# Skew in Parallel Query Processing

Paul Beame, Paraschos Koutris and Dan Suciu
University of Washington, Seattle, WA
{beame,pkoutris,suciu}@cs.washington.edu

## ABSTRACT

We study the problem of computing a conjunctive query $q$ in parallel, using $p$ of servers, on a large database. We consider algorithms with one round of communication, and study the complexity of the communication. We are especially interested in the case where the data is skewed, which is a major challenge for scalable parallel query processing. We establish a tight connection between the *fractional edge packing* of the query and the *amount of communication* in two cases. First, in the case when the only statistics on the database are the cardinalities of the input relations, and the data is skew-free, we provide matching upper and lower bounds (up to a polylogarithmic factor of $p$) expressed in terms of fractional edge packings of the query $q$. Second, in the case when the relations are skewed and the heavy hitters and their frequencies are known, we provide upper and lower bounds expressed in terms of packings of *residual queries* obtained by specializing the query to a heavy hitter. All our lower bounds are expressed in the strongest form, as number of bits needed to be communicated between processors with unlimited computational power. Our results generalize prior results on uniform databases (where each relation is a matching) [4], and lower bounds for the MapReduce model [1].

## Categories and Subject Descriptors

H.2.4 [**Systems**]: Parallel Databases

## Keywords

Parallel Computation; Skew; Lower Bounds

## 1. INTRODUCTION

While in traditional query processing the main complexity is dominated by the disk access time, in modern massively distributed systems the dominant cost is that of the communication. A data analyst will use a cluster with sufficiently many servers to ensure that the entire data fits in main memory. Unlike MapReduce [6], which stores data on disk between the Map and the Reduce phase for recovery purposes, newer systems like Spark [17] and its SQL-extension Shark [14] perform the entire computation in main memory, and use replay to recover. In these systems the new complexity parameter is the communication cost, which depends on both the amount of data sent and the number of rounds.

A key requirement in such systems is that the data be uniformly partitioned on all servers, and this requirement is challenging to enforce when the input data is *skewed*. A value in the database is skewed, and is called a *heavy hitter*, when it occurs with much higher frequency than some predefined threshold. Since data reshuffling is typically done using hash-partitioning, all records containing a heavy hitter will be sent to the same server, causing it to be overloaded. Skew for parallel joins has been studied intensively since the early days of parallel databases, see [13]. The standard parallel join algorithm that handles skew is the *skew join* [11], which consists of first detecting the heavy hitters (e.g. using sampling), then treating them differently from the others values, e.g. by partitioning tuples with heavy hitters on the other attributes; a detailed description is in [15]. None of these algorithms has been proven to be optimal in any formal sense, and to the best of our knowledge there are no lower bounds for the communication required to compute a join in the presence of skew.

Complex queries often involve multiple joins, and the traditional approach that computes one join at a time leads to a number of communication rounds at least as large as the depth of the query plan. It is possible, however, to compute a multiway join in a single communication round, using a technique that can be traced back to Ganguli, Silberschatz, and Tsur [8, Sec.7], and was described by Afrati and Ullman [2] in the context of MapReduce algorithms. We will refer to this technique as the HyperCube *algorithm*, following [4]. The $p$ servers are organized into a hypercube with $k$ dimensions, where $k$ is the number of variables in the query. During a single reshuffling step, every tuple is sent to all servers in a certain subcube of the hypercube (with as many dimensions as the number of query variables missing from the tuple). One challenge in this approach is to determine the size of the hypercube in each of the $k$ dimensions. In [2] this is treated as a non-linear optimization problem and solved using Lagrange multipliers. In [4] it is shown that, in the case where all relations have the same cardinality, the optimal dimensions can be expressed in terms of the optimal fractional vertex cover of the query. All hypercube-based techniques described in [2, 4], and elsewhere (e.g. in [12] for computing triangles) assume that the


*PODS'14,* June 22–27, 2014, Snowbird, UT, USA.

ACM 978-1-4503-2375-8/14/06 ...$15.00.
http://dx.doi.org/10.1145/2594538.2594558 .

data has no skew. The behavior of the algorithm on skewed data has not been studied before and no techniques for addressing skew have been proposed.

**Our Contribution.** In this paper we study the problem of computing a full conjunctive query (multi-way join query) when the input data may have arbitrary skew. We are given $p$ servers that have to compute a query on a database with $m$ tuples; we assume that $m \gg p$. We prove upper and lower bounds for the amount of communication needed to compute the query in one round. We assume the following statistics on the input database to be known: the cardinality of each input relation, the identity of the heavy hitters, and their (approximate) frequency in the data. We note that this is a reasonable assumption in today's distributed query engines. In our settings there are at most $O(p)$ heavy hitters because we choose a threshold for the frequency of heavy hitters that is $\geq m/p$, and therefore the number of heavy hitters is tiny compared to the size of the database. We assume that at the beginning of the computation all servers know the identity of all heavy hitters, and the (approximate) frequency of each heavy hitter. Given these statistics, we present upper and lower bounds for the amount of communication needed to compute the query on the class of databases satisfying those statistics. There is a small gap remaining between the upper and lower bound; for the join query, however, we prove that the bounds match. Our results are significant extensions of our previous results [4], which hold only in the absence of skew and for relations of the same cardinality.

Grohe and Marx [9] and Atserias et al. [3] give upper bounds on the query size in terms of a fractional *edge cover*; this is also a lower bound on the running time of any sequential algorithm that computes the query. Recently, Ngo at al. [10] described a sequential algorithm that matches that bound. Thus, the sequential complexity of a query is captured by the edge cover; our results show that the communication complexity for parallel evaluation is instead captured by the *edge packing*.

**Overview of the results.** Our analysis of skew starts with an analysis of skew-free databases, but with unequal cardinalities. Consider a simple cartesian product, $q(x, y) = S_1(x), S_2(y)$, of two relations with cardinalities $m_1, m_2$. Assume $1/p \leq m_1/m_2 \leq p$[1]. Let $p_1 = \sqrt{m_1 p/m_2}$, $p_2 = \sqrt{m_2 p/m_1}$ and assume that they are integer values. Organize the $p$ servers into a $p_1 \times p_2$ rectangle, and assign to each server two coordinates $(i, j) \in [p_1] \times [p_2]$. During the communication phase, the algorithm uses two random hash functions $h_1, h_2$ and sends every tuple $S_1(x)$ to all servers with coordinates $(h_1(x), j)$, $j \in [p_2]$ (thus, every server receives with high probability $O(m_1/p_1) = O(\sqrt{m_1 m_2/p})$ tuples from $S_1$), and sends every tuple $S_2(y)$ to all servers with coordinates $(i, h_2(y))$, $i \in [p_1]$. The load per server is $L = O(\sqrt{m_1 m_2/p})$, and it is not hard to see that this is optimal[2]. This observation generalises to any $u$-way cartesian product: the minimum load per server needed to compute $S_1 \times \ldots \times S_u$ is $\Omega((m_1 m_2 \cdots m_u/p)^{1/u})$

Consider now some arbitrary full conjunctive query $q$ over relations $S_1, \ldots, S_\ell$, and assume that the cardinality of $S_j$ is $m_j$. Choose some subset $S_{j_1}, S_{j_2}, \ldots, S_{j_u}$; the subset is called an *edge packing*, or an *edge matching*, if no two relations share a common variable. Any one-round algorithm that computes the query correctly must also compute the cartesian product of the relations $S_{j_1}, S_{j_2}, \ldots, S_{j_u}$. Indeed, since no two relations share variables, any tuple in their cartesian product could potentially be part of the query answer; without knowing the content of the other relations, the input servers that store (fragments of) $S_{j_1}, S_{j_2}, \ldots$ must ensure that any combination reaches some output server. Therefore, the load per server of any one-round algorithm is at least $\Omega((m_{j_1} \cdots m_{j_u}/p)^{1/u})$. Thus, every edge packing gives a lower bound for computing $q$. For example the load per server needed to compute the query $q(x, y, z, w) = S_1(x, y), S_2(y, z), S_3(z, w)$ is at least $L \geq \sqrt{m_1 m_3/p}$, because of the packing $\{S_1, S_3\}$; it must also be $L \geq m_2/p$, because of the packing $\{S_2\}$. We prove in this paper that this property extends to any *fractional edge packing*. Let $M_j$ denote the number of bits needed to represent the relation $S_j$. We show:

THEOREM 1.1. *Let $\mathbf{u} = (u_1, \ldots, u_\ell)$ be any fractional edge packing for the query $q$, and $u = \sum_j u_j$. Let $K(\mathbf{u}, \mathbf{M}) = \prod_j M_j^{u_j}$ and $L(\mathbf{u}, \mathbf{M}, p) = (K(\mathbf{u}, \mathbf{M})/p)^{1/u}$. If an algorithm computes $q$ in one step, then at least one server has a load $\Omega(L(\mathbf{u}, \mathbf{M}, p))$. Conversely, let $L_{\mathtt{lower}} = \max_{\mathbf{u}} L(\mathbf{u}, \mathbf{M}, p)$ be the maximum over all fractional edge packings. Then, there exists a randomized algorithm for $q$ (HYPERCUBE or HC algorithm) whose maximum load per server is $O(L_{\mathtt{lower}} \ln^k p)$ with high probability on all databases without skew.*

In the case when all relations have the same size $M$, then the lower bound is $L_{\mathtt{lower}} = \max_{\mathbf{u}}(M/p^{1/u}) = M/p^{1/\tau^*}$, where $\tau^*$ is the value of the maximal fractional edge packing, and is equal to the *fractional vertex covering* number for $q$; thus, we recover our prior result in [4], which was stated for the special case when all relations are *matchings* and have equal cardinalities.

Theorem 1.1 completes the analysis of the HC algorithm on skew-free databases with arbitrary cardinalities. In addition, we prove a rather surprising result: the HC algorithm is resilient to skew, in the sense that, even on skewed databases, it can still offer a non-trivial upper bound for the maximum load per server: namely $L = O(M/p^{1/k})$, where $M$ is the size of the largest relation, and $k$ the total number of variables in the query. For example, using HC one can compute the join of two relations and guarantee a maximum load of $O(M/p^{1/3})$, *even without any knowledge about skew or heavy hitters*. In contrast, a standard hash-join algorithm may incur a load of $\Omega(M)$ when the join attributes have a single value.

Next, we consider the case when information about heavy hitters is known. In addition to knowing the cardinalities of the input relations, we assume that the identities of the heavy hitters are known, and that the frequency in the data of every heavy hitter is also known. For example, if the relation $S_j$ contains an attribute $x$, then we assume to know the set of heavy hitters $H$, together with the frequencies $m_j(h) = |\sigma_{x=h}(S_j)|$, which, by definition, are $\geq m_j/p$.

[1] if $m_1 < m_2/p$ then we can broadcast $S_1$ to all servers and compute the query with a load increase of at most $m_2/p$ per server thus at most double that of any algorithm, because $m_2/p$ is the load required to store $S_2$.

[2] Let $a_i, b_i$ be the number of $S_1$-tuples and $S_2$-tuples received by server $i \in [p]$. On one hand $\sum_i a_i b_i = \langle \bar{a}, \bar{b} \rangle \geq m_1 m_2$ because the servers must report all $m_1 m_2$ tuples; on the other hand $\langle \bar{a}, \bar{b} \rangle \leq \|\bar{a} + \bar{b}\|_2^2/4 \leq p\|\bar{a} + \bar{b}\|_\infty^2/4 = pL^2/4$.

For this setting, we generalize the results for skew-free databases by proving both lower bound and upper bounds. Our lower bound is an elegant generalization of that in Theorem 1.1, and is expressed in terms of fractional edge packings of *residual queries*: for each set of variables $\mathbf{x}$, the residual query $q_{\mathbf{x}}$ is obtained from $q$ by simply removing the variables $\mathbf{x}$. The upper bound is based on the idea of running the main query on the subset of the database that consists of light hitters, then handling each heavy hitter separately, by computing a residual query. The algorithm is difficult, because of two challenges. First, one needs to consider sets of attributes of each relation $S_j$ that may be heavy hitters jointly, even if none of them is a heavy hitter by itself. Second, an attribute value may become a heavy hitter in the residual query even though it was light in the main query. Our algorithm addresses these challenges by creating, for each subset of attributes of each relation, $O(\log p)$ bins of heavy hitters, where all heavy hitters in a bin have frequencies that differ by at most a factor of two (because of this it suffices for our algorithm to have access only to approximate frequencies of heavy hitters). By considering separately all combinations of bins, we can run residual queries on databases where the frequencies are guaranteed to be uniform, thus avoiding the difficulties that arise from recursion. Denote $M_j(h)$ the number of bits needed to represent the subset $\sigma_{x=h}(S_j)$ of $S_j$. Our second main result is:

THEOREM 1.2. *Consider all database instances defined by a set of statistics consisting of the cardinalities of the relations, the set of heavy hitters, and the frequency of each heavy hitter. For a set of variables* $\mathbf{x}$ *and any packing* $\mathbf{u}$ *of the residual query* $q_{\mathbf{x}}$ *that saturates the variables in* $\mathbf{x}$*, let* $L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p) = \left(\sum_{\mathbf{h}} K(\mathbf{u}, \mathbf{M}(\mathbf{h}))/p\right)^{1/u}$*: then any deterministic algorithm that computes* $q$ *on these databases must have a load* $\geq L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p)$.

*Moreover, there exists a randomized algorithm for computing* $q$ *with maximum load* $O(L_{upper} \log^{O(1)} p)$ *with high probability, where* $L_{upper} = \max_{\mathbf{x},\mathbf{u}} L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p)$ *and* $\mathbf{u}$ *now ranges over all packings of the residual query* $q_{\mathbf{x}}$.

The gap between the upper and lower bound comes from the fact that in the upper bound the possible edge packings for $q_{\mathbf{x}}$ are not restricted to only those which saturate the variables in $\mathbf{x}$.

As a final contribution of our paper, we discuss the connection between our results in the MPC model and the results of [1] on models for computation in MapReduce. We show that our results provide new upper and lower bounds for computing conjunctive queries in [1], and in a stronger computational model.

The paper is organized as follows. We describe the computational model and review the basic definitions from [4] in section 2, then present in section 3 our results for the case when the statistics known about the database are restricted to cardinalities. The case of databases with known heavy hitters is discussed in section 4. We present the connection with [1] in section 5 and finally conclude in section 6. Several proofs are relegated to the full version of the paper [5].

# 2. PRELIMINARIES

## 2.1 Massively Parallel Communication

We define here the MPC model. The computation is performed by $p$ servers connected by a complete network of private[3] channels. The input data is initially distributed evenly among the $p$ servers. The computation proceeds in rounds, where each round consists of *local computation* at the servers interleaved with *global communication*. The servers have unlimited computational power, but may be limited in the amount of bits they receive. In this paper, we discuss query evaluation in this model, and consider a single round of communication. The *load* of a server is the number of bits received by the server during the communication; we write $L$ for the maximum load among all servers.

If the size of the input is $M$ bits, an ideal algorithm would split the data equally among the servers, and so we would like to have $L = M/p$; in this case, the total amount of data communicated is $M$ and thus there is no *replication*. Depending on the query, $L$ is higher than the ideal $M/p$ by some factor called *replication factor*. In [4] we considered the case when the input data is perfectly uniform and all relations have the same size, and showed that the replication factor for any conjunctive query is $O(p^{\varepsilon})$, where $0 < \varepsilon \leq 1$ is a constant that depends only on the query. In this work we consider arbitrary input data, and the replication factor becomes a more complex formula that depends on the database statistics.

**Randomization.** The MPC model allows randomization during the computation. The random bits are available to all servers at the beginning of computation, and are independent of the input data.

**Random Instances and Yao's Lemma.** Our lower bounds are stated by showing that, if the database instance is chosen at random from some known probability space, then any algorithm with a load less than a certain bound can report only $o(1)$ fraction of the expected number of answers to the query. Using Yao's lemma [16] this implies that for any randomized algorithm there exists an instance on which the algorithm will fail with high probability; see [4] for details.

**Input Servers.** In our upper bounds we assume that the input relations $S_j$ are initially partitioned uniformly on the servers: all our algorithms treat tuples in $S_j$ independently of other tuples. For our lower bounds, we assume a more powerful model, where at the beginning of the algorithm each relation $S_j$ is stored on a separate server, called an *input server*, which can examine the entire relation in order to determine what message to send. These assumptions are the same as in [4].

**Database Statistics.** In this paper we assume that all input servers know certain database statistics. *Simple database statistics* consists of the cardinalities $m_j$ of all input relations $S_j$; we discuss this case in section 3. *Complex database statistics* add information about heavy hitters; we discuss these in the rest of the paper. The size of these statistics is $O(1)$ in the first case, and $O(p)$ in the second. Both upper and lower bounds assume that these statistics are available to all input servers.

## 2.2 Conjunctive Queries

We study the problem of computing answers to conjunctive queries over an input database in the MPC model. We fix an input vocabulary $S_1, \ldots, S_\ell$, where each relation $S_j$ has arity $a_j$; let $a = \sum_{j=1}^{\ell} a_j$. The input data consists of one relation instance for each symbol. We consider full con-

[3]"Private" means that when server $i$ sends a message to server $j$ no other server sees its content.

junctive queries without self-joins[4]:

$$q(x_1, \ldots, x_k) = S_1(\bar{x}_1), \ldots, S_\ell(\bar{x}_\ell) \quad (1)$$

The query is *full*, meaning that every variable in the body appears in the head (for example $q(x) = S(x, y)$ is not full), and *without self-joins*, meaning that each relation name $S_j$ appears only once (for example $q(x, y, z) = S(x, y), S(y, z)$ has a self-join). The *hypergraph* of a query $q$ is defined by introducing one node for each variable in the body and one hyperedge for each set of variables that occur in a single atom. With some notational abuse we write $i \in S_j$ to mean that the variable $x_i$ occurs in the the variables $vars(S_j)$ of the atom $S_j$.

**Fractional Edge Packing.** A *fractional edge packing* (also known as a *fractional matching*) of a query $q$ is any feasible solution $\mathbf{u} = (u_1, \ldots, u_\ell)$ of the following linear constraints:

$$\forall i \in [k] : \sum_{j: i \in S_j} u_j \leq 1 \quad (2)$$
$$\forall j \in [\ell] : u_j \geq 0$$

The edge packing associates a non-negative weight $u_j$ to each atom $S_j$ such that for every variable $x_i$, the sum of the weights for the atoms that contain $x_i$ do not exceed 1. If all inequalities are satisfied as equalities by a solution to the LP, we say that the solution is *tight*.

For a simple example, an edge packing of the query $L_3 = S_1(x_1, x_2), S_2(x_2, x_3), S_3(x_3, x_4)$ is any solution to $u_1 \leq 1$, $u_1 + u_2 \leq 1$, $u_2 + u_3 \leq 1$ and $u_3 \leq 1$. In particular, the solution $(1, 0, 1)$ is a tight and feasible edge packing. A *fractional edge cover* is a feasible solution $\mathbf{u} = (u_1, \ldots, u_\ell)$ to the system above where $\leq$ is replaced by $\geq$ in Eq.2. Every tight fractional edge packing is a tight fractional edge cover, and vice versa.

## 2.3 Friedgut's Inequality

Friedgut [7] introduces the following class of inequalities. Each inequality is described by a hypergraph, which in our paper corresponds to a query, so we will describe the inequality using query terminology. Fix a query $q$ as in (1), and let $n > 0$. For every atom $S_j(\bar{x}_j)$ of arity $a_j$, we introduce a set of $n^{a_j}$ variables $w_j(\mathbf{a}_j) \geq 0$, where $\mathbf{a}_j \in [n]^{a_j}$. If $\mathbf{a} \in [n]^k$, we denote by $\mathbf{a}_j$ the vector of size $a_j$ that results from projecting on the variables of the relation $S_j$. Let $\mathbf{u} = (u_1, \ldots, u_\ell)$ be a fractional *edge cover* for $q$. Then:

$$\sum_{\mathbf{a} \in [n]^k} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) \leq \prod_{j=1}^{\ell} \left( \sum_{\mathbf{a}_j \in [n]^{a_j}} w_j(\mathbf{a}_j)^{1/u_j} \right)^{u_j} \quad (3)$$

We illustrate Friedgut's inequality on $C_3$:

$$C_3(x, y, z) = S_1(x, y), S_2(y, z), S_3(z, x) \quad (4)$$

$C_3$ has cover $(1/2, 1/2, 1/2)$. Thus, we obtain the following, where $a, b, c$ stand for $w_1, w_2, w_3$ respectively:

$$\sum_{x,y,z \in [n]} a_{xy} \cdot b_{yz} \cdot c_{zx} \leq \sqrt{\sum_{x,y \in [n]} a_{xy}^2 \sum_{y,z \in [n]} b_{yz}^2 \sum_{z,x \in [n]} c_{zx}^2}$$

Friedgut's inequalities immediately imply a well known result developed in a series of papers [9, 3, 10] that give an upper bound on the size of a query answer as a function on the cardinality of the relations. For example in the case of $C_3$, consider an instance $S_1, S_2, S_3$, and set $a_{xy} = 1$ if $(x, y) \in S_1$, otherwise $a_{xy} = 0$ (and similarly for $b_{yz}, c_{zx}$). We obtain then $|C_3| \leq \sqrt{|S_1| \cdot |S_2| \cdot |S_3|}$.

[4]For queries with self-joins, the upper bounds hold unchanged, while the lower bounds hold up to constant factor.

# 3. SIMPLE DATABASE STATISTICS

In this section we consider the case when the statistics on database consist of the cardinalities $m_1, \ldots, m_\ell$ of the relations $S_1, \ldots, S_\ell$. All input servers know these statistics. We denote $\mathbf{m} = (m_1, \ldots, m_\ell)$ the vector of cardinalities, and $\mathbf{M} = (M_1, \ldots, M_\ell)$ the vector of the sizes expressed in bits, where $M_j = a_j m_j \log n$, and $n$ is the size of the domain of each attribute.

## 3.1 The HyperCube Algorithm

We present here the HyperCube (HC) algorithm and its analysis.

The HC algorithm, first described in [2], expresses the number of servers $p$ as $p = p_1 \cdot p_2 \cdots p_k$, where each $p_i$ is called the *share* for the variable $x_i$. The algorithm uses $k$ independently chosen random hash functions $h_i : [n] \to [p_i]$, one for each variable $x_i$. During the communication step, the algorithm sends every tuple $S_j(\mathbf{a}_j) = S_j(a_{i_1}, \ldots, a_{i_{r_j}})$ to all servers $\mathbf{y} \in [p_1] \times \cdots \times [p_k]$ such that $h_{i_m}(a_{i_m}) = \mathbf{y}_{i_m}$ for any $1 \leq m \leq r_j$. In other words, for every tuple in $S_j$, after applying the hash functions the algorithm knows the coordinates for the dimensions $i_1, \ldots, i_{r_j}$ in the hypercube, but does not know the other coordinates, and it simply replicates the tuple along those other dimensions. The algorithm finds all answers, because each potential output tuple $(a_1, \ldots, a_k)$ is known by the server $\mathbf{y} = (h_1(a_1), \ldots, h_k(a_k))$.

Since the HC algorithm is parametrized by the choice of shares, we next address two issues. First, we choose the shares $p_i$ so as to minimize the expected load per server. Second, we prove that, with high probability on the choices of the random hash functions, the expected load is not exceeded by more than a factor for any server. We start with the latter, which was not addressed in [2], and was addressed only in a limited setting in [4]: our analysis reveals a previously unknown property of the HC algorithm.

**Analysis of the Load Per Server.** Our analysis is based on the following lemma about hashing.

Lemma 3.1. *Let $R(A_1, \ldots, A_r)$ be a relation of arity $r$ with $m$ tuples. Let $p_1, \ldots, p_r$ be integers and denote $p = \prod_i p_i$ where $m \geq p^2$. Suppose that we hash each tuple $(a_1, \ldots, a_r)$ to the bucket $(h_1(a_1), \ldots, h_r(a_r))$, where $h_1, \ldots, h_r$ are independent and perfectly random hash functions. Then:*

1. *The expected load in every bucket is $m/p$.*
2. *If for every $i \in [r]$ every value of the attribute $A_i$ occurs at most once, then the maximum load per bucket is $O(m/p)$ with high probability[5].*
3. *If for every $S \subseteq [r]$, every tuple of values of attributes $(A_i)_{i \in S}$ occurs at most $am/\prod_{i \in S} p_i$ times, for $a \geq e^3/\ln(p)$ then the maximum load is $O((\frac{12a \ln p}{\ln \ln p + \ln(3a)})^r m/p)$ with high probability.*
4. *The maximum load per bucket is $O(m/\min_i(p_i))$ with high probability, independent of the instance.*

[5]high probability means polynomially small in $p$

We prove this lemma in the full paper [5]. The proof of the lemma is based on the balls-into-bins framework. The bounds provided for the case where $r \geq 2$ require novel arguments, to the best of our knowledge.

We apply the lemma to analyze the behavior of the HC algorithm under two conditions: over skew-free databases, and over arbitrary databases. For a vector of shares $(p_1, \ldots, p_k)$, we say that a relation $S_j$ is *skew-free* w.r.t. the shares if for every subset of variables $\mathbf{x} \subseteq vars(S_j)$, every value has frequency at most $m_j / \prod_{x_i \in \mathbf{x}} p_i$. Our prior analysis in [4] was only for the special case when the frequency of each value at each attribute is at most 1.

COROLLARY 3.2. *Let $\mathbf{p} = (p_1, \ldots, p_k)$ be the shares of the HC algorithm.*

*(i) If $S_j$ is skew-free w.r.t. $\mathbf{p}$, then with high probability the maximum load per server is*

$$O\left(\max_j \frac{M_j}{\prod_{i:i\in S_j} p_i} \ln^k(p)\right)$$

*(ii) For any given database, with high probability the maximum load per server is*

$$O\left(\max_j \frac{M_j}{\min_{i:i\in S_j}(p_i)}\right)$$

In [2], it is assumed that the database is skew-free and that the load per server is the expected load; item (i) of our result confirms that the load does not exceed the expected load by more than a poly-log factor with high probability, and defines precisely the skew threshold that gives the optimal behavior. Item (ii) is novel, because it describes how the HC algorithm behaves on skewed data: it shows that the algorithm is resilient to skew, and gives an upper bound even on skewed databases. We illustrate with an example.

EXAMPLE 3.3. *Let $q(x, y, z) = S_1(x, z), S_2(y, z)$ be a simple join, where both relations have cardinality $m$. We show two instances of the HC algorithm, the first optimized for skewed databases, and the second optimized for skew-free databases. The first share allocation is $p_1 = p_2 = p_3 = p^{1/3}$, thus every processor is identified by $(w_1, w_2, w_3) \in [p_1] \times [p_2] \times [p_3]$. The algorithm sends every tuple $S_1(a, c)$ to all processors $(h_1(a), w_2, h_3(c))$ for $w_2 \in [p_3]$ and every tuple $S_2(b, c)$ to all processors $(w_1, h_2(b), h_3(c))$ for $w_1 \in [p_1]$. By Corollary 3.2, on skew-free databases the load per server is $O(m/p^{2/3})$ (times some polylog factor). But even on skewed database the load per server is $O(m/p^{1/3})$. The second algorithm allocates shares $p_1 = p_2 = 1, p_3 = p$. This corresponds to a standard hash-join on the variable $z$. On a skew-free database (equivalently, when every value of $z$ has frequency at most $m/p$ in both relations) the load per server is $O(m/p)$ with high probability. However, if it is skewed, then the load can be as bad as $O(m)$: this occurs when all tuples have the same value $z$.*

Generalizing the example, for every conjunctive query with $k$ variables, we can execute the HC algorithm with equal shares $p_1 = \ldots = p_k = p^{1/k}$. Then, the algorithm achieves a maximum load per server of at most $O(\max_j M_j/p^{1/k})$.

However, in practice, in applications where skew is expected, it is better to design specialized algorithms, as we further discuss in section 4. Therefore, we focus our analysis on skew-free databases, and optimize the expected load.

**Choosing the Shares.** Here we discuss how to compute the shares $p_i$ to optimize the expected load per server. Afrati and Ullman compute the shares by optimizing the total load $\sum_j m_j / \prod_{i:i\in S_j} p_i$ subject to the constraint $\prod_i p_i = 1$, which is a non-linear system that can be solved using Lagrange multipliers. Here we take a different approach. First, we write the shares as $p_i = p^{e_i}$ where $e_i \in [0, 1]$ is called the *share exponent* for $x_i$, and denote $L$ the maximum load per server, thus $M_j / \prod_{i:i\in S_j} p_i \leq L$ for every $j$. Denote $\lambda = \log_p L$ and $\mu_j = \log_p M_j$ (we will assume w.l.o.g. that $m_j \geq p$, hence $\mu_j \geq 1$ for all $j$). Then, we optimize the LP:

$$\begin{aligned} \text{minimize} \quad & \lambda \\ \text{subject to} \quad & \sum_{i\in[k]} -e_i \geq -1 \\ & \forall j \in [\ell] : \sum_{i\in S_j} e_i + \lambda \geq \mu_j \\ & \forall i \in [k] : e_i \geq 0, \quad \lambda \geq 0 \end{aligned} \tag{5}$$

Denote $L_{\texttt{upper}} = p^{e^*}$ where $e^*$ is the objective value of the optimal solution to the above LP. We have:

THEOREM 3.4. *For a query $q$ and $p$ servers, with statistics $\mathbf{M}$, let $\mathbf{e} = e_1, \ldots, e_k$ be share exponents that are optimal for the above LP. Then, the expected load per server is $L_{upper}$. Moreover, if every $S_j$ is skew-free w.r.t. to $\mathbf{e}$, then the maximum load per server is $O(L_{upper} \cdot \ln^k(p))$ with high probability.*

In subsection 3.3 we will give a closed form expression for $L_{\texttt{upper}}$ and also provide an example. But first, we prove a matching lower bound.

## 3.2 The Lower Bound

We prove a lower bound for the maximum load per server over databases with statistics $\mathbf{M}$. Fix some constant $0 < \delta < \min_j\{a_j\}$, and assume that for every relation $S_j$ its cardinality satisfies $m_j \leq n^\delta$, where $n$ is the domain size of each attribute.

Consider the probability space where each relation $S_j$ is chosen independently and uniformly at random from all subsets of $[n]^{a_j}$ with exactly $m_j$ tuples. Denote $\mathbf{E}[|q(I)|]$ the expected number of answers to $q$: we can show that $\mathbf{E}[|q(I)|] = n^{k-a} \prod_{j=1}^{\ell} m_j$.

Fix a query $q$ and a fractional edge packing $\mathbf{u}$ of $q$. Denote $u = \sum_{j=1}^{\ell} u_j$ the value of the packing, and:

$$K(\mathbf{u}, \mathbf{M}) = \prod_{j=1}^{\ell} M_j^{u_j} \tag{6}$$

$$L(\mathbf{u}, \mathbf{M}, p) = \left(\frac{K(\mathbf{u}, \mathbf{M})}{p}\right)^{1/u} \tag{7}$$

Further denote $L_{\texttt{lower}} = \max_{\mathbf{u}} L(\mathbf{u}, \mathbf{M}, p)$, where $\mathbf{u}$ ranges over all fractional edge packings for $q$. Let $c$ be a constant, $c = \frac{a_j - \delta}{3a_j}$, where $a_j$ is the maximum arity of all relations. We prove in the full paper:

THEOREM 3.5. *Fix statistics $\mathbf{M}$, and consider any deterministic MPC algorithm that runs in one communication round on $p$ servers. Let $\mathbf{u}$ be any edge packing of $q$. Any*

*server $i$ with load $L_i$ reports at most*

$$\frac{L_i^u}{c^u K(\mathbf{u},\mathbf{M})} \cdot \mathbf{E}[|q(I)|]$$

*answers in expectation, where $I$ is a randomly chosen database with statistics $\mathbf{M}$. Therefore, the $p$ servers of the algorithm report at most*

$$\left(\frac{L}{c \cdot L(\mathbf{u},\mathbf{M},p)}\right)^u \cdot \mathbf{E}[|q(I)|]$$

*answers in expectation, where $L$ is the maximum load.*

*As a consequence, any algorithm that computes $q$ correctly over any database with statistics $\mathbf{M}$ must have load $L \geq cL_{\texttt{lower}}$ bits*[6].

In our previous work [4], we presented a matching lower and upper bound for computing $q$ on some restricted database instances, where the relations $S_j$ are *matchings* and have the same cardinalities; the proof of Theorem 3.5 is an extension of the lower bound proof in [4]. We explain here the relationship. When all cardinalities are equal, $M_1 = \ldots = M_\ell = M$, then $L_{\texttt{lower}} = M/p^{1/u}$, and this quantity is maximized when $\mathbf{u}$ is a maximum fractional edge packing, whose value is denoted $\tau^*$: by duality, this is equal to the fractional vertex covering number for $q$. The bound in [4] is $c'M/p^{1/\tau^*}$ (the constant $c'$ in [4] is tighter). Theorem 3.5 generalizes the lower bound to arbitrary cardinalities, in which case $L(\mathbf{u},\mathbf{M},p)$ is not necessarily maximized at $\tau^*$. In the rest of this section we prove that $L_{\texttt{lower}} = L_{\texttt{upper}}$.

## 3.3 Proof of Equivalence

The feasible solutions of the edge packing constraints in (2) define a feasible and bounded convex polytope. An *extreme point* of a polytope is one that cannot be written as a convex combination of two other distinct points of the polytope. Each extreme point can be obtained by choosing $k$ constraints, transform them into equalities and solve the corresponding linear system. The set of extreme points for (2), denoted $pk(q)$, is thus finite and can be bounded by $|pk(q)| \leq \binom{k+\ell}{k}$, *i.e.* it depends only on the query.

We prove here:

THEOREM 3.6. *For any vector of statistics $\mathbf{M}$ and number of processors $p$ , we have:*

$$L_{lower} = L_{upper} = \max_{\mathbf{u} \in pk(q)} L(\mathbf{u},\mathbf{M},p)$$

Before we prove the theorem we discuss its implications. We start with an example.

EXAMPLE 3.7. *Consider the triangle query*

$$C_3 = S_1(x_1,x_2), S_2(x_2,x_3), S_3(x_3,x_1)$$

*and assume the three cardinalities are $m_1, m_2, m_3$. Then, $pk(C_3)$ has four vertices, and each gives a different value for $L(\mathbf{u},\mathbf{M},p)$:*

| $\mathbf{u}$ | $L(\mathbf{u},\mathbf{M},p)$ |
|---|---|
| $(1/2,1/2,1/2)$ | $(M_1M_2M_3)^{1/3}/p^{2/3}$ |
| $(1,0,0)$ | $M_1/p$ |
| $(0,1,0)$ | $M_2/p$ |
| $(0,0,1)$ | $M_3/p$ |

[6]This follows by observing that, when $L(\mathbf{u},\mathbf{M},p)$ is maximized, then $u = \sum_j u_j \geq 1$.

*The first vertex is the solution to $u_1+u_2 = u_1+u_3 = u_2+u_3 = 1$; the second the solution to $u_1+u_2 = 1, u_2 = u_3 = 0$, etc. Thus, the load of the algorithm is the largest of these four quantities, and this is also the lower bound of any algorithm. In other words, the optimal solution to the LP (5) can be given in closed form, as the maximum over four expressions.*

Next, we use the theorem to compute the *space exponent*. In [4] we showed that, for every query $q$, the optimal load over databases restricted to matchings of equal size $M$ is $O(M/p^{1-\varepsilon})$, where $0 \leq \varepsilon < 1$ is called the space exponent for $q$. Consider now a database with arbitrary statistics, and denote $M = \max_j M_j$. Thus, we may assume w.l.o.g. that for every $j$, $M_j = M/p^{\nu_j}$ for some $\nu_j \geq 0$. Then, $L(\mathbf{u},\mathbf{M},p) = M/p^{(\sum_j \nu_j u_j + 1)/u}$. To obtain the optimal load, one needs to find $\mathbf{u} \in pk(q)$ that minimizes $v = (\sum_j \nu_j u_j + 1)/(\sum_j u_j)$. Denoting $v^*$ the minimal value, the load is $M/p^{v^*}$. Thus, the space exponent for given statistics is $1 - v^*$.

PROOF OF THEOREM 3.6. Recall that $L_{\texttt{upper}}$ is $p^{e^*}$, where $e^*$ is the objective value of the optimal solution to the *primal* LP problem (5). Consider its *dual* LP:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j\in[\ell]} \mu_j f_j - f \\
\text{subject to} \quad & \sum_{j\in[\ell]} f_j \leq 1 \\
& \forall i \in [k] : \sum_{j:i\in S_j} f_j - f \leq 0 \\
& \forall j \in [\ell] : f_j \geq 0, \quad f \geq 0
\end{aligned} \tag{8}$$

By the primal-dual theorem, the objective is also maximized at $e^*$. Writing $u_j = f_j/f$ and $u = 1/f$, we transform it into the following non-linear optimization problem:

$$\begin{aligned}
\text{maximize} \quad & \frac{1}{u} \cdot \left(\sum_{j\in[\ell]} \mu_j u_j - 1\right) \\
\text{subject to} \quad & \sum_{j\in[\ell]} u_j \leq u \\
& \forall i \in [k] : \sum_{j:i\in S_j} u_j \leq 1 \\
& \forall j \in [\ell] : u_j \geq, 0 \quad u \geq 0
\end{aligned} \tag{9}$$

The optimal solution of the above non-linear problem, with value $u^*$, must satisfy $u = \sum_j u_j$, otherwise we simply replace $u$ with $\sum_j u_j$ and obtain a feasible solution with at least as good objective function (indeed, $\mu_j \geq 1$ for any $j$, and hence $\sum_j \mu_j u_j \geq \sum_j u_j \geq 1$, since any optimal $\mathbf{u}$ will have sum at least 1). Therefore, the optimal is given by a fractional edge packing $\mathbf{u}$. Furthermore, for any packing $\mathbf{u}$, the objective function $\sum_j \frac{1}{u} \cdot (\mu_j u_j - 1)$ is $\log_p L(\mathbf{u},\mathbf{M},p)$. To conclude the proof of the theorem, we show that (a) $e^* = u^*$ and (b) the optimum is obtained when $\mathbf{u} \in pk(q)$. This follows from:

LEMMA 3.8. *Let $F : \mathbf{R}^{k+1} \to \mathbf{R}^{k+1}$: $F(x_0, x_1, \ldots, x_k) = (1/x_0, x_1/x_0, \ldots, x_k/x_0)$. Then:*

- *$F$ is its own inverse, $F = F^{-1}$.*

- *$F$ maps any feasible solution to* (8) *to a feasible solution to* (9)*, and conversely.*
- *$F$ maps a convex set to a convex set.*

PROOF. If $y_0 = 1/x_0$ and $y_j = x_j/x_0$, then obviously $x_0 = 1/y_0$ and $x_j = y_j/y_0$. The second item can be checked directly. For the third item, it suffices to prove that $F$ maps a convex combination $\lambda\mathbf{x} + \lambda'\mathbf{x}'$ where $\lambda + \lambda' = 1$ into a convex combination $\mu F(\mathbf{x}) + \mu' F(\mathbf{x}')$, where $\mu + \mu' = 1$. Assuming $\mathbf{x} = (x_0, x_1, \ldots, x_k)$ and $\mathbf{x}' = (x'_0, x'_1, \ldots, x'_k)$, this follows by setting $\mu = x_0/(\lambda x_0 + \lambda x'_0)$ and $\mu' = x'_0/(\lambda x_0 + \lambda x'_0)$. □

This completes the proof of Theorem 3.6.

# 4. COMPLEX DATABASE STATISTICS

In this section, we discuss algorithms and lower bounds for the case where the input servers are provided by additional information regarding skew.

## 4.1 A Simple Case: Join

We start with a simple example, the join of two tables, $q(x, y, z) = S_1(x, z), S_2(y, z)$, to illustrate the main algorithmic and proof ideas. Let $m_1, m_2$ be the cardinalities of $S_1, S_2$. For any value $h \in [n]$ that variable $z$ may assume, let $m_j(h)$ denote the frequency of $h$ in $S_j$, $j = 1, 2$; $h$ is called a *heavy hitter* in $S_j$ if $m_j(h) \geq m_j/p$. For general queries, the threshold for a heavy hitter will differ. We assume that heavy hitters and their frequencies are known initially by the algorithm.

The algorithm uses the same principle popular in virtually all parallel join implementations to date: identify the heavy hitters and treat them differently. However, the analysis and optimality proof is new, to the best of our knowledge.

**The Algorithm.** Let $H$ denote the set of heavy hitters either in $S_1$ or in $S_2$. Note that $|H| \leq 2p$. The algorithm will deal with the tuples that have no heavy hitter values (*light tuples*) by running the vanilla HC algorithm. However, it will adapt its function for heavy hitters.

To compute $q$, the algorithm must compute for each $h \in H$ the subquery $q[h/z] = S_1(x, h), S_2(y, h)$, which is equivalent to computing the cartesian product $q_z = S'_1(x), S'_2(z)$, where $S'_1(x) = S_1(x, h)$ and $S'_2(y) = S_2(y, h)$, and the relations have cardinality $m_1(h)$ and $m_2(h)$ respectively (and size in bits $M_1(h), M_2(h)$). We call $q_z$ the *residual query*. The algorithm will allocate $p_h$ servers to compute $q[h/z]$ for each $h \in H$, such that $\sum_{h\in H} p_h = \Theta(p)$. Since the unary relations have no skew, Theorem 3.6 says that the maximum load $L_h$ for each $h$ is given by

$$L_h = \tilde{O}\left(\max_{\mathbf{u}\in pk(q_z)} L(\mathbf{u}, \mathbf{M}(h), p_h)\right)$$

where $\tilde{O}$ hides the polylog dependence on $p$. One can compute that $pk(q_z) = \{(1, 1), (1, 0), (0, 1)\}$. At this point, since $p_h$ is not specified, it is not clear which edge packing maximizes the above quantity for each $h$. To overcome this problem, we further refine the assignment of servers to heavy hitters: we allocate $p_{h,\mathbf{u}}$ servers to each $h$ and each $\mathbf{u} \in pk(q_z)$, such that $p_h = \sum_{\mathbf{u}} p_{h,\mathbf{u}}$.

Now, for a given $\mathbf{u} \in pk(q)$, we can evenly distribute the load among the heavy hitters by choosing $p_{h,\mathbf{u}}$ such that for any $h, h' \in H$, we have $L(\mathbf{u}, \mathbf{M}(h), p_{h,\mathbf{u}}) = L(\mathbf{u}, \mathbf{M}(h'), p_{h',\mathbf{u}})$. In particular, we will choose the server allocation proportionally to the "heaviness" of executing the residual query:

$$p_{h,\mathbf{u}} = \left\lceil p \cdot \frac{K(\mathbf{u}, \mathbf{M}(h))}{\sum_{h'\in H} K(\mathbf{u}, \mathbf{M}(h'))} \right\rceil$$

It is easy to check that the total number of servers will be $\Theta(p)$, and that the load $L_h$ will be

$$L_h = \tilde{O}\left(\max_{\mathbf{u}\in pk(q_z)} \left(\frac{\sum_{h\in H} M_1(h)^{u_1} M_2(h)^{u_2}}{p}\right)^{1/(u_1+u_2)}\right)$$

By plugging in the possible values of $pk(q_z)$, as well as the load for the vanilla HC algorithm that runs on the light tuples, we obtain that the maximum load will be

$$\tilde{O}\left(\max\left\{\frac{M_1}{p}, \frac{M_2}{p}, \left(\frac{\sum_{h\in H} M_1(h)M_2(h)}{p}\right)^{1/2}\right\}\right) \quad (10)$$

The first two terms are exactly what we would get from the analysis of the HC algorithm, and do not depend on the occurrence of heavy hitters, while the third term depends on the frequencies of the heavy hitters and can be much larger than the first two. In the extreme, a single heavy hitter $h$ with $m_j(h) = m_j$ for $j = 1, 2$ will demand maximum load equal to $\tilde{O}(\sqrt{M_1 M_2/p})$.

**The Lower Bound.** We show here that the above algorithm is optimal within a polylog factor of $p$. Recall that in section 3 we have shown that any algorithm that computes correctly a query $q$ must have maximum load at least $\Omega(\max_{\mathbf{u}\in pk(q)}\{L(\mathbf{u}, \mathbf{M}, p)\})$. Since $pk(q) = \{(1, 0), (0, 1)\}$ in the case of the join, we already have a lower bound of $\Omega(\max\{M_1/p, M_2/p\})$ for the load.

Hence, to show optimality it suffices to show that the load $L$ is further lower bounded by the third term of (10). We sketch here the main idea of the proof. Recall that in section 3 we constructed a uniformly random instance to show the lower bound. For skewed data, we have to construct a random instance $I$ that agrees with the frequency information $m_j(h)$ for each $h \in [n]$. To do this, we create $I$ by choosing a uniformly random subinstance for each residual query $q[h/z]$. Notice that, by our construction, the size of the join will be $\sum_{h\in[n]} m_1(h)m_2(h)$.

Now, let $L_j^s(h)$ denote the expected number of tuples from the subinstance $S_j(h)$ that are known by some server $s$ ($s = 1, \ldots, p$) after communication. We show in subsection 4.3 that, in order for the servers to report correctly all join tuples, we must have that for each $h \in [n]$:

$$\sum_{s=1}^{p} L_1^s(h)L_2^s(h) \geq m_1(h)m_2(h)$$

We can now sum up the above inequalities for all $h \in [n]$:

$$\sum_{h\in[n]} m_1(h)m_2(h) \leq \sum_{h\in[n]}\sum_{s=1}^{p} L_1^s(h)L_2^s(h)$$

$$= \sum_{s=1}^{p}\sum_{h\in[n]} L_1^s(h)L_2^s(h) \leq \sum_{s=1}^{p}\left(\sum_{h\in[n]} L_1^s(h)\right)\left(\sum_{h\in[n]} L_2^s(h)\right)$$

Observe that $\sum_h L_j^s(h)$ denotes the expected number of tuples from $S_j$ known by a server $s$. Since $s$ will receive at most $L$ bits, it can be shown (details in subsection 4.3) that

$\sum_h L_j^s(h) \leq L/(2c\log(n))$ for some constant $c$. We now obtain:

$$\sum_{h\in[n]} M_1(h)M_2(h) \leq pL^2$$

which proves our lower bound.

## 4.2 An Algorithm for the General Case

We now generalize some of the ideas for the simple join to an arbitrary conjunctive query $q$. Extending the notion for simple joins, for each relation $S_j$ with $|S_j| = m_j$ we say that a partial assignment $\mathbf{h}_j$ to a subset $\mathbf{x}_j \subset vars(S_j)$ is a *heavy hitter* if and only if the number of tuples, $m_j(\mathbf{h}_j)$, from $S_j$ that contain $\mathbf{h}_j$ satisfies $m_j(\mathbf{h}_j) > m_j/p$. As before, there are $O(p)$ such heavy hitters. We will assume that each input server knows the entire set of heavy hitters for all relations.

**Bin Combinations.** For simplicity we assume that $p$ is a power of 2. We will not produce quite as smooth a bound as we did for the simple join, since we will initially group the frequencies to *bins*, which will add a $\log^{O(1)} p$ factor to the bound. In particular, for each relation $S_j$ and subset of variables $\mathbf{x}_j$, we define $\log_2 p$ bins for the frequencies, or degrees of each of the heavy hitters. The $b$-th bin, for $b = 1, \ldots, \log_2 p$ will contain all heavy hitters $\mathbf{h}_j$ with $m_j/2^{b-1} \geq m_j(\mathbf{h}_j) > m_j/2^b$. The last bin, a bin of light hitters with $b = \log_2 p + 1$, will contain all assignments $\mathbf{h}_j$ to $\mathbf{x}_j$ that are not heavy hitters. Notice that, when $\mathbf{x}_j = \emptyset$, the only non-empty bin is the first bin, the only heavy hitter is the empty tuple $\mathbf{h}_j = ()$, and $m_j(\mathbf{h}_j) = m_j$.

For a bin $b$ on $\mathbf{x}_j$ define $\beta_b = \log_p(2^{b-1})$; observe that for each heavy hitter bin, there are at most $2p^{\beta_b}$ heavy hitters in this bin, and for the last bin we have $\beta_b = 1$. Instead of identifying each bin using its index $b$, we identify each bin by $\beta_b$, called its *bin exponent*, along with the index of the relation $S_j$ for which it is defined, and the set $\mathbf{x}_j \subset vars(S_j)$. Note that $0 = \beta_1 < \beta_2 < \cdots < \beta_{\log_2 p+1} = 1$.

Definition 4.1 (Bin Combination). *Consider a set of variables $\mathbf{x} \subset V = vars(q)$, and define $\mathbf{x}_j = \mathbf{x} \cap vars(S_j)$. A pair $\mathcal{B} = (\mathbf{x}, (\beta_j)_j)$ is called a* bin combination *if (1) $\beta_j = 0$ for every $j$ where $\mathbf{x}_j = \emptyset$, and (2) there is some consistent assignment $\mathbf{h}$ to $\mathbf{x}$ such that for each $j$ with $\mathbf{x}_j \neq \emptyset$ the induced assignment $\mathbf{h}_j$ to $\mathbf{x}_j$ has bin exponent $\beta_j$ in relation $S_j$. We write $C(\mathcal{B})$ for the set of all such assignments $\mathbf{h}$.*

**Algorithm BinHC.** The algorithm BinHC allocates $p$ virtual processors to each bin combination and handles associated inputs separately. There are $O(\log p)$ bin choices for each relation and therefore the algorithm requires at most $\log^{O(1)} p$ virtual processors in total. Let $N_{\text{bc}}$ be the number of possible bin combinations. As in the join algorithm (subsection 4.1), within each bin combination we partition the $p$ servers among the heavy hitters, using $p_{\mathbf{h}} = p/|C(\mathcal{B})|$ servers for heavy hitter $\mathbf{h}$ (note that $p_{\mathbf{h}}$ is independent of $\mathbf{h}$, since we have ensured complete uniformity within a bin combination). Unfortunately, we can only process $\leq p$ heavy hitters in every bin combination, while in genera, we may have $C(\mathcal{B}) > p$: *e.g.* if $\mathbf{x}$ contains variables $x_1$ in $S_1$ and $x_2$ in $S_2$, there may be up to $p \times p$ heavy hitters in this bin combination.

To solve this issue, for each $\mathcal{B}$ we will define a set $C'(\mathcal{B}) \subseteq C(\mathcal{B})$ with $|C'(\mathcal{B})| \leq p$ and sets $S_j^{(\mathcal{B})} \subseteq S_j$ of tuples for $j \in [\ell]$ that extend $\mathbf{h}_j$ for some $\mathbf{h} \in C'(\mathcal{B})$.

The BinHC algorithm for $\mathcal{B}$, in short BinHC($\mathcal{B}$), will compute all query answers for the subinstance $I_{\mathcal{B}} = (S_j^{(\mathcal{B})})_j$. If $\alpha(\mathcal{B}) = \log_p |C'(\mathcal{B})|$, the algorithm will run the HC algorithm on $p^{1-\alpha(\mathcal{B})}$ virtual processors for each of the heavy hitters $\mathbf{h} \in C'(\mathcal{B})$ so as to compute $q(I_{\mathcal{B}})$.

The share exponents for the HC algorithm will be provided by a modification of the vertex covering primal LP (5), which describes an algorithm that suffices for all light hitters. Recall that in this LP, $\mu_j = \log_p M_j$ and $\lambda$ is $\log_p L$ for the load $L$. That LP corresponds to the bin combination $\mathcal{B}_\emptyset$ which has $\mathbf{x} = \emptyset$ and all $\beta_j = 0$. More generally, the LP associated with our algorithm for bin combination $\mathcal{B}$ is:

$$\begin{aligned} \text{mininimize} \quad & \lambda \\ \text{subject to} \quad & \\ \forall j \in [\ell]: \quad & \lambda + \sum_{x_i \in vars(S_j)-\mathbf{x}_j} e_i \geq \mu_j - \beta_j \\ & \sum_{i \in V-\mathbf{x}} e_i \leq 1 - \alpha(\mathcal{B}) \\ \forall i \in V-\mathbf{x}: \quad & e_i \geq 0, \quad \lambda \geq 0 \end{aligned} \tag{11}$$

Let $(e_i^{(\mathcal{B})})_{i \in V-\mathbf{x}'})$ be the optimal solution for the above LP and $(\lambda^{(\mathcal{B})}$ the minimum value of the objective function.

To complete the description of BinHC, we need to define $C'(\mathcal{B})$ and $(S_j^{(\mathcal{B})})_j$. We define $C'(\mathcal{B})$ inductively. For the bin combination $\mathcal{B}_\emptyset$, $C'(\mathcal{B}_\emptyset) = C(\mathcal{B}_\emptyset)$ and it has 1 element, the empty partial assignment. For $\mathcal{B} \neq \mathcal{B}_\emptyset$, $C'(\mathcal{B})$ is defined based on optimal solutions to the above LP applied to bin combinations $\mathcal{B}'$ with $\mathbf{x}' \subset \mathbf{x}$ (such solutions may not be unique but we fix one arbitrarily for each bin combination).

For $\mathbf{h}' \in C'(\mathcal{B}')$, we say that a heavy hitter $\mathbf{h}_j$ of $S_j$ that is an extension of $\mathbf{h}'_j$ to $\mathbf{x}_j$ is *overweight* for $\mathcal{B}'$ if there are more than $N_{\text{bc}} \cdot m_j/p^{\beta_j + \sum_{i \in \mathbf{x}_j - \mathbf{x}'_j} e_i^{(\mathcal{B}')}}$ elements of $S_j$ consistent with $\mathbf{h}_j$. $C'(\mathcal{B})$ consists of all assignments $\mathbf{h} \in C(\mathcal{B})$ such that there is some $j \in [\ell]$, some bin combination $\mathcal{B}'$ on set $\mathbf{x}' \subset \mathbf{x}$ such that $\mathbf{x} - \mathbf{x}' \subseteq vars(S_j)$, and some $\mathbf{h}' \in C'(\mathcal{B}')$ such that $\mathbf{h}$ is an extension of $\mathbf{h}'$ and $\mathbf{h}_j$ is an overweight heavy hitter of $S_j$ for $\mathcal{B}'$. The following lemma, which is proved in the full paper, shows that $\alpha(\mathcal{B}) \leq 1$.

Lemma 4.2. *For all bin combinations $\mathcal{B}$, $|C'(\mathcal{B})| \leq p$.*

Let $A_{\mathcal{B}} \subseteq [\ell]$ be the set of all $j$ such that $\mathbf{x}_j \neq \emptyset$. For each $j \in [\ell] - A_{\mathcal{B}}$, let $S_j^{(\mathcal{B})}$ consist of all tuples in $S_j$ that do not contain any heavy hitter $\mathbf{h}''_j$ of $S_j$ that is overweight for $\mathcal{B}$. For each $j \in A_{\mathcal{B}}$, and $\mathbf{h} \in C'(\mathcal{B})$ let $S_j^{(\mathcal{B})}(\mathbf{h})$ consist of all tuples in $S_j$ that contain $\mathbf{h}_j$ on $\mathbf{x}_j$ (with bin exponent $\beta_j$) but do not contain any heavy hitter $\mathbf{h}''_j$ of $S_j$ that is overweight for $\mathcal{B}$ and a proper extension of $\mathbf{h}_j$. $S_j^{(\mathcal{B})}$ will be the union of all $S_j^{(\mathcal{B})}(\mathbf{h})$ for all $\mathbf{h} \in C'(\mathcal{B})$.

**Analysis.** We analyze here the BinHC algorithm. We show first that it correctly computes all answers to the query $q$ on the relations $S_j$.

Lemma 4.3 (Correctness). *Every tuple in the join of $(S_j)_{j\in[\ell]}$ is contained in a join of subrelations $(S_j^{(\mathcal{B})})_{j\in[\ell]}$ for some bin combination $\mathcal{B}$.*

Proof. Observe first that every join tuple is vacuously consistent with the empty bin combination $\mathcal{B}_\emptyset$. Therefore

the join of $(S_j^{(\mathcal{B}_\emptyset)})_{j\in[\ell]}$ contains all join tuples that do not contain an overweight heavy hitter $\mathbf{h}_j$ for any relation $S_j$ with respect to $\mathcal{B}_\emptyset$ (and therefore contains all join tuples that are not consistent with any heavy hitter). Now fix a join tuple $\mathbf{t}$ that is overweight for $\mathcal{B}_\emptyset$. By definition, there is an associated relation $S_{j_1}$ and $\mathbf{x}^1 \subset vars(S_{j_1})$ such that $\mathbf{h}^1 = (\mathbf{t}_{\mathbf{x}^1})$ is an overweight heavy hitter of $S_{j_1}$ for $\mathcal{B}_\emptyset$. Let $\mathcal{B}_1$ be the bin combination associated with $\mathbf{h}^1$. By definition $\mathbf{h}^1 \in C'(\mathcal{B}_1)$. Now either $\mathbf{t}$ is contained in the join of $(S_j^{(\mathcal{B}_1)})_{j\in[\ell]}$ and we are done or there is some relation $S_{j_2}$ and $\mathbf{x}^2$ such that $\mathbf{x}^2 - \mathbf{x}^1 \subset vars(S_{j_2})$ such that $\mathbf{h}^2 = (\mathbf{t}_{\mathbf{x}^2})$ has the property that $\mathbf{h}^2_{j_2}$ is an overweight heavy hitter of $S_{j_2}$ for $\mathcal{B}_1$. Again, in the latter case, if $\mathcal{B}_2$ is the bin combination associated with $\mathbf{h}^2$ then $\mathbf{h}^2 \in C'(\mathcal{B}_2)$ by definition and we can repeat the previous argument for $\mathcal{B}_2$ instead of $\mathcal{B}_1$. Since the number of variables grows at each iteration, we can repeat this at most $k$ times before finding a first $\mathcal{B}_r$ such that $\mathbf{t}$ is not associated with any overweight heavy hitter for $\mathcal{B}_r$. In this case $\mathbf{t}$ will be computed in the join of $(S_j^{(\mathcal{B}_r)})_{j\in[\ell]}$. □

We next analyze the load for BinHC($\mathcal{B}$), and show that it is within a $\log^{O(1)} p$ factor of $p^{\lambda^{(\mathcal{B})}}$, where $\lambda^{(\mathcal{B})}$ is the optimal value given by the LP for $\mathcal{B}$.

LEMMA 4.4. *Let $\mathbf{h}$ be an assignment to $\mathbf{x}$ that is consistent with bin combination $\mathcal{B}$. If we hash each residual relation $S_j^{(\mathcal{B})}(\mathbf{h})$ on $vars(S_j) - \mathbf{x}_j$ using $p^{e_i^{(\mathcal{B})}}$ values for each $x_i \in vars(S_j) - \mathbf{x}_j$, each processor has load (in bits)*

$$O\left((N_{\text{bc}} \cdot \ln p)^{r'} \cdot M_j/p^{\min(\beta_j + \sum_{i\in vars(S_j)-\mathbf{x}_j} e_i^{(\mathcal{B})}, 1)}\right)$$

*with high probability, where $r' = \max_j(r_j - |\mathbf{x}_j|)$.*

PROOF. For $j \in [\ell] - A_{\mathcal{B}}$, $S_j^{(\mathcal{B})}$ only contains tuples of $S_j$ that are not overweight for $\mathcal{B}$, which means that for every $\mathbf{x}''_j \subseteq vars(S_j)$ and every heavy hitter assignment $\mathbf{h}''$ to the variables of $\mathbf{x}''_j$, there are at most

$$N_{\text{bc}} \cdot m_j/p^{\beta_j + \sum_{i\in\mathbf{x}''_j} e_i^{(\mathcal{B})}} = N_{\text{bc}} \cdot m_j/p^{\beta_j + \sum_{i\in\mathbf{x}''_j - \mathbf{x}_j} e_i^{(\mathcal{B})}}$$

elements of $S_j$ consistent with $\mathbf{h}''$. Every other assignment $\mathbf{h}''$ to the variables of $\mathbf{x}''_j$ is a light hitter and therefore is contained in at most $m_j/p$ consistent tuples of $S_j$. For $j \in A_{\mathcal{B}}$, we obtain the same bound, where the only difference is that we need to restrict things to extensions of $\mathbf{h}_j$. This bound gives the smoothness condition on $S_j^{(\mathcal{B})}(\mathbf{h})$ necessary to apply Lemma 3.1 to each relation $S_j^{(\mathcal{B})}(\mathbf{h})$ and yields the claimed result. □

As a corollary, we obtain:

COROLLARY 4.5. *Let $L_{min} = \max_j(M_j/p)$. The maximum load of BinHC($\mathcal{B}$) is $O((N_{\text{bc}} \cdot \ln p)^{r_{\max}} \cdot \max(L_{min}, p^\lambda))$ with high probability, where $\lambda = \lambda^{(\mathcal{B})}$ is the optimum of the LP for $\mathcal{B}$ and $r_{\max}$ is the maximum arity of any $S_j$.*

PROOF. There are $p^{1-\alpha(\mathcal{B})}$ processors allocated to each $\mathbf{h}$ and $p^{\alpha(\mathcal{B})}$ such assignments $\mathbf{h}$ so that the maximum load per $\mathbf{h}$ is also the maximum overall load. Given the presence of the $L_{min}$ term, it suffices to show that the maximum load per $\mathbf{h}$ due to relation $S_j^{(\mathcal{B})}(\mathbf{h})$ is at most $(\ln p)^{k-|\mathbf{x}|} \cdot \max(M_j/p, p^\lambda)$ for each $j \in [\ell]$. Observe that by construction, $p^\lambda$ is the smallest value s.t. $p^\lambda \cdot p^{\beta_j + \sum_{x_i\in vars(S_j)-\mathbf{x}_j} e_i^{(\mathcal{B})}} \geq M_j$ for all $j$ and $\sum_{i\in\mathbf{x}} e_i^{(\mathcal{B})} \leq 1 - \alpha(\mathcal{B})$. Lemma 4.4 then implies that the load due to relation $S_j^{(\mathcal{B})}(\mathbf{h})$ is at most a polylogarithmic factor times

$$\max(M_j/p, M_j/p^{\beta_j + \sum_{x_i\in vars(S_j)-H_j} e_i^{(\mathcal{B})}})$$

which is at most $\max(M_j/p, p^\lambda)$. □

We can now upper bound the maximum load of the BinHC algorithm. Denote $\mathbf{M}(\mathcal{B}) = (M_1/p^{\beta_1}, \ldots, M_\ell/p^{\beta_\ell})$. Then:

THEOREM 4.6 (MAXIMUM LOAD). *The BinHC algorithm has with high probability maximum load*

$$L = O\left(\log^{O(1)} p \cdot \max_{\mathcal{B},\mathbf{u}\in pk(q_{\mathbf{x}})} L(\mathbf{u}, \mathbf{M}(\mathcal{B}), p^{1-\alpha(\mathcal{B})})\right)$$

PROOF. There are only $\log^{O(1)} p$ choices of $\mathcal{B}$ and for each choice of $\mathcal{B}$, by Corollary 4.5, the load with $p$ virtual processors is $O(\log^{O(1)} p \cdot \max(L_{min}, p^{\lambda^{(\mathcal{B})}}))$. We first show that we can remove the $L_{min}$ term. Indeed, observe that in the original LP which corresponds to an empty bin combination $\mathcal{B}$, we have $\lambda + \sum_{i\in S_j} e_i \geq \mu_j$ for each $j \in [\ell]$ and $\sum_i e_i \leq 1$. This implies that $\lambda \geq \mu_j - 1$ and hence $p^\lambda \geq M_j/p$ for each $j$, so $p^\lambda \geq L_{min}$. Finally, by applying a duality argument to (11), we have:

$$\lambda^{(\mathcal{B})} = \max_{\mathbf{u}\in pk(q_{\mathbf{x}})} \{L(\mathbf{u}, \mathbf{M}(\mathcal{B}), p^{1-\alpha(\mathcal{B})})\}$$

This completes the proof. □

## 4.3 Lower Bound

In this section we give a lower bound for the load of any deterministic algorithm that computes a query $q$, and generalize the lower bound in Theorem 3.5, which was over databases with cardinality statistics $\mathbf{M}$. Our new lower bound generalizes this to databases with a fixed degree sequence: if the degree sequence is skewed, then the new bounds can be stronger, proving that skew in the input data makes query evaluation harder.

For a relation $S_j$, let $\mathbf{x}_j \subseteq vars(S_j)$ and $d_j = |\mathbf{x}_j|$. A *statistics of type* $\mathbf{x}_j$, or *$\mathbf{x}_j$-statistics*, is a function $m_j : [n]^{d_j} \to \mathbb{N}$. An instance of $S_j$ satisfies the statistics $m_j$ if for any tuple $\mathbf{h}_j \in [n]^{d_j}$, its frequency is precisely $m_j(\mathbf{h}_j)$, in other words $|\sigma_{\mathbf{x}_j=\mathbf{h}_j}(S_j)| = m_j(\mathbf{h}_j)$. As an example, if $S(x, y)$ is a binary relation, then an $x$-statistics is a degree sequence $m(1), m(2), \ldots, m(n)$; also, if $\mathbf{x}_j = \emptyset$ then an $\mathbf{x}_j$-statistics consists of a single number, which denotes the cardinality of $S_j$. In general, the $\mathbf{x}_j$-statistics define uniquely the cardinality of $S_j$, as $|S_j| = \sum_{\mathbf{h}_j\in[n]^{d_j}} m_j(\mathbf{h}_j)$.

Fix a set of variables $\mathbf{x}$ from $q$, let $d = |\mathbf{x}|$, and denote $\mathbf{x}_j = \mathbf{x} \cap vars(S_j)$ for every $j$. A *statistics of type* $\mathbf{x}$ *for the database* is a vector $\mathbf{m} = (m_1, \ldots, m_\ell)$, where each $m_j$ is an $\mathbf{x}_j$-statistics for $S_j$. We associate with $m$ the function $m : [n]^k \to (\mathbb{N})^\ell$, $m(\mathbf{h}) = (m_1(\mathbf{h}_1), \ldots, m_\ell(\mathbf{h}_\ell))$; here and in the rest of the section, $\mathbf{h}_j$ denotes $\pi_{\mathbf{x}_j}(\mathbf{h})$, *i.e.* the restriction of the tuple $\mathbf{h}$ to the variables in $\mathbf{x}_j$. When $\mathbf{x} = \emptyset$, then $m$ simply consists of $\ell$ numbers, each representing the cardinality of a relation; thus, a $\mathbf{x}$-statistics generalizes the cardinality statistics from section 3. Recall that we use upper case $\mathbf{M} = (M_1, \ldots, M_\ell)$ to denote the same statistics

expressed in bits, i.e. $M_j(\mathbf{h}_j) = a_j m_j(\mathbf{h}_j) \log n$. As before, we fix some constant $0 < \delta < 1$, and assume every relation $S_j$, has cardinality $\leq n^\delta$.

In this section, we fix $\mathbf{x}$-statistics $\mathbf{M}$ and consider the probability space where the instance is chosen uniformly at random over all instances that satisfy $\mathbf{M}$.

To prove the lower bound we need some notations. Let $q_{\mathbf{x}}$ be the *residual query*, obtained by removing all variables $\mathbf{x}$, and decreasing the arities of $S_j$ as necessary: the new arity of $S_j$ is $a_j - d_j$. Clearly, every fractional edge packing of $q$ is also a fractional edge packing of $q_{\mathbf{x}}$, but the converse does not hold in general. Let $\mathbf{u}$ be a fractional edge packing of $q_{\mathbf{x}}$. We say that $\mathbf{u}$ *saturates* a variable $x_i \in \mathbf{x}$, if $\sum_{j:i\in S_j} u_j \geq 1$; we say that $\mathbf{u}$ saturates $\mathbf{x}$ if it saturates all variables in $\mathbf{x}$. For every fractional edge packing $\mathbf{u}$ of $q_{\mathbf{x}}$ that saturates $\mathbf{x}$, denote as before $u = \sum_{j=1}^{\ell} u_j$ and, using $K$ defined in (6):

$$L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p) = \left( \frac{\sum_{\mathbf{h}\in[n]^d} K(\mathbf{u}, \mathbf{M}(\mathbf{h}))}{p} \right)^{1/u} \quad (12)$$

Further denote $L_{\mathtt{lower}} = \max_{\mathbf{u}} L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p)$, and let $c$ be the constant $c = \min_j \frac{a_j - d_j - \delta}{3a_j}$.

THEOREM 4.7. *Given a query $q$, fix statistics $\mathbf{M}$ of type $\mathbf{x}$, where $\mathbf{x}$ is a strict subset of the variables, and consider any deterministic MPC algorithm that runs in one communication round on $p$ servers and has maximum load $L$. Then, for any edge packing $\mathbf{u}$ of $q$ that saturates $\mathbf{x}$, any algorithm that computes $q$ correctly must have maximum load $L \geq cL_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p)$ bits.*

Note that, when $\mathbf{x} = \emptyset$ then $L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p) = L(\mathbf{u}, \mathbf{M}, p)$, defined in (7); therefore, our theorem is a generalization of the simpler lower bound Theorem 3.5. Before we prove the theorem, we show an example.

EXAMPLE 4.8. *We first revisit the lower bound we described in subsection 4.1 for $q(x, y, z) = S_1(x, z), S_2(y, z)$. If $\mathbf{x} = \emptyset$ then the lower bound is $\max_{\mathbf{u}} L(\mathbf{u}, (M_1, M_2), p)$, which is $\max(M_1/p, M_2/p)$, because there are two packings in $pk(q)$: $(1, 0)$ and $(0, 1)$. For $\mathbf{x} = \{z\}$, the residual query is $q_{\mathbf{x}} = S_1(x), S_2(y)$ and its sole packing is $(1, 1)$, which saturates the variable $z$. The packing produces an additional new lower bound: $\sqrt{\sum_{h\in[n]} M_1(h)M_2(h)/p}$. The lower bound is the maximum of these quantities.*

*Next, consider $C_3$. In addition to the lower bounds in Example 3.7, we obtain new bounds by setting $\mathbf{x} = \{x_1\}$. The residual query is $S_1(x_2), S_2(x_2, x_3), S_3(x_3)$ and $(1, 0, 1)$ is a packing that saturates $x_1$ (while for example $(0, 1, 0)$ does not). This gives us a new lower bound, of the form $\sqrt{\sum_{h\in[n]} m_1(h)m_3(h)/p}$.*

In the rest of the section we prove Theorem 4.7. Fix a concrete instance $S_j$, and let $\mathbf{a}_j \in S_j$. We write $\mathbf{a}_j|\mathbf{h}$ to denote that the tuple $\mathbf{a}_j$ from $S_j$ matches with $\mathbf{h}$ at their common variables, and denote $(S_j)_{\mathbf{h}}$ the subset of tuples $\mathbf{a}_j$ that match $\mathbf{h}$: $S_j(\mathbf{h}) = \{\mathbf{a}_j \mid \mathbf{a}_j \in S_j, \mathbf{a}_j|\mathbf{h}\}$. Let $I_{\mathbf{h}}$ denote the restriction of $I$ to $\mathbf{h}$, in other words $I_{\mathbf{h}} = (S_1(\mathbf{h}), \ldots, S_\ell(\mathbf{h}))$.

When $I$ is chosen at random over the probability space defined by the $\mathbf{x}$-statistics $\mathbf{M}$, then, for a fixed tuple $\mathbf{h} \in [n]^d$, the restriction $I_{\mathbf{h}}$ is a uniformly chosen instance over all instances with cardinalities $\mathbf{M}(\mathbf{h})$, which is precisely the probability space that we used in the proof of Theorem 3.6. In particular, for every $\mathbf{a}_j \in [n]^{d_j}$ such that $\mathbf{a}_j|\mathbf{h}$, the probability that $S_j$ contains $\mathbf{a}_j$ is $P(\mathbf{a}_j \in S_j) = m_j(\mathbf{h}_j)/n^{a_j - d_j}$; thus, our proof below is an extension of that of Theorem 3.6. We first compute the expected number of answers for $q$ on the subinstance $I_{\mathbf{h}}$:

LEMMA 4.9. $\mathbf{E}[|q(I_{\mathbf{h}})|] = n^{k-d} \prod_{j=1}^{\ell} \frac{m_j(\mathbf{h}_j)}{n^{a_j - d_j}}$

PROOF. We can write:

$$\mathbf{E}[|q(I_{\mathbf{h}})|] = \sum_{\mathbf{a}|\mathbf{h}} P(\bigwedge_{j=1}^{\ell} (\mathbf{a}_j \in S_j)) = \sum_{\mathbf{a}|\mathbf{h}} \prod_{j=1}^{\ell} P(\mathbf{a}_j \in S_j)$$
$$= \sum_{\mathbf{a}|\mathbf{h}} \prod_{j=1}^{\ell} m_j(\mathbf{h}_j) n^{d_j - a_j} = n^{k-d} \prod_{j=1}^{\ell} m_j(\mathbf{h}_j) n^{d_j - a_j}$$

where the last equality follows from the fact that the number of tuples $\mathbf{a}|\mathbf{h}$ is exactly $n^{k-d}$. □

### *Single Server.*

Let us fix some server and let $m(I)$ be the message the server receives on input $I$. For any fixed value $m$ of $m(I)$, let $K_m(S_j)$ be the set of tuples from relation $S_j$ *known* by the server. Let $w_j(\mathbf{a}_j)$ to denote the probability that the server knows the tuple $\mathbf{a}_j \in S_j$. In other words $w_j(\mathbf{a}_j) = P(\mathbf{a}_j \in K_{m_j(S_j)}(S_j))$, where the probability is over the random choices of $S_j$. This is upper bounded by $P(\mathbf{a}_j \in S_j)$:

$$w_j(\mathbf{a}_j|\mathbf{h}) \leq m_j(\mathbf{h}_j)/n^{a_j - d_j} \quad (13)$$

We derive a second upper bound by exploiting the fact that the server receives a limited number of bits.

LEMMA 4.10. *Let $L$ be the number of bits a server receives. If $a_j > d_j$, then $\sum_{\mathbf{a}_j\in[n]^{a_j}} w_j(\mathbf{a}_j) \leq \frac{L}{ca_j \log(n)}$ for some constant $c > 0$.*

PROOF. Since $\sum_{\mathbf{a}_j\in[n]^{a_j}} w_j(\mathbf{a}_j) = \mathbf{E}[|K_{m(S_j)}(S_j)|]$, we will bound the right hand side. Now, notice that:

$$H(S_j) = H(m(S_j)) + \sum_m P(m(S_j) = m) \cdot H(S_j|m(S_j) = m)$$
$$\leq L + \sum_m P(m(S_j) = m) \cdot H(S_j|m(S_j) = m) \quad (14)$$

For every $\mathbf{h}$, let $K_m(S_j(\mathbf{h}))$ denote the known tuples from the restriction of $S_j$ to $\mathbf{h}$. We can now show that:

$$H(S_j|m(S_j) = m) \leq \sum_{\mathbf{h}} \left(1 - \frac{|K_m(S_j(\mathbf{h}))|}{cm_j(\mathbf{h}_j)}\right) \log \binom{n^{a_j - d_j}}{m_j(\mathbf{h}_j)}$$
$$= H(S_j) - \sum_{\mathbf{h}} \frac{|K_m(S_j(\mathbf{h}))|}{cm_j(\mathbf{h}_j)} \log \binom{n^{a_j - d_j}}{m_j(\mathbf{h}_j)}$$
$$\leq H(S_j) - \sum_{\mathbf{h}} \frac{|K_m(S_j(\mathbf{h}))|}{cm_j(\mathbf{h}_j)} m_j(\mathbf{h}_j)(a_j - d_j - \delta)\log(n)$$
$$= H(S_j) - (1/c) \cdot |K_m(S_j)|(a_j - d_j - \delta)\log(n)$$

where the proof for the first inequality is in the full paper. Plugging this in Equation 14, we have:

$$H(S_j) \leq L + H(S_j) - (1/c) \cdot \mathbf{E}[|K_m(S_j)|](a_j - d_j - \delta)\log(n)$$

or equivalently, since $c \leq 3$:

$$\mathbf{E}[|K_m(S_j)|] \leq \frac{3L}{(a_j - d_j - \delta)\log(n)}$$

This concludes our proof. □

In the case where $a_j = d_j$, the instance $I_{\mathbf{h}}$ specifies exactly the relation $S_j$, and so $w_j(\mathbf{a}_j) \in \{0, 1\}$ for every $\mathbf{a}_j$. Denote by $J(\mathbf{x})$ the set of relations $S_j$ for which $a_j > d_j$.

Recall that $\mathbf{u}$ is a fractional edge packing for the residual query $q_{\mathbf{x}}$ that saturates $\mathbf{x}$. Define the *extended query* $q_{\mathbf{x}}'$ to consists of $q_{\mathbf{x}}$, where we add a new atom $S_i'(x_i)$ for every variable $x_i \in vars(q_{\mathbf{x}})$. Define $u_i' = 1 - \sum_{j:i\in S_j} u_j$. In other words, $u_i'$ is defined to be the slack at the variable $x_i$ of the packing $\mathbf{u}$. The new edge packing $(\mathbf{u}, \mathbf{u}')$ for the extended query $q_{\mathbf{x}}'$ has no more slack, hence it is both a tight fractional edge packing and a tight fractional edge cover for $q_{\mathbf{x}}$. By adding all equalities of the tight packing we obtain:

$$\sum_{j=1}^{\ell}(a_j - d_j)u_j + \sum_{i=1}^{k-d} u_i' = k - d$$

We next compute how many output tuples from $q(I_{\mathbf{h}})$ will be known in expectation by the server. Note that $q(I_{\mathbf{h}}) = q_{\mathbf{x}}(I_{\mathbf{h}})$, and thus:

$$\begin{aligned}\mathbf{E}[|K_m(q(I_{\mathbf{h}}))|] = \mathbf{E}[|K_m(q_{\mathbf{x}}(I_{\mathbf{h}}))|] &= \sum_{\mathbf{a}\in[n]^{k-d}}\prod_{j=1}^{\ell} w_j(\mathbf{a}_j|\mathbf{h}) \\ &= \sum_{\mathbf{a}\in[n]^{k-d}}\prod_{j=1}^{\ell} w_j(\mathbf{a}_j|\mathbf{h})\prod_{i=1}^{k-d} w_i'(\mathbf{a}_i) \\ &\le \prod_{i=1}^{k-d} n^{u_i'}\cdot\prod_{j=1}^{\ell}\left(\sum_{\mathbf{a}\in[n]^{a_j-d_j}} w_j(\mathbf{a}|\mathbf{h})^{1/u_j}\right)^{u_j}\end{aligned}$$

By writing $w_j(\mathbf{a}|\mathbf{h})^{1/u_j} = w_j(\mathbf{a}|\mathbf{h})^{1/u_j-1}w_j(\mathbf{a}|\mathbf{h})$, we can bound the quantity as follows:

$$\begin{aligned}\sum_{\mathbf{a}\in[n]^{a_j-d_j}} w_j(\mathbf{a}|\mathbf{h})^{1/u_j} &\le \left(\frac{m_j(\mathbf{h}_j)}{n^{a_j-d_j}}\right)^{1/u_j-1}\sum_{\mathbf{a}\in[n]^{a_j-d_j}} w_j(\mathbf{a}|\mathbf{h}) \\ &= (m_j(\mathbf{h}_j)n^{d_j-a_j})^{1/u_j-1}L_j(\mathbf{h})\end{aligned}$$

where $L_j(\mathbf{h}) = \sum_{\mathbf{a}\in[n]^{a_j-d_j}} w_j(\mathbf{a}|\mathbf{h})$. Notice that for every relation $S_j$, $\sum_{\mathbf{h}_j\in[n]^{d_j}} L_j(\mathbf{h}_j) = \sum_{\mathbf{a}_j\in[n]^{a_j}} w_j(\mathbf{a}_j)$.

$$\begin{aligned}&\mathbf{E}[|K_m(q(I_{\mathbf{h}}))|] \\ &\le n^{\sum_{i=1}^{k-d}u_i'}\prod_{j=1}^{\ell}\left(L_j(\mathbf{h})m_j(\mathbf{h}_j)^{1/u_j-1}n^{(d_j-a_j)(1/u_j-1)}\right)^{u_j} \\ &= \prod_{j=1}^{\ell} L_j(\mathbf{h})^{u_j}\cdot\prod_{j=1}^{\ell} m_j(\mathbf{h}_j)^{-u_j}\cdot\mathbf{E}[|q(I_{\mathbf{h}})|]\end{aligned}$$

### *All Servers.*

Let us first index the quantities $K_m, L_j$ with the server $s = 1, \ldots, p$ they correspond to: $K_m^s, L_j^s$. Let $\mathcal{H}$ be the set of $\mathbf{h} \in [n]^d$ such that for every $j \notin J(\mathbf{x})$, $m_j(\mathbf{h}) = 1$. Observe that if $\mathbf{h} \notin \mathcal{H}$, then for some $j \notin J(\mathbf{x})$ we have $m_j(\mathbf{h}) = 0$, which implies that $q(I_{\mathbf{h}}) = \emptyset$ independent of the random subinstance. For some $\mathbf{h} \in [n]^d$, the expected number of answers that all servers will produce for the subinstance $I_{\mathbf{h}}$ is $\mathbf{E}[|K_m(q(I_{\mathbf{h}}))|] = \sum_s \mathbf{E}[|K_m^s(q(I_{\mathbf{h}}))|]$. If some $\mathbf{h} \in \mathcal{H}$ this number is not at least $\mathbf{E}[|q(I_{\mathbf{h}})|]$, the algorithm will fail to compute $q(I)$ (since any $\mathbf{h} \notin \mathcal{H}$ never produces answers, we do not need to consider it). Consequently, for every $\mathbf{h} \in \mathcal{H}$ we must have that

$$\sum_s \prod_{j\in J(\mathbf{x})} L_j^s(\mathbf{h}_j)^{u_j} \ge \prod_{j\in J(\mathbf{x})} m_j(\mathbf{h}_j)^{u_j} \tag{15}$$

Summing the inequalities for every $\mathbf{h} \in \mathcal{H}$:

$$\begin{aligned}\sum_{\mathbf{h}\in\mathcal{H}}\prod_{j\in J(\mathbf{x})} m_j(\mathbf{h}_j)^{u_j} &\le \sum_{\mathbf{h}\in\mathcal{H}}\sum_s\prod_{j\in J(\mathbf{x})} L_j(\mathbf{h}_j)^{u_j} \\ &= \sum_s\left(\sum_{\mathbf{h}\in\mathcal{H}}\prod_{j\in J(\mathbf{x})} L_j(\mathbf{h}_j)^{u_j}\right) \\ &\le \sum_s\prod_{j\in J(\mathbf{x})}\left(\sum_{\mathbf{h}_j} L_j^s(\mathbf{h}_j)\right)^{u_j}\end{aligned}$$

where the last inequality comes from the following application of Friedgut's inequality.

LEMMA 4.11. *If* $\mathbf{u}$ *saturates* $\mathbf{x}$ *in query* $q$,

$$\sum_{\mathbf{h}}\prod_j L_j(\mathbf{h}_j)^{u_j} \le \prod_j\left(\sum_{\mathbf{h}_j} L_j(\mathbf{h}_j)\right)^{u_j}$$

Since we have $M_j = a_j m_j \log(n)$, and for any relation where $j \in J(\mathbf{x})$ also $\sum_{\mathbf{h}_j} L_j^s(\mathbf{h}_j) \le L/(ca_j\log(n))$ (from Lemma 4.10) for every server $s$, we obtain that :

$$\begin{aligned}\sum_{\mathbf{h}}\prod_{j=1}^{\ell} M_j(\mathbf{h}_j)^{u_j} &= \sum_{\mathbf{h}\in\mathcal{H}}\prod_{j\in J(\mathbf{x})} M_j(\mathbf{h}_j)^{u_j} \le \sum_s\prod_{j\in J(\mathbf{x})}(L/c)^{u_j} \\ &= p\left(\frac{L}{c}\right)^{\sum_{j\in J(\mathbf{x})}u_j} \le p\left(\frac{L}{c}\right)^{\sum_j u_j}\end{aligned}$$

which completes the proof of Theorem 4.7.

**Upper and Lower Bound Comparison.** To compare the lower bound with the upper bound we obtained from the BinHC algorithm in Theorem 4.6, we apply the lower bound for a particular bin combination $\mathcal{B}$ with set $\mathbf{x}$. In this case, the definition of bin exponents implies that $m_j(\mathbf{h}_j) \ge m_j/(2p^{\beta_j})$ for the heavy hitters. Then:

$$\begin{aligned}L &\ge \max_{\mathbf{u}'}\left(\frac{\sum_{\mathbf{h}\in[n]^d}\prod_j M_j(\mathbf{h}_j)^{u_j'}}{p}\right)^{1/u'} \\ &\ge \max_{\mathbf{u}'}\left(\frac{p^\alpha\prod_j(M_j/2p^{\beta_j})^{u_j'}}{p}\right)^{1/u'} \\ &= \frac{1}{2}\max_{\mathbf{u}'}\left(\frac{\prod_j(M_j/p^{\beta_j})^{u_j'}}{p^{1-\alpha}}\right)^{1/u'} \\ &= \frac{1}{2}\max_{\mathbf{u}'}\{L(\mathbf{u}', \mathbf{M}(\mathcal{B}), p^{1-\alpha})\}\end{aligned}$$

where, in contrast to Theorem 4.6, the edge packing $\mathbf{u}'$ ranges only over edge packings of the residual query $q_{\mathbf{x}}$ that *saturate* all variables in $\mathbf{x}$.

## 5. MAP-REDUCE MODELS

In this section, we discuss the connection between the MPC model and the Map-Reduce model presented by Afrati et. al [1]. In contrast to the MPC model, where the number of servers $p$ is the main parameter, in the model of [1]

the main parameter is an upper bound $q$ on the number of input tuples a reducer can receive, which is called *reducer size*. Given an input $I$, a Map-Reduce algorithm is restricted to deterministically send each input tuple independently to some reducer, which will then produce all the outputs that can be extracted from the received tuples. If $q_i \leq q$ is the number of inputs assigned to the $i$-th reducer, where $i = 1, \ldots, p$, we define the *replication rate* $r$ of the algorithm $r = \sum_{i=1}^{p} q_i/|I|$.

In [1], the authors provide lower and upper bounds on $r$ with respect to $q$ and the size of the input. However, their results are restricted to binary relations where all sizes are equal, and they provide matching upper and lower bounds only for a subclass of such queries. We show next how to apply the results of this paper to remove these restrictions, and further consider an even stronger computation model for our lower bounds.

First, we express the bound on the data received by the reducers in *bits*: let $L$ be the maximum number of bits each reducer can receive. The input size $|I|$ is also expressed in bits. We will also allow any algorithm to use randomization. Finally, we relax the assumption on how the inputs are communicated to the reducers: instead of restricting each input tuple to be sent independently, we assume that the input data is initially partitioned into a number of *input servers* $p_0$ (where $p_0$ must be bigger than the query size), and allow the algorithm to communicate bits to reducers by accessing all the data in one such input server. Notice that this setting allows for stronger algorithms that use input statistics to improve communication.

If each reducer receives $L_i$ bits, the replication rate $r$ is defined as $r = \sum_{i=1}^{p} L_i/|I|$. Notice that any algorithm with replication rate $r$ must use $p \geq (r|I|)/L$ reducers. Now, let $q$ be a conjunctive query, where $S_j$ has $M_j = a_j m_j \log n$ bits ($n$ is the size of the domain).

THEOREM 5.1. *Let $q$ be a conjunctive query where $S_j$ has size (in bits) $M_j$. Any algorithm that computes $q$ with reducer size $L$, where $L \leq M_j$ for every $S_j$[7] must have replication rate*

$$r \geq \frac{c^u L}{\sum_j M_j} \max_{\mathbf{u}} \prod_{j=1}^{\ell} \left(\frac{M_j}{L}\right)^{u_j}$$

*where $\mathbf{u}$ ranges over all fractional edge packings of $q$.*

PROOF. Let $f_i$ be the fraction of answers returned by server $i$, in expectation, where $I$ is a randomly chosen database with statistics $\mathbf{M}$. By Theorem 3.5, $f_i \leq \frac{L_i^u}{c^u K(\mathbf{u},\mathbf{M})}$. Since we assume all answers are returned,

$$1 \leq \sum_{i=1}^{p} f_i = \sum_{i=1}^{p} \frac{L_i^u}{c^u K(\mathbf{u},\mathbf{M})} = \frac{\sum_{i=1}^{p} L_i L_i^{u-1}}{c^u K(\mathbf{u},\mathbf{M})}$$
$$\leq \frac{L^{u-1} \sum_{i=1}^{p} L_i}{c^u K(\mathbf{u},\mathbf{M})} = \frac{L^{u-1} r|I|}{c^u K(\mathbf{u},\mathbf{M})}$$

where we used the fact that $u \geq 1$ for the optimal $\mathbf{u}$. The claim follows by using the definition of $K$ (6) and noting that $|I| = \sum_j M_j$. □

We should note here that, following from our analysis in section 3, the bound provided by Theorem 5.1 is matched by the HYPERCUBE algorithm with appropriate shares. We illustrate Theorem 5.1 with an example.

EXAMPLE 5.2 (TRIANGLES). *Consider the triangle query $C_3$ and assume that all sizes are equal to $M$. In this case, the edge packing that maximizes the lower bound is the one that maximizes $\sum_j u_j$, $(1/2, 1/2, 1/2)$. Thus, we obtain a bound $\Omega(\sqrt{M/L})$ for the replication rate. This is exactly the formula proved in [1], but notice that we can derive bounds even if the sizes are not equal. Further, observe that any algorithm must use at least $\Omega((M/L)^{3/2})$ reducers.*

# 6. CONCLUSIONS

In this paper we have studied the parallel query evaluation problem on databases in two settings: the first with known cardinalities, and the second with additionally known heavy hitters and their frequency. In the first case we have given matching lower and upper bounds (within a polylog factor of $p$) that are described in terms of *fractional edge packings* of the query. In the second case we have shown both lower and upper bounds described in terms of fractional packings of the residual queries, one residual query for each type of heavy hitter.

**Acknowledgements.** This work was partially supported by NSF grants IIS-1247469 and CCF-1217099.

# 7. REFERENCES

[1] F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.
[2] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, pages 99–110, 2010.
[3] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748, 2008.
[4] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *PODS*, pages 273–284, 2013.
[5] P. Beame, P. Koutris, and D. Suciu. Skew in parallel query processing. *CoRR*, abs/1401.1872, 2014.
[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
[7] E. Friedgut. Hypergraphs, entropy, and inequalities. *American Mathematical Monthly*, pages 749–760, 2004.
[8] S. Ganguly, A. Silberschatz, and S. Tsur. Parallel bottom-up processing of datalog queries. *J. Log. Program.*, 14(1&2):101–126, 1992.
[9] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.
[10] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms: [extended abstract]. In *PODS*, pages 37–48, 2012.
[11] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD Conference*, pages 1099–1110, 2008.
[12] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614, 2011.
[13] C. B. Walton, A. G. Dale, and R. M. Jenevein. A taxonomy and performance model of data skew effects in parallel joins. In *VLDB*, pages 537–548, 1991.
[14] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In *SIGMOD Conference*, pages 13–24, 2013.
[15] Y. Xu, P. Kostamaa, X. Zhou, and L. Chen. Handling data skew in parallel joins in shared-nothing systems. In *SIGMOD Conference*, pages 1043–1052, 2008.
[16] A. C. Yao. Lower bounds by probabilistic arguments. In *FOCS*, pages 420–428, Tucson, AZ, 1983.
[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

[7] if $L > M_j$, we can send the whole relation to any reducer without cost