Deterministic Load Balancing for Parallel Joins (Extended Abstract)

Paraschos Koutris
University of Wisconsin-Madison
Madison, WI
paris@cs.wisc.edu

Nivetha Singara Vadivelu
University of Wisconsin-Madison
Madison, WI
nivethavadivelu@cs.wisc.edu

ABSTRACT

We study the problem of distributing the tuples of a relation to a number of processors organized in an r-dimensional hypercube, which is an important task for parallel join processing. In contrast to previous work, which proposed randomized algorithms for the task, we ask here the question of how to construct efficient *deterministic* distribution strategies that can optimally load balance the input relation. We first present some general lower bounds on the load for any dimension; these bounds depend not only on the size of the relation, but also on the maximum frequency of each value in the relation. We then construct an algorithm for the case of 1 dimension that is optimal within a constant factor, and an algorithm for the case of 2 dimensions that is optimal within a polylogarithmic factor. Our 2-dimensional algorithm is based on an interesting connection with the *vector load balancing problem*, a well-studied problem that generalizes classic load balancing.

CCS Concepts

•Information systems → Relational parallel and distributed DBMSs; •Theory of computation → Parallel computing models;

1. INTRODUCTION

A central task in large-scale data analytics systems [7, 10, 6] is the computation of multi-way joins. To compute a join efficiently in parallel, the most widely used techniques are hash-based methods. For example, the standard parallel hash-join algorithm computes one join at a time, and computes the join between two relations by partitioning each relation through the application of a hash function on the join attribute. In recent years there has been a development of new parallel multi-way join algorithms (the SHARES or HYPERCUBE algorithm [1, 3]), which compute a multi-way join in a single step by organizing the available servers in the cluster on a multi-dimensional hypercube and then hashing each relation on multiple attributes.

BeyondMR'16, June 26-July 01 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2926534.2926536

In this family of parallel join algorithms, the core problem is how to distribute a relation $R(X_1, \ldots, X_\alpha)$ among p available servers. These servers have been organized in an r-dimensional hypercube with dimensions $p_1, p_2, \ldots, p_r$, such that $\prod_{i=1}^{r} p_i = p$. (Choosing the dimensions of the hypercube is done by the join algorithm and thus the p_i's are given.) The distribution of the tuples of R must be performed in such a way that for any two tuples t, t' if $t.X_i = t'.X_i$ (i.e. they agree on attribute X_i), then t, t' must be located in servers that have the same i-th coordinate.

Since there are multiple ways we can distribute the tuples such that the above condition is satisfied, our goal is to do this by minimizing the maximum number of tuples, called the *maximum load* L, that each server receives; this achieves the best possible load balancing of the data across the servers. If the relation R has m tuples and we have p available servers, ideally we would like to achieve the optimal maximum load of $L = m/p$. However, this is not always feasible. For example, say that we have a binary relation $R(X_1, X_2)$, and suppose that a single value appears at the X_1 attribute of all m tuples. It is easy to see that the problem then becomes equivalent to distributing a unary relation $R'(X_2)$ over p_2 servers; hence, the load cannot be better than m/p_2 in this case.

It has been shown in [4] that the optimal load m/p can be achieved with high probability within a polylogarithmic factor if the relational instance satisfies certain skew conditions. For example, in the case of the binary relation, each value in the X_1 attribute must appear at most $c \cdot m/p_1$ times, and each value in the X_2 attribute must appear at most $c \cdot m/p_2$ times for some constant c. Then it can be shown that distributing the tuples by applying a random hash function independently on each attribute achieves a load of $O(\frac{m}{p} \log^2 p)$ with probability $1 - 1/p^b$ for some constant $b > 1$.

In this paper, we study the following question: *can we construct an efficient deterministic algorithm that distributes the tuples in R such that the load is always as close to the optimal value as possible?* Further, we seek to obtain optimality guarantees under any skew conditions, and not only for the case of no data skew.

Our contributions. We summarize the contributions.

- We present a general lower bound (Section 3) for the load. The bound depends not only on the size m and the number of servers p, but also on the maximum degrees of each value (or combination of values) in R.
- We present a fast deterministic algorithm, which we call GREEDYBALANCE, for the case of 1 dimension

(Section 4). The algorithm is optimal within a constant factor of the lower bound.

- We construct and analyze a fast deterministic algorithm, called 2-BALANCE, for the case of 2 dimensions (Section 6), which is optimal within a polylogarithmic factor of the lower bound. Our algorithm uses as components the greedy algorithm for one dimension, and an algorithm for the *vector load balancing* problem.

We should note here two things. First, even for the case of one dimension, the problem of finding the optimal distribution is already NP-complete. Thus, our only hope is to construct fast approximation algorithms. For the case of two dimensions, we show in Section 3 that we cannot hope for even a constant approximation. Second, we are interested in proving that our algorithms achieve a load that is close to the lower bound that depends on the maximum degrees in the relation, instead of the optimal value. This is necessary since size information and maximum degrees are used to optimize the choice of dimensions for the hypercube.

We leave as an open problem whether we can construct efficient and optimal (or almost-optimal) algorithms for the case of more than 2 dimensions.

2. BACKGROUND

Let $R(X_1, \ldots, X_\alpha)$ be a relation of arity α. Throughout this paper, we use m to denote the number of tuples in R. We denote by $\mathsf{adom}(X_i)$ the active domain of X_i, which is all possible values that attribute X_i can take in R. For a given value a of attribute X_i, we define the *degree* of a, denoted $d_i(a)$, as $d_i(a) = |\{t \in R \mid t.X_i = a\}|$. More generally, given a tuple $\vec{a}$ over a set of attributes $U \subseteq \{X_1, \ldots, X_r\}$, we define the degree of $\vec{a}$ as $d_U(\vec{a}) = |\{t \in R \mid \forall X_i \in U : t.X_i = a.X_i\}|$. We assume that there are p processors, which are organized in an r-dimensional hypercube ($1 \leq r \leq \alpha$) with dimensions $p_1, p_2, \ldots, p_r$, such that $\prod_{i=1}^{r} p_i = p$. Each server s is identified with a point in the r-dimensional space $\mathcal{P} = [p_1] \times [p_2] \times \cdots \times [p_r]$.

DEFINITION 2.1 (DISTRIBUTION STRATEGY). *A distribution strategy $\mathcal{D}$ is a mapping from each tuple of R to $\mathcal{P}$, such that for any two tuples $t_1, t_2 \in R$, if for some $i = 1, \ldots, r$ we have $t_1.X_i = t_2.X_i$ then $\mathcal{D}(t_1)[i] = \mathcal{D}(t_2)[i]$.*

In other words, when two tuples have the same value for attribute X_i, they must be mapped to servers that have the same i-th coordinate in the hypercube. This type of restriction is necessary in order to allow us to perform correctly join queries on the input relations (for example see the HYPERCUBE algorithm), since we have to make sure that two tuples from different relations that join on some attribute will meet in a common server. Notice also that we use only the first r attributes to distribute the relation; we can assume this without any loss of generality.

Given a distribution strategy $\mathcal{D}$, the load of a server s, denoted L_s is defined as: $L_s = |\{t \in R \mid \mathcal{D}(t) = s\}|$.

DEFINITION 2.2 (LOAD). *The* maximum load *L of a distribution strategy $\mathcal{D}$ is defined as the maximum number of tuples that are assigned to any server, $L = \max_s\{L_s\}$.*

In this paper we examine the following question. *Given a relation R, can we compute efficiently a distribution strategy over an r-dimensional hypercube $\mathcal{P}$ that achieves the minimum possible load L?*

2.1 Randomized Algorithm

In previous work [4], a simple randomized distribution strategy was proposed. The strategy picks independently for each $i = 1, \ldots, m$ a perfectly random hash function $h_i : \mathsf{adom}(X_i) \to \{1, \ldots, p_i\}$. Then, we send each tuple $R(a_1, \ldots, a_r, \ldots)$ to the server with coordinates specified by the hash functions: $(h_1(a_1), \ldots, h_r(a_r))$. The following result can be shown about the load achieved:

THEOREM 2.1. *[4] Suppose that for every set of attributes $U \subseteq \{X_1, \ldots, X_r\}$, any tuple $\vec{a}$ over U has degree $d_U(\vec{a}) \leq c \cdot m/p_U$ for some constant c, where $p_U = \prod_{i:X_i \in U} p_i$. Then, the randomized distribution strategy achieves maximum load $O(m/p \cdot \ln^r(p))$ with high probability.*

In other words, if the degrees of each value, or combination of values, is small enough, the randomized strategy will give an almost optimal load (up to a polylogarithmic factor), since m/p is a lower bound. In this work we will focus on deterministic distribution strategies.

2.2 Vector Load Balancing

We discuss here the problem of *vector load balancing* (VLB). Vector load balancing is a special type of load balancing where we do not distribute scalar values, but vectors of values. The problem was first introduced by Chekuri and Khanna [5]. Later work [8, 2, 9] approached vector load balancing from an online perspective.

DEFINITION 2.3 (VECTOR LOAD BALANCING). *There exist n machines, each with d components. A* job *j is associated with a d-dimensional vector a_j, where the k^{th} coordinate a_j^k denotes the load placed on component k by job j. Let l_i^k denote the sum of a_j^k over all jobs j that are assigned to machine i. The goal is to assign jobs to machines such that we minimize the* makespan $\max_{i,k} l_i^k$.

The authors in [5] show that it is hard to approximate VLB to within *any constant* when the number of components d can be arbitrarily large. Subsequent work [9] showed that an online algorithm (where in this case we look at one job at a time and we have to allocate this job to a machine irrevocably) can achieve a $O(\log d)$ competitive ratio. This result implies an offline algorithm with the same approximation ratio. [8] improves the competitive ratio to $O(\log d / \log\log d)$, and proves that it is tight.

3. LOWER BOUNDS AND HARDNESS

In this section, we provide lower bounds for the optimal load of any (deterministic or randomized) distribution strategy, and also show some hardness results. The lower bound depends not only on the size of the relation m and the number of servers p, but also on the maximum degrees of each value in the relation. For any subset of attributes U, let us denote $d_U = \max_{\vec{a}}\{d_U(\vec{a})\}$.

It is straightforward to see that an immediate lower bound for the maximum load for any $r \geq 1$ is $L \geq m/p$. However, by using the maximum degree information (which captures the amount of skew), we can obtain better lower bounds.

LEMMA 3.1. *Any distribution strategy on an r-dimensional hypercube with dimensions $p_1, \ldots, p_r$ requires load*

$$L \geq \max\left(\frac{m}{p}, \max_U \frac{d_U \cdot p_U}{p}\right) \quad (1)$$

For the case of one dimension, where we distribute a relation $R(X,\dots)$ using only the attribute X, the lemma gives a lower bound $L \geq \max(m/p, d_X)$. In the case of a 2-dimensional hypercube that distributes the first two attributes of the relation $R(X,Y,\dots)$, the above lemma gives similarly a lower bound of $L \geq \max\left(\frac{m}{p}, \frac{d_X}{p_Y}, \frac{d_Y}{p_X}, d_{XY}\right)$.

LEMMA 3.2. *Any distribution strategy on an r-dimensional hypercube with dimensions $p_1,\dots,p_r$ requires load $L \geq \frac{m}{\min_i p_i}$. Moreover, there exists a data distribution for R for which the above lower bound is tight.*

Regarding the hardness of computing an optimal solution, we can show that the problem is already NP-complete for the case of one dimension.

LEMMA 3.3. *Computing the optimal strategy for distributing R along a 1-dimensional hypercube is NP-complete.*

For the case of two dimensions or more, we can show an even stronger result, which is that it is not possible to approximate the optimal solution within a constant. We prove this result by a reduction from vector load balancing.

LEMMA 3.4. *Unless $NP = ZPP$, for any constant $c > 1$, there is no deterministic polynomial algorithm that approximates the optimal value of Eq.(1) to within a factor of c.*

4. THE GREEDY ALGORITHM

In this section, we present a deterministic distribution strategy that achieves a load that is only a constant factor away from the lower bound in the 1-dimensional case. We call this algorithm GREEDYBALANCE; we will further use this strategy as a component of our algorithm for 2 dimensions. GREEDYBALANCE takes as input a relation $R(X_1,\dots)$ of size m, and distributes the tuples along a 1-dimensional hypercube (where dimension X_1 has size p).

GREEDYBALANCE iterates in an arbitrary order through all $a \in \mathsf{adom}(X_1)$; for each a, it allocates all tuples with $t.X_1 = a$ to the first X_1-coordinate with load $< m/p$.

Algorithm 1 GREEDYBALANCE

```
1: i ← 1
2: for all a ∈ adom(X_1) do
3:     if L_(i) ≤ m/p then
4:         for all t ∈ R s.t. t.X_1 = a do
5:             D(t) ← (i)
6:     else  i ← i + 1
```

LEMMA 4.1. GREEDYBALANCE *allocates all tuples in R to some processor and achieves maximum load $L \leq \frac{m}{p} + d_{X_1}$.*

Since $\frac{m}{p} + d_{X_1} \leq 2\max(\frac{m}{p}, d_{X_1})$, GREEDYBALANCE achieves a load that is a factor 2 away from the lower bound. The example below shows that this factor is essentially tight.

EXAMPLE 4.1. *Consider an instance with the following structure. We have $|\mathsf{adom}(X_1)| = p+1$, where p values have degree $\frac{m}{p} - \frac{m}{p^2}$ and one value has degree $\frac{m}{p}$ (observe that the total number of tuples is m). GREEDYBALANCE in this case will achieve a load of at most $2\frac{m}{p}$. Now, since we have $p+1$ values, at least two of these values must be allocated to the same X_1 coordinate; hence the load cannot be smaller than $2(\frac{m}{p} - \frac{m}{p^2}) = 2\frac{m}{p}(1-1/p)$. Notice that as p grows, this lower bound in the limit becomes $2\frac{m}{p}$, which implies that GREEDYBALANCE is asymptotically optimal.*

5. VECTOR LOAD BALANCING

To see how vector load balancing is related to the problem of finding a distribution strategy, consider the 2-dimensional hypercube, and suppose that we are already given a partitioning of the tuples in $R(X,Y,\dots)$ according to the Y attribute into the p_Y coordinates. To complete the distribution strategy, we need to specify how to partition the values in $\mathsf{adom}(X)$ to the X coordinates of the hypercube. We can now view each value $j \in \mathsf{adom}(X)$ as a job of VLB, and each one of the p_X coordinates as a machine. The k-th coordinate a_j^k of job j in this case is the number of tuples of the form $R(j,b,\dots)$ for which b is assigned to the k-th Y coordinate. Observe that by solving the vector load balancing problem we seek to find the minimum makespan, which is equivalent to minimizing the maximum load L.

For our application of vector load balancing, we use a restricted version of VLB, for which we have additional constraints on the vector structure. In particular, we assume there exists a parameter Λ such that:

$$\Lambda \geq \max\left(\max_{j,k} a_j^k, \max_k \left(\sum_j a_j^k\right)/n\right)$$

We denote this instance of vector load balancing Λ-VLB. We will next solve this restricted version of vector load balancing by applying a simple modified version of the algorithm of [9]. Notice that the choice of Λ is not arbitrary. In particular, Λ is a lower bound for the best possible load that any algorithm can achieve. We will use this bound as an estimation of the optimal load for the algorithm. Let $\beta = (1+\frac{1}{\gamma})^{1/\Lambda}$ for some constant $\gamma > 1$. We use $l_i^k(j)$ to denote the load on component k of machine i once the jobs up to job j have been assigned.

Algorithm 2 Online Algorithm for Λ-VLB

```
1: β ← (1 + 1/γ)^(1/Λ)
2: for all jobs j do
3:     assign job j to machine
           s = arg min_i Σ_{k=1}^{d} [ β^(l_i^k(j-1)+a_j^k) − β^(l_i^k(j-1)) ]
4:     For all k, let l_s^k(j) ← l_s^k(j−1) + a_j^k
```

We prove the following theorem on the makespan of the algorithm in the full version of this paper.

THEOREM 5.1. *Algorithm 2 always achieves a makespan of $\Lambda \cdot O(\log nd)$ for the Λ-VLB problem.*

6. AN ALGORITHM FOR 2 DIMENSIONS

In this section, we present a deterministic algorithm, which we call 2-BALANCE, that constructs a distribution strategy that approximates the optimal lower bound within a polylogarithmic factor for the case of two dimensions. We assume a relation $R(X,Y,\dots)$ which we distribute across

a 2-dimensional hypercube (rectangle) using the X, Y attributes. The rectangle has p_X rows and p_Y columns. The total number of processors is $p = p_X \cdot p_Y$.

Let the degree of the X attribute be at most d_X and the degree of Y attribute be at most d_Y. We need to make two additional assumptions: (a) $m \geq p^{3/2}$, in other words the number of tuples is much larger than the number of processors, and (b) $d_{XY} = 1$, in other words each pair a, b for the X, Y attributes appears exactly once. The second assumption is equivalent to the case where the relation R contains only X, Y and so it is of the form $R(X, Y)$.

Let us assume without loss of generality that $p_X \geq p_Y$. The algorithm 2-BALANCE first splits the values in $\mathsf{adom}(X)$ into two disjoint subsets:

1. **Heavy** (R^H): values for which $d_X(a) \geq p_Y$.
2. **Light** (R^L): values for which $d_X(a) < p_Y$.

The key idea is as follows. We will first decide the distribution of the values in $\mathsf{adom}(X)$, and then we will use vector load balancing to distribute the values in $\mathsf{adom}(Y)$ conditioned on the assignment of the X values. To distribute the values in $\mathsf{adom}(X)$, we will use the splitting into heavy and light values. The values in the heavy set this case will be partitioned using GREEDYBALANCE. As for the light values, we will partition first according to the Y coordinate (i.e. columns) using GREEDYBALANCE, and then apply vector load balancing algorithm to distribute the light X values along the rows. We next present a detailed description and analysis of 2-BALANCE.

Distributing Heavy X. We apply GREEDYBALANCE to assign each tuple to a specific row based on its X attribute (hence all tuples having that value for X are assigned to the same row). Lemma 4.1 implies that the total load on each row will be at most $2\max(\frac{m}{p_X}, d_X)$. Moreover:

LEMMA 6.1. *If we apply* GREEDYBALANCE *to distribute R^H according to the X attribute, every value in $\mathsf{adom}(Y)$ appears at most $2\max(\frac{m}{p}, \frac{d_X}{p_Y})$ times in each row.*

Distributing Light X. To distribute properly the heavy values of $\mathsf{adom}(X)$, we start by assigning tuples to columns according to the Y attribute by applying GREEDYBALANCE. As we noted in the previous section, we can now view this as an instance of VLB, where each job j corresponds to a value in $\mathsf{adom}(X)$, each component to a column of the rectangle, and the value a_j^k of a job vector is the number of tuples t such that $t.X = j$ and t is distributed to column k. The bound on the total load of each column guaranteed by GREEDYBALANCE implies that for all k, $\sum_j a_j^k \leq 2\max(\frac{m}{p_Y}, d_Y)$. We next show that a_j^k will be appropriately bounded as well.

LEMMA 6.2. *If we apply* GREEDYBALANCE *to distribute R^L according to the Y attribute, every value in $\mathsf{adom}(X)$ appears at most $2\max(\frac{m}{p}, \frac{d_Y}{p_X})$ times in each column.*

Thus, for every j, k we have $a_j^k \leq 2\max\{\frac{m}{p}, \frac{d_Y}{p_X}\}$. The number of machines for VLB is now $n' = p_X$ and the number of components $d' = p_Y$. Consequently, we have an instance of the Λ'-VLB problem for $\Lambda' = 2\max(\frac{m}{p}, \frac{d_Y}{p_X})$. Thus, if we run the algorithm for Λ'-VLB to distribute the light X values, from Lemma 5.1 we obtain that the maximum load will be

$$L = \Lambda' \cdot O(\log(n'd')) = \max\left(\frac{m}{p}, \frac{d_Y}{p_X}\right) \cdot O(\log p).$$

This implies that for the distribution of the light values, every value in $\mathsf{adom}(Y)$ appears at most $\max(\frac{m}{p}, \frac{d_Y}{p_X}) \cdot O(\log p)$ times in each row. Also, the load per row will be at most $\max(\frac{m}{p_X}, \frac{d_Y}{p_X} p_Y) \cdot O(\log p)$.

Distributing Y. We have thus far provided a distribution of both the heavy and light values in $\mathsf{adom}(X)$. We use this distribution and apply VLB along the columns. In this case, each job is a value in $\mathsf{adom}(Y)$, each component of the job is a row of the rectangle (hence $d = p_X$), and a_j^k is the number of tuples t such that $t.Y = j$ and t is distributed to row k. We also have $n = p_Y$. Let $\Lambda = O(\log p) \cdot \max(\frac{m}{p}, \frac{d_X}{p_Y}, \frac{d_Y}{p_X})$. Our analysis so far guarantees that for each j, k we have $a_j^k \leq \Lambda$, and also for each k, $\sum_j a_j^k \leq \Lambda \cdot p_Y$. We can now apply Lemma 5.1 to obtain that after we run the algorithm for Λ-VLB to distribute the Y values, the load is $\Lambda \cdot O(\log nd)$.

THEOREM 6.3. *Suppose we have a binary relation $R(X, Y)$ with size $m \geq p^{3/2}$. Then,* 2-BALANCE *describes a deterministic distribution strategy that achieves maximum load*

$$L = \max\left(\frac{m}{p}, \frac{d_Y}{p_X}, \frac{d_X}{p_Y}\right) \cdot O(\log^2 p)$$

Following directly from the above theorem, we see that our algorithm is optimal within a $O(\log^2 p)$ factor.

7. CONCLUSION

We studied the problem of distributing a relation to a number of processors organized in an r-dimensional hypercube. We make progress by discussing lower bounds, and constructing algorithms for the case of 1 and 2 dimensions. There are several open questions. For example, can we extend our techniques to construct an algorithm that works for dimensions ≥ 3? Also, although our algorithms are very efficient, they still require centralized decisions in order to distribute the data. This raises the challenge of designing deterministic strategies that work well in a distributed setting, where the coordination must be minimal.

8. REFERENCES

[1] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, pages 99–110, 2010.

[2] Y. Azar, I. R. Cohen, S. Kamara, and B. Shepherd. Tight bounds for online vector bin packing. In *STOC*, pages 961–970, 2013.

[3] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *PODS*, pages 273–284, 2013.

[4] P. Beame, P. Koutris, and D. Suciu. Skew in parallel query processing. In *PODS*, pages 212–223, 2014.

[5] C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, Apr. 2004.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[7] D. Halperin, V. T. de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, S. Xu, M. Balazinska, B. Howe, and D. Suciu. Demonstration of the Myria big data management service. In *SIGMOD*, pages 881–884, 2014.

[8] S. Im, N. Kell, J. Kulkarni, and D. Panigrahi. Tight bounds for online vector scheduling. In *FOCS*, pages 525–544, 2015.

[9] A. Meyerson, A. Roytman, and B. Tagiku. Online multidimensional load balancing. In *APPROX-RANDOM*, pages 287–302, 2013.

[10] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In *SIGMOD*, pages 13–24, 2013.