The VLDB Journal (2009) 18:1279–1312
DOI 10.1007/s00778-009-0140-7

REGULAR PAPER

Statistical structures for Internet-scale data management

Nikos Ntarmos · Peter Triantafillou · Gerhard Weikum

Received: 21 December 2007 / Revised: 27 January 2009 / Accepted: 9 February 2009 / Published online: 5 March 2009

Abstract Efficient query processing in traditional database management systems relies on statistics on base data. For centralized systems, there is a rich body of research results on such statistics, from simple aggregates to more elaborate synopses such as sketches and histograms. For Internet-scale distributed systems, on the other hand, statistics management still poses major challenges. With the work in this paper we aim to endow peer-to-peer data management over structured overlays with the power associated with such statistical information, with emphasis on meeting the scalability challenge. To this end, we first contribute efficient, accurate, and decentralized algorithms that can compute key aggregates such as Count, CountDistinct, Sum, and Average. We show how to construct several types of histograms, such as simple Equi-Width, Average-Shifted Equi-Width, and Equi-Depth histograms. We present a full-fledged open-source implementation of these tools for distributed statistical synopses, and report on a comprehensive experimental performance evaluation, evaluating our contributions in terms of efficiency, accuracy, and scalability.

Keywords Distributed information systems · Data management over peer-to-peer data networks · Distributed data synopses

N. Ntarmos · P. Triantafillou (✉)
R.A. Computer Technology Institute and Computer Engineering and Informatics Department, University of Patras, Rio, Greece
e-mail: peter@ceid.upatras.gr

N. Ntarmos
e-mail: ntarmos@ceid.upatras.gr

G. Weikum
Max-Planck-Institut für Informatik, Saarbrücken, Germany
e-mail: weikum@mpi-inf.mpg.de

1 Introduction

In the last decade, we have witnessed a proliferation of global data management applications that involve a large number of nodes spread across the Internet. Prominent examples are:

- *Peer-to-peer (P2P) file sharing* in overlay networks like Gnutella or BitTorrent [82]. These networks have millions of users (peers) that provide storage and bandwidth for searching and fetching files, and they exhibit a high degree of churn with users joining and leaving at high rates.
- *Grid-based sharing of scientific data* [33] such as virtual observatories in astronomy or models and experiments on biochemical networks in life sciences. Here the data typically needs a much higher degree of data consistency, compared to P2P applications, and calls for advanced querying.
- *Distributed data analysis over structured records* as part of data-fusion applications over relational, XML, or RDF sources [53,84]. A grand challenge along these lines could be to perform real-time analysis of Internet-traffic data in order to combat (and ideally prevent) attacks, spam, and other anomalies [40]. This class of applications comes with a rich repertoire of query types including relational operators like join and group-by [31].

All these Internet-scale application areas have a need for *distributed aggregation queries* over a large number of network nodes which can be dynamically selected by a filter predicate. In many cases, the aggregates have a statistical nature, and can thus sometimes be estimated with sufficient accuracy and without computing the full, exact result; this paradigm is known as approximate query processing (AQP) [2,3,12,41,50]. For example, in P2P file sharing, one may be

interested in estimating the total number of distinct files that the entire network has available at a given point—a *Count-Distinct* aggregation. In e-science, the total number of X-ray spectras for triple star systems available anywhere in a Grid network could be an interesting measure, requiring a distributed *Count* operation. For the analysis of Internet-traffic logs, the total amount of bytes transferred to clients in a particular range of IP addresses can be computed by a distributed *Sum* operation. Last but not least, the advanced join-and-group queries over structured data sources typically require choosing a low-cost query execution plan, and this query optimization in turn relies on sufficiently accurate statistical estimators for the selectivity of query predicates (i.e., the cardinality of intermediate results). If the data itself is widely distributed, we thus face a problem of having to compute *histograms* and other statistical synopses over a large number of network nodes each of which holds some data fragment.

In centralized settings, the statistics management for aggregation queries, selectivity estimation, and approximate query processing has reached a fairly mature state. For distributed data, however, the issues are much less understood. And for Internet-scale, widely decentralized settings, building distributed statistical synopses and computing accurate estimates for the aforementioned kinds of queries still poses major challenges. This paper takes a first step towards addressing these challenges, and providing solutions to some of the involved issues.

1.1 Problem formulation

The general framework within which our solution is envisaged to operate is the following. We consider a networked data system of cooperating peer nodes, built over a structured P2P overlay. The network consists of possibly a large number of nodes, which collectively form the system's infrastructure. These nodes contribute and/or store data items and are thus involved in operations such as computing synopses and building histograms. In general, queries do not affect all nodes. In particular, aggregation queries compute aggregation functions over data sets dynamically determined by a filter predicate of the query. The relevant data items are stored in unpredictable ways in a subset of all nodes. Further, the networked data management system is not single-purpose: it is intended to be providing a large number of "data services" concurrently. That is, a large number of different data sets are expected to exist, stored at (perhaps overlapping) subsets of the network. And, relevant queries and synopses may be built and used over any of these data sets.

Our target design is depicted in Fig. 1. We assume operation in a structured overlay, such as either traditional or newer locality-preserving Distributed Hash Table (DHT) overlays.

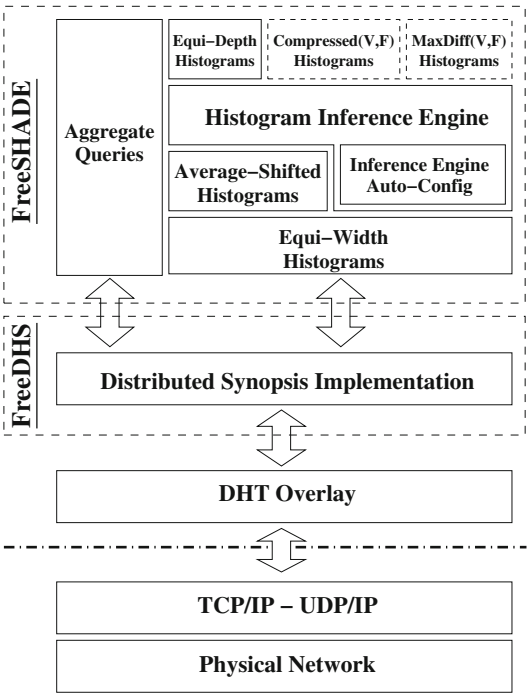


Fig. 1 Building blocks of our target system: all algorithms presented in this work, collectively coined FreeSHADE [29], are built on top of our distributed synopsis layer, FreeDHS [27,68], operating on top of a DHT, which in turn is a network overlay above the IP network

Data stored in this P2P network are assumed to be structured[1] in relations. Each such relation R consists of $(k+l)$ attributes or columns, $R(a_1, \ldots, a_k, b_1, \ldots, b_l)$; attributes a_i are used as single-attribute indices of the tuples of R, while no index information is kept for attributes b_i. Each attribute a_i is characterized by its value domain $a_i \cdot \mathscr{D} : \{a_i \cdot v_{\min}, a_i \cdot v_{\max}\}$, consisting of the minimum and maximum values of the attribute,. Every tuple t in R is uniquely identified by a primary key $t.key$. This key can be either one of the attributes of the tuple, or can be calculated otherwise (e.g. based on the values of a combination of its attributes). The set of nodes on which a data tuple t is stored, is defined in one of three ways:

1. The tuple is replicated at the nodes in the P2P overlay dictated by its key $t.key$ and by each of the $t.a_i$ attributes, using hash functions of the DHT infrastructure.
2. Alternatively, one can store just one instance of the data tuple, for example at the node responsible for $t.key$, and store pointers to this node at the other locations.

[1] The notions of structure and relations are not as strict as they are in centralized database management systems. For example, all MP3 files in the system can be thought of as belonging to a relation, since they are all annotated using a predefined set of attributes, such as "artist", "title", "album", etc. In essence, in such a setting relations are akin to "namespaces".

3. Last, there is also the scenario of only the node contributing a given tuple actually storing a complete instance of the latter, and nodes dictated by $t.key$ and $t.a_i$ storing just pointers to this one node.

In all three cases, we can efficiently identify all nodes holding tuples that match a single-attribute filter predicate on any of the indexed attributes a_i.

In [68] we introduced Distributed Hash Sketches (DHS); a distributed counting mechanism extending hash sketches [24, 26]. The solution proposed there supports efficient counting in a decentralized and load-balanced way, both in a duplicate-sensitive and a duplicate-insensitive fashion. In this work we use DHS as a building block for our solutions for decentralized aggregate query processing and histogram creation and use, while also presenting and comparing alternative designs and implementations for creating distributed synopses.

The key desiderata that an acceptable solution should satisfy are:

D1 Efficiency: the number of nodes that need to be contacted for query-answering must be small in order to enjoy small latency and bandwidth requirements;
D2 Scalability and availability, seemingly contradicting the efficiency goal: notwithstanding the goal of minimizing the number of involved nodes, in bad cases an arbitrarily large numbers of nodes may be involved (e.g., when counting with a non-selective filter or when adding elements to multiple multisets), which dictates the need for a truly decentralized solution, avoiding single-point-based scalability, bottleneck, and availability problems;
D3 Access *and* storage load balancing: query-answering and related overheads should be distributed fairly across all nodes; this should pertain to both the cost of inserting items in the overlay and the cost of disseminating data synopses to interested nodes;
D4 Accuracy: tunable, highly accurate estimation of statistical synopses, with robustness to network dynamics (churn) and failures;
D5 Ease of integration: special-purpose indexing structures, and their required extra (routing) state to be maintained by nodes, should be avoided;
D6 Duplicate (in)sensitivity when counting: the proposed solution must be able to count both the total number of items as well as the number of unique items in multisets. For example, counting the total number of distinct MP3 songs in a network where each node holds some subset of songs and popular songs are held by many nodes, or the number of all songs whose title contains some specific, popular word (e.g. love). These aggregations are query-driven and involve a variable, a priori unknown number of nodes.

These desiderata suggest a number of design decisions. First, we have chosen to build our solutions over a DHT overlay, solely using DHT primitives and the DHT paradigm. The choice of a DHT overlay is made primarily because of its scalability and efficiency in storing and locating data items of interest and also because of mature solutions for handling network dynamics [77,82]. Second, our proposed solution should require no extra structures and associated (routing) state that needs to be maintained. For example, we would be hesitant to introducing additional multicast trees into the system (other than the mechanisms that are provided by the DHT anyway), as these would require maintenance of additional routing information. Third, we wish to contribute a statistics maintenance infrastructure that should be thought of as the counterpart of (a part of) catalog information in centralized environments, in the following notion: if a query is posed in the system and there exist relevant data in the catalog, then that data will be readily used for processing and optimization of that query; otherwise, the query is executed without any optimization step, since computing the relevant catalog data is too expensive to be computed at query run-time. Fourth and last, we opt for a single DHT-based solution, to be utilized for concurrently providing many data services, answering multiple aggregation queries and building multiple histograms. This goes a long way towards reducing the complexity associated with such large networked data system infrastructures. And this, in turn, goes a long way towards understanding and maintaining the system better.

We will first leverage basic statistical structures (such as hash sketches and distributed hash sketches) to compute high precision estimates of many known aggregate queries in a P2P data system. We will further exploit these structures as building blocks in order to scalably and efficiently produce and utilize higher-level statistical structures, such as various types of histograms, for query optimization purposes. Our hope is that, by doing so, it becomes feasible to harness the wealth of research results produced for query processing and optimization in centralized and distributed database systems and port it into the P2P realm.

1.2 Contributions

Query optimization in data base management systems (DBMSs) relies on precomputed statistics and related data synopses such as histograms or sketches (see, e.g., [15,44, 58,59] and further references given there). With this work, we make the first step towards an in-depth treatment of developing and maintaining such statistical information in a decentralized fashion that is appropriate for large-scale distributed data management systems. We show how to perform decentralized aggregate query processing and how to construct, maintain, and use in a decentralized fashion several types of histograms. Our implementation and extensive performance

evaluation, on the one hand, testify that the proposed algorithms and structures enjoy efficiency, scalability, and accuracy and, on the other, help bring to the surface related trade-offs.

Our specific contributions are:

1. Algorithms for computing important aggregates, such as COUNT-DISTINCT, COUNT, SUM, and AVG.
2. Algorithms for constructing, maintaining, and utilizing several histogram types, including Equi-Width, Average-Shifted, and Equi-Depth histograms, satisfying the aforementioned design desiderata.
3. A full implementation of the above algorithms for decentralized aggregate query processing and histogram construction, use, and maintenance. Specifically, our implementation—coined FreeSHADE for "Statistics, Histograms, and Aggregates in a DHT-based Environment"—is carried out over our distributed synopsis implementation, built on top of the open-source FreePastry [28] overlay network. Our software is itself available to the community to test, validate, and use[29].
4. We contribute a comprehensive performance evaluation of our algorithms in terms of statistical estimation errors, hop-count efficiency, network bandwidth requirements, scalability, and load distribution fairness among network nodes. For comparison, we have additionally implemented a rendezvous-based solution.

We have developed an arsenal of basic building blocks (basic aggregates and histogram types) and provide fully distributed implementations for each of them, believing that more elaborate queries and statistical structures can stem from this thread of research. Our histogram-related contributions in this paper, first create distributed equi-width histograms, utilizing the DHS distinct-value estimator structure and then proceed to infer more elaborate histogram types based on the equi-width histograms. We stress that this work claims neither that creating histograms over a distinct-value estimator is the best possible approach, nor that our approach for inferring these complex histograms is the best way to do this. Rather the focus of this paper is on the DHS-based architectural principles and composability properties towards our design desiderata D1 through D6.

1.3 Outline

The rest of this paper proceeds as follows. Section 2 outlines the foundational preliminaries and related work that this paper is based on, including a query optimization primer, overviews of histograms and hash sketches, and query processing and optimization in peer-to-peer data systems. Section 3 discusses rendezvous-based approaches to computing hash sketches in a distributed manner, and leverages DHS as a fully decentralized implementation of hash sketches. Sections 4 and 5 discuss ways to compute basic aggregates in a P2P setting and methods of computing various types of histograms. Section 6 presents our implementation and experimental performance evaluation of the proposed solutions. Section 7 concludes the paper.

2 Preliminaries and related work

This work strives to bridge the gap between traditional query processing and optimization and P2P data management systems. To do so, we leverage tools from both the peer-to-peer and centralized world. In this section we present a short overview of traditional and P2P query processing and optimization and related statistical structures and techniques.

2.1 Query optimization primer

DBMS's depend on statistics for efficient query processing, typically encompassing various data aggregates, sketches of base data, and histograms. Such statistics are used in many ways throughout the lifetime of a query, either as part of the query optimizer logic—for example, to optimize the access paths for single-relation queries [79] and to calculate the selectivity of predicates [21,46,73,74] or the optimal order of predicate evaluation in multi-predicate queries [45,72]—or as high-quality samples of the base data in approximate query answering systems [2,3,71]. Even if exact answers are sought, such quick approximate answers may be desirable as a feedback to the user prior to execution of a large and time/resource-consuming query [41].

System R [79] was among the first to use cost-based optimization. It maintained in the system catalogs such statistics as the number of tuples and data pages per relation, the number of data pages and distinct values per index, and the ratio of data pages per segment that hold information for any given relation. The cost of candidate access paths was then estimated using a set of formulas with static factors and the above statistics as input. Later on, the industry turned to histograms as a more accurate and compact way of summarizing information about stored data. [44] and [74] give a taxonomy and a brief history of the evolution of histograms, while [52] presents a survey of distributed query processing in the pre-P2P era.

Chaudhuri [11] identified the following information as necessary for an optimizer to reach an informed decision: (1) number of tuples in a relation, (2) number of physical pages used by a table, (3) statistical information for table columns, in the form of histograms, minimum and maximum, or second-lowest and second-highest, values, and number of distinct values in the column, and (4) information on the correlations among attribute values, in the form of either

multi-dimensional histograms (having the disadvantage of growing very big with the number of dimensions) or a single-dimensional histogram on the leading column, plus the total count of distinct combinations of column values present in the data, for multi-column indices. In the following sections we shall discuss techniques to compute such statistics in a P2P setting.

Sketches, histograms, aggregates, and data synopses in general, have found uses in many other fields of computer science. Research on data streams [17,18,35,66] has turned to such synopses of base data to alleviate large data transfers and ease storage and processing requirements on stream processors. Histograms and sketches have also been proposed as a means of decreasing the amount of data transmitted—and, consequently, the amount of power consumed—by nodes of sensor networks [16,85]. Along the same line of thought, distributed systems such as publish/subscribe systems [10, 87], distributed web proxy caches [25], and peer-to-peer web search engines [55,61,62] have all used statistical synopses to quickly and efficiently exchange information about data stored on the various nodes in the system.

2.2 Histograms

Histograms are by far the most common technique used by commercial databases as a statistical summary and an approximation of the distribution of values in base relations. For a given attribute/column, a histogram is a grouping of attribute values into "buckets" whose collective frequency is approximated by statistics maintained in each such bucket.

All histograms make some basic assumptions concerning the distribution of items in each bucket. One of the most prominent such assumptions is the *uniform spread assumption* [74]. According to it, values in a bucket are assumed to exist only at the points of equal spread (equal to the bucket average), and have a frequency equal to the ratio of the frequency of the cell over the number of distinct values in it. In order to calculate this, one needs to store the number of distinct attribute values along with the minimum and maximum values per bucket.

The most basic histogram variant—the Equi-Width histogram—partitions the attribute value domain into cells (buckets) of equal spread and assigns to each the number of tuples with an attribute value within the cell's boundaries. Relevant research has focused on improving the statistical properties of histograms, with many interesting results. [74] present a taxonomy of histograms along with several novel histogram types, and take an extensive look into their accuracy and efficiency with regard to both construction time and storage space requirements. In the following sections, we shall discuss tools, protocols, and techniques to build and compute several types of histograms. Namely, apart from plain Equi-Width histograms, we deal with Average Shifted Equi-Width and Equi-Depth histograms, described shortly.

"Average Shifted Histograms" or ASHs [78] pose an interesting alternative to simple Equi-Width histograms. An ASH consists of a set of Equi-Width histograms with the same number of buckets and the same bucket spread but different starting points. The frequency of each value in a bucket is then computed as the average of the estimations given by each of these histograms. ASH—in essence a kernel estimator—can provide a much smoother approximation of the actual distribution of tuple values compared to plain Equi-Width histograms.

Equi-depth histograms [67,80] have been widely used in commercial database management systems. They consist of a partitioning of the attribute value domain in disjoint intervals, such that the number of data tuples whose attribute value falls in each such interval is (almost) equal among all intervals. That is, in an Equi-Depth histogram all buckets have equal frequencies but not (necessarily) equal spreads.

2.3 Hash sketches

Hash sketches were first proposed by Flajolet and Martin in [26] under the name of *Probabilistic Counting with Stochastic Averaging* or *PCSA*, as a means of estimating the number of distinct items in a multiset $\mathscr{D}$ of data in a database,[2] building on the counting algorithm of [65]. The estimate obtained is (virtually) unbiased, while the authors also provide upper bounds on its standard deviation. The only assumption underlying hash sketches is the existence of a pseudo-uniform hash function $h() : \mathscr{D} \rightarrow [0, 1, \ldots, 2^L)$—an assumption also present in most (if not all) P2P-related research. Durand and Flajolet presented a similar algorithm [24], coined *super-LogLog counting*, which reduced the space complexity and relaxed the assumptions on the statistical properties of the hash function of [26].[3] Hash sketches have been used in many application domains where counting distinct elements in multi-sets is of some importance, such as approximate query answering in very large databases [54], data mining on the Internet graph [69], and stream processing [22,30]. The selection of such sketches is dictated by the fact that related data synopsis techniques relying on a global ordering of (hashed) data—such as, for example, count-min sketches [17] and the distinct value estimators of [7]— cannot be implemented in a completely decentralized manner following the DHT paradigm, without resorting to such techniques as gossiping, multi-broadcasting, or a rendezvous-based approach, all of which have undesirable properties in our setting as we shall see shortly.

[2] For a survey of distinct-value estimators, see [7].

[3] The analysis leading to the equations used in this section is well beyond the scope of this paper and can be found in [24,26].

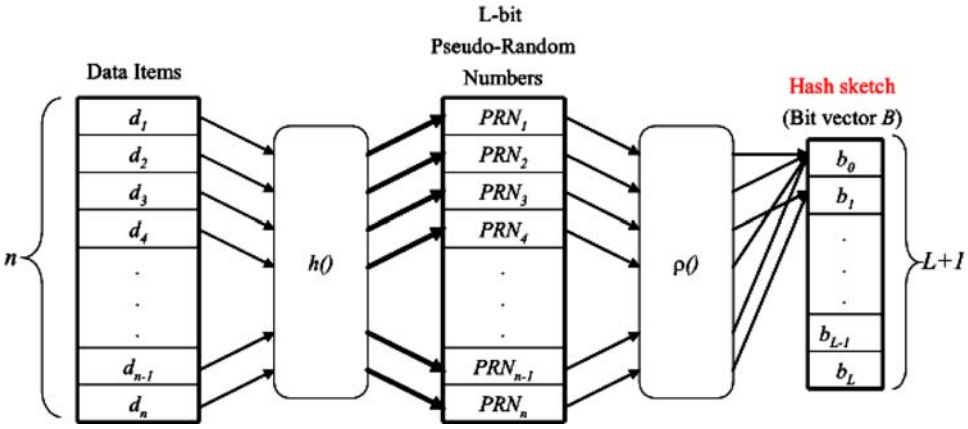


Fig. 2 Inserting items into a hash sketch: single bitmap case

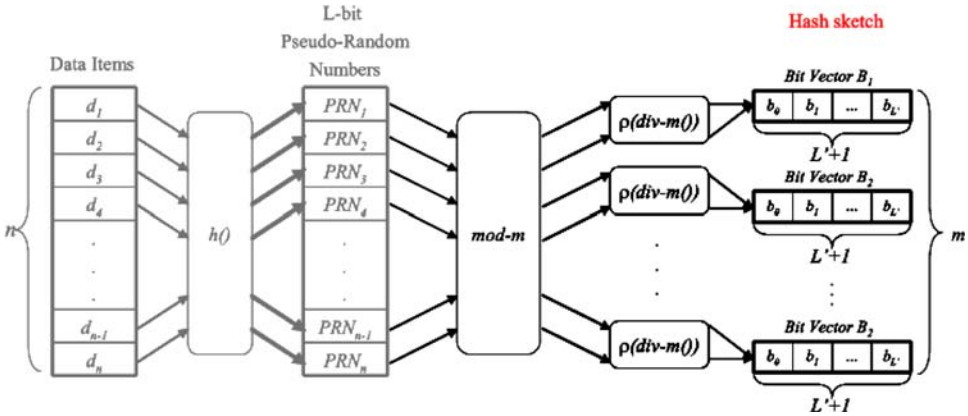


Fig. 3 Inserting items into a hash sketch: multiple bitmaps case

A hash sketch consists of a bit vector $B[\cdot]$ of length L, with all bits initially set to 0, and a hash function $h()$ as above. Let $\rho(y) : [0, 2^L) \rightarrow [0, L)$ be the position of the least significant (leftmost) 1-bit in the binary representation of y; that is, $\rho(y) = \{\min(k \geq 0) : bit(y, k) \neq 0\}$, $y > 0$, and $\rho(0) = L$, where $bit(y, k)$ denotes the kth bit in the binary representation of y (bit-position 0 corresponds to the least significant bit). In order to estimate the number n of distinct elements in a multiset $\mathscr{D}$ we apply $\rho(h(d))$ to all $d \in \mathscr{D}$ and record the results in the bitmap vector $B[0 \ldots L-1]$ (see Fig. 2). Since $h()$ distributes values uniformly over $[0,\ 2^L)$, it follows that $P(\rho(h(d)) = k) = 2^{-k-1}$. Thus, when counting elements in an n-item multiset, $B[0]$ will be set to 1 approximately $\frac{n}{2}$ times, $B[1]$ approximately $\frac{n}{4}$ times, etc. This fact is rather intuitive: imagine all n possible L-bit numbers; the least significant bit (bit 0) will be 1 for half of them (odd numbers); of the remaining $\frac{n}{2}$ numbers, half will have bit 1 set, or $\frac{n}{4}$ overall, and so on.

Then, the quantity $R(\mathscr{D}) = \max_{d \in \mathscr{D}} \rho(h(d))$ provides an estimation of the value of $\log(n)$, with an additive bias of 1.33 and a standard deviation of 1.87. Thus, 2^R estimates "logarithmically" n within 1.87 binary orders of magnitude. However, the expectation of 2^R is infinite and thus cannot be used to estimate n. To this extent, [24] propose the following technique (similar to the *stochastic averaging* technique in [26]): (1) use a set of $m = 2^c$ different $B^{\langle i \rangle}[\cdot]$ vectors (c being a non-negative integer), each resulting to a different $R^{\langle i \rangle}$ estimate, (2) for each element d, select one of these using the first c bits of $h(d)$, and (3) update the selected vector and compute $R^{\langle i \rangle}$ using the remaining bits of $h(d)$ (see Fig. 3).

If $M^{\langle i \rangle}$ is the (random) value of the parameter R for vector i, then the arithmetic mean $\frac{1}{m} \sum_{i=1}^{m} M^{\langle i \rangle}$ is expected to approximate $\log(\frac{n}{m})$ plus an additive bias. The estimate of n is then computed by the formula: $E(n) = \alpha_m \cdot m \cdot 2^{\frac{1}{m} \cdot \sum_{i=1}^{m} M^{\langle i \rangle}}$, where $\alpha_m = (-m \cdot \frac{2^{-\frac{1}{m}} - 1}{\log(2)} \cdot \int_0^\infty e^{-t} \cdot t^{-\frac{1}{m}} dt)^{-m}$([24]). The authors further propose a *truncation rule*, consisting of taking into account only the $m_0 = \lfloor \theta_0 \cdot m \rfloor$ smallest M values. θ_0 is a real number between 0 and 1, with $\theta_0 = 0.7$ producing near-optimal results. With this modification, the estimate formula becomes: $E(n) = \tilde{\alpha}_m \cdot m_0 \cdot 2^{\frac{1}{m_0} \cdot \sum^* M^{\langle i \rangle}}$, where $\sum^*$ indicates the truncated sum, and the modified constant $\tilde{\alpha}_m$ ensures that the estimate remains unbiased (see Fig. 4). The resulting estimate has a standard deviation of $\frac{1.05}{\sqrt{m}}$, while the hash function must have a length of at least $H_0 = \log(m) + \lceil \log(\left(\frac{n_{\max}}{m}\right)) + 3 \rceil$, $n_{\max}$ being the maximum cardinality estimated.

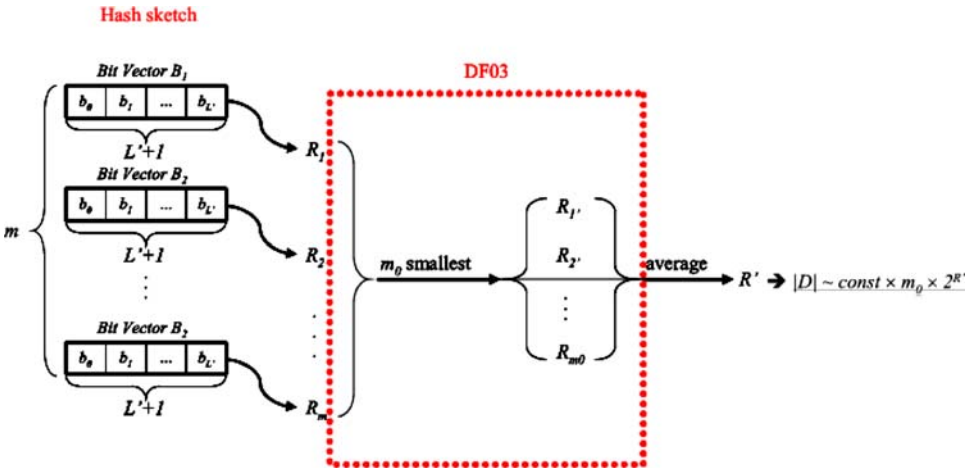


Fig. 4 Counting items using a hash sketch

PCSA counting. The algorithm in [26] is based on the same hashing scheme (i.e. using $\rho(\cdot)$) and the same observations as [24]. The PCSA algorithm differs from the superLogLog algorithm in the following: (1) [26] rely on the existence of an explicit family of hash functions exhibiting ideal random properties, while [24] have relaxed this assumption, (2) [26] set R to be the position of the leftmost 0-bit in the bitmap $B[\cdot]$, as opposed to the position of the rightmost 1-bit in the bitmap with [24], (3) [24] use in the order of $\log\log(\textit{max cardinality})$ bits per bitmap while [26] need in the order of $\log(\textit{max cardinality})$ bits per bitmap, (4) the estimation in [26] is computed as:

$$E(n) = \frac{1}{0.77351} \cdot m \cdot 2^{\frac{1}{m}\sum_0^{m-1} M^{\langle i \rangle}}.$$

and (5) the bias and standard error of [26] are closely approximated by $1 + 0.31/m$ and $0.78/\sqrt{m}$ respectively. Note that the data insertion algorithm is the same for both [24] and [26] (with the sole difference of the assumptions on the hash function).

Hash sketches exhibit a natural distributivity; the hash sketch of the union of any number of sets can be computed from the hash sketches of these sets by a bitwise OR of the corresponding bit vectors, given that all of them have the same number of bit vectors and length and that they have been built using the same set of hash functions. Thus, if an initial set A is spread across several hosts (e.g. across a P2P network), one can compute the global hash sketch for A from each of the locally computed hash sketches corresponding to the subset of A that each peer is responsible for.

2.4 Summation sketches

Considine et al. [16] have introduced the notion of "summation sketches", building on hash sketches [24,26]. Assume again we have a multiset $\mathscr{D} = \{d_1, d_2, d_3, \ldots\}$ of data items $d_i = (k_i, v_i)$, each identified by a key k_i and bearing a value v_i. Then, the distinct summation problem consists of computing the quantity:

$$\mathscr{S} = \sum_{\text{distinct}((k_i, v_i) \in \mathscr{D})} v_i.$$

The basic idea is to model data item values as a series of item insertions. That is, in order to estimate the sum of the values of two distinct data items $d_1 = (k_1, v_1)$ and $d_2 = (k_2, v_2)$, with $k_1 \neq k_2$, the algorithm in [16] proceeds as follows: first, (1) insert v_1 distinct items in a first hash sketch; then (2) insert v_2 distinct items in a second hash sketch; (3) take the bitwise OR of the resulting sketches; finally, (4) use the standard PCSA or superLogLog estimator on the combined sketch to compute the actual sum result. This technique takes $O(v_i)$ expected time to add a data item $d_i = (k_i, v_i)$ to the summation sketch. Obviously, this does not scale well for large values of v_i. [16] address this by *emulating* the v_i insertions. The proposed method consists of two steps:

1. First, set the lowest $\delta_i = \lfloor \log(v_i) - 2\log\log(v_i) \rfloor$ of the summation sketch bits to all ones, since these bits are all set to one with high probability after v_i insertions (see the proof of Theorem 2 in [26]).
2. Simulate the insertions that set bits δ_i and higher in the hash sketch.

An item d_i sets a given bit position $p \geq \delta_i$ if and only if $\forall_{0 \leq j < p}(bit(h(d_i), j) = 0)$, which happens with probability 2^{-p}. This means that, for a set of v_i insertions, the number of insertions setting bits above a position p follows a binomial distribution with parameters v_i and $2^{-\delta_i}$. Thus, in order to add an item $d_i = (k_i, v_i)$, the authors advocate first to draw a random sample y from $B(v_i, 2^{-\delta_i})$, consider each of these y insertions as having reached bit δ_i, then use the classic hash sketch insertion process to set the remaining bits beyond δ_i.

In order to take advantage of multiple bitmaps in the hash sketch without losing in accuracy, the above algorithm is modified as follows. First, each value v_i is transformed to $q_i \cdot m + r_i$, for some integer q_i and r_i, with $0 \leq r_i < m$. Then, in order to add such an item to the summation sketch, [16] first add r_i distinct items once, as in standard PCSA, and then add q_i to each bitmap independently. The time cost of this insertion algorithm is in $O(m \log^2(\frac{v_i}{m}))$, while the result has the same precision guarantees as that of the standard PCSA and/or superLogLog counting.

2.5 Statistics in P2P systems for aggregate queries and histograms

The peer-to-peer research corpus has already begun to investigate ways of providing DBMS functionality over P2P data networks [34]. Most prominent such systems, mainly targeted by this work, are built on top of Distributed Hash Tables (DHTs). Distributed Hash Tables are a family of structured peer-to-peer network overlays exposing a hash-table-like interface. The main advantage of DHTs over unstructured P2P networks, lies in the strict theoretical probabilistic (in the presence of node failures and network dynamics) performance guarantees offered by the former. Prominent examples of traditional DHTs include Pastry[23], CAN [75], Chord [83], Kademlia [60], etc.

DHTs offer two basic primitives: *insert(key, value)* and *lookup(key)*. Nodes are assigned unique identifiers from a circular ID space—usually computed as the hash (SHA-1, MD5, etc.) of their IP address and port number on which the P2P application is operating—and arranged according to a predefined geometry and distance function[36]. This results in a partitioning of the node-ID space among nodes, so that each node is responsible for a well-defined set (arc) of identifiers. Each item is also assigned a unique identifier from the same ID space—usually by simply feeding the item to the same cryptographic hash function used to generate the node IDs—and is stored at the node whose ID is closest to the item's ID, according to the DHT's distance function. Each node in an N-node DHT maintains direct IP links (aka fingers) to $O(\log(N))$ other nodes in appropriate positions in the overlay, as dictated by the DHT's geometry, so that routing between any two nodes takes $O(\log(N))$ hops in the worst-case.[4]

In general, due to the pseudo-random output of cryptographic hash functions, each DHT node will be responsible for storing a (possibly random) subset of the attribute values—and hence (pointers to) tuples—of each relation. This holds regardless of the specific DHT employed and has grave implications on the efficiency of several query types. For example, aggregate or range queries may introduce a messaging overhead that is in $O(N)$ in an N-node network. It is this challenge that this paper endeavors to meet in an effort to make efficient and scalable peer-to-peer query processing and optimization feasible.

Distributed counting/aggregation solutions proposed by the peer-to-peer research corpus so far, can be categorized in one of the following groups:

1. Rendezvous-based protocols,
2. Gossip-based protocols,
3. Broadcast/convergecast-type protocols, also known as aggregation-tree approaches,
4. Sampling-based protocols.

Rendezvous-based protocols. The first type of solution is also the first that comes to mind when using a structured overlay (DHT): select a node in the overlay (e.g. by using the hash function(s) of the DHT overlay) and use it to maintain the aggregate value (e.g. see the distributed counting mechanism outlined in [26]). Hash-partitioned counters—where the counting space is partitioned into disjoint intervals, with each such interval mapped to a (set of) node(s) in the overlay—or "coordinator"-based solutions (abundant in sensor networks and distributed data stream processing [18, 39])—where summaries of data are gathered at a central aggregation point to be processed—also fall in this category. Solutions of this type suffer from potential shortcomings regarding our design desiderata D1 through D6. Having a central aggregation node means that this node will be contacted on every update of, and on every query for, the current value of the aggregate, potentially violating (D2). Moreover, each of these central aggregation nodes withstands a high access and storage load, violating (D3), while one can argue that such highly loaded nodes will exhibit high response times, also violating (D1). Using a (fixed) number of rendezvous nodes does not really solve the problem but merely mitigates the scalability issues to the cost of inserting items to and/or querying the value of such an aggregate, as these grow linearly to the number of rendezvous nodes engaged in the computation, while also violating (D1). Similar arguments hold for the case when multiple rendezvous nodes have to be contacted, as the result of simultaneously maintaining multiple aggregates (i.e. counting multiple quantities at the same time). Despite the theoretical and practical advantages of the other types of solutions to be discussed shortly, rendezvous-based solutions are by far the most popular in real-world implementations, mainly due to their simplicity and excellent hop-count performance in the single aggregate case; for example, IETF's Service Location Protocol,[5]

[4] All $\log(\cdot)$ notation refers to base-2 logarithms.

[5] http://www.ietf.org/rfc/rfc2608.txt.

Skype,[6] iTunes,[7] TiVo,[8] the Asterisk PBX,[9] as well as any protocol relying on super-peers to function, all use some sort of rendezvous-based protocol to bring together resources and coordinate access to them. For this reason, we have chosen to compare our proposal against variants of the rendezvous-based solution.

Gossip-based protocols. The second type of solutions, based on gossiping, (e.g. [4,48,49,51,63,64]) usually provide probabilistic semantics of "eventual consistency" for their outcome; gossip-based protocols are based on an iterative procedure, according to which every node exchanges information with a (set of) its neighboring node(s) on every iteration. Eventual consistency means that, in the presence of failures and dynamicity in the P2P overlay, the algorithm will eventually converge to a stable state after the overlay has itself stabilized. Although the bandwidth requirements of these approaches are low when amortized over all nodes, the overall bandwidth consumption and hop-count are usually very high. Moreover, the fact that all nodes have to actively participate in a gossip-based computation, even if it is of no interest to them, coupled with the multi-round property of these solutions violates (D1) and (D2), while their semantics violates (D4). Of course, in unstructured overlays it is not clear if one can do better than this anyway. All in all, gossiping solutions for aggregation are completely decentralized; however, they are best suited for an environment where simple aggregations can take place and small amounts of extra data can be passed along (piggybacked) with regular messaging behavior between neighbors. In a DHT-based infrastructure, on the other hand, approaches like the one advocated in this paper are much more advantageous.

Aggregation-tree-based protocols. The third type of solutions [5,6,16,56,81,85,88,89] is based on a two-round procedure: (1) a broadcast phase, during which the querying node broadcasts a query through the network, creating a (virtual) tree of nodes as the query propagates in the overlay; and (2) a convergecast phase, during which each node sends its local part of the answer, along with answers received from nodes deeper down the tree, to its "parent" node. Solutions that are based on pre-built tree structures also belong in this group. Of these works, Astrolabe [88] was among the first to talk of aggregation in the peer-to-peer landscape; the authors proposed the creation and maintenance of a hierarchical, tree-like overlay, used to propagate complex queries and their results through the peer-to-peer overlay. A similar idea has been proposed in [89]. Bawa et al. [5] propose building a (set of) multicast overlay tree(s) to propagate queries and results back and forth, while using flood-like methods to send messages around the network. Although these structures have nice properties and are capable of computing aggregates in a wide scale, they are not directly applicable for the creation and maintenance of multiple simultaneous counters/aggregates. First, they require the creation and maintenance of a separate (possibly extra) network overlay, thus violating the desired property (D5). Second, similarly to rendezvous-based approaches, the cost of maintaining multiple counters at the same time grows linearly with the number of such counters (e.g. when having to maintain a different tree per counter). Furthermore, even with just one counter, if the number of nodes containing items to be counted is in $O(N)$, then the counting cost (total hop count and number of messages) is also in $O(N)$, thus violating (D1). Moreover, even if the load is balanced, functionality is not; although all intermediate nodes communicate with the same number of neighbors (that is, if the tree is full), nodes closer to the root of the tree are more "important" than leaf nodes, thus violating (D2) and (D3). The time for computing an aggregate is in $O(\log N)$ if all queries go out in parallel across links between parent and children nodes (that is, if the tree is balanced), but this statement obscures the fact that the total number of messages is in $O(N)$. In brief, our solution can also do $O(\log N)$ if we send out queries in parallel. However, the total number of messages is a better metric as far as resource consumption (e.g., network bandwidth and per-node processors) and scalability are concerned; in this regard, our solutions do much better than the $O(N)$ performance of aggregation trees.

Sampling-based protocols. The core idea of the last type of solutions [8,57] is to estimate the value of the counter in question, by selectively querying (sampling) a set of nodes in the network. Bharambe et al. [8] attempt to compute approximate histograms of system statistics by using random sampling of nodes in the network. Manku [57] estimate the number of nodes in the overlay by also using a random sampling algorithm. First, there is no obvious way to generalize these techniques to count arbitrary quantities (other than the ones they were designed for). Second, with data tuples arbitrarily replicated across the network overlay and with a possibly highly skewed tuple popularity distribution, simplistic random selection schemes will lead to highly biased samples of the actual dataset. This is caused by the fact that sampling-based techniques are known to suffer when duplicates exist in the base data [14,38], thus conflicting with both desiderata (D4) and (D6). On the other hand, if the sample is big enough to guarantee a certain level of confidence, then these solutions may fall short of satisfying (D1). Even if distributed sampling was practical, arguments similar to those presented earlier for rendezvous and tree-based approaches hold for the reuse of sampled data for future queries. In general, we discern

[6] http://www.skype.com/.

[7] http://www.apple.com/itunes/.

[8] http://www.tivo.com/.

[9] http://www.asterisk.org.

two categories: pre-computed synopses-based versus online approaches. To our knowledge, with the exception of our DHS-based approach, there has not been a method for storing or utilizing such pre-computed synopses in a decentralized manner in distributed environments. In Sect. 5, we will adopt a two-step procedure to create and maintain complex histograms. Keen readers may suggest borrowing algorithms and constructions from online histograms [32,86]. However, online histogram algorithms rely heavily on sampling and are thus impractical for the reasons mentioned above.

In summary, each of the above families of protocols has its specific strengths and weaknesses. By and large, the strengths are more congruent with unstructured overlay networks, whereas this paper focuses on DHT-based structured overlays. Moreover, as we anticipate the need to compute aggregations and other statistical estimates in combination with filter predicates of the posed queries, we demand that we can efficiently identify the subset of network nodes that hold the base statistics for such dynamically restricted aggregations. None of the above four approaches is well suited for this requirement. Hybrid methods that combine different elements of several of the above paradigms or modifications of these methods are conceivable as well, but would entail new research and thus fall outside the scope of the current paper.

3 Distributing data synopses

In the following sections, we will present various distributed algorithms and protocols for computing aggregates and histograms, to be used in distributed query optimization. The common denominator of all these, is the need for a distributed synopsis infrastructure. Assume we wish to compute a hash sketch for a set A distributed across a peer-to-peer data network. We identify two major directions of attacking this problem: (1) the "conservative" but popular rendezvous based approach, and (2) the completely decentralized and highly scalable way of DHS, in which no node has some sort of special functionality.

3.1 The rendezvous approach

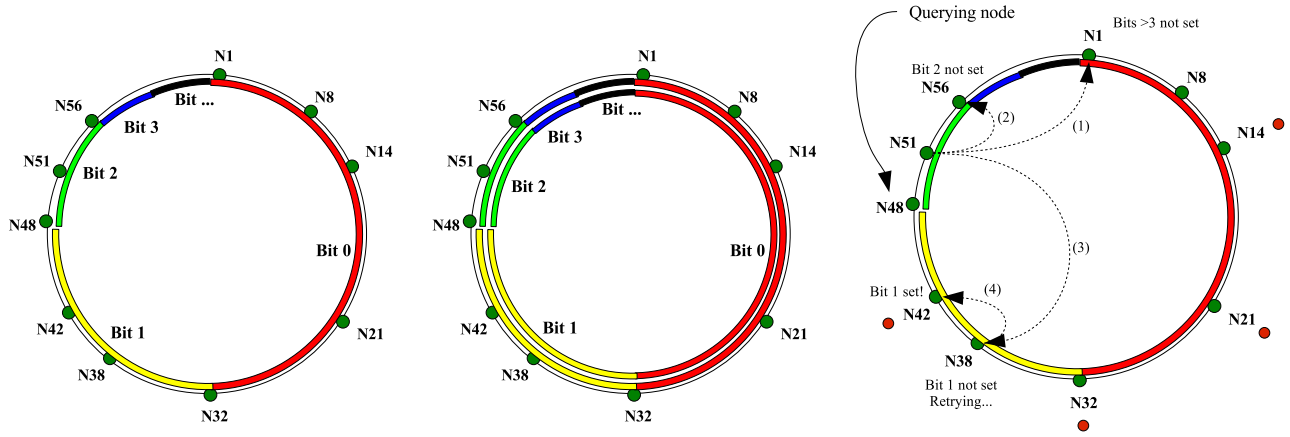
The rendezvous approach is what one would call the natural evolution of client–server architectures in the distributed world of peer-to-peer networks. Nodes storing items belonging to the set A first compute a rendezvous ID (for example, by feeding "A" to the underlying DHT's base hash function). Then, they compute locally the synopsis of choice and send the outcome to the node whose ID is closest to the above ID (called the "rendezvous node"). The rendezvous node is responsible for combining the individual synopses (by bitwise OR) into the global synopsis for A. Interested nodes can then acquire the global synopsis for A by querying the rendezvous node. As is obvious, the message cost for a node to "insert" an item to this distributed synopsis, as well as the cost for a node to acquire the global synopsis, is in $O(\log(N))$.

Although clean, simple, and highly efficient in terms of hop count, this solution suffers from two major scalability issues. First, the rendezvous node is burdened with disproportionally higher communication cost than the rest of the nodes. In the worst case that all (or a large portion of) nodes in the system store items from A—or a large portion of nodes wish to acquire the global synopsis— then the rendezvous node will have to withstand $O(N)$ incoming connections and the corresponding bandwidth overhead.

The "easy way out" is to use multiple rendezvous nodes per distributed synopsis so as to spread the load among them. This can happen in any of two ways: either (1) nodes spread their items across rendezvous nodes on insertion and maintenance time (proactive replication), but pay the extra hop-count cost of having to visit every one of them to reconstruct the complete synopsis, or (2) they always update all rendezvous nodes on insertion, thus paying this extra cost during insertions, but only need to visit a single rendezvous node to acquire the complete synopsis. Second, along the same lines of the above observation, in order to compute the global synopses for multiple sets, one has to contact as many rendezvous nodes as are the sets. That is, the overall message cost for computing multiple synopses grows linearly to the number of synopses. However, for the sake of completeness, we shall not drop this approach, since it appears to be quite popular in the relevant literature.

3.2 Distributed hash sketches

In [68], we presented a DHT-based implementation of hash sketches, coined Distributed Hash Sketches (or DHS). DHTs already feature a pseudo-uniform hash function; node and document IDs are (usually) computed as either the secure hash of some object-specific piece of information [23,83] (e.g. the IP address and port of nodes, the content for files, etc.), or as the outcome of a pseudo-uniform random number generator [60]. In both cases, the resulting ID is an L-bit pseudo-uniform number (for some fixed, system-specific L), thus satisfying the main assumption of hash sketches. We denote by $k \leq L$ the length of the DHS bitmap vectors and assume that items are added to the DHS using the k lower-order bits of their corresponding DHT keys. In [68] we discussed techniques and protocols to implement both PCSA and superLogLog counting within DHS. In this work we use only the latter for clarity of presentation but we would like to note that the solutions discussed in this work are also applicable to the PCSA-based DHS. We'll first discuss the case with a single bitmap vector and a single



(a) Mapping of bit positions to nodes – Single bitmap

(b) Mapping of bit positions to nodes – Multiple bitmaps

(c) Counting items using a DHS

Fig. 5 Distributed hash sketches

estimated quantity, extending our design for multiple vectors and multiple quantities later.

3.2.1 Mapping DHS bits to DHT nodes

The core observation is that higher bit positions in the hash sketch bitmaps are set with exponentially decreasing frequency. We thus proposed to also partition the node ID space into consecutive disjoint regions of exponentially decreasing size, and to associate every bit position with the region whose size corresponds to the bit's frequency. That is, we partition the node ID space, $[0, 2^L)$, into k consecutive, non-overlapping intervals $\mathscr{I}_r = [thr(r), thr(r-1)), r \in [0, k)$, where: $thr(r) = 2^{L-r-1}$. Using this partitioning, bit r of $B[\cdot]$ is mapped to node IDs randomly (uniformly) chosen from $\mathscr{I}_r$ (bit k is mapped to the interval $[0, thr(k-1))$). Remember that when counting distinct items in an n-object multiset, bit r of the bitmap vector is "visited" $n \cdot 2^{-r-1}$ times. With the k-bit IDs used in DHS, this translates to a maximum of 2^k distinct objects in any possible multiset, or to a maximum of 2^{k-r-1} objects being mapped to position r in the bitmap vector. Now, note that intervals $\mathscr{I}_r$ have exponentially decreasing sizes $|\mathscr{I}_r| = 2^{L-r-1}$. The above result in a distribution of information across all nodes in the network, as uniform as the hash function used. With this partitioning scheme, bit position 0 is mapped to the first half of the ID space, bit position 1 is mapped to the next quarter of the ID space, bit 2 to the next eighth, and so on (Fig. 5a). The mapping of bit positions to ID-space regions is such that the bit position to region correspondence is the same for all bitmaps of the distributed hash sketch—that is, different bitmaps coincide in the ID space (see Fig. 5b).

3.2.2 DHS data insertion

Assume a node wishes to add an item to the DHS. First it inserts the item to a temporary local hash sketch instance. This operation consists of taking the item's ID as produced by the DHT's base hash function, applying the $\rho(\cdot)$ function, and setting the corresponding bit in the local sketch. For an object o with ID $o.id$, we first compute $r = \rho(lsb_k(o.id))$, where $lsb_k(\cdot)$ returns the k lower-order bits of its argument; then we select a random ID in the interval $[thr(r), thr(r-1))$ in the ID space, corresponding to bit position r set during the previous step, and send a "set-to-1" message to the node responsible for that ID. This in essence means that the mapping of bits to nodes within the bit's specified region is done in a randomized fashion: every time an operation (set a bit to 1, check the current bit's value, etc.) is to be performed on a bit, a random ID is chosen uniformly from the corresponding ID space region and the operation is carried out by the node responsible for that ID. The DHT overlay guarantees, with high probability, that the message will reach the target node in $O(\log(N))$ hops in the worst case. Each DHS tuple is of the form $< metric\_id, bit, time\_out >$, where $metric\_id$ is an identifier uniquely identifying the metric[10] to be estimated, $bit = r$ denotes the position in the distributed vector of the bit that is to be set, and $time\_out$ defines a time-to-live interval for the current tuple, reset at every updates of the tuple, allowing for aging out of DHS entries.

When multiple (m) bitmaps are used, the item insertion is performed in the exact same manner as in the single-bitmap case, only now selecting one out of m vectors using

[10] For the time being, we will assume that there is only one metric estimated in the overlay; counting multiple metrics at once (also called "multi-dimensional counting") will be discussed later.

$lsb_k(o.id)\ mod\ m$, and then using $r = \rho(lsb_k(o.id)\ div\ m)$ as the position of the bit to be set. The DHS tuple data must now be extended to $< metric\_id, vector\_id, bit, time\_out >$, where $vector\_id$ is the ID of the vector being updated. The worst-case hop-count cost for a node to insert an item in such a DHS is in $O(\log N)$—that is, *independent of the number of bitmaps*—since each insertion only touches a single bitmap.

When multiple items are to be inserted, nodes can either iterate over the itemset repeating the above algorithm one item at a time, or they can first insert all these items to a local hash sketch and then send a "set-to-1" message to only one node per bit position/ID-space region. The latter operation is carried out by (1) first constructing a message containing the bits to be set, then (2) sending it to a random node in the region corresponding to the least-significant set bit in the message, and (3) iterating the last step for the next higher set bit in the message. For I items per node, setting on average $\log(I)$ bits in the local hash sketch, the per-node average worst-case insertion hop-count cost becomes $O(\log(I) \cdot \log(N))$.

3.2.3 DHS data maintenance

The maintenance of the DHS bits stored at a node can be provided using either a "soft-state" approach, whereby the bits are periodically refreshed by the nodes responsible for setting them, or using an "explicit-update" approach, whereby bits are set until explicitly deleted by the responsible nodes. Our contributions here are orthogonal with respect to the above decision. Both alternatives can be supported. For concreteness, we discuss one alternative, based on the soft-state approach.

Remember that a time-to-live value is stored along with every piece of information; data items are then deleted if not updated within this time period, so deleting an item incurs no extra cost. The computation of this $time\_out$ field poses an interesting trade-off. Larger time-out values will result in less updates per time unit needed to keep the DHS up-to-date. On the other hand, a smaller value will allow for faster adaptation to abrupt fluctuations in the value of the metric estimated, but will incur a higher maintenance cost as far as (primarily) network resources are concerned. However, we have to point out once again that the per-node bandwidth and storage requirements of DHS are very low, thus even a high update rate might translate to a negligible bandwidth consumption.

3.2.4 Counting with DHS

Just like in the insertion phase, in order to check for the state of a bit position, the querying node first computes a random ID in the region corresponding to the probed bit position and then sends a "probe" message to the node responsible for that ID. The worst-case message cost for this operation is also in $O(\log(N))$. Furthermore, the fact that the ID space region corresponding to a given bit position is the same for all bitmaps, allows the querying node to acquire state information for the probed bit position for *any and all bitmaps and metrics* with just a *single operation*. This design achieves a balanced storage and maintenance load across all nodes in the system.

In [68], superLogLog counting using a populated DHS is performed by starting from the most significant bit region of the ID space and probing nodes in regions corresponding to successively less significant bit positions, until all bitmaps have at least one 1-bit (see Fig. 5c). For PCSA hash sketches, counting proceeds in the opposite direction; starting from the least significant bit region of the ID space and probing nodes in regions corresponding to successively more significant bit positions, until at least one 0 bit has been located for every bitmap. Probed nodes respond with a *<metricID, bit-vector>* tuple (*metricID* corresponds to the quantity being estimated). For a m-bitmap DHS, *bit-vector* is a m-bit vector with bit positions corresponding to the value of the probed bit for each of the bitmaps in the DHS. The worst-case message cost for this procedure is in $O(L \log(N))$ for L bits per hash sketch bitmap, N nodes in the P2P overlay, independently of the number of bitmaps, items, or dimensions.

We have slightly modified the above counting procedures to perform counting in a recursive manner, since recursive routing has been proved to be more efficient than iterative routing in real-world systems [19,77], and in order to use caching of the probe results along the path of recursion to deal with some of the shortcoming of the techniques in [68]. In more detail, in both the superLogLog and PCSA cases, counting proceeds from the least-significant-bit position to bit positions of increasingly higher significance (depicted later in Fig. 7). Suppose a random node wishes to execute the counting procedure for a given metric. Counting begins with the querying node and an empty hash sketch (i.e. a hash sketch whose bits are all set to an "undefined" value) and proceeds as follows:

1. The current node first checks its local cache for a full hash sketch for the desired metric.
2. If such a sketch is found:
   (i) The sketch is returned to the previous node in the path (or to the user if this is the querying node).
   (ii) The target node caches the received sketch and recurses to step (i).
3. Else:
   (a) The current node computes the hash sketch based on its local data and merges it with the hash sketch retrieved from the previous node (if any).

(b) If the merged hash sketch is complete (i.e. for PCSA counting, there is at least a 0-bit for any bitmap in the sketch, while for superLogLog counting, there is a bit position which is set to 0 for all bitmaps in the sketch):[11]
   – The node caches the final result, and goes to step (a).

(c) Else:
   – The node computes a random ID in the region for the next higher bit position (or bit position 0 if this is the querying node), and
   – Sends a probe for the desired metric, along with the merged hash sketch, to the node responsible for that ID.

Note that since the algorithm visits at most L regions and it takes $O(\log(N))$ hops in the worst case to go from one region to the next during the forward phase (propagation of the probe) of the algorithm, the resulting worst-case hop count complexity is again in $O(L \cdot \log(N))$.

3.2.5 Compensating for random selection errors

We have further examined analytically the error introduced by the above-mentioned randomized distribution of bit information to multiple nodes, and have presented a trade-off between accuracy, availability, and efficiency. In brief, the randomizing algorithms used when setting bits and when checking their values in the DHS, introduces a probability of probing nodes not storing any bit data during the query phase. In order to deal with this, when a node answers back with an empty result set, we retry the probe to its immediate neighbors in the overlay, for up to a precomputed number of times. In order to compute this threshold, assume that n' items have been uniformly distributed to N' bins (i.e. mapped to an N'-node *interval* in the DHS). The counting process corresponds to uniformly and independently picking a bin from the set of bins without replacement, and checking for whether there is any item stored in it. The probability $P(X = t)$ that t empty bins are selected in the first t probes, equals: $P(X = t) = \left(\frac{N'-t}{N'}\right)^{n'}$ (see [68] for a sketch of proof). By solving this equation for t, and taking into account that with multiple (m) bitmaps, items are quasi-uniformly partitioned among them, we get that, in order to choose a non-empty bin with probability of at least p, one has to visit at least: $\lceil N' \cdot (1 - p^{\frac{m}{n'}}) \rceil$ nodes. Empirically we have found that, when there are as many items as there are nodes in the overlay, up to 5 nodes have to be visited in the worst case in order to guarantee that a non-empty node will be found with a probability of at least 99%, if such a node does exist (i.e. if the bit probed has actually been set during the insertion phase). Last, these are all 1-hop operations and the number of nodes visited during the retry phase can be considered a small constant.

[11] For superLogLog counting, although there may be more 1-bits after an all-0 bit position, we have empirically verified that (1) the probability that this happens is negligible, and (2) in the rare opposite case, these bit positions are always among those omitted during the truncation phase of the superLogLog algorithm.

3.2.6 Simultaneously counting multiple quantities

As noted earlier, when counting, one probes a node ID in a region for its bit information for all bitmaps and all metrics of interest, and the probed node responds with a sequence of tuples, one for each estimated dimension/quantity. Thus, counting in multiple dimensions (i.e. estimating multiple quantities) has *the same worst-case hop-count complexity as counting in a single dimension*. Such quantities, for example, can be the frequencies of the cells of a DHS-based histogram, to be discussed shortly, thus allowing a node to gather *all* cell frequencies in just $O(L \cdot \log(N))$ hops.

4 Decentralized aggregate query processing for P2P data management

We move on now to discuss ways of computing certain basic aggregates in a P2P data management system.

COUNT-DISTINCT

Both rendezvous-based hash sketches and DHS are directly applicable for the estimation of the number of (distinct) items in a multiset. Assume we wish to be able to estimate the number of distinct values in a column C of a relation R stored in our Internet-scale data management system. In the rendezvous-based scenario, the algorithm proceeds as follows: (1) nodes storing tuples of R, first hash together the identifiers of R and C to compute the rendezvous ID; (2) each such node adds the relevant *values* it stores into a locally populated synopsis, and (3) sends its synopsis to the node responsible for the rendezvous ID. That node will then be responsible for gathering all synopses and constructing (via bitwise OR) the overall synopsis. Any node will then be able to ask the rendezvous node for the estimated cardinality of $R.C$ (see Fig. 6). The worst-case number of messages to populate the overall synopsis is in $O(N \log(N))$ in an N-node network—translating to a $O(b \cdot N \log(N))$ bandwidth usage figure, for b bytes per local synopsis—while querying the populated synopsis needs $O(\log(N))$ messages.

Note that, when counting distinct items, a synopsis-based rendezvous solution is much more efficient than the simple approach of having nodes send lists of tuple-ID/value pairs they store to a rendezvous node; the rendezvous node would

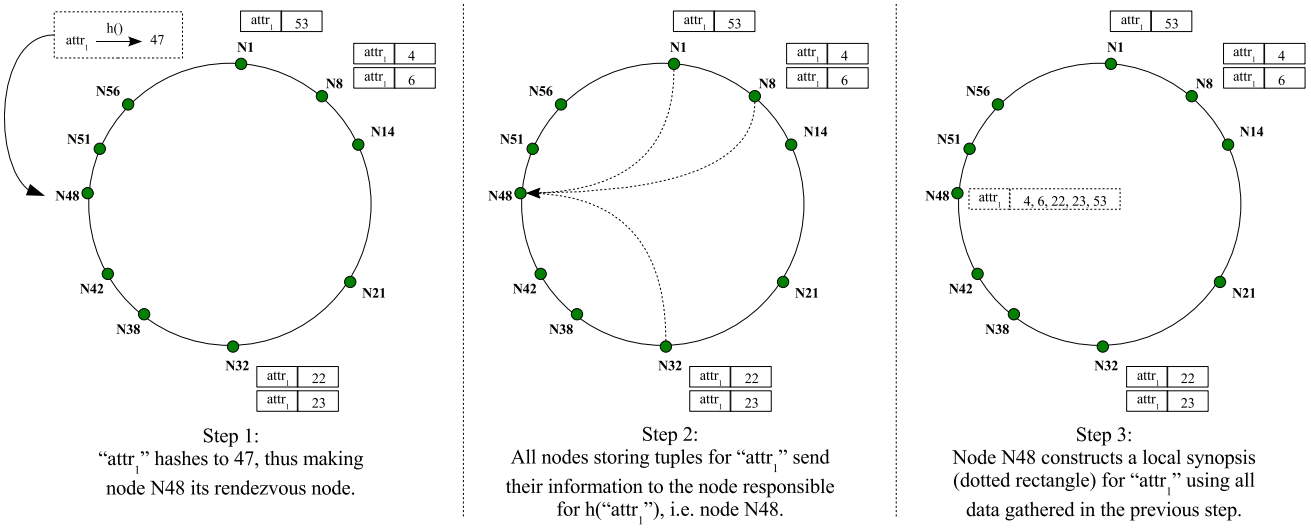


Fig. 6 Example of a rendezvous-based approach: node 48 is designated as the rendezvous node and all nodes with tuples for attribute$_1$ send it their data

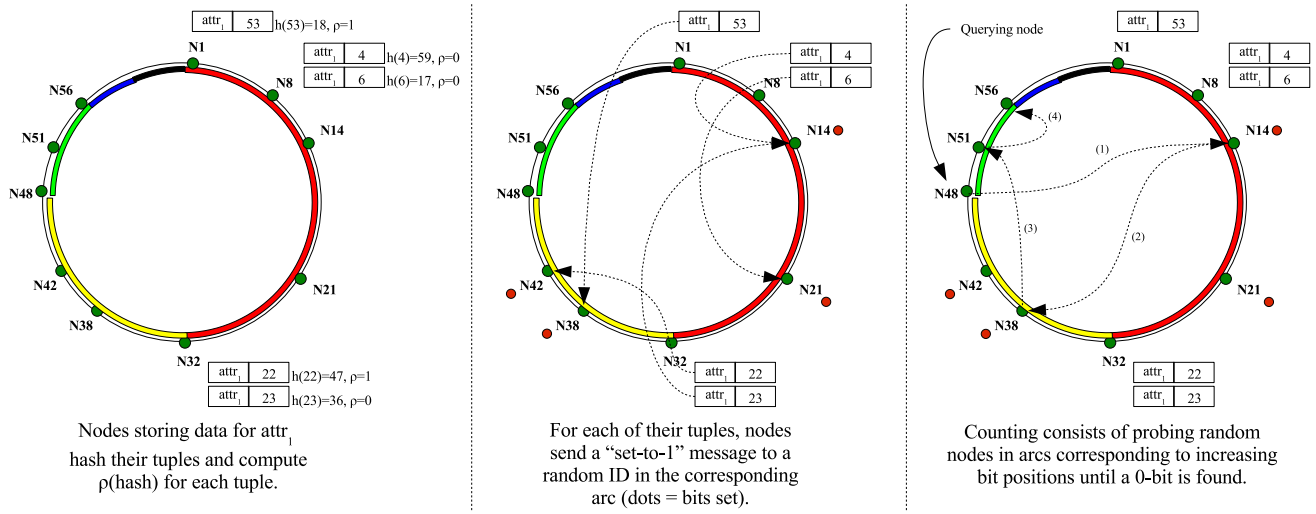


Fig. 7 Example of a DHS-based approach: nodes insert their tuples in the DHS and the load is spread across the overlay

need space proportional to the actual estimated quantity to be able to discern duplicates, while hash sketches considerably trim this figure down (roughly in the order of the logarithm of the measured quantity), while a similar argument also holds for bandwidth usage for transferring the lists to the rendezvous node.

In the DHS-based case, nodes storing tuples of R insert them into the DHS, by:

1. Hashing them on their C value ($C.v_i$) and inserting them to a local hash sketch, as described in Sect. 3.2,
2. Selecting a random ID in the interval corresponding to the bit set during the previous step, and
3. Sending a "set-to-1" message to the node responsible for this ID (see Fig. 7).

Assume the DHS uses L-bit bitmaps. Then, the worst-case message count cost to populate the DHS is in $O(N \cdot L \cdot \log(N))$, which translates to a $O(b' \cdot N \cdot L \cdot \log(N))$ overall bandwidth consumption, with b' bytes per "set-to-1" message. In a real-world scenario, such a message would merely include the identifier of the measured quantity (i.e. the "metric ID" per the lingo of [68]) and the bitmap and bit position to set. Querying the populated DHS requires $O(L \cdot \log(N))$ messages in the worst case.

Comparing the two approaches, we can see that populating the DHS-based estimator needs a factor of L more messages and a factor of L more overall bandwidth[12] in the worst case, while querying it also requires a factor of L more messages in

[12] Divide by N to get the corresponding cost figures per node, as opposed to overall costs.

the worst case, compared to the rendezvous based estimator. However note that: (1) in the rendezvous-based approach, the rendezvous node receives all the load of counting and constitutes a single-point-of-failure and performance bottleneck, while the DHS-based estimator is completely decentralized and balanced with regard to the per-node load, and (2) when estimating multiple quantities, the hop-count cost of populating and querying the DHS-based estimator remains unchanged, while the relevant costs of the rendezvous solution grows linearly to the number of estimated quantities.

COUNT

In order to count the number of items in a column *including duplicates* we would proceed as above, only adding the tuple IDs to the corresponding synopsis (hash sketch or DHS), instead of the values of the column in question. Furthermore, if identical tuples stored on different nodes should also be accounted for, nodes hash together the node and tuple IDs and use the result as the input to the synopsis insertion procedures.

Node cardinality

One of the metrics used in the optimizer of System-R and a very important one when it comes to actually executing a query, is the number of data pages per relation or per index. This metric defines the number of secondary storage accesses that the query processor will have to make when doing a full scan of the relation/index. We propose an analogy between P2P overlay nodes and secondary storage disk blocks. In the traditional, centralized database world, the number of secondary storage accesses is the main cost component of query access plans. Furthermore, it is generally known that disk I/O delays are mainly due to seek time. In the distributed, peer-to-peer setting, nodes can be thought of as the counterpart of disk blocks; routing to a node in the network overlay corresponds to a seek in a disk subsystem, while accessing items on a node corresponds to a disk block access. As a matter of fact, by P2P norms, the communication overhead dominates by far all local processing costs, to the extent that any local operation may safely be (and actually usually is) ignored in the cost formulas.

In order to estimate the number of nodes that store tuples of a given relation, each node storing tuples for the relation/column in question, inserts its ID into a global synopsis. Since node IDs are unique across the P2P overlay this operation is equivalent to a COUNT-DISTINCT aggregate over an imaginary column populated with the IDs of nodes storing data for the relation in question. As of this, we can use any of the techniques discussed earlier (i.e. hash sketches with rendezvous nodes or the DHS) to estimate this quantity.

When applied to a whole relation R, the "node cardinality" metric corresponds to the number of nodes in the system storing tuples of R. One can easily think of important cases where this computation may prove useful. For example, in later sections we shall show how to compute histograms over attributes in a relation stored in an Internet-scale data management system. In order to reach an informed decision, a possible optimizer will need, in addition to the attribute-value frequency distribution histograms, further information as to how many nodes store tuples belonging to a given histogram cell or having a given attribute value. By building node cardinality histograms, one can answer such questions using standard statistical techniques from the relevant literature. Such statistics, coupled with a method for computing overlaps between the data sets of various nodes (such as [61]), will greatly improve the efficiency and quality of query processing in such distributed settings.

SUM and AVG

In order to compute the SUM aggregate, we follow the algorithm in [16]: each node locally computes the sum of values of the column tuples it stores, populates a local hash sketch using the algorithm in [16], and then either posts this hash sketch to the rendezvous node in the former case, or sends batches of "set-to-1" messages to the appropriate nodes on the DHS in the latter case. For the rendezvous case, the message count, bandwidth consumption, and query overhead figures are obviously the same as those discussed previously. For the DHS case, note that the $O(N \cdot L \cdot \log(N))$ figure also holds here since, with L-bit bitmaps, at most L bits can be set. Therefore, the cost analysis is the same as that of earlier aggregates. Similarly, computing the average (AVG) of the values of a column consists of estimating the SUM and COUNT of the column and then taking their ratio.

5 Decentralized histogram creation and use for P2P data management

In this section, we shall attempt to harness the tools discussed so far and present techniques to implement Average Shifted Equi-Width histograms, thus improving the accuracy of the approach in [68], and a method to construct Equi-Depth histograms, over an Internet-scale data management system. We also discuss the case of extending our approach for computing more complex histogram types, such as MaxDiff(V,F) and Compressed(V,F) histograms.

Equi-Width histograms

The core idea is to partition the attribute value domain into a fixed number of equally sized intervals/buckets (say B)

and use a different distributed synopsis "metric" for each bucket to estimate the number of data items belonging to it. This construct corresponds to a distributed Equi-Width histogram (coined DEWH) over the given attribute. If using a DHS-based distributed synopsis, this technique requires $O(N \cdot L \cdot \log(N))$ overall messages in the worst case for all nodes to insert all of their items in the DHS and $O(L \cdot \log(N))$ hops to reconstruct the histogram (remember that counting in multiple dimensions with DHS incurs no additional hop-count overhead). For a rendezvous-based approach, these costs are in $O(N \cdot B \cdot \log(N))$ and $O(B \log(N))$, respectively, since there will (on average) be B different rendezvous nodes. An even more extreme case would be to store all histogram buckets on a single rendezvous node; in this case, the message cost for data insertion and histogram reconstruction is in $O(\log(N))$, as discussed earlier, but the load imbalance is even more severe, as we shall shortly see in the experimental evaluation section.

Note that, even in this extreme case, the rendezvous-based approach is still approximate and not exact. Due to the arbitrary replication of data tuples across nodes in the overlay, for such an approach to be exact, the rendezvous node should maintain complete lists of at least the primary keys of every item mapped to every bucket, so that identical items inserted by different nodes do not get counted twice in the bucket frequencies. However, this clearly does not scale well with the number of nodes and/or the number of items in the overlay. Thus, the rendezvous node has to either (1) keep a synopsis of the primary keys mapped to every bucket (e.g. a Bloom filter—but without a-priori knowledge of the amount of items mapped to each bucket, this should be made large enough to be able to record the primary keys of all data items in the overlay, which in turn is also not a known quantity), or (2) use a synopsis for the per-bucket frequency (e.g. a hash sketch), which is based on the exact same notion as the per-bucket distributed hash sketch of the DHS-based approach.

Average shifted Equi-Width histograms

By taking advantage of the dimension-free counting of DHS, we can also implement a distributed version of ASHs (coined Distributed ASH or DASH), using the technique outlined earlier for standard Equi-Width distributed histograms. Each DASH will consist of several DEWH with the same bucket widths but different starting positions in the value space.

In a DHS-based approach, each node constructs a local hash sketch for each cell of each of the equi-width histograms comprising the ASH, and sends batches of "set-to-1" messages for every non-zero bit position in the resulting hash sketches, containing the identifiers of the metrics/ASH cells for which this bit is to be set. Thus, the message cost of populating DASH is the same as in the previous case, while the cost of reconstructing the histogram is again in $O(L \cdot \log(N))$.

With rendezvous-based DEWH as a base, on the other hand, the cost of inserting the items is in $O(N \cdot S \cdot B \cdot \log N)$ for S shifted DEWHs and the cost of reconstructing the histogram is in $O(S \cdot B \cdot \log N)$, since in this case there will be $S \cdot B$ different rendezvous nodes. Again, one could take the rendezvous case to the limits and store all buckets of all shifted histograms on the same rendezvous node so as to bring these figures down to $O(\log(N))$, incurring however an even greater load imbalance than in the previous approach.

Equi-Depth histograms

Distributed synopses, as described earlier, are good for estimating the number of items belonging to a predefined group (e.g. value range), but cannot count items belonging to a range whose boundaries change on-the-fly. This makes distributed synopses inappropriate for direct use in building histogram types other than Equi-Width or Average-Shifted ones.

To this extent, first note that although researchers usually reject Equi-Width histograms in favor of more complex types, such as Compressed or MaxDiff histograms, there are situations where the former perform as well or even better than the latter [9], while there may be settings where even a one-bucket histogram is adequate [21].

Studying the relevant literature, we noted that more complex histogram types, such as Equi-Depth histograms, usually require much fewer cells to provide a good approximation of the attribute value distribution than Equi-Width histograms [74]. Moreover remember that, when using DHS, we can build Equi-Width or Average Shifted histograms with many cells for no extra cost, having a separate metric for each histogram cell, due to the counting properties of DHS.

We thus propose to use either DEWH or DASH histograms as our base and try to infer more complex histogram types from them. The general approach is to retrieve information from a DEWH or DASH histogram, and then to do a series of local computations to infer the more complex histogram types. In more detail, our solution for computing an Equi-Depth histograms consists of the following steps:

1. First, build a relatively large DEWH or DASH histogram—for example, having 10-to-20 times more buckets than the target histogram.
2. Use the uniform spread assumption to project frequencies to values within each cell,
3. Consider the set of values of all cells and
4. Find a partitioning of them into c groups, such that the order of values is not changed and all groups have equal (or almost equal) sums of value frequencies.

More complex histogram types

One could advocate using the above inference technique to compute even more complex histogram types, such as MaxDiff(V,F) histograms—used by the Microsoft SQL Server [44]—and Compressed(V,F) histograms—proved to be the perfect trade-off between optimality under the variance metric [46] and practicality due to their size and construction time overheads, and is used by the DB2 optimizer [44]. In short, MaxDiff(V,F) histograms first sort items on their values and then place cell boundaries in the points in the value domain where items have the largest differences in their frequency, while in a c-cell Compressed(V,F) histogram, the $c-k$ higher frequency items are placed in single-value buckets, and the remaining values are placed in k buckets in an Equi-Depth manner.

After projecting frequencies to values within each cell using the uniform spread assumption, a c-cell MaxDiff(V, F) histogram could be inferred from a large plain Equi-Width or DASH histogram, by:

1. Creating a sorted list of the differences in frequency of the above values,
2. Selecting the $c-1$ largest differences as cell boundaries,
3. Grouping together values within each new cell, and
4. Computing the cell's frequency as the sum of the values' frequencies.

while for a c-cell Compressed(V, F) histogram we could:

1. Locate the $c-k$ highest-frequency items,
2. Place them in singleton buckets, and
3. Assign the remaining values to k Equi-Depth buckets as in the Equi-Depth case above.

However, as we shall se shortly in the experimental evaluation section, these cases are somewhat problematic. This is mainly due to the fact that the width (and, therefore, the number) of the buckets of the histogram in the first step of the above inference algorithm, defines the accuracy and granularity of the subsequent inference steps. In order not to lose any information there should be a separate histogram bucket for each attribute value (i.e. buckets should be 1 value wide). On the other hand, if the attribute value domain is very large (e.g. millions of values), having singleton buckets will not be practical. Moreover, the use of DHS imposes a trade-off between the number of items falling into each bucket and the accuracy of the corresponding estimate, as outlined in [68], with more items resulting in better accuracy. Since the bucket frequency is projected to values in the bucket using the uniform spread assumption, the proposed method loses in accuracy as the buckets get wider; this holds especially for the Compressed(V, F) and MaxDiff(V, F) histograms whose construction relies on accurately locating the highest-frequency or highest-frequency-difference value positions; for example, if the value distribution is very skewed and there is one very popular value in a multi-valued bucket, then the proposed inference method will assign a fraction of the total bucket frequency to each of the distinct equal-spread values in the bucket, and will thus fail to accurately compute the frequency-to-value mapping in that bucket.

We shall revisit this trade-off in the experimental evaluation section. As we shall also see there, this issue affects almost solely the Compressed and MaxDiff histogram construction; for Equi-Depth histograms, a DEWH or DASH with 10-to-20 times more buckets than the target histogram achieves nearly excellent accuracy with a very low cost. On the other hand, for Compressed and MaxDiff histograms we require a DEWH or DASH with a few values, 1-to-5, per bucket. There is a new hidden trade-off here regarding bandwidth requirements; unlike the number of hops, the amount of data fetched during reconstruction of the base DEWH or DASH histograms grows linearly with the number of buckets in the histograms. To this extent, we have considered adding a module in our inference engine that is responsible for automatically tuning the number and width of buckets and/or shifts in the underlying DEWH or DASH histograms, according to the domain size, the type of histogram to construct, and a user-supplied network bandwidth threshold. The current implementation of this module is in preliminary stages and remains a subject of ongoing and future work.

We also wish to explicitly state that we do not deal with multi-dimensional histograms. The main thrust behind this work is not to provide implementations for all specific histogram types or to solve all well-known issues in histogram-related research. We simply wish to show that such statistical structures can be maintained in a purely distributed manner, to present alternatives and discuss issues arising in this setting.

5.1 Applications in P2P query optimization

As already mentioned, we think of remote nodes and network accesses as if they were blocks and secondary storage accesses in a centralized database management system. Thus, a query optimizer for a P2P DBMS should, in addition to taking into account the statistics of the data values stored on the P2P overlay, also consider statistics on the node population, such as the "number of nodes per relation" (as opposed to the "number of data pages per relation"). We thus advocate storing, along with value frequencies, the node cardinality of each histogram cell, and to take into consideration both the number of data items and the number of nodes involved in a query, to further optimize the query access paths. This

operation roughly corresponds to building "node histograms" in addition to the histograms on the data value frequencies.

With the infrastructure discussed so far, we can compute several pieces of information required by optimizers, as discussed in [11] (and mentioned earlier). More specifically:

1. The number of tuples in a relation. This quantity corresponds to the relation cardinality and can now be computed with a single COUNT and/or COUNT-DISTINCT operation, depending on whether we want to account for duplicates or not.
2. The number of physical pages used by a table. This quantity corresponds to the "node cardinality" of the relation, which can also be directly computed, as discussed earlier.
3. Statistical information for table columns, in the form of:
   (a) Histograms. Earlier in the current section we discussed techniques to compute several types of histograms over base data.
   (b) The number of distinct values in a column. Again, this quantity can be computed with a single COUNT-DISTINCT operation on the column of interest.
   (c) The minimum and maximum, or second-lowest and second-highest, values. As discussed earlier, this kind of functionality is usually provided by locality-preserving DHTs, on top of which we have assumed to operate. Otherwise, we can use histograms and the inference technique outlined earlier to compute an estimate of these values for each bucket, or for the column as a whole.
4. Information on the correlations among attribute values, in the form of:
   (a) Multi-dimensional histograms, which could be created using the infrastructure presented so far, by defining the boundaries of the multi-dimensional buckets, and assigning a DHS metric to each such bucket, instead of the single-dimensional buckets we had up to now. This solution may well suffer from the curse of dimensionality and probably be subject to a degradation in accuracy with increasing number of dimensions; however, there is also the following alternative.
   (b) A single-dimensional histogram on the leading column, plus the total count of distinct combinations of column values present in the data, for multi-column indices[11]. This can also be computed with our infrastructure, even on a per-bucket basis, by having nodes insert combinations of values under appropriate DHS metrics.

Furthermore, we can provide for range predicate selectivity estimation. To this extent, we consider the uniform spread assumption, with a twist. Remember that rendezvous hash sketches and DHS can count both the total number of items in a set including duplicates and the number of distinct values, depending on whether we insert the values or the keys of tuples. Since we are dealing with range queries at this point, we assume that the underlying DHT efficiently supports them, so we can find the minimum and maximum values of a cell by probing the nodes responsible for the values corresponding to the cell's boundaries; if on the other hand there is no such functionality available or we do not wish to make these two extra network accesses, we can assume that the minimum and maximum values coincide with the cell's boundaries, or estimate them as earlier.

Given a range predicate $\alpha \leq X \leq \beta$, we proceed as follows:

1. First determine the histogram buckets affected by the range query (the buckets with which the range predicate overlaps).
2. Using the DHS, determine the frequency of each bucket involved in the query, following the algorithms outlined earlier.
3. Using the uniform spread assumption [74], determine the values of each bucket and their frequencies.
4. Among these values, find those that belong to the range predicate, and
5. Return the sum of their frequencies as the estimated size of the range query result set.

Moreover, we could compute an estimate of a relation's self-join size, by using Compressed(V,F) histograms (due to the inefficiencies of basic types of histograms as estimators when the values of the joined relations are correlated [21]). What we need to do is:

1. Use the DHS to reconstruct the base Equi-Width histogram and infer the Compressed(V, F) histogram,
2. Iterate over all histogram buckets, and
3. Take the sum of the squares of the bucket frequencies.

Due to the counting properties of DHS, reconstructing the histogram has a worst-case message cost of $O(L \log(N))$. Note that the above computation should be used only as a heuristic, as the accuracy of Compressed(V, F) histograms is subject to many subtle factors, as we shall shortly see. We are currently investigating ways of applying our methods in other classic query optimization scenarios, such as projections, single- and multi-way joins, etc.

6 Implementation and performance evaluation

6.1 Setup

We initially implemented a basic Chord-like DHT, DHS, and the solutions proposed in this work, in an event-driven simulator coded in C++ from scratch [68]. The simulator supported all basic DHT primitives—that is, node joins and leaves/failures and data addition and deletion—and all the functionality of DHS and subsequent constructions. We used this simulator for preliminary performance evaluation purposes and sanity checks, and in order to observe how our solutions behave and scale in a fully controlled environment. We now implemented DHS and DHS-based aggregates and histograms over FreePastry [28], a publicly available, implementation of the Pastry overlay [23] in the Java programming language. In Pastry (and FreePastry) lookups have a hop-count cost in $O(\log_{2^b} N)$, with b a positive integer. We tweaked FreePastry so that its routing cost is in $O(\log_2 N)$ (i.e. $b = 1$), so that hop-count results are easily comparable to other DHTs. Both implementations gave similar results, so we chose to show only results computed using the latter, to showcase the performance of our solutions as implemented in a real-world system. The source code of our implementation, coined FreeSHADE for "Statistics, Histograms, and Aggregates in a DHT-based Environment", consists of approximately 8,500 lines of Java code[13] and is publicly available on the world-wide web through [29].

In both cases, the performance evaluation was carried out in the following steps:

1. We generated the workload; that is, the data tuples of the base relations and the queries to be executed on them later on. As is standard practice in the relevant literature, we synthetically generated data sets that (1) can test our system for various value distributions ranging from near uniform to highly skewed, and (2) correspond to value distributions also observed in real-world systems [37,76].
2. We populated the network with peers and allowed enough time for the DHT to stabilize. We chose to simulate a P2P DBMS running on a 1000-node DHT overlay, as an example of a mid-range distributed system, but larger networks are naturally supported by both our solutions and the code base.
3. We randomly assigned data tuples from the base data to nodes in the overlay. This corresponds to the data contributed by each overlay node.
4. Then we had all nodes insert into the P2P DBMS and the distributed synopses all relevant information (attribute values, tuple IDs, node ID) for the tuples they were assigned during the previous step.
5. Finally, we selected random nodes and had them execute our algorithms for reconstructing histograms and computing aggregates.

For the DHS-based cases, we used 32 bits per hash sketch bitmap. Note that the expected performance figures are directly affected by the bitmap length, and that with 32-bit bitmaps we are able to count over *trillions* of items. A more conservative (and better looking, with regard to the resulting hop-count figures) approach would be to use smaller (e.g. 20-bit) bitmaps; however, we present the 32-bit case as an example of a worst-case scenario. Finally, we varied the number of bitmaps from 64 to 512. The results we shall present shortly were averaged over multiple runs for every case, to avoid statistical artifacts.

Our main focus is on distribution transparency, thus we mainly want to prove that the proposed solutions (1) are as accurate as their centralized counterparts, (2) impose low run-time overhead, and (3) scale well with the size of the network and of the overall data collection of peers. We thus measure the accuracy and (message count) performance of the proposed solutions, attempting to showcase their applicability under the wide-scale distribution setting of a P2P DBMS.

More specifically, for aggregate query processing, we were primarily interested in the number of hops required to do the estimation, the accuracy of the estimation itself, as well as the fairness of the load distribution across nodes in the network. We instrumented FreePastry with the appropriate hooks to allow us to measure the number of hops each message needs to reach its destination node and the corresponding bandwidth usage. We present hop-count results for both inserting items to the DHS and for doing the actual estimation. We present results for three aggregates; namely, COUNT, SUM, and AVG, as examples of aggregates based on classic hash sketches, on summation sketches, and on a combination of these two respectively. Moreover, we report on the mean error of the estimation, computed as the percentage by which the distributed estimation differed to the actual value of the estimated aggregate computed over the base data in a centralized manner (i.e. as if all data was stored on a single host). Last, we report on the distribution of insertion and query load across nodes in the overlay, in order to showcase the trade-off of performance versus scalability/load distribution between the DHS and rendezvous-based approaches.

For histograms, our main concern was the accuracy of the estimated bucket frequencies and bucket boundaries (for inferred histograms), as well as the cost of reconstructing the base DEWH or DASH histogram in terms of number of hops and bandwidth usage (the inference step is a local operation and hence considered of negligible cost in the P2P

[13] Counted using David A. Wheeler's 'SLOCCount'.

environment) and the load distribution. For the hop-count and bandwidth measurements we proceeded as above. In order to measure the accuracy of the estimated histograms, we first computed the histograms of choice over the base data in a centralized manner, then reconstructed and inferred the corresponding distributed histograms. Accuracy is again measured as the percentage by which the per-bucket estimated frequencies or bucket boundaries differ to the corresponding bucket frequencies/boundaries of the centralized histograms.

As far as the (fairness of the) distribution of load across participating hosts is concerned, we measure the load on any given node as the insertion and query/probe "hits" on this node; that is, the number of times this node is the target of an insertion or query/probe operation. We further instrumented FreePastry to report on the above metrics; namely Node Insertion Hits and Node Query Hits. Now, assume l_i, $1 \leq i \leq N$ is the load on the i^{th} node, and that μ_l and σ_l is the mean and sample standard deviation of these loads respectively. That is:

$$\mu_l = \frac{1}{N} \cdot \sum_{i=1}^{N} l_i$$

and

$$\sigma_l = \sqrt{\frac{1}{N-1} \cdot \sum_{i=1}^{N} (l_i - \mu_l)^2}.$$

We employed a multitude of metrics to visualize the impact of the different approaches outlined earlier in this paper. More specifically, we used:

- The Gini Coefficient [20], as proposed in [70], calculated by:

$$\mathrm{GC} = \frac{1}{2 \cdot N^2 \cdot \mu_l} \sum_{i=1}^{N} \sum_{j=1}^{N} (|l_i - l_j|).$$

  That is, GC is the mean of the absolute difference of every possible pair of load values. The Gini Coefficient takes values in the interval [0, 1), where a GC value of 0.0 is the best possible state, with 1.0 being the worst. The Gini Coefficient roughly represents the amount of imbalance in the system, so that for example a GC value of 0.25 translates to ≈75% of the total load being equally distributed across all nodes in the system.
- The Fairness Index [47]:

$$\mathrm{FI} = \frac{\left(\sum_{i=1}^{N} l_i\right)^2}{N \cdot \sum_{i=1}^{N} l_i^2}.$$

  The Fairness Index (also known as "Jain's Fairness Index" after the first of the authors of [47]) takes values in the interval (0, 1], with 0 and 1 being the worst and the best value respectively. FI is to some extent the inverse of the GC in that it represents the amount of balance in the system, so that for example a FI value of 0.25 roughly translates to the load being equally spread across 0.25 of the nodes in the system.
- The maximum and total loads for DHS- and rendezvous-based approaches.

6.2 Results

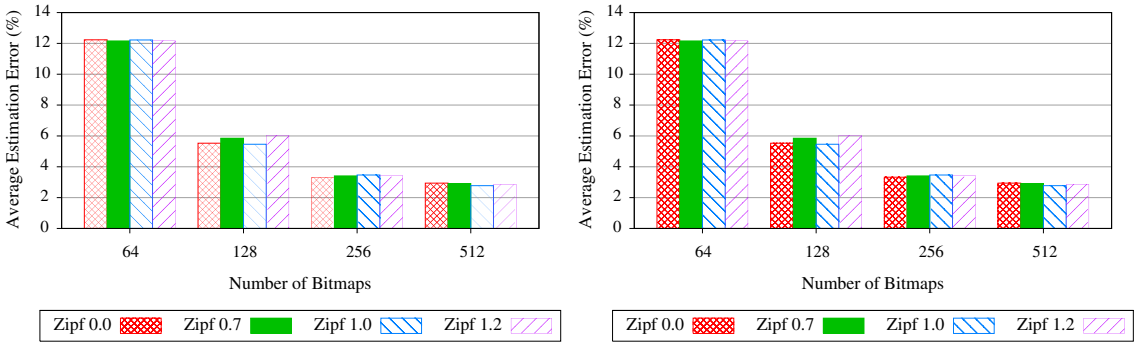
6.2.1 Aggregate queries

In this part of our experimental evaluation we primarily measured the hop-count efficiency and the accuracy of rendezvous-based hash sketches and of the DHS as the substrate for computing aggregates in a P2P DBMS. We initially created single-attribute relations, with integer values in the intervals [0, 1000), following either a uniform distribution (depicted as a Zipf with θ equal to 0.0), or a shuffled Zipf distribution with θ equal to 0.7, 1.0, and 1.2 [37,76]). Relations contained 300,000 tuples each, distributed across all nodes in the overlay. Note that we use single-column relations for presentation reasons and ease of modeling, as a means of evaluating queries on statistical structures over index attributes of a (possibly much) larger relation; we have also tested our system with larger configurations (more and larger tuples per relation, real-valued attributes, etc.) with similar results.

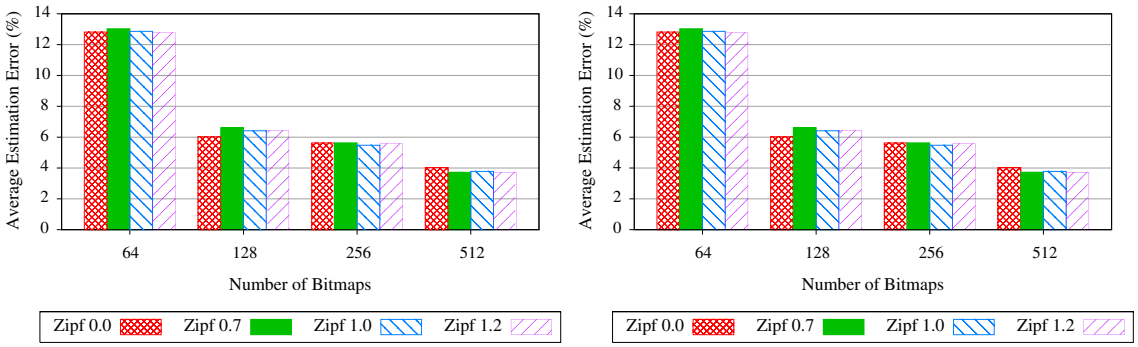
Estimation error and hop-count costs

Figures 8a and b plots the average error of the resulting estimate for the COUNT aggregate, using a rendezvous- and a DHS-based approach respectively. In both cases, the resulting error is due to the use of hash sketches to estimate the aggregate, thus both approaches exhibit the same average error. As expected, the higher the number of bitmaps in the synopsis, the better the accuracy. Specifically, for 64 bitmaps the average error is ≈12%, dropping to ≈5.7% for 128 bitmaps, ≈3.4% for 256 bitmaps, and ≈2.9% for 512 bitmaps. The estimation error is slightly larger for the SUM aggregate, due to the error introduced by the use of the summation technique of [16] (Figs. 8c, d). Last, the estimation error curve for the AVG aggregate closely follows that of the SUM aggregate.

We have measured and show the per-node average hop count for inserting all tuples to the distributed synopsis, and the hop count for computing the aggregate. Figures 9a and b depict the insertion hop-count cost for all of the evaluated aggregates. Figure 10a depicts the query hop-count cost for the rendezvous-based approach; since there is a single

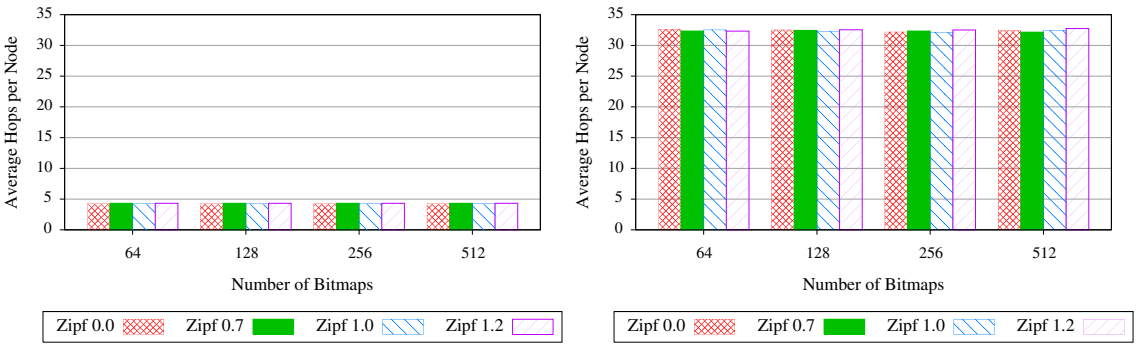


(a) COUNT aggregate – Rendezvous approach **(b)** COUNT aggregate – DHS approach



(c) SUM/AVG aggregate – Rendezvous approach **(d)** SUM/AVG aggregate – DHS approach

Fig. 8 Average % estimation error for the COUNT and SUM/AVG aggregates versus their centralized values



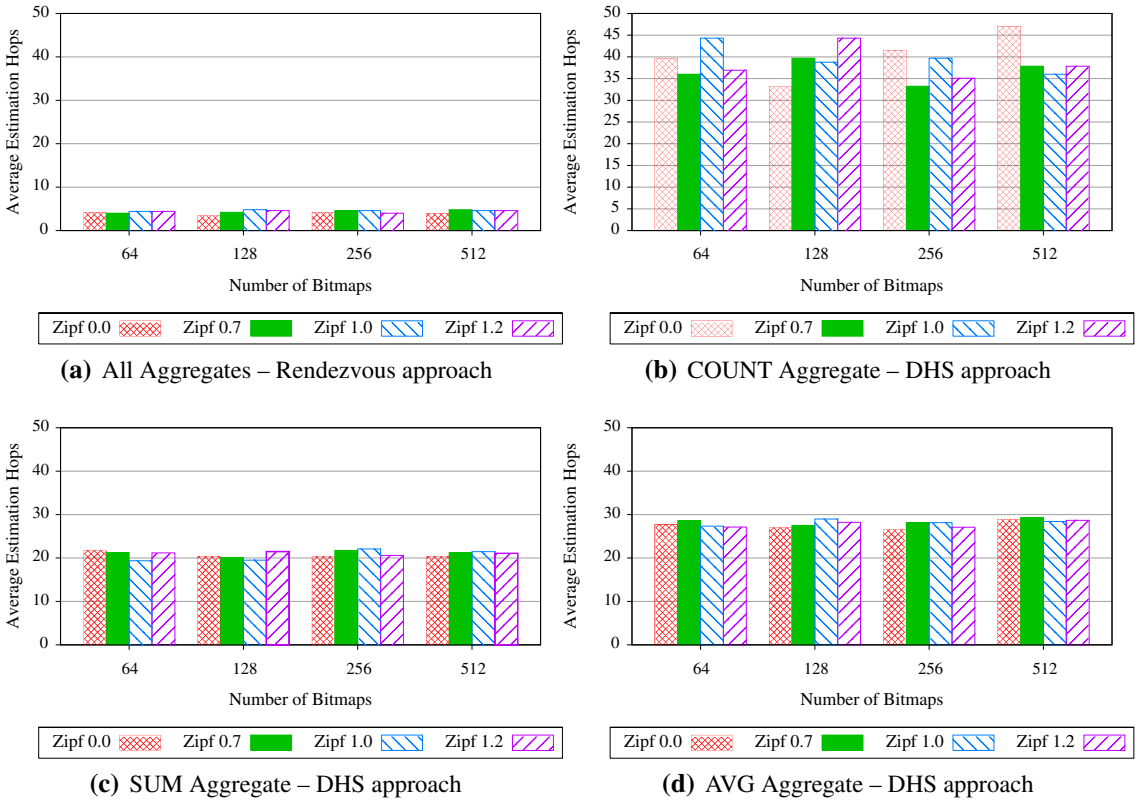
(a) Insertion hop count – Rendezvous approach **(b)** Insertion hop count – DHS approach

Fig. 9 Hop count for populating the distributed synopsis using a rendezvous-based approach without replication and a DHS-based approach

rendezvous node per aggregate, this curve is the same for all evaluated aggregates. Finally, Fig. 10b, c, and d depicts the query hop-count cost for the DHS-based approach, for the COUNT, SUM, and AVG aggregates, respectively. As can be seen, the per-node hop count costs are higher for the DHS-based approach by a factor of approximately 8× for both the insertion and query cases. Given that each node does one DHT lookup to insert all of its tuples and to query the distributed synopsis in the rendezvous-based case, this translates to approximately 8 DHT lookups per node in the DHS-based case. The query hop-count cost for the DHS-based approaches for SUM and AVG (Fig. 10a, d) is somewhat lower compared to the COUNT aggregate, as more bits are set to 1 due to the algorithm in [16] and thus it is easier for the DHS probing algorithm to find set bits.

Load distribution

The extra hop-count cost of the DHS-based approach pays back when it comes to load distribution fairness. Figures 11a, b and 12a, b plot the Gini Coefficient and the Fairness Index of the load imposed on nodes in the system during tuples



(a) All Aggregates – Rendezvous approach

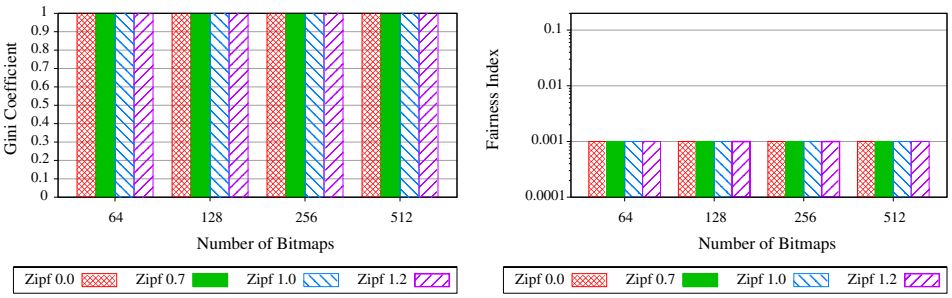
(b) COUNT Aggregate – DHS approach

(c) SUM Aggregate – DHS approach

(d) AVG Aggregate – DHS approach

Fig. 10 Hop count for computing the COUNT, SUM, and AVG aggregates using a rendezvous-based approach without replication and a DHS-based approach

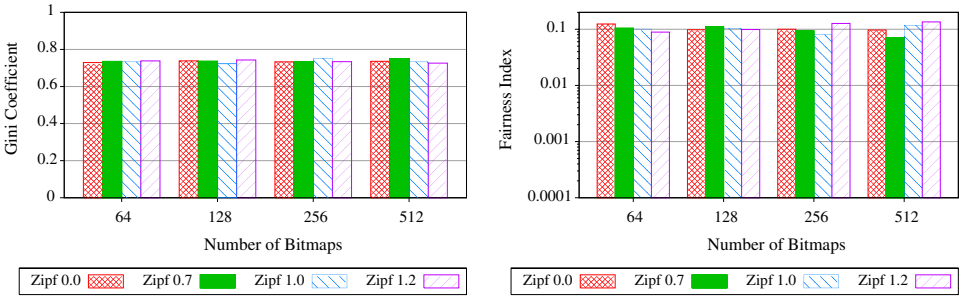
Fig. 11 Gini Coefficient (smaller is better) and Fairness Index (larger is better—note the logarithmic scale on the Y axis) of the load on nodes for populating a rendezvous-based synopsis



(a) Gini Coefficient – Rendezvous approach

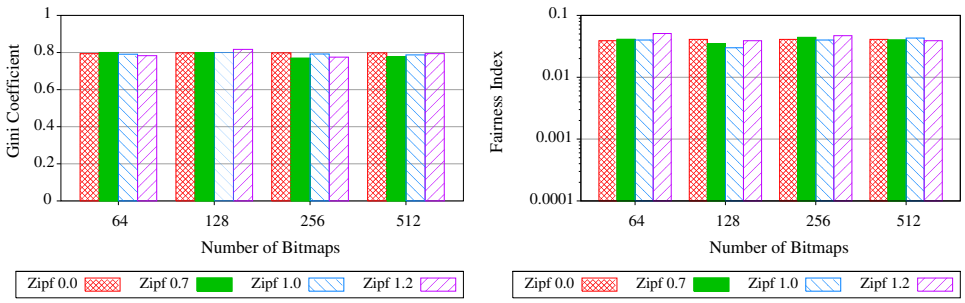
(b) Fairness Index – Rendezvous approach

Fig. 12 Gini Coefficient (smaller is better) and Fairness Index (larger is better—note the logarithmic scale on the Y axis) of the load on nodes for populating the DHS-based synopsis for the COUNT aggregate



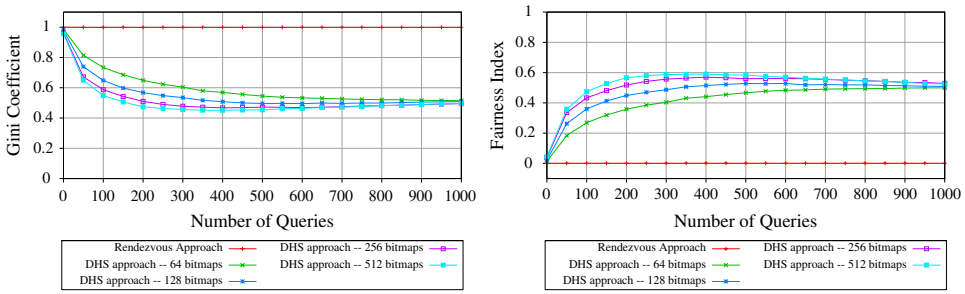
(a) Gini Coefficient – DHS approach

(b) Fairness Index – DHS approach



(a) DHS approach **(b)** DHS approach

Fig. 13 Gini Coefficient (smaller is better) and Fairness Index of the load on nodes for populating the DHS-based synopsis for the SUM and AVG aggregates (larger is better—note the logarithmic scale on the Y axis)



(a) Gini Coefficient **(b)** Fairness Index

Fig. 14 Evolution over time of the Gini coefficient and Fairness index for querying the distributed synopsis for all three aggregates

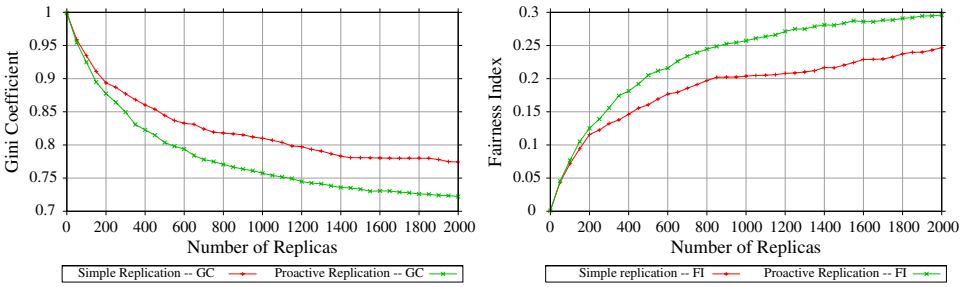
insertion for the COUNT aggregate (the figures for the rendezvous-based approach are practically identical for all three aggregates tested, so we will reuse Fig. 11a and b). Recall that the load on any given node is measured as the number of times that node is visited (a.k.a. node hits) during data insertion and/or query processing. There is approximately an improvement of more than two orders of magnitude for both metrics. More specifically, according to the Gini Coefficient, from a ≈0.1% of the load being equally distributed across all nodes in the rendezvous approach, to a ≈26.4% of the load being equally spread for the DHS approach; likewise, the FI value jumped from ≈0.001 for the rendezvous-based case to ≈0.100 for the DHS-based approach. This also holds for the SUM and AVG aggregate (see Fig. 13); the DHS-based approach manages on average to outperform the rendezvous-based approach by a factor of ≈200× and ≈40× as far as GC and FI are concerned, respectively.

Last, Fig. 14 plots the evolution of the Gini coefficient, Fairness index, and total query load (node hits) over time, as queries are executed in the system. With the rendezvous-based approach there is a single node that is burdened with all of the query load, thus the GC and FI are constant over time and equal or worse than ≈0.999 and ≈0.001, respectively. The DHS-based approaches converge to a GC an FI value of ≈0.5, which equal the GC and FI values of the distribution of the distances between consecutive nodes in the ID space and is thus the best respective values achievable by any algorithm that uses a randomized assignment of items to nodes.

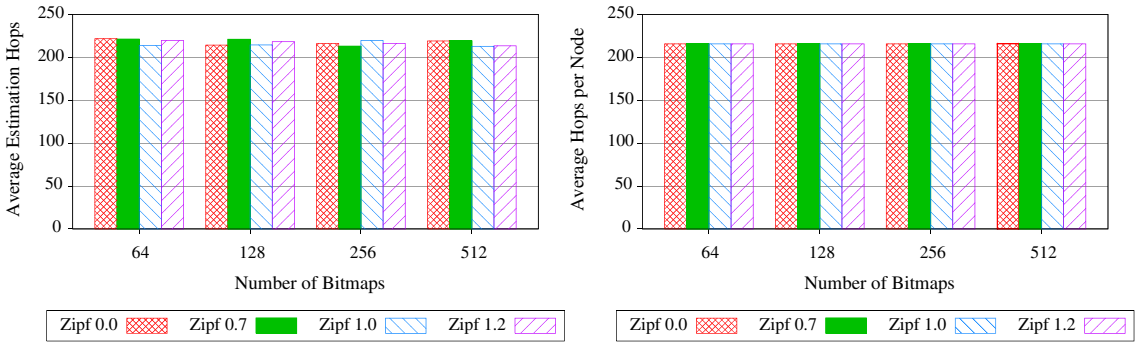
Replicated rendezvous

Supporters of the rendezvous-based approach might think that the GC and FI figures can be greatly improved by using multiple nodes to store the distributed synopsis and one of the replication schemes outlined in Sect. 3—namely, simple replication (i.e. each node chooses randomly one of the rendezvous nodes and insert all of its tuples there) or proactive replication (i.e. each node inserts all of its tuples on all rendezvous nodes), representing the two extremes in load distribution for the rendezvous-based case. Figure 15 plots the GC and FI values for an increasingly larger number of rendezvous nodes (replicas). As can be seen, even for 2, 000 replicas the GC value does not drop below 0.725 and the FI merely climbs to just below 0.3.

This might seem somewhat counterintuitive to the uninformed reader; one might expect that the load would be totally balanced with as many replicas as there are nodes (i.e. 1000). It all boils down to the way these replica nodes are selected; since a node does not (and cannot) know a priori the IDs of all nodes in the system (and even if it did, this information would soon become obsolete with nodes entering and leaving the P2P overlay), the best possible way of choosing the rendezvous IDs is by picking random IDs (following a uniform distribution) from the node ID space. Although node IDs are selected in a quasi-uniform manner (due to the use of a cryptographic hash function for their computation), this does not mean that the distances between consecutive



(a) Gini Coefficient (smaller is better) **(b)** Fairness Index (larger is better)



(a) Simple replication – Query hop count **(b)** Proactive replication – Insertion hop count

Fig. 15 Gini coefficient and Fairness index of the load on nodes for populating the distributed synopsis for the COUNT aggregate for increasingly larger amount of rendezvous nodes sharing the computation (either through replication or because of multiple computed aggregates)

Fig. 16 Hop counts for the COUNT aggregate using a rendezvous-based approach with 50 replicas

IDs are all equal; quite on the contrary. For example, in our 1000-node, 32-bit ID-space setup, the minimum, average, and maximum distances between any two consecutive nodes were $\approx 2^{11}$, 2^{22}, and 2^{25}, respectively, with a standard deviation of $\approx 2^{22}$. Note that this distribution of the distances between consecutive nodes in the network has a GC and a FI value of ≈ 0.5. Therefore, choosing among these nodes by first picking a random (uniform) ID and then selecting the node responsible for it, ends up in a rather skewed load distribution.

Now assume that there were a way for a node to know the IDs of all other nodes currently present in the system at any given point in time. Even in that case, it would take at least ≈ 275 rendezvous node replicas (either with simple or with proactive replication), for the rendezvous-based approach to achieve the GC and FI figures of the DHS-based approach. However, in that case the hop-count cost for inserting tuples into and/or computing the distributed synopsis would be abysmal. As a rule of thumb, Fig. 16 depicts the hop-count cost for inserting all tuples under proactive replication and for computing the COUNT aggregate under simple replication, using a rendezvous-based distributed synopsis, for a much milder replication scenario using just 50 replicas.

So far, there seems to be a clear-cut trade-off between the rendezvous- and the DHS-based approaches; the former is better when raw hop-count cost is more important, but the latter is the best choice when load distribution comes into play. This is only half the truth, though; let us take a look at Figs. 15 and 16 from a different perspective; that of multiple computed aggregates, as opposed to multiple replicas of a single aggregate. Assume, for example, that we are to compute multiple COUNT, SUM, AVG, etc. aggregates over the same peer-to-peer data network. Unless we store all such aggregates on a single rendezvous node—a nightmare from a load distribution and, hence, from a scalability and fault tolerance point of view—the only other choice is to assign these aggregates to randomly chosen nodes in the overlay. This scenario, however, is identical to that of a single aggregate with a proactive replication scheme and as many replicas as there are aggregates. The bottom line is that, although in the single-aggregate (or single-"metric") case the rendezvous approach seems to be the winner with regard to raw hop count, its advantage vanishes quickly with more than a handful of estimated aggregates.

In order to further illustrate this, Fig. 17 plots the total load imposed on the nodes in the system during query time for 10, 20, 50, and 100 COUNT aggregates (or conversely for a single COUNT aggregate and proactive replication with a factor of 1, 2, 5, and 10%, respectively, the figure for the simple replication rendezvous scenario would be even worse). Note that the y-axis is in logarithmic scale. This figure showcases in the best possible way the linear increase in the total

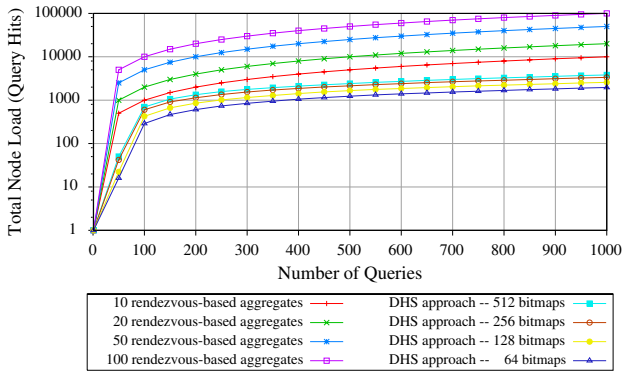


Fig. 17 Total query load in the system when multiple aggregates are to be computed concurrently

query load with the number of computed aggregates for the rendezvous-based approaches. Note that only one curve is shown the DHS-based solution for each of the possible number of bitmaps since the total query load (and hop count cost) is the same irrespective of the number of aggregates/"metrics" estimated, due to the properties of the DHS outlined in Sect. 3.2.

Scalability

Given the above discussions, we now turn our attention to studying the scalability of the DHS-based solutions. As made obvious thus far, the greatest strengths of the DHS-based solutions is their scalability for large numbers of computed aggregates and their fair load distribution across nodes in the overlay. The DHS-based approach further exhibits excellent scalability with regard to the number of nodes and tuples in the system. In this section we repeat the measurements for the COUNT aggregate, only now varying the number of nodes in the system from 1,000 to 2,500, 5,000, and 10,000 nodes. The number of data tuples managed by the overlay is also scaled to either 30× or 300× the number of nodes (e.g. 30,000 and 300,000 tuples for the 1000-node network, 75,000 and 750,000 tuples for the 2,500-node network, etc).

Figures 18a, 19a, and 20a plot the per-node average insertion load, maximum node insertion load, and hop-count cost respectively for the 30× case—that is, the average number of times a node is hit during insertion of all tuples into the DHS synopsis, the maximum such number, and the average number of hops each node pays in order to insert all of its tuples into the DHS—for various combinations of input value distributions and number of bitmaps in the DHS. Similarly, Figs. 18b, 19b,and 20b depict the same metrics for the 300× case. As we can see, the per-node insertion load is nearly constant, while the maximum insertion load scales logarithmically to the number of nodes and sublinearly to the number of items in the overlay (the relevant figures for the rendezvous approach would obviously show a linear dependence on the number of nodes and independence on the number of items). Moreover, the per-node average insertion hop count grows also nearly logarithmically to the number of nodes and tuples in the system—there is less than double the hop-count cost for a ten-times increase in the number of nodes and a one-hundred-times increase in the number of tuples in the overlay!

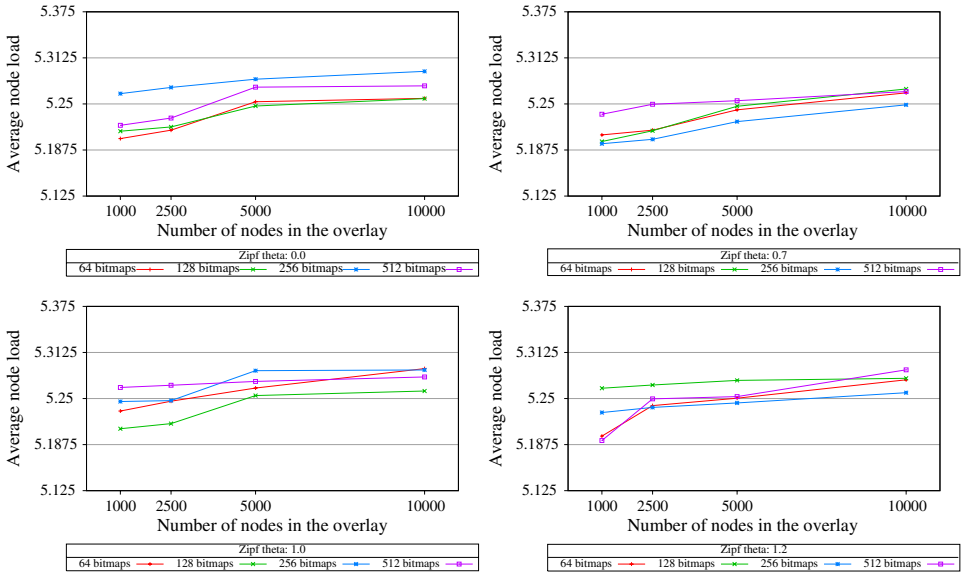
Likewise, Figs. 21a and 22a plot the average number of nodes visited and the hop-count cost for computing the COUNT aggregate for the 30× case, with the 300× case being depicted in Figs. 21b and 22b. From this set of figures we can conclude that the average query hop-count is independent on the number of items and grows logarithmically to the number of nodes in the system. The average query load also grows logarithmically to the number of nodes. Note, however, that our DHS-based solutions perform better, load-wise, for a larger number of items in the system, depending on the number of bitmaps used. This is due to the fact that the larger number of items inserted into the DHS-based synopsis combined with the lower number of bitmaps in the synopsis, means that less nodes will have to be visited because of retries (see Sect. 3.2.5). Average estimation error, Gini coefficient and Fairness index were as shown earlier in Figs. 8b, 12a and 12b, respectively. Last, the figures for the other aggregates were similar to these presented here for COUNT and are thus omitted.

6.2.2 Histograms

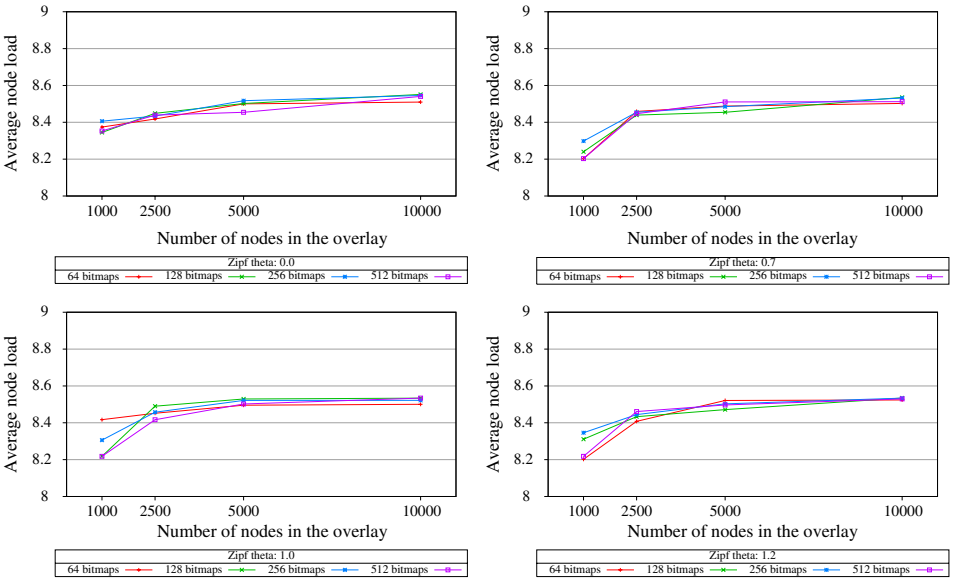
We would like to emphasize again that our key goal is distribution transparency. We are thus not concerned with the statistical properties of the histograms per se, or with whether a given histogram type is optimal or not; these have been extensively studied in the literature. Instead, we compare our results with histograms of the same type and characteristics built using a centralized approach.

Our experiments proceed as follows: (1) first, we generate the base data and compute the histogram(s) of choice over it in a centralized manner; (2) then, we configure and populate the simulated 1000-node network with peers and items as described earlier; (3) finally, we choose random nodes and have them reconstruct and/or infer the same set of histograms, following the approach outlined in Sect. 5. Note that we are dealing solely with single-dimensional histograms.

We first populate and reconstruct a 100-cell Equi-Width DHS-based histogram, and then use it to infer 10-bucket Equi-Depth, Compressed(V,F), and MaxDiff(V,F) histograms. In this part of our experimental evaluation we use only 256 bitmaps per DHS hash sketch, as this number poses a good trade-off between cost and accuracy, as showcased by the previous results. All relations have a single integer-valued attribute, with values drawn from the attribute value domain



(a) Number of data tuples = 30 × Number of nodes



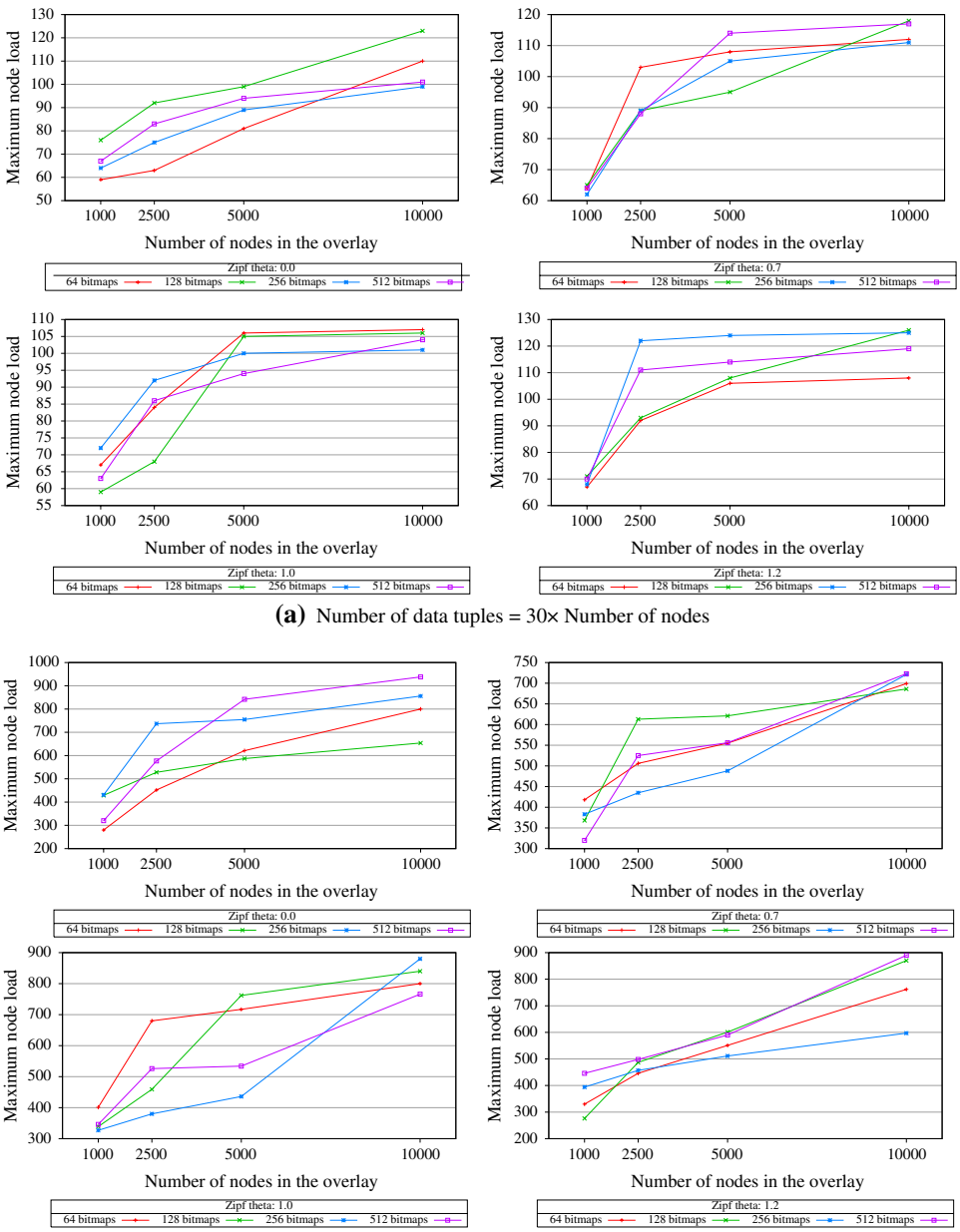
(b) Number of data tuples = 300 × Number of nodes

Fig. 18 Per-node average insertion load for a COUNT aggregate using the DHS, for various number of nodes in the overlay; a tenfold increase in the number of items leads to a ≈60% increase in the number of times each node is visited during the insertion phase

using either a uniform or a shuffled Zipf distribution with θ equal to 0.7, 1.0, and 1.2 [37,76].

For each of the inferred histogram types we measured the average cell frequency error versus the real attribute-value frequency distribution. For the Equi-Depth histogram, we further show what we call the "positioning error": the error at the estimation of the cell boundaries. For the latter two histogram types we also show the recall rate—the ratio of top frequencies or single-valued buckets (respectively) that were inferred correctly out of the 10 total. Finally, for the Compressed histograms we further show the "ranking error", that is the fraction of top-frequency values that were inferred at the correct ranks. Again, the reported values were averaged over several runs of the simulation.

Initially we used a 10-million tuples relation, with values in the interval [0, 1,000,000). Figure 23a summarizes the results from this set of experiments. With regard to the hop count costs and load distribution, the same argumentation holds as in the multiple aggregates case, as the proposed way of implementing distributed Equi-Width histograms is by using a different DHS "metric" or a different rendezvous ID per histogram bucket. This means that a rendezvous-based solution would have to either (ii) use multiple rendezvous nodes, thus incurring the outrageous hop-count costs and total query load depicted in Figs. 16 and 17, or (i) store all histogram buckets on a single node, hence creating a severe load imbalance in the system. We thus omit any further discussion of such costs in this part of



(a) Number of data tuples = 30× Number of nodes

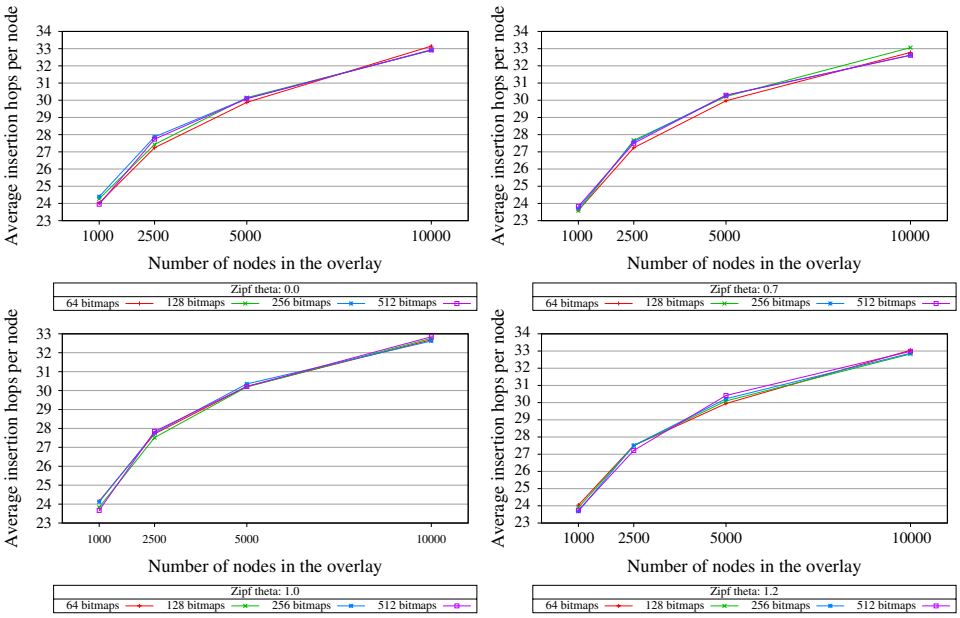
(b) Number of data tuples = 300× Number of nodes

Fig. 19 Maximum node insertion load for a COUNT aggregate using the DHS, for various number of nodes in the overlay; a tenfold increase in the data tuples in the overlay leads to a ≈6-fold increase in the maximum number any node is visited during the insertion phase, yet still being an order of magnitude lower than that of the rendezvous-based approach

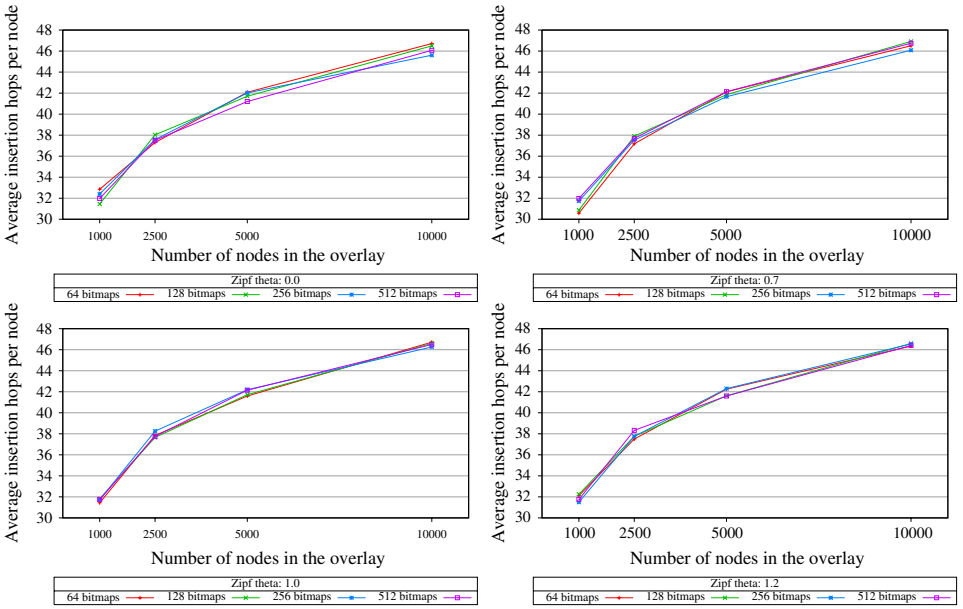
our experimental evaluation and concentrate on DHS-based histograms.

As far as accuracy is concerned, the average cell frequency error is nearly excellent for the Equi-Width, Average-Shifted and Equi-Depth histograms. The very small error in the Equi-Depth cases is somewhat artificial; since all buckets have approximately the same frequency there is little margin for error. What makes more sense in this case, is the cell boundary positioning error, depicted in Fig. 23b. Again, our algorithms achieve a nearly excellent accuracy, even in the highly skewed case where input values are drawn from a Zipf with θ equal to 1.2. On the cost side, the average hop count for reconstructing the initial DHS histogram is ≈26 hops (i.e. equaling the cost of merely ≈6 DHT lookups), while the average bandwidth consumption is ≈60–70 kbytes.

No results are shown in Fig. 23a for the MaxDiff(V, F) and Compressed(V, F) histograms, as they proved to be quite problematic. The main reason for this is that, with 100 buckets in the initial DHS histogram and 1 million values in the attribute value domain, 10,000 values are mapped into every histogram cell. This makes it nearly impossible to guess, based solely on the number of unique values in each



(a) Number of data tuples = 30× Number of nodes



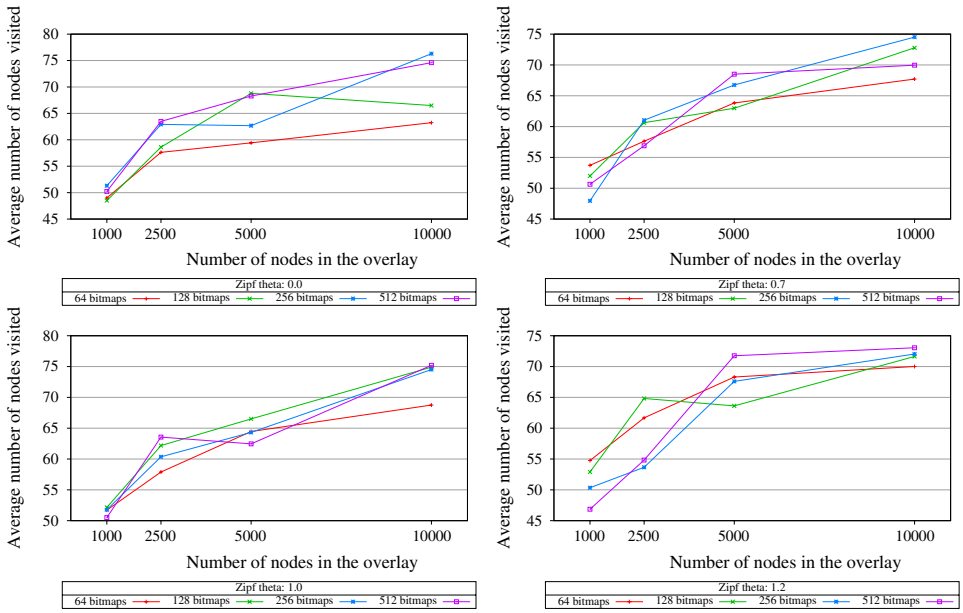
(b) Number of data tuples = 300× Number of nodes

Fig. 20 Per-node average average insertion hop-count cost for a COUNT aggregate using the DHS, for various number of nodes in the overlay; a tenfold increase in the number of tuples in the overlay leads to a ≈35% increase in the hop-count cost each node has to pay to insert all of its items in the DHS

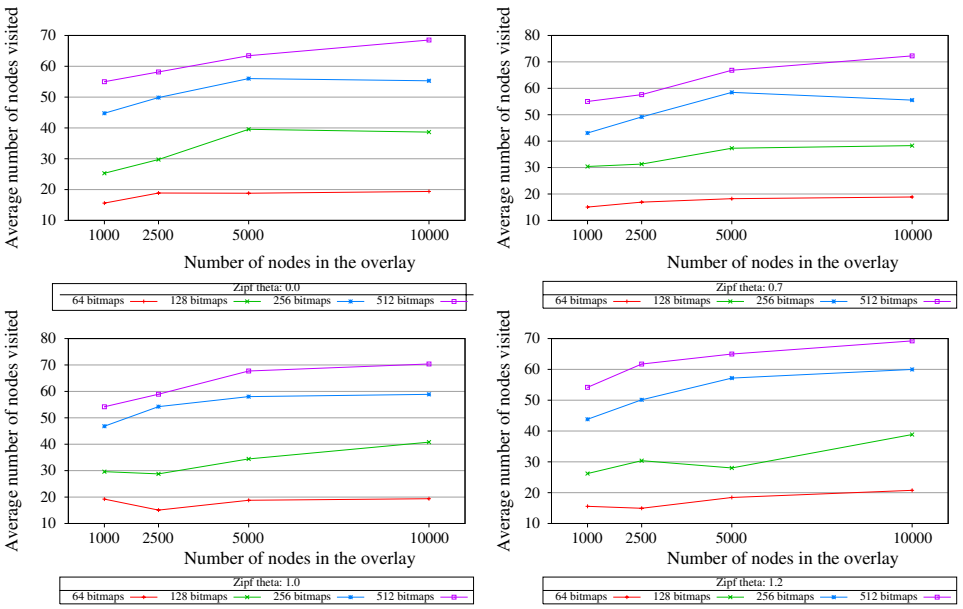
bucket and the bucket's overall frequency, the exact mapping of frequencies to values, required for these histogram types.

To this extent, we switched our experimental evaluation towards a different direction: again, we used a 1,000-node overlay and tried to build the same histograms as before, only now using a 10-million tuples relation, drawing tuple values from the interval [0,1,000) and building a 1,000-bucket initial DEWH histogram. This corresponds to having a separate DHS counter for every possible value in the attribute value domain.

The results are depicted in Figs. 24a–d. As we can see, the average cell frequency error is at the same levels as before (Fig. 24a), and so is also the cell boundary positioning error for the Equi-Depth histogram case (Fig. 24b). As far as the average ranking error (Fig. 24c) and recall rate (Fig. 24d) are concerned, we discern a certain trend: in both cases our algorithms lose in accuracy when the input values are drawn from the uniform distribution, and get better as the input distribution becomes more skewed. Note, however, that the strengths of these two histogram types manifest with skewed value distributions; if the tuple values are uniformly distributed over



(a) Number of data tuples = 30× Number of nodes



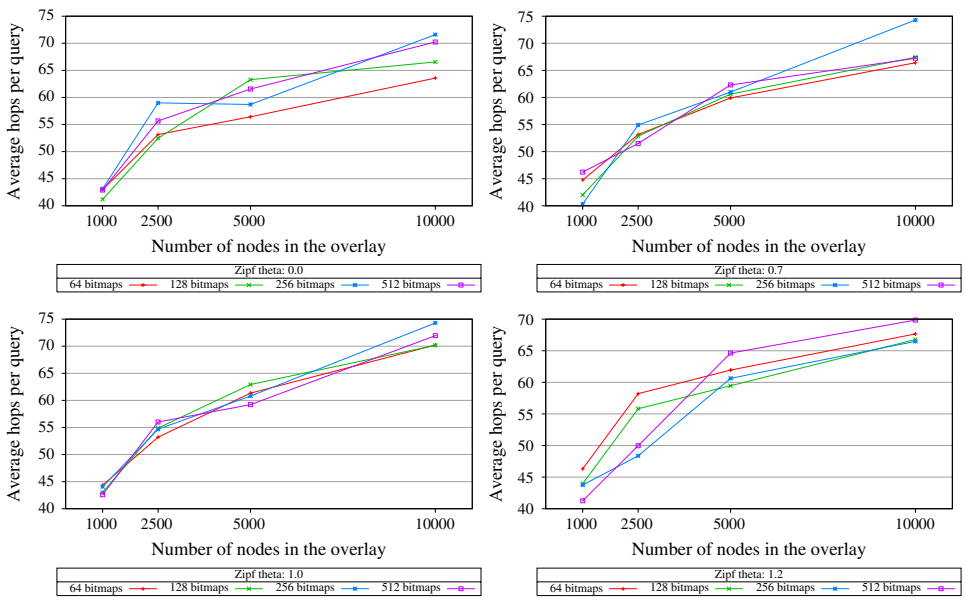
(b) Number of data tuples = 300× Number of nodes

Fig. 21 Average number of nodes visited when computing a COUNT aggregate using the DHS, for various number of nodes in the overlay

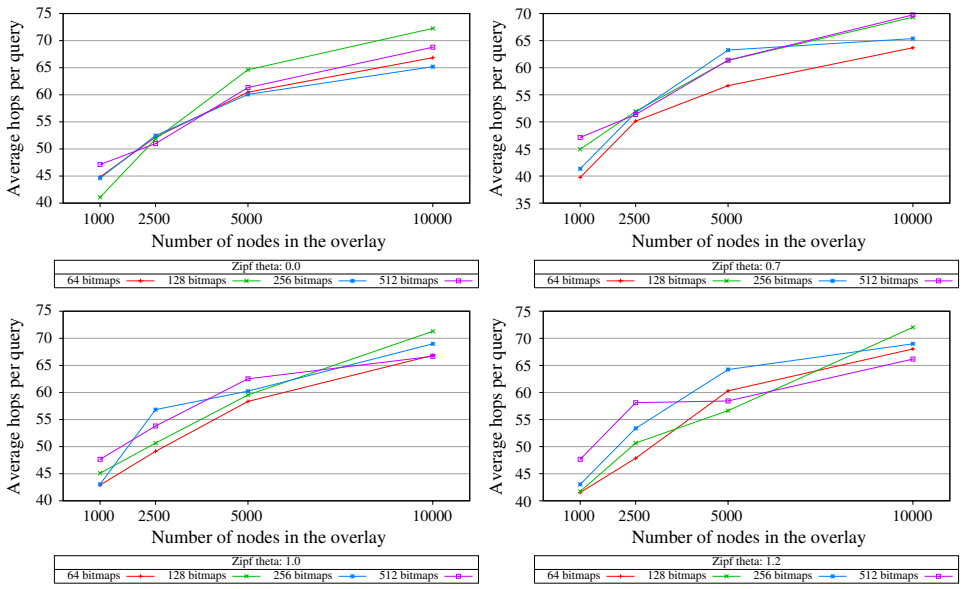
their value domain, then one gains nothing from using such a histogram, or any histogram at all to that extent!

However, the most interesting thing about this set of experiments is the hop-count cost for reconstructing the initial DEWH histogram: across all sets of parameters, this hop-count cost didn't exceed 30 hops (for an average of ≈27 hops)! The average bandwidth cost for reconstructing the base DEWH histogram was ≈600–700 kbytes. As already mentioned, the lurking trade-off here is related to bandwidth requirements; with 1000 buckets versus the 100 buckets we had earlier, each node visited during histogram reconstruction may respond with up to 10 times more data. Remember that this data is merely a set of <metricID, bit-vector> tuples whose size is very small (24–84 bytes per tuple in our real-world implementation).

This trade-off becomes much more apparent when dealing with large attribute value domains. For example, if attribute values were drawn from the interval [0, 1,000,000) so that the interval is densely populated (i.e. there were tuples for most of the 1M values in the domain) and we required a 1-million DEWH base histogram, then the histogram reconstruction bandwidth cost would go up to around 600–700 Mbytes! A first note here is that this figure refers to uncompressed DHS data; an appropriate compression algorithm could probably cut down bandwidth requirements by one to two orders of magnitude. Besides that, (1) to our knowledge, most realistic



(a) Number of data tuples = 30× Number of nodes



(b) Number of data tuples = 300× Number of nodes

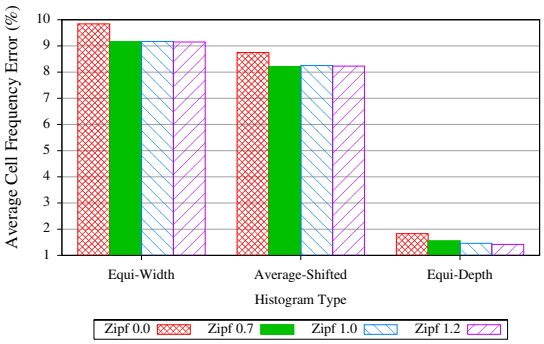
Fig. 22 Average hop-count cost per query when computing a COUNT aggregate using the DHS, for various number of nodes in the overlay

databases and workloads build indices and histograms for columns with up to a few (tens of) thousands of distinct values, (2) in most queries we shall not need information for all of the buckets but for a handful of them, and (3) even if the previous conditions do not hold, we can still build highly accurate Equi-Width, ASH, and Equi-Depth histograms over the base data with a very low hop-count and bandwidth consumption cost.

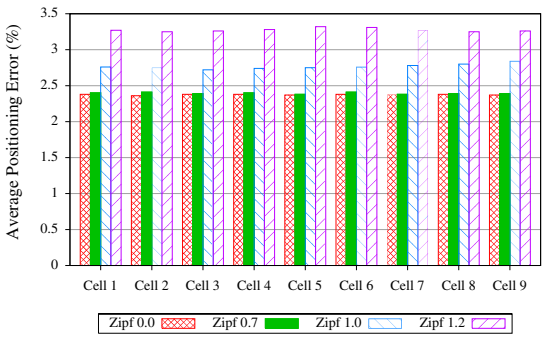
Discussion

We would like to pinpoint the necessity and appropriateness of the techniques presented, implemented, and evaluated in this work, for solving basic problems in the P2P DBMS realm. The above experimental results showed that the message count and bandwidth costs for reconstructing the base DEWH or DASH histograms and inferring the more complex histogram types can be quite low. Now, note that (1) after a node has paid the cost to reconstruct the histograms, choosing the optimal execution plan is a local operation, and (2) the above figures refer to the cost of reconstructing the whole histogram; query processing may require estimation of the cardinality of only specific buckets, depending on the query predicate constraints.

Now assume that a query optimizer, armed with the above histograms, is (at least) able to select the optimal query

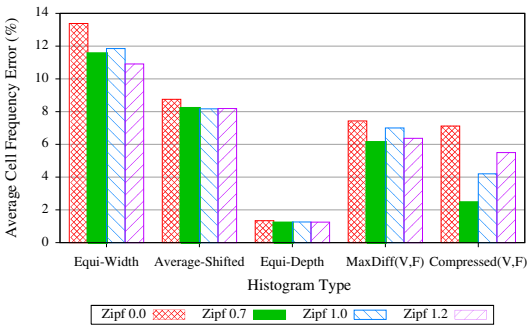


(a) Average cell frequency error

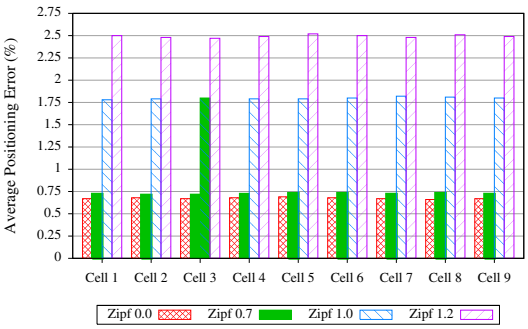


(b) Average cell boundary positioning error (Equi-Depth histogram case only)

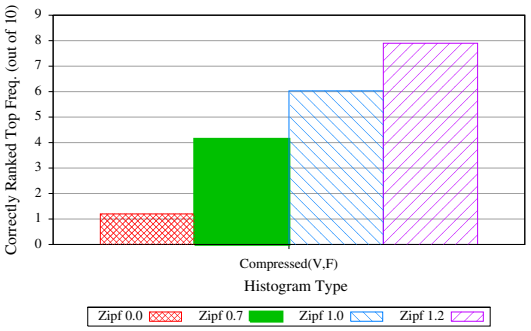
Fig. 23 Histogram inference results: 1,000 nodes, 10,000,000 items, values in [0, 1,000,000), 100 initial DHS buckets, 10 buckets per inferred histogram



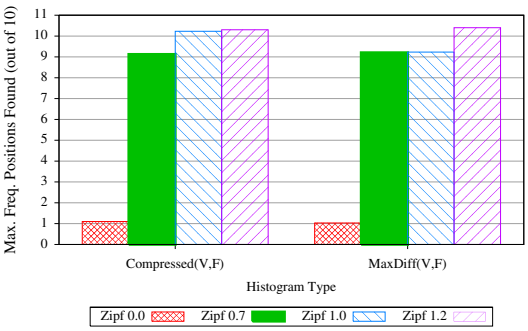
(a) Average cell frequency error



(b) Average cell boundary positioning error – Equi-Depth histogram case



(c) Average ranking error – Compressed(V,F) histogram case



(d) Average recall rate – Compressed(V,F) and MaxDiff(V,F) histogram cases

Fig. 24 Histogram inference results: 1,000 nodes, 10,000,000 items, values in [0, 1,000), 1,000 initial DHS buckets, 10 buckets per inferred histogram

execution plan. In this case, the savings in bandwidth and response time will be considerable [8,43]. For example, in [43] the authors consider multi-way joins in a much smaller setting than the one discussed above (256 nodes and relations with 256,000 tuples each or 100 tuples per node). The optimal join strategy in the three-way join case results in a data transfer of 47Mbytes, as opposed to 71Mbytes transferred by FREddies, both of which are orders of magnitude larger than the ≈0.7Mbytes required to reconstruct the histograms using DHS. Thus, for example, if DHS-based histograms were added to the PIER query processing logic, PIER would select the optimal join plan at a very reasonable additional cost, resulting in major overall bandwidth and latency savings. This gives an insight to the impact that DHS can have on Internet-scale query engines.

7 Conclusions

In this paper, we have developed a new framework for distributed statistical synopses that are particularly well suited for

Internet-scale overlay networks such as P2P systems. Our rationale has been to start with the best known techniques for centralized settings and extend them towards distributed settings with particular care about scalability.

Our contributions rest on utilizing specific statistical structures, such as hash sketches and distributed hash sketches, and on showing new methods for harnessing them to process important types of aggregate queries in a P2P environment. Using this substrate, we have shown how to develop DHT-based higher-level synopses such as Equi-Width, Average-Shifted Equi-Width, and Equi-Depth histograms, which figure prominently in the literature on traditional DBMS statistics management for query optimization. Our implementation and extensive performance evaluation show that these structures can be constructed and exploited efficiently and scale well with growing network size, while achieving high accuracy. The implementation of our solutions, over the popular stable DHT software FreePastry, is available as open source for anyone to use and test [29].

There are several routes to explore for future research. First, we would like to examine the design and implementation of auto-tuning capabilities for our histogram inference engine; these issues have been addressed in centralized database systems [1], but they are completely untouched by the P2P research community. Integrating our solutions with Internet-scale query processing systems, in the spirit of [42], comes up next in the list; we believe that the outcome would bring us closer to the elusive goal of completely self-organized P2P solutions to advanced data management. Third, we intend to look into implementing further types of synopses, aggregates, and histogram variants. Finally, using our tools for approximate query answering, in the sense of [2,3,13,50], would be a very intriguing direction as well.

Acknowledgments Nikos Ntarmos was supported by the 03ED719 research project, implemented within the framework of the "Reinforcement Program of Human Research Manpower" (PENED), co-financed by National and Community Funds (25% from the Greek Ministry of Development, General Secretariat of Research and Technology and 75% from the European Union, European Social Fund). Peter Triantafillou was funded by the 6th Framework Program of the European Union through the Integrated Project DELIS (#001907) on Dynamically Evolving Large-scale Information Systems. We would like to sincerely thank the associate editor and the reviewers for their careful reading of our paper and thoughtful and valuable comments.

References

1. Aboulnaga, A., Chaudhuri, S.: Self-tuning histograms: Building histograms without looking at data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1999)
2. Acharaya, S., Gibbons, P.B., Poosala, V., Ramaswamy, S.: The AQUA approximate query answering system. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1999)
3. Acharya, S., Gibbons, P., Poosala, V., Ramaswamy, S.: Join synopses for approximate query answering. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1999)
4. Babaoğlu, Ö., Meling, H., Montresor, A.: Anthill: A framework for the development of agent-based peer-to-peer systems. In: Proceedings of the IEEE International Conference on Distributed Computing and Systems (ICDCS) (2002)
5. Bawa, M., Garcia-Molina, H., Gionis, A., Motwani, R.: Estimating aggregates on a peer-to-peer network. Technical report, Computer Science Department, Stanford University (2003)
6. Bawa, M., Gionis, A., Garcia-Molina, H., Motwani, R.: The price of validity in dynamic networks. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (2004)
7. Beyer, K., Haas, P.J., Reinwald, B., Sismanis, Y., Gemulla, R.: On synopses for distinct-value estimation under multiset operations. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (2007)
8. Bharambe, A., Agrawal, M., Seshan, S.: Mercury: Supporting scalable multi-attribute range queries. In: Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM) (2004)
9. Blohsfeld, B., Korus, D., Seeger, B.: A comparison of selectivity estimators for range queries on metric attributes. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1999)
10. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst. (TOCS) **19**(3), 332–383 (2001)
11. Chaudhuri, S.: An overview of query optimization in relational systems. In: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS) (1998)
12. Chaudhuri, S., Das, G., Narasayya, V.R.: A robust, optimization-based approach for approximate answering of aggregate queries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (2001)
13. Chaudhuri, S., Das, G., Narasayya, V.R.: Optimized stratified sampling for approximate query processing. ACM Trans. Database Syst. (TODS) **32**(2), 9 (2007)
14. Chaudhuri, S., Motwani, R., Narasayya, R.: Random sampling for histogram construction: how much is enough? In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1998)
15. Chaudhuri, S., Narasayya, V.R.: Automating statistics management for query optimizers. IEEE Trans. Knowl. Data Eng. (TKDE) **13**(1), 7–20 (2001)
16. Considine, J., Li, F., Kollios, G., Byers, J.: Approximate aggregation techniques for sensor databases. In: Proceedings of the International Conference on Data Engineering (ICDE) (2004)
17. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. In: Proceedings of the Latin American Symposium on Theoretical Informatics (LATIN) (2004)
18. Cormode, G., Muthukrishnan, S.: Space efficient mining of multigraph streams. In: Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS) (2005)
19. Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M.F., Morris, R.: Designing a DHT for low latency and high throughput. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI) (2004)
20. Damgaard, C., Weiner, J.: Describing inequality in plant size or fecundity. Ecology **81**, 1139–1142 (2000)

21. Dobra, A.: Histograms revisited: When are histograms the best approximation method for aggregates over joins? In: Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS) (2005)
22. Dobra, A., Garofalakis, M., Gehrke, J., Rastogi, R.: Sketch-based multi-query processing over data streams. In: Proceedings of the International Conference on Extending Database Technology (EDBT) (2004)
23. Druschel, P., Rowstron, A.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proceedings of the IFIP/ACM IFIP/ACM International Conference on Distributed Systems Platforms (Middleware) (2001)
24. Durand, M., Flajolet, P.: Loglog counting of large cardinalities. In: Proceedings of the Annual European Symposium on Algorithms (ESA) (2003)
25. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. In: Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM) (1998)
26. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci. **31**(2), 182–209 (1985)
27. FreeDHS: Homepage (2006). http://netcins.ceid.upatras.gr/DHS.php
28. FreePastry: Homepage (2002). http://freepastry.org/FreePastry/
29. FreeSHADE (2006). http://netcins.ceid.upatras.gr/SHADE.php
30. Ganguly, S., Garofalakis, M., Rastogi, R.: Processing set expressions over continuous update streams. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (2003)
31. Garofalakis, M. (ed.): Special issue on in-network query processing. IEEE Data Eng. Bull. **28**(1) (2005)
32. Gibbons, P.B., Matias, Y., Poosala, V.: Fast incremental maintenance of approximate histograms. ACM Trans. Database Syst. (TODS) **27**(3), 261–298 (2002)
33. Gray, J., Liu, D.T., Nieto-Santisteban, M.A., Szalay, A.S., DeWitt, D.J., Heber, G.: Scientific data management in the coming decade. SIGMOD Record **34**(4), 34–41 (2005)
34. Gribble, S., Halevy, A., Ives, Z., Rodrig, M., Suciu, D.: What can peer-to-peer do for databases, and vice versa? In: Proceedings of the International Workshop on the Web and Databases (WebDB) (2001)
35. Guha, S., Koudas, N., Shim, K.: Data-streams and histograms. In: Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) (2001)
36. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM) (2003)
37. Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), pp. 314–329 (2003)
38. Haas, P., Naughton, J.F., Seshadri, S., Stokes, L.: Sampling-based estimation of the number of distinct values of an attribute. In: Proceedings of the International Conference on Very Large Data Bases (VLDB) (1995)
39. Hadjieleftheriou, M., Byers, J.W., Kollios, G.: Robust sketching and aggregation of distributed data streams. Technical Report 2005-011, Computer Science Department, Boston University (2005)
40. Hellerstein, J.M., Condie, T., Garofalakis, M., Loo, B.T., Maniatis, P., Roscoe, T., Taft, N.: Public health for the internet (PHI). In: Proceedings of the ACM SIGMOD/VLDB Biennial Conference on Innovative Data Systems Research (CIDR) (2007)
41. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1997)
42. Huebsch, R., Chun, B.N., Hellerstein, J.M., Loo, B.T., Maniatis, P., Roscoe, T., Shenker, S., Stoica, I., Yumerefendi, A.R.: The architecture of PIER: an internet-scale query processor. In: Proceedings of the ACM SIGMOD/VLDB Biennial Conference on Innovative Data Systems Research (CIDR) (2005)
43. Huebsch, R., Jeffery, S.: FREddies: DHT-based adaptive query processing via FedeRated Eddies. Technical Report UCB/CSD-4-1339, University of Californa at Berkeley (2004)
44. Ioannidis, Y.: The history of histograms (abridged). In: Proceedings of the International Conference on Very Large Data Bases (VLDB) (2003)
45. Ioannidis, Y., Christodoulakis, S.: Optimal histograms for limiting worst-case error propagation in the size of join results. ACM Trans. Database Syst. (TODS) **18**(4), 709–748 (1993)
46. Ioannidis, Y., Poosala, V.: Balancing histogram optimality and practicality for query result size estimation. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1995)
47. Jain, R., Chiu, D., Hawe, W.: A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report TR-301, DEC (1984)
48. Jelasity, M., Montresor, A.: Epidemic-style proactive aggregation in large overlay networks. In: Proceedings of the IEEE International Conference on Distributed Computing and Systems (ICDCS) (2004)
49. Jelasity, M., Montresor, A., Babaoğlu, Ö.: Gossip-based aggregation in large dynamic networks. ACM Trans. Comput. Syst. (TOCS) **23**(3), 219–252 (2005)
50. Jermaine, C.M., Arumugam, S., Pol, A., Dobra, A.: Scalable approximate query processing with the DBO engine. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (2007)
51. Kempe, D., Dobra, A., Gehrke, J.: Computing aggregate information using gossip. In: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS) (2003)
52. Kossmann, A.: The state of the art in distributed query processing. ACM Comput. Surv. **32**(4), 422–469 (2000)
53. Koudas, N. (ed.): Special issue on data quality. IEEE Data Eng. Bull. **29**(2) (2006)
54. Krishnan, P.: Online prediction algorithms for databases and operating systems. Ph.D. thesis, Brown University (1995)
55. Li, J., Loo, B.T., Hellerstein, J.M., Kaashoek, M.F., Karger, D., Morris, R.: On the feasibility of peer-to-peer web indexing and search. In: Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS) (2003)
56. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: A Tiny AGgregation service for ad-hoc sensor networks. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI) (2002)
57. Manku, G.: Routing networks for Distributed Hash Tables. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC) (2003)
58. Markl, V., Haas, P.J., Kutsch, M., Megiddo, N., Srivastava, U., Tran, T.M.: Consistent selectivity estimation via maximum entropy. VLDB J. **16**(1), 55–76 (2007)
59. Matia, Y., Matias, Y.: Calibration and profile based synopses error estimation and synopses reconciliation. In: Proceedings of the International Conference on Data Engineering (ICDE) (2007)
60. Maymouknov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS) (2002)

61. Michel, S., Bender, M., Ntarmos, N., Triantafillou, P., Weikum, G., Zimmer, C.: Discovering and exploiting keyword and attribute-value co-occurrences to improve P2P routing indices. In: Proceedings of the ACM Conference on Information and Knowledge Management (CIKM) (2006)
62. Michel, S., Triantafillou, P., Weikum, G.: KLEE: A framework for distributed top-k query algorithms. In: Proceedings of the International Conference on Very Large Databases (VLDB) (2005)
63. Montresor, A., Jelasity, M., Babaoğlu, Ö.: Robust aggregation protocols for large-scale overlay networks. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN) (2004)
64. Montresor, A., Meling, H., Babaoğlu, Ö.: Messor: Load-balancing through a swarm of autonomous agents. In: Proceedings of the Workshop on Agent and Peer-to-Peer Systems (2002)
65. Morris, R.: Counting large numbers of events in small registers. Commun. ACM **21**(10), 840–842 (1978)
66. Munro, J.I., Paterson, M.S.: Selection and sorting with limited storage. Theor. Comput. Sci. **12**(3), 315–323 (1980)
67. Muralikrishna, M., DeWitt, D.J.: Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1988)
68. Ntarmos, N., Triantafillou, P., Weikum, G.: Counting at large: Efficient cardinality estimation in internet-scale data networks. In: Proceedings of the International Conference on Data Engineering (ICDE) (2006)
69. Palmer, C.R., Siganos, G., Faloutsos, M., Faloutsos, C., Gibbons, P.B.: The connectivity and fault-tolerance of the internet topology. In: Proceedings of the Workshop on Network-Related Data Management (NRDM) (2001)
70. Pitoura, T., Triantafillou, P.: Load distribution fairness in p2p data management systems. In: Proceedings of the International Conference on Data Engineering (ICDE) (2007)
71. Poosala, V., Ganti, V., Ioannidis, Y.: Approximate query answering using histograms. IEEE Data Eng. Bull. **22**(4), 5–14 (1999)
72. Poosala, V., Ioannidis, Y.: Estimation of query-result distribution and its application in parallel-join load balancing. In: Proceedings of the International Conference on Very Large Data Bases (VLDB) (1996)
73. Poosala, V., Ioannidis, Y.: Selectivity estimation without the attribute value independence assumption. In: Proceedings of the International Conference on Very Large Data Bases (VLDB) (1997)
74. Poosala, V., Ioannidis, Y., Haas, P., Shekita, E.: Improved histograms for selectivity estimation of range predicates. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1996)
75. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM) (2001)
76. Reynolds, P., Vahdat, A.: Efficient peer-to-peer keyword searching. In: Proceedings of the IFIP/ACM IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp. 21–40 (2003)
77. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a DHT. In: Proceedings of the USENIX Annual Technical Conference (USENIX) (2004)
78. Scott, D.W.: Average shifted histograms: effective nonparametric density estimators in several dimensions. Ann. Stat. **13**(3), 1024–1040 (1985)
79. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1979)
80. Shapiro, G.P., Connell, C.: Accurate estimation of the number of tuples satisfying a condition. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (1984)
81. Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S.: Medians and beyond: New aggregation techniques for sensor networks. In: Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys) (2004)
82. Steinmetz, R., Wehrle, K. (eds.): Peer-to-Peer Systems and Applications. Springer, Berlin (2005)
83. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM) (2001)
84. Suciu, D. (ed.): Special issue on web-scale data, systems, and semantics. IEEE Data Eng. Bull. **29**(4) (2006)
85. Tao, Y., Kollios, G., Considine, J., Li, F., Papadias, D.: Spatio-temporal aggregation using sketches. In: Proceedings of the International Conference on Data Engineering (ICDE) (2004)
86. Thaper, N., Guha, S., Indyk, P., Koudas, N.: Dynamic multidimensional histograms. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (2002)
87. Triantafillou, P., Economides, A.: Subscription summarization: A new paradigm for efficient publish/subscribe systems. In: Proceedings of the IEEE International Conference on Distributed Computing and Systems (ICDCS) (2004)
88. van Renesse, R., Birman, K.P., Vogels, W.: Astrolabe: a robust and scalable technology for distributed system monitoring, management, and data mining. ACM Trans. Comput. Syst. (TOCS) **21**(2), 164–206 (2003)
89. Yalagandula, P., Dahlin, M.: A scalable distributed information management system. In: Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM) (2004)