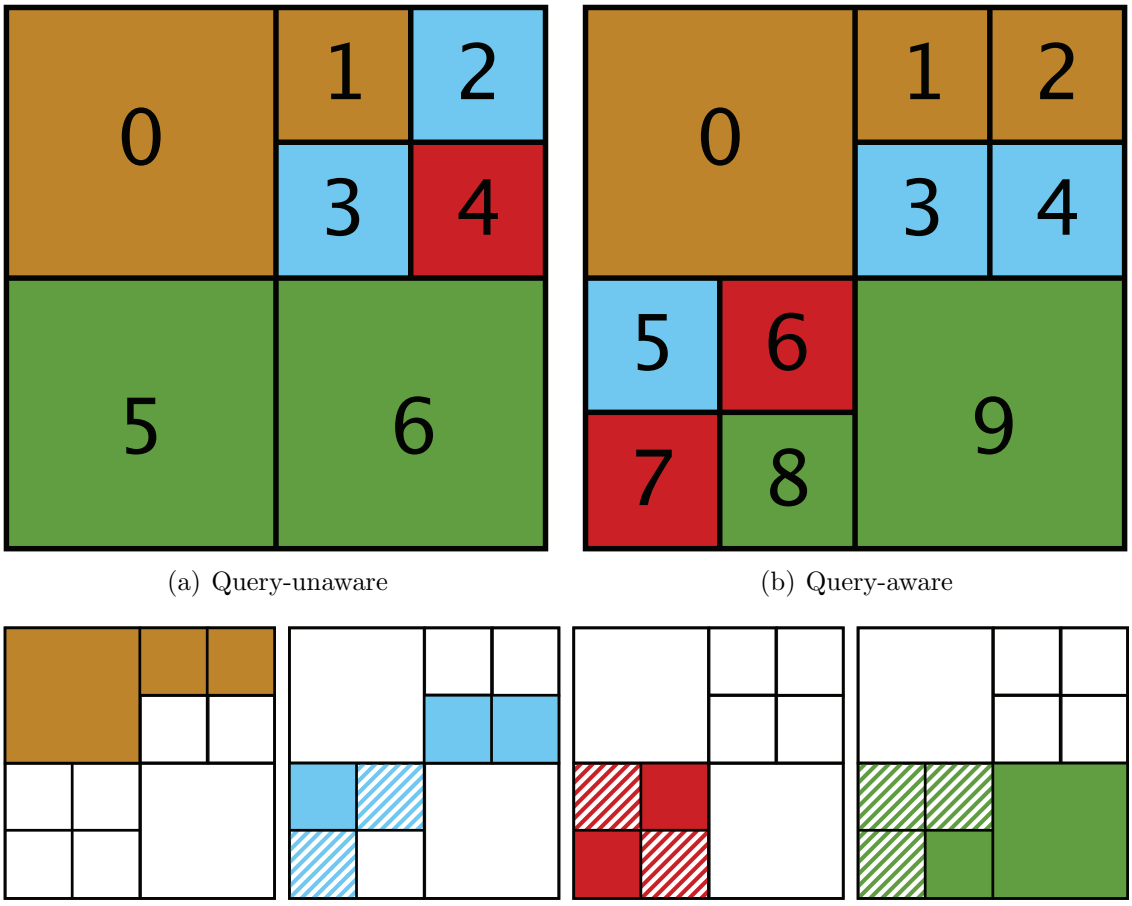
For negotiating the actual replication scheme determining which peer replicates which region there are different strategies, which we look at in the following sections.

3.3.1 Static strategy

In keeping with the static approach to data dissemination already used in HiSbase, we first look at a static replication strategy. For that, we define a configuration parameter *data capacity*, whose base value C_{base} is the total amount of data divided by the expected number of peers.

After the primary data dissemination phase, if the total amount of data stored by a peer is less than its data capacity, the peer replicates neighboring regions until its capacity is saturated. Choosing neighboring regions to replicate has the advantage of avoiding data fragmentation, as the interval on the key space covered by the peer is simply extended.

An example from [19] can be seen in figure 3.2. Region 5 in the query-unaware histogram contains a hot spots, so it is split up into four regions. As all the regions in the original histogram stored about the same amount of data, the newly created regions contain less data than the other regions, so the peers covering them have free capacities, which they fill by covering additional regions. After replication, the original region 5 is now covered by three peers, which can use load balancing to distribute the query load among them.



(a) Query-unaware

(b) Query-aware

(c) Query-aware with replication

Figure 3.2: Data dissemination schemes using static replication. Solid squares indicate primary regions, hatched squares replicated ones.

Because of the normal variations in the data amounts stored by the peers, peers which just by chance have a lower amount of data also start replicating additional data, achieving an even smoother data distribution at the cost of possibly unnecessary replication. If this is not desired, the data capacity can be set to a lower value than C_{base}.

Likewise, if the data capacity is set to a higher value, one can expect a "basic" replication unaware of the query workload: A replication factor of n can be achieved by setting the data capacity to $n \cdot C_{base}$.

3.3.2 Dynamic strategy

Contrary to the static replication strategy described in the previous section, the normal replication approach used in a DHT is a more dynamic one like in [5], based on the actual query load at runtime. In the context of HiSbase, this would mean that peers

with a high query load try to get other peers to replicate their regions. However, this presents several difficulties:

First, for each region it would have to be determined when to replicate it. Because burdening peers already working at full capacity even more with the task of replicating other regions is undesirable, the query load on a peer would have to be taken in comparison to the load on the other peers. Furthermore, for short spikes in traffic the better approach would probably be to "sit them out" instead of initiating costly data transfers.

Likewise, if a region needs to be replicated, it can prove difficult to find a suitable peer as a target for replication. While peers could announce free capacities via a broadcast mechanism like Scribe [4], the task of choosing a peer therefrom still remains. Because of the spatial locality, neighboring peers are likely to have the same query load, so replication wouldn't help in alleviating it. On the other hand, replicating a region at the site of a distant peer (on the keyspace) could lead to data fragmentation.

In addition, a mechanism would be necessary for freeing resources no longer needing to be replicated, because otherwise the peer databases would fill up with replicated data until they could no longer accept new replicated regions.

By treating replication as an offline rather than an online problem, we can avoid most of these difficulties; at the same time, dynamic replication for HiSbase remains a topic of ongoing research.

3.4 Query processing

In order to deal with replicated data and avoid duplicate results during query processing, a sequential processing strategy as illustrated in figure 3.3 has to be employed rather than the normal parallel strategy described in section 2.2.3. This means that the coordinator, instead of sending out all requests for regions at once, forwards the request to one other peer, who processes the query and in turn forwards it to another peer, until all regions relevant to the query have been processed. The database itself can still be accessed in parallel.

Before the query is issued to the database, it has to be rewritten to restrict it to regions not processed yet (figure 3.4), because otherwise duplicated regions would run the chance of being processed by more than one peer. As a side effect of the sequential processing strategy, keeping track of recent queries is no longer necessary, as every peer is contacted at most once for one query.

3.4.1 Load balancing using routing

While correctly working for replicated data, the processing algorithm described so far does not yet offer any actual load balancing capabilities. The first variant for that uses the routing mechanism built into the DHT: On the route to its destination, the intermediate peers try to answer a query as early as possible instead of simply forwarding it. If some part of the query can already be processed, the whole query never arrives at

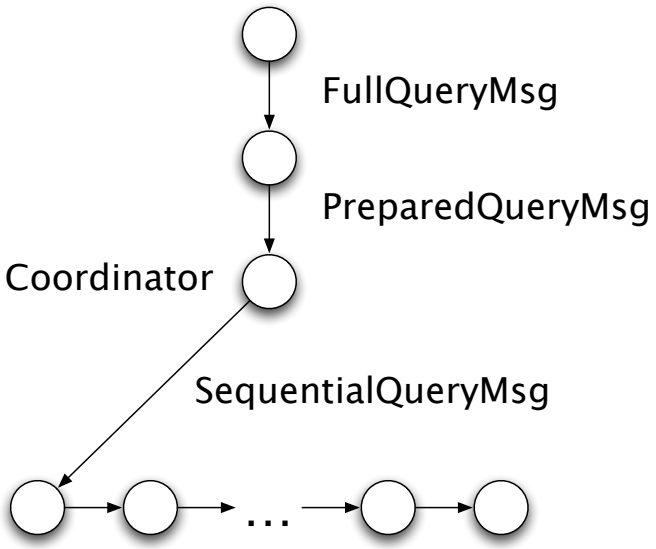
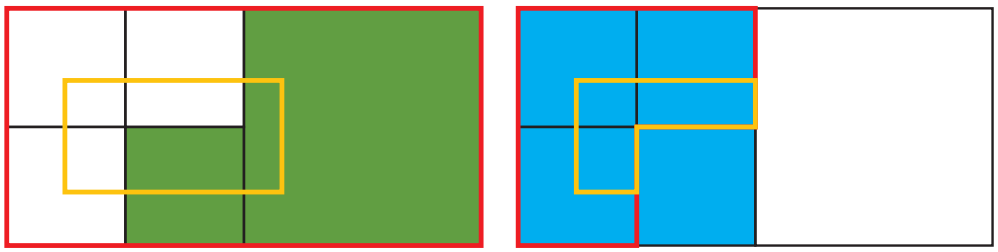


Figure 3.3: Sequential query processing.

its original target; instead, the intercepting peer completely takes over query processing, picking a new region to forward the query to if any unprocessed regions remain.

Because rerouted queries never even arrive at their destination, this strategy is best suited for workloads with high numbers of low-complexity queries, similar to DNS load balancing for web servers.

The drawback for this strategy is that it depends on the structure of the overlay network; in particular, it needs a network topology with a certain diameter. Small Pastry networks tend to use direct connections for performance reasons; artificially lowering the number of direct connections (for example, by reducing the size of the leaf set in Pastry) could impede the routing performance.



(a) Peer 1 (b) Peer 2

Figure 3.4: Query rewriting with the sequential query processing strategy. The original query (yellow) is rewritten at the site of the second peer in order to restrict it to regions not processed yet (red).

3.4.2 Load balancing using delegation

Another approach is to delegate queries to another peer replicating one of the queried regions, as shown in figure 3.5. This approach offers much flexibility, as the primary peer can choose any one of the peers replicating the targeted regions as well as itself to further process the message.

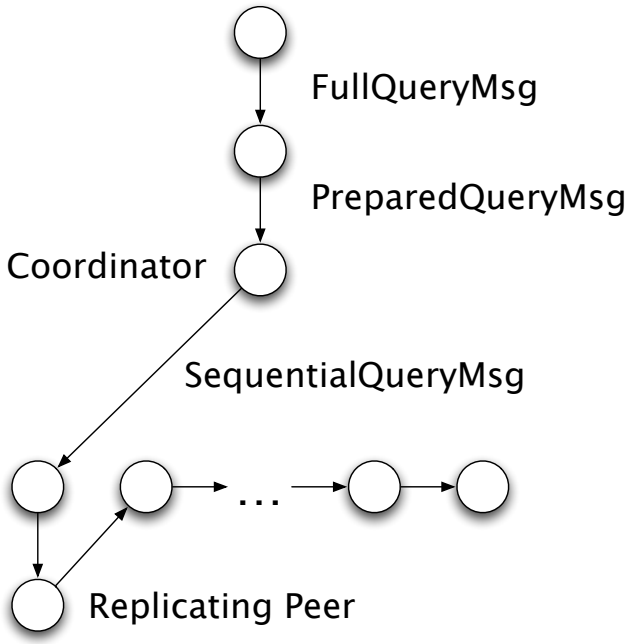


Figure 3.5: Load balancing using delegation with the sequential processing strategy.

In order to avoid unlimited delegation, a delegated query has to be processed in the next step and cannot be further delegated. However, after processing, the remaining query can be forwarded again to another peer.

When processing a query, every peer uses its primary as well as its replicated regions in order to minimize the number of peers to contact and therefore the number of database accesses needed.

Compared to the previous one, this strategy is more suited for small numbers of high-complexity queries (similar to front-end proxies for large application servers in the context of web servers). Incidentally, this is also the direction for future extension of HiSbase to more complicated distributed data mining tasks.

It should be noted that the two load balancing strategies described here are not mutually exclusive, as they leverage different aspects of the query processing mechanism and therefore could very well be employed both at the same time.

4 Evaluation

In this chapter, we describe the evaluation process we used to assess the feasibility of the query-aware training process, detailing the experimental setup as well as comparing the obtained results to our expectations.

4.1 Experimental setup

We conducted an evaluation of the modified training process using our HiSbase prototype in Java on two different data sets from the astronomical domain, together with corresponding query workloads. Because evaluating the actual performance of the whole system is subject to random fluctuations owing – among other factors – to the random placement of peers on the key space, we restricted the experiments to the training phase, leaving a "real-world" performance evaluation to further research.

4.1.1 Data sets

Observational data. The first data set consists of a 0.1% sample from subsets of the ROSAT, SDSS and 2MASS catalogs (already shown in figure 1.1), together making for about 150 000 data points. The associated query workload was constructed from queries submitted to the SkyServer web interface during August 2006 [10]. Because the biggest part of the queries uses radial searches, we constructed similar rectangular queries from them using the bounding box of the original circular query 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 whole query log. Out of the remaining queries, 1 000 000 were used for training and 100 000 queries for testing. Both the data and the query workload exhibit a highly skewed distribution, as evidenced in figures 1.1 and 1.2 in the introductory chapter.

Simulation data. In order to assess the effect of query-aware training on uniformly distributed data with a high query skew, we used a 0.1% data sample (figure 4.1) from the millennium simulation [26] consisting of about 150 000 points distributed in a roughly uniform manner on the area $[-45°, 45°] \times [-45°, 45°]$. The query areas were artificially generated with their midpoints (p_x, p_y) following a 2-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

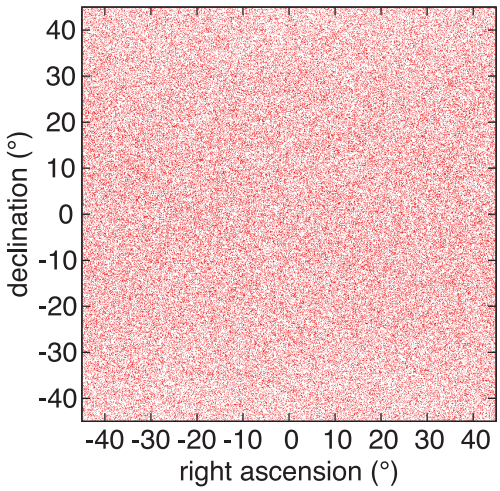


Figure 4.1: The uniformly distributed simulation data set from [26].

correspond to the 5 most frequent query radii on the SDSS catalog. In this way 10 000 queries were generated for training and 1 000 queries for testing the resulting histograms.

4.1.2 Expected results

As a direct effect of the modified training process, we expected regions with high query load to be split further, adapting to the query workload used in training.

For a quantitative evaluation, we examined again the heat for the histogram regions as defined by the heat-based weight function w_{pq} in equation 3.6, using the set of testing queries as defined in the previous section.[1] Basing the quantative evaluation on the notion of heat is motivated by the assumption that more data inside a region results in more work when processing a query intersecting this region, even if the query area itself does not contain all of the points stored inside the region.

We expected an overall reduction in hot-spots, as well as an even distribution of the remaining heat. For the former, we calculated the total heat over all regions, for the latter we constructed Lorenz curves and calculated the Gini coefficient as in [16], which is defined as the area between the Lorenz curve and the diagonal.

4.1.3 Testing parameters

Weight function. As a baseline we used the uniform weight function w_p which counts the number of points, comparing it to the query-based point weight function w_{Q_p} with various parameters as well as the heat-based leaf weight function w_{pq}. For the weight function w_{Q_p} we set the default weight $\sigma = 1$ and the weight factor $\lambda = 1$ for the uniform data set and $\lambda = 0.01$ for the skewed data set, as we used a much bigger query set for

[1] Because the data set *is* known in advance – contrary to the query workload – we used the same data set for training and testing.

it. For the uniform data set, we varied the query scaling factor ϕ between the values 10, 50, 100, 200 and 400, and between the values 10, 20, 40 and 80 for the skewed data set.

Histogram construction. We constructed histograms with 16 (4^2) to 65 536 (4^8) leaves using the standard splitting strategy from [22] as well as the median heuristic. When using the latter, we based the computation of the median only on the data points, without taking into regard their weight.

Amount of training queries. In analogy to [22], in a separate set of experiments we varied the amount of queries used for training, using 5% of the query workload as well as 10 to 90% in increases of 10% for training and the rest for testing. We adjusted for the different amount of test queries by dividing the heat for a region by the total number of testing queries, resulting in a "relative" heat value.

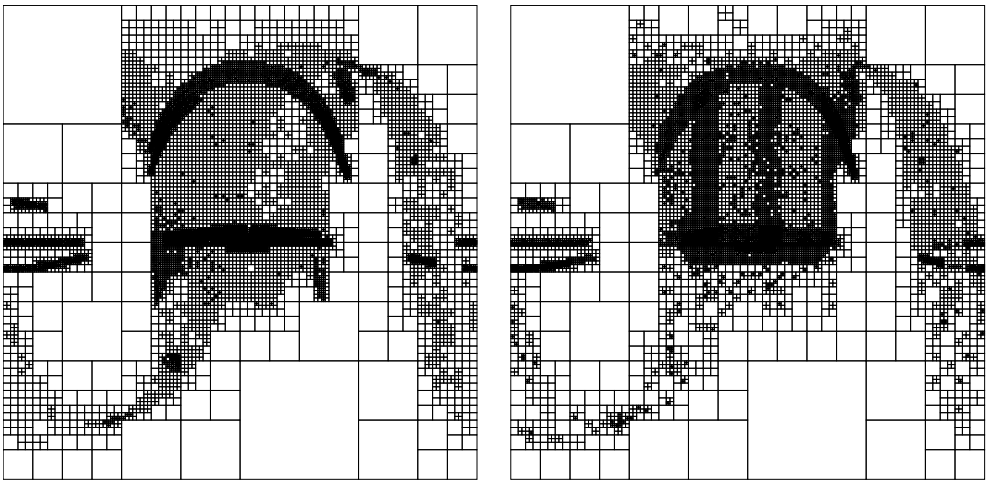
Amount of training data. We also conducted the same experiments using bigger samples of the data sets, namely 1% and 10% samples. As the results obtained in these experiments were quite similar to those with the 0.1% sample, we only present the latter in the following.

4.2 Results

Direct effects. The effect of using query-aware training on the observational data set can be seen in figure 4.2, where we compare the histograms created with uniform (w_p) and heat-based (w_{pq}) weight functions. Compared to the query-unaware histogram, we can clearly see an imprint of the W-like shape visible in the query workload in figure 1.2, suggesting that the histogram is adapted to the query workload.

With the uniformly distributed data set, the effect on the data density can be seen more clearly, for example in figure 4.3. For w_{pq}, the amount of data per region is clustered into three bands, because splitting a histogram leaf exactly quarters its area, thereby also approximately quartering the amount of data, as can be seen nicely in figure 4.4 While the data distribution is non-uniform for w_{pq}, the region load is distributed much more evenly, with the big spikes (or "flames") from the uniform weight function being smoothed out.

Quantitative evaluation. As can be seen in figures 4.5 and 4.6, the overall best results were achieved with the heat-based weight function w_{pq}. The parameters for w_{Q_p} prove quite difficult to tune, as was hinted at in section 3.2.1; using the wrong parameters can lead to either no difference to query-unaware training or to suboptimal results. We can see this in figure 4.4, where in the leftmost histogram a scaling factor of $\sigma = 10$ does not change the training process at all. In the next histogram, with $\sigma = 50$ some regions near the center are still not split up enough to reduce the effect of the query hot spot in the center, leaving a "burn mark". When increasing σ even further, the regions in



(a) Query-unaware (b) Query-aware

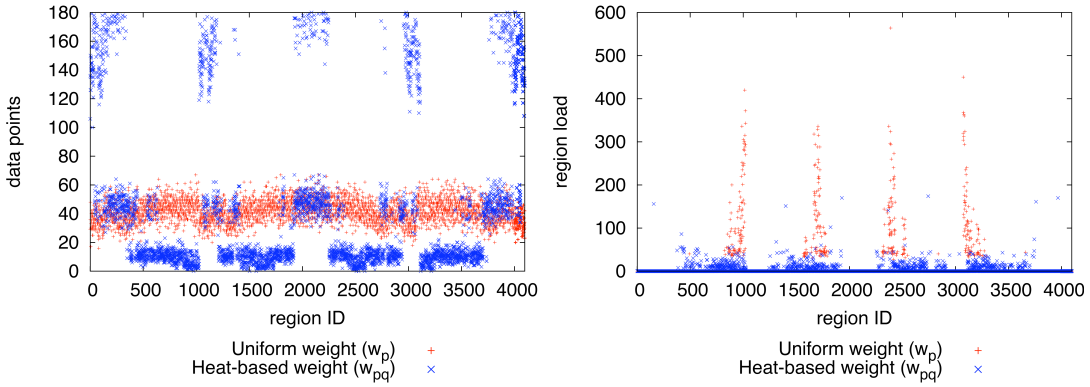
Figure 4.2: Histograms of the observation data set generated with query-unaware and query-aware training, respectively.

the center are split up, but at the expense of the regions lying further outside, resulting in artificial hot spots outside the center. Using the heat-based weight function in the rightmost histogram yields the best distribution of heat.[2] More results can be found in appendix A.

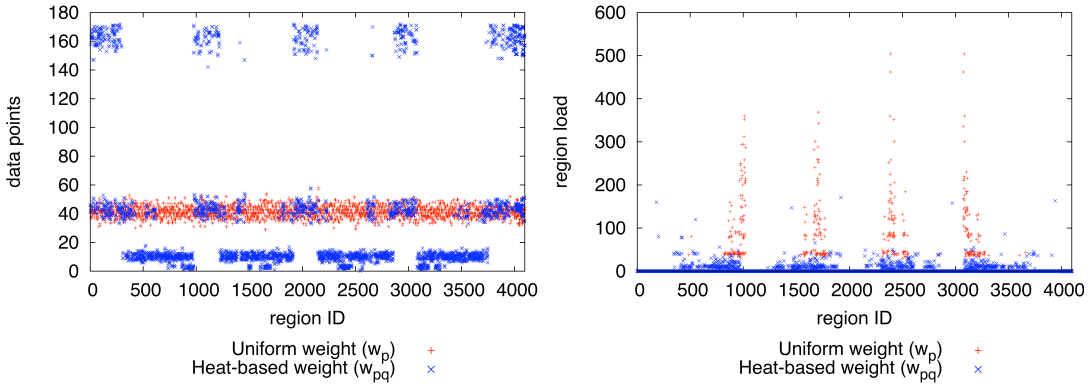
Figures 4.7 and 4.8 show that using only a small part of the available queries for training is already sufficient to achieve good values for the Gini coeffient or the total heat, so in analogy to the data sampling from [22], sampling queries in combination with the heat-based weight function seems to be an apt way to obtain an approximation of the expected query workload.

All in all the results seem to show the feasibility of the query-aware training appraoch and give reason for hope that the histogram-based approach leads to adequate load balancing.

[2]Which is not that surprising, as the heat-based weight function is designed specifically for this task.

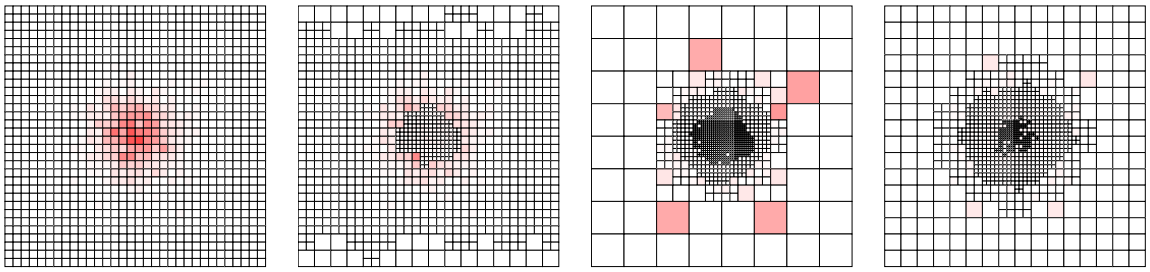


(a) midpoint splitting



(b) median splitting

Figure 4.3: Data density and region load (heat) over 4096 regions of the simulation data set.



(a) w_p or $w_{Q_p,\phi=10}$ (b) $w_{Q_p,\phi=50}$ (c) $w_{Q_p,\phi=400}$ (d) w_{pq}

Figure 4.4: Visualization of the heat generated by histograms over the uniform data set created with various weight functions.

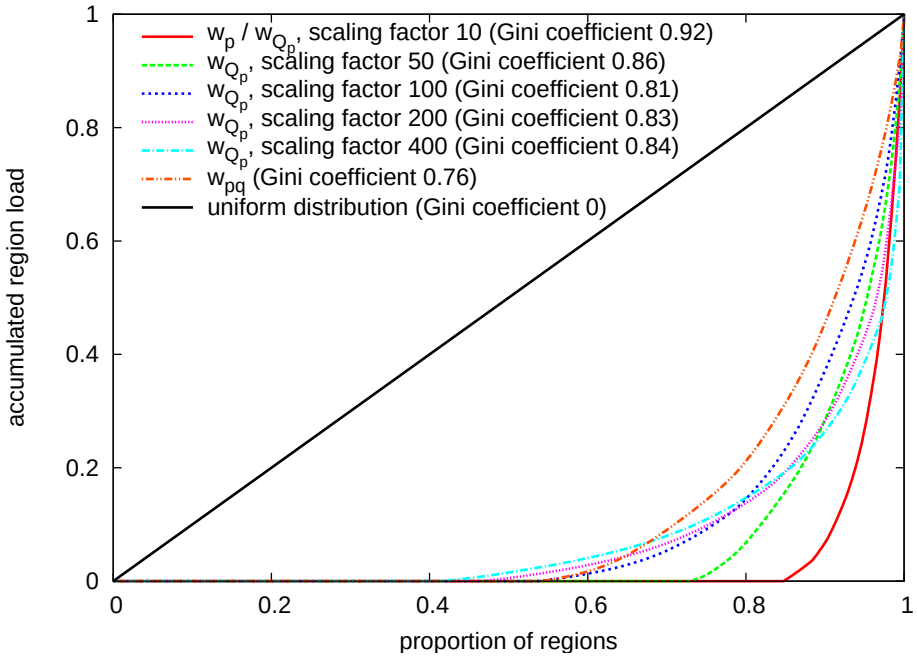


Figure 4.5: Lorenz curves for histograms with 1024 regions over the simulation data set, created using midpoint splitting.

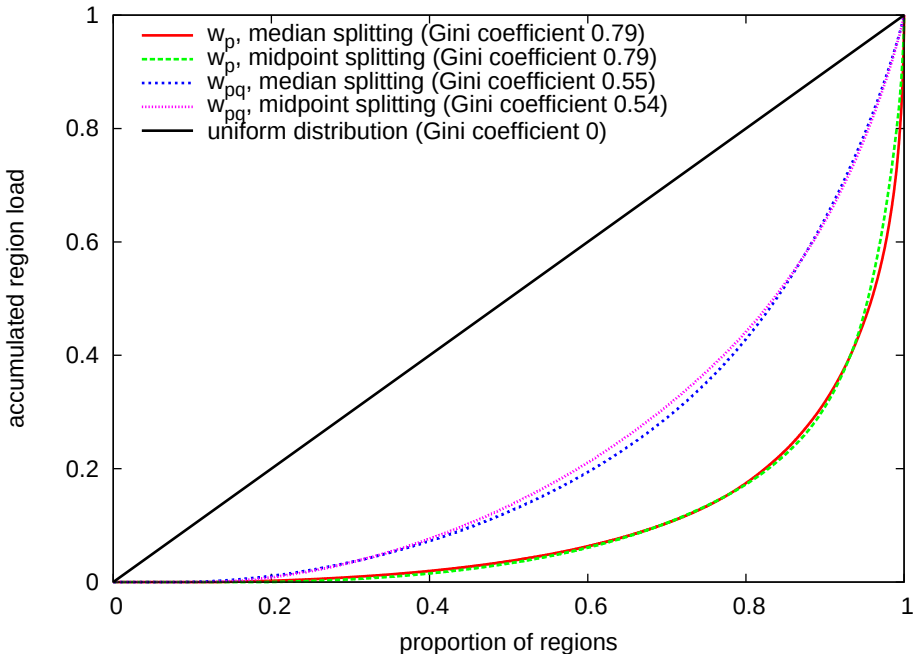


Figure 4.6: Lorenz curves for histograms with 4096 regions over the observational data set.

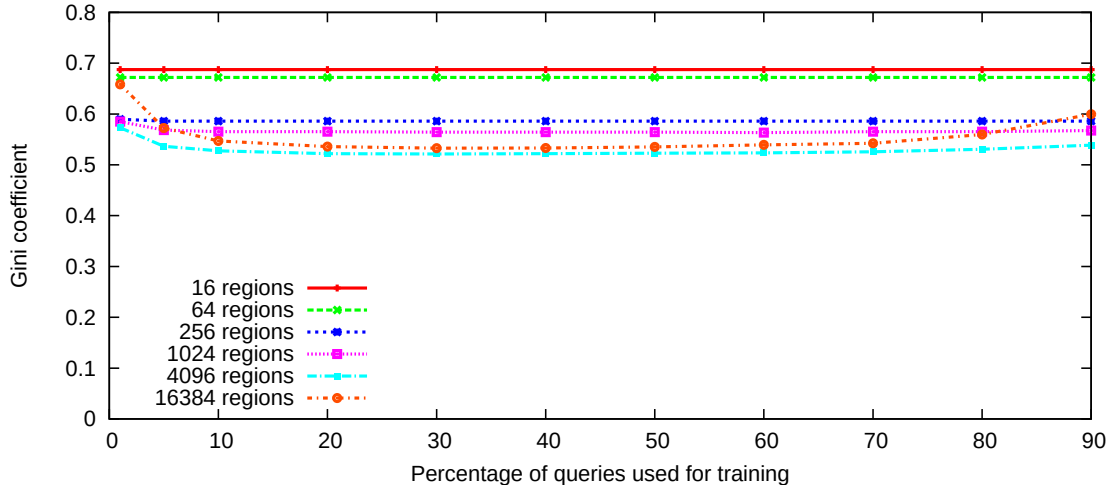


Figure 4.7: Gini coefficient for different percentages of the query workload on the *observational* data set used for training with w_{pq}.

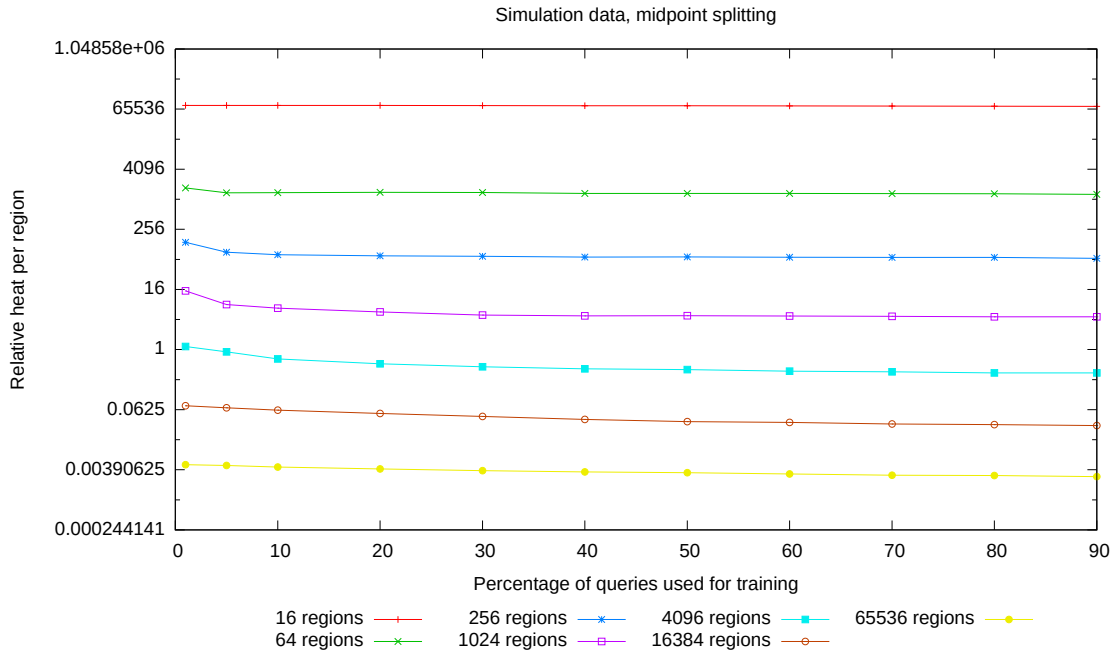


Figure 4.8: Relative heat for different percentages of the query workload on the *simulation* data set used for training with w_{pq}.

5 Related work

Distributed data processing. In the realm of astrophysics, different ways for distributed data processing are being developed. The SkyQuery web service described in [3] provides query processing for distributed astronomical data sets using sophisticated query scheduling algorithms [29] to optimize throughput. Research is ongoing to implement pipelining on top of VOSpace services [8] to avoid congestion of available system resources. In a sense, this approach is orthogonal to the approach used by HiSbase: While HiSbase first distributes data and then routes queries to the data, the proposed VOPipes approach first sets up the processing scheme and then feeds chunks of data into the "pipes".

Load balancing. Much of the work in load balancing for distributed systems is based on skip graphs [2], for example [1] and [23]. Skip graphs, which are randomized data structures based on skip lists, allow *arbitrary* identifiers on the key space (rather than pseudo-random hash values), as the key space does not need a metric but only a total order on the keys used (like for example the Z-order of the quadtree leaves in the histogram). However, in order to even out data skew, the distribution of peers has to follow the data distribution, which is difficult for the decentralized placement of peers. In comparison, metric DHT's like Pastry can use random identifiers on the key space, which can be assigned independently.

In [13], an approach similar to skip graphs is presented, insofar as it also uses an order preserving routing mechanism, so the same issues as for skip graphs apply.

HotRoD [16] stores replicates on a second, displaced ring but does not deal with data skew. P-Ring [6] uses peers storing no data at first as "helper peers", which then pick up part of the data stored by an "owner peer" in order to even out data skew. While HiSbase supports a similar notion by utilizing free data capacities, we use them for *replicating* data, having already addresed the data skew in the training phase. P-Ring on the other hand does not offer replication capabilities.

Some of the issues associated with replication of data are adressed in [12], like ensuring consistency in the face of data updates. However, the use of versioned data sets in astronomy eliminates this problem.

6 Summary

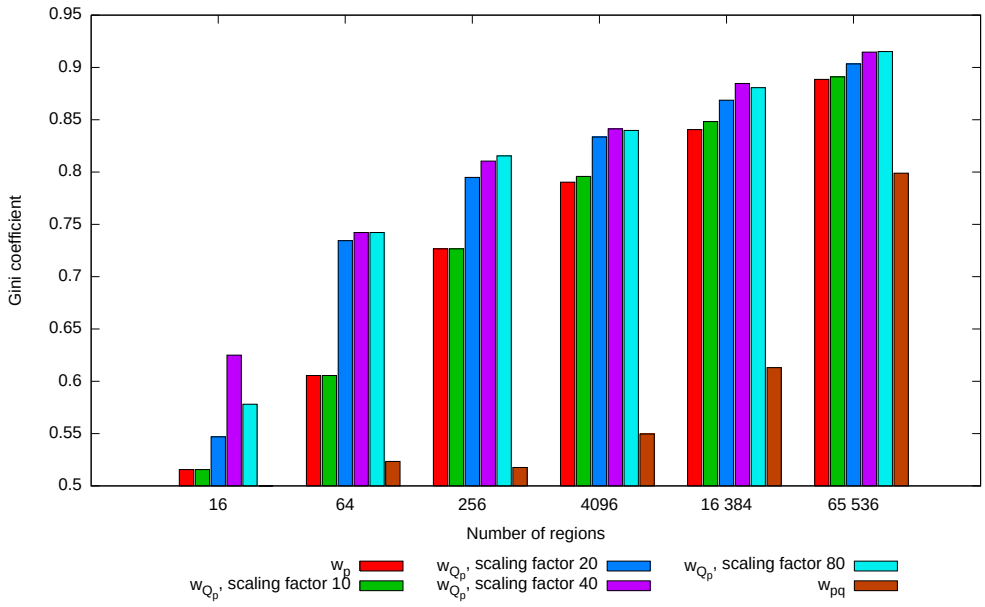
In this thesis we presented a histogram-centric approach to load balancing in a distributed data management system.

During the training phase, we artificially thin out the data distribution in regions with high query load. For that, we define query-aware weight functions that are used together with a sample of the expected query workload to even out the "heat", a measure of the expected load, over the regions instead of the pure data distribution.

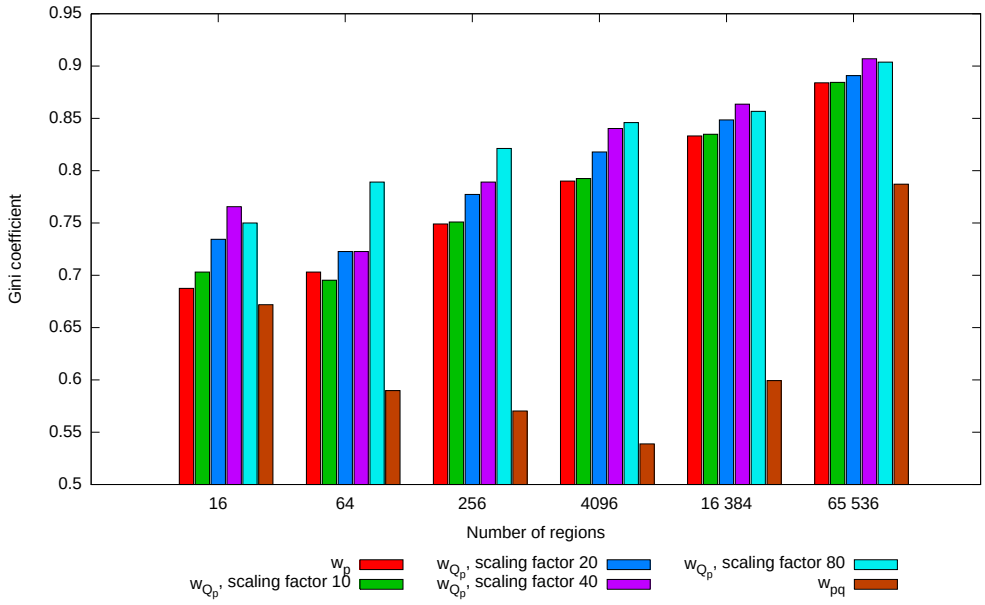
As peers with high query load and little data replicate the data among themselves, requests can be delegated to replicating peers in a sequential query processing strategy.

The HiSbase system presents a flexible framework for using different strategies for data dissemination as well as query processing. First benchmarks of the training phase show promising results for the efficiency of the complete system.

A Benchmark results

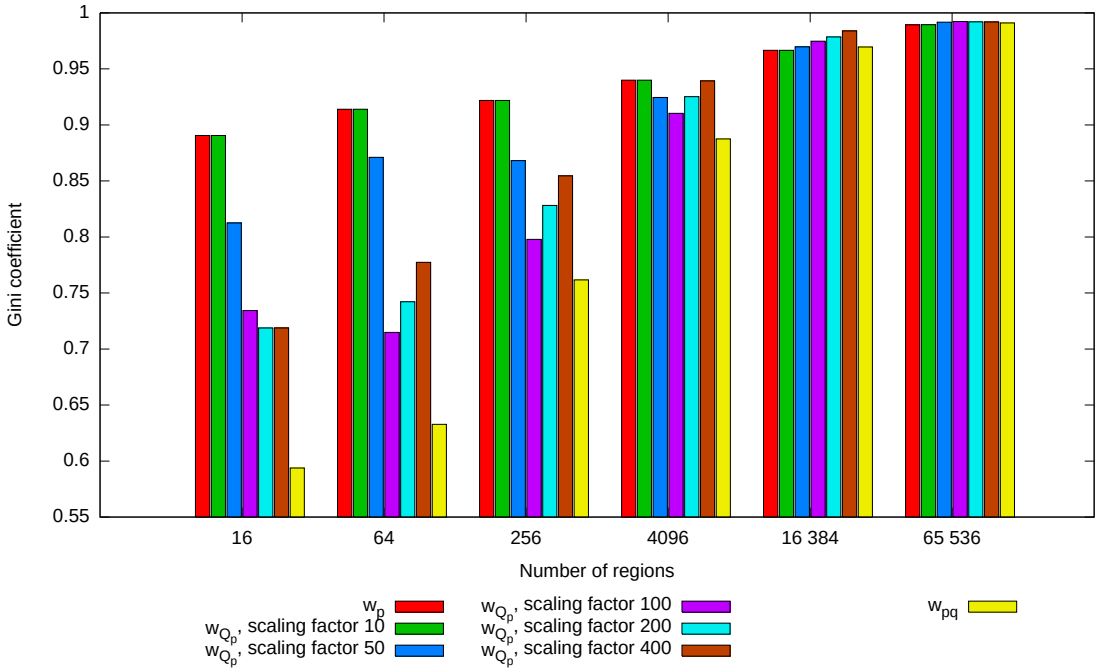


(a) Midpoint splitting

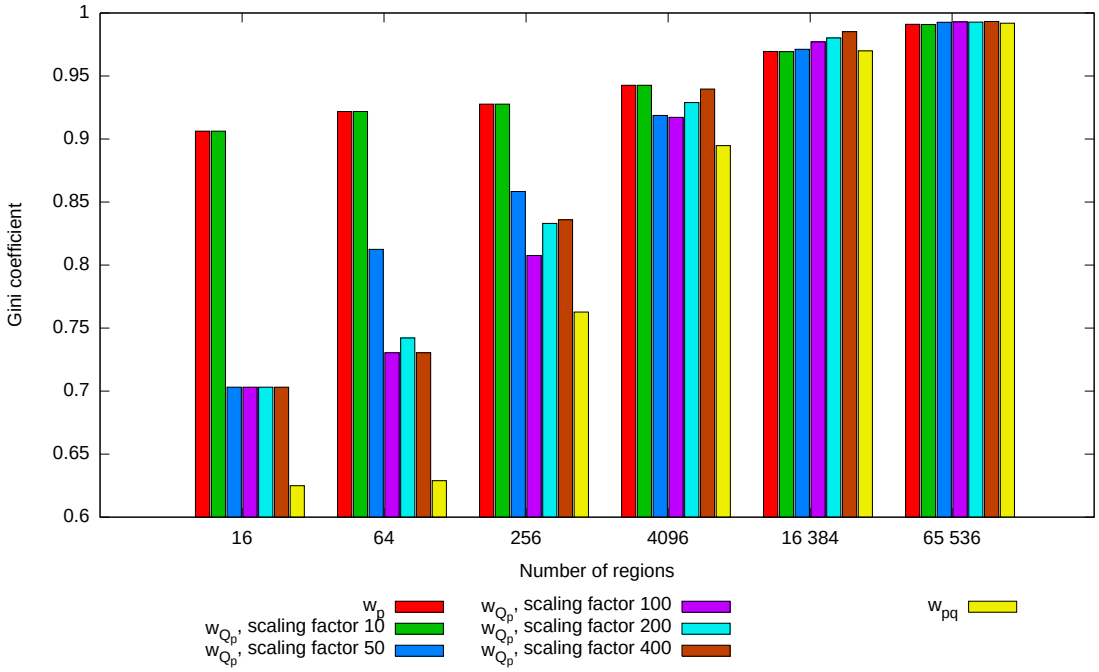


(b) Median splitting

Figure A.1: Gini coefficients for the observational data set.

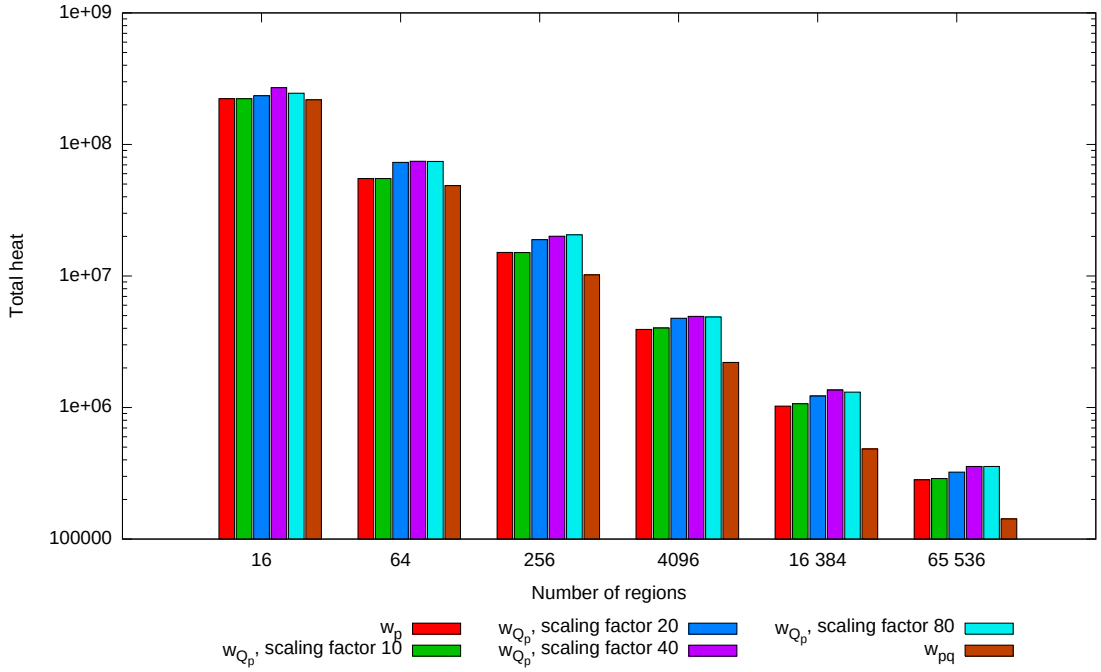


(a) Midpoint splitting

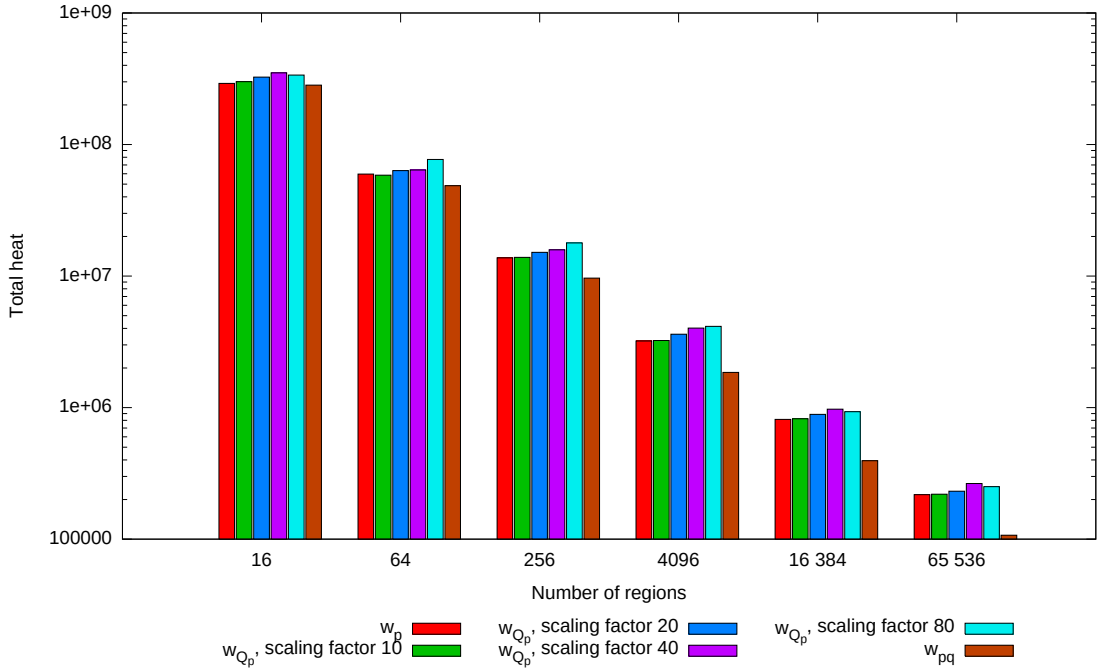


(b) Median splitting

Figure A.2: Gini coefficients for the simulation data set.

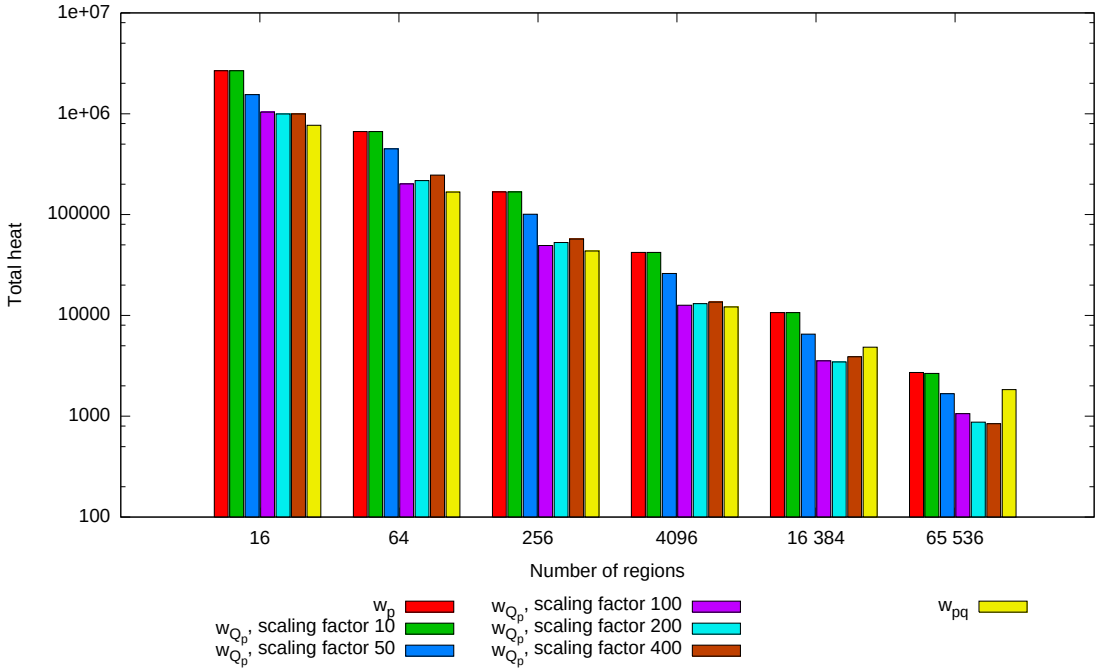


(a) Midpoint splitting

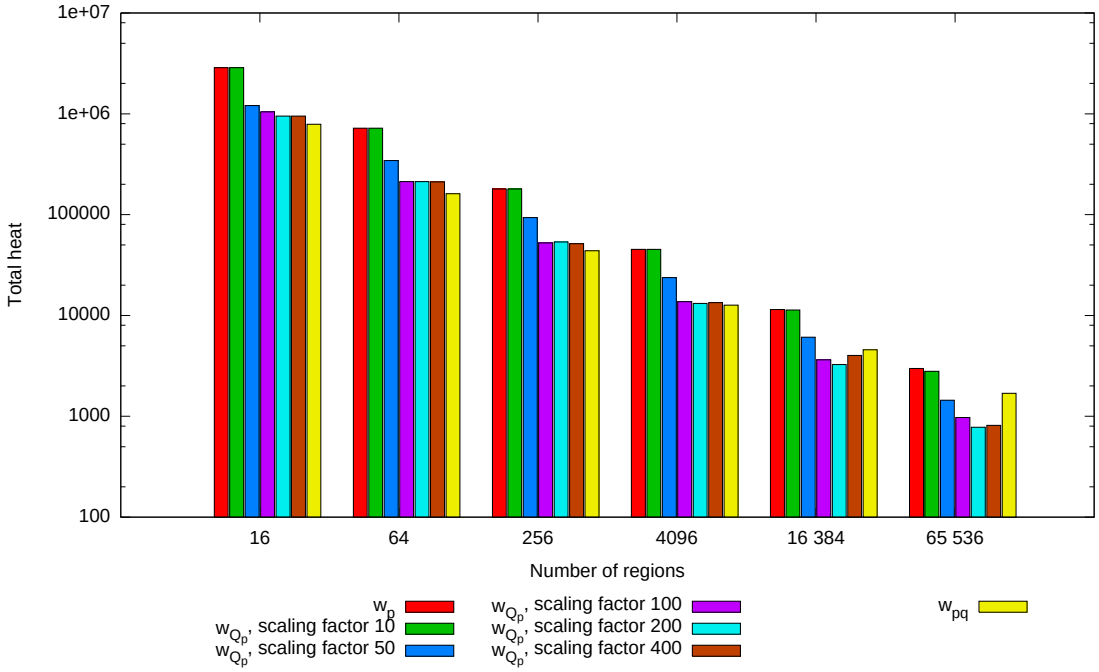


(b) Median splitting

Figure A.3: Total heat for the observational data set.



(a) Midpoint splitting



(b) Median splitting

Figure A.4: Total heat for the observational data set.

B Bibliography

[1] J. Aspnes, J. Kirsch, and A. Krishnamurthy. Load Balancing and Locality in Range-Queriable Data Structures. In *Proceedings of ACM Symposium on Principles of Distributed Computing*, pages 115–124, St. John's, Newfoundland, Canada, July 2004.

[2] J. Aspnes and G. Shah. Skip Graphs. In *Proceedings of the ACM/SIAM Symposium on Discrete Algorithms*, pages 384–393, Baltimore, MD, USA, Jan. 2003.

[3] T. Budavári, T. Malik, A. Szalay, A. Thakar, and J. Gray. SkyQuery - A Prototype Distributed Query Web Service for the Virtual Observatory. In *Astronomical Data Analysis Software and Systems XII*, volume 295, 2003.

[4] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66. Springer, 2000.

[6] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. P-Ring: An Efficient and Robust P2P Range Index Structure. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 223–234, Beijing, China, June 2007.

[7] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[8] M. J. Graham, D. Morris, and G. Rixon. VOSpace: a Prototype for Grid 2.0. In R. A. Shaw, F. Hill, and D. J. Bell, editors, *Astronomical Data Analysis Software and Systems XVI*, volume 376 of *Astronomical Society of the Pacific Conference Series*, page 567, Oct. 2007.

[9] J. Gray, A. S. Szalay, G. Fekete, W. O'Mullane, M. A. N. Santisteban, A. R. Thakar, G. Heber, and A. H. Rots. There Goes the Neighborhood: Relational Algebra for Spatial Data Search. Technical Report MSR-TR-2004-32, Microsoft Research, Microsoft Cooperation, Redmond, WA, USA, Apr. 2004.

[10] M. Ivanova, N. Nes, R. Goncalves, and M. Kersten. MonetDB/SQL Meets SkyServer: the Challenges of a Scientific Database. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, page 13, Banff, Canada, July 2007.

[11] D. Jewitt. Project Pan-STARRS and the Outer Solar System. *Earth Moon and Planets*, 92:465–476, June 2003.

[12] D. Klan, K.-U. Sattler, K. Hose, and M. Karnstedt. Decentralized managing of replication objects in massively distributed systems. In *Proc. of the Intl. Workshop on Data Management in P2P Systems*, 2008.

[13] F. Klemm, S. Girdzijauskas, J. L. Boudec, and K. Aberer. On routing in distributed hash tables. In *Proceedings of the Seventh IEEE international Conference on Peer-To-Peer Computing*, pages 113–122. IEEE Computer Society, September 2007.

[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 *Proceedings of the International Conference on Web Services*, pages 33–40, Orlando, FL, USA, July 2005.

[15] J. Orenstein and T. Merrett. A class of data structures for associative searching. In *Proceedings of the ACM SIGACT-SIGMOD Symp. on Principles of Database Sys.*, 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 *Proceedings of the International Conference on Extending Database Technology*, 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 *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, Nov. 2001.

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

[19] T. Scholl. PhD thesis, Technische Universität München. Draft.

[20] T. Scholl, B. Bauer, B. Gufler, R. Kuntschke, A. Reiser, and A. Kemper. Scalable Community-Driven Data Sharing in e-Science Grids. *Elsevier Future Generation Computer Systems*, 2008. Accepted for publication.

[21] T. Scholl, B. Bauer, B. Gufler, R. Kuntschke, D. Weber, A. Reiser, and A. Kemper. HiSbase: Histogram-based P2P Main Memory Data Management (Demonstration Proposal). In *Proceedings of the 33rd International Conference on Very Large Data Bases*, 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 *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing*, pages 195–203, Bangalore, India, December 2007.

[23] Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 173–180, Washington, DC, USA, 2005. IEEE Computer Society.

[24] 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.

[25] M. Skrutskie, R. Cutri, R. Stiening, M. Weinberg, S. Schneider, J. Carpenter, C. Beichman, R. Capps, T. Chester, J. Elias, J. Huchra, J. Liebert, C. Lonsdale, D. Monet, S. Price, P. Seitzer, T. Jarrett, J. Kirkpatrick, J. Gizis, E. Howard, T. Evans, J. Fowler, L. Fullmer, R. Hurt, R. Light, E. Kopan, K. Marsh, H. McCallon, R. Tam, S. V. Dyk, and S. Wheelock. The Two Micron All Sky Survey (2MASS). *Astrophysical Journal*, 131(1163), 2006.

[26] 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.

[27] 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 *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 570–581, Madison, WI, USA, 2002.

[28] W. Voges, B. Aschenbach, T. Boller, H. Bräuninger, U. Briel, W. Burkert, K. Dennerl, J. Englhauser, R. Gruber, F. Haberl, G. Hartner, G. Hasinger, M. Kürster, E. Pfeffermann, W. Pietsch, P. Predehl, C. Rosso, J. H. M. M. Schmitt, J. Trümper, and H. U. Zimmermann. The ROSAT all-sky survey bright source catalogue. *Astronomy and Astrophysics*, 349:389–405, Sept. 1999.

[29] X. Wang, R. Burns, and A. Terzis. Throughput-Optimized, Global-Scale Join-Processing in Scientific Federations. In *International Workshop on Networking Meets Databases*, Cambridge, UK, Apr. 2007.

[30] D. Weber. Distributed query processing for locality-aware data in P2P networks. diploma thesis, Technische Universität München, 2006.