A New Framework for Join Product Skew

Foto Afrati[1], Victor Kyritsis[1], Paraskevas Lekeas[2], Dora Souliou[1]

[1] National Technical University of Athens, Athens, Greece,
{afrati, vkyri}@cs.ntua.gr, dsouliou@mail.ntua.gr
[2] Department of Applied Mathematics, University of Crete, Herakleio, Greece,
plekeas@tem.uoc.gr

Abstract. Different types of data skew can result in load imbalance in the context of parallel joins under the shared nothing architecture. We study one important type of skew, join product skew (JPS). A static approach based on frequency classes is proposed which takes for granted the data distribution of join attribute values. It comes from the observation that the join selectivity can be expressed as a sum of products of frequencies of the join attribute values. As a consequence, an appropriate assignment of join sub-tasks, that takes into consideration the magnitude of the frequency products can alleviate the join product skew. Motivated by the aforementioned remark, we propose an algorithm, called Handling Join Product Skew (HJPS), to handle join product skew.

Key words: Parallel DBMS, join operation, data distribution, data skew, load imbalance, shared nothing architecture

1 Introduction

The limited potentials of centralized database systems in terms of the storage and the process of large volumes of data has led to the advent of parallel database management systems (PDBMS) that adopt the shared-nothing architecture. According to this architecture, each computational node (database processor) has its own memory and CPU and independently accesses its local disks while it is provided with the ability to perform locally relational operations. By definition, the aforementioned architecture favors the deployment of data intensive scale computing applications [13] by reducing the complexity of the underlying infrastructure and the overall cost as well.

Within the scope of the parallel evaluation of the relational operators by splitting them into many independent operators (*partitioned parallelism*), sort-merge join and hash-join constitute the main algorithms for the computation of the equijoin. Equijoin is a common special case of the join operation $R \bowtie S$, where the join condition consists solely of equalities of the form $R.X = S.Y$ (X and Y are assumed to be attributes of the relations R and S respectively). Both algorithms are subject to parallel execution. However, the hash-based algorithm has prevailed since it has linear execution cost, and it performs better in the presence of data skew as well [3].

The parallel hash-based join processing is separated into three phases. In the first phase, each relation is fully declustered horizontally across the database processors by applying a partition function on the declustering attribute, which in general is different from the join attribute. Next, at the redistribution phase, each database processor applies a common hash function h on the join attribute value for its local fragments of relations R and S. The hash function h ships any tuple belonging to either relation R or S with join attribute value b_i to the $h(b_i)$-th database processor. At the end of the redistribution process both relations are fully partitioned into disjoint fragments. Lastly, each database processor p performs locally with the most cost-effective way an equijoin operation between its fragments of relations R and S, denoted by R^p and S^p respectively. The joined tuples may be kept locally in each database processor instead of being merged with other output tuples into a single stream.

Skewness, perceived as the variance in the response times of the database processors involved in the previously described computation, is identified as one of the major factors that affects the effectiveness of the hash-based parallel join [7]. [9] defines four types of the data skew effect: Tuple placement skew, selectivity skew, redistribution skew and join product skew. Query load balancing in terms of the join operation is very sensitive to the existence of the redistribution skew and/or the join product skew. Redistribution skew can be observed after the end of the redistribution phase. It happens when at least one database processor has received large number of tuples belonging to a specific relation, say R, in comparison to the other processors after the completion of the redistribution phase. This imbalance in the number of redistributed tuples is due to the existence of naturally skewed values in the join attribute. Redistribution skew can be experienced in a subset of database processors. It may also concern both the relations R and S (double redistribution skew). Join product skew occurs when there is an imbalance in the number of join tuples produced by each database processor. [8] points the impact of this type of skewness to the response time of the join query. Especially, join product skew deteriorates the performance of subsequent join operation since this type of data skew is propagated into the query tree.

In this paper we address the issue of join product skew. Various techniques and algorithms have been proposed in the literature to handle this type of skew ([1], [4], [11], [2], [6], [12]). We introduce the notion of frequency classes, whose definition is based on the product of frequencies of the join attribute values. Under this perspective we examine the cases of homogeneous and heterogeneous input relations.

We also propose a new static algorithm, called HJPS (Handling Join Product Skew) to improve the performance of the parallel joins in the presence of this specific type of skewness. The algorithm is based on the intuition that join product skew comes into play when the produced tuples associated with a specific value overbalance the workload of a processor. HJPS algorithm constitutes a refinement of the PRPD algorithm [11] in the sense that the exact number of the needed processors is defined for each skewed value instead of duplicat-

ing or redistributing the tuples across all the database processors. Additionally, HJPS is advantageous in the case of having join product skew without having redistribution skew.

The rest of this paper is organized as follows. Section 2 discusses the related work. In section 3 we illustrate the notion of division of join attribute values into classes of frequencies by means of two generic cases. In section 4 an algorithm that helps in reducing join product skew effect is proposed and section 5 concludes the paper.

2   Related Work

The achievement of load balancing in the presence of redistribution and join product skew is related to the development of static and dynamic algorithms. In static algorithms it is assumed that adequate information on skewed data is known before the application of the algorithm. [1], [4] and [11] expose static algorithms. On the contrary, [2], [6] and [12] propose techniques and algorithms according to which data skew is detected and encountered dynamically at run time.

[2], [12] address the issue of the join product skew following a dynamic approach. A dynamic parallel join algorithm that employs a two-phase scheduling procedure is proposed in [12]. The authors of [2] present an hybrid frequency-adaptive algorithm which dynamically combines histogram-based balancing with standard hashing methods. The main idea is that the processing of each sub-relation, stored in a processor, depends on the join attribute value frequencies which are determined by its volume and the hashing distribution.

[1], [4] and [11] deal with the join product skew in a static manner. In [11], authors addresses the issue of the redistribution skew by proposing the PRPD algorithm. However, except for redistribution skew, their approach handles the join product skew that results from the former. In PRPD algorithm, the redistribution phase of the hash-join has been modified to some degree. Especially, for the equijoin operation $R_1 \bowtie R_2$, the tuples of each sub-relation of R_1 with skewed join attribute values occurring in R_1 are kept locally in the database processor. On the other hand, the tuples that have skewed values happening in R_2 are broadcast to all the database processor. The remaining tuples of sub-relation are hash redistributed. The tuples of each sub-relation of R_2 are treated in the respective way. The algorithm captures efficiently the case where some values are skewed in both relations. Using the notion of the splitting values stored in a split vector, virtual processor partitioning [4] assigns multiple range partitions instead of one to each processor. Finally, authors in [1] assign a work weight function to each join attribute value in order to generate partitions of nearly equal weight.

Finally, OJSO algorithm [10] handles data skew effect in an outer join, which is a variant of the equijoin operation.

3   Two Motivating Examples

We will assume the simple case of a binary join operation $R_1(A, B) \bowtie R_2(B, C)$, in which the join predicate is of the form $R_1.B = R_2.B$. The m discrete values $b_1, b_2, \ldots, b_m$ define the domain D of the join attribute B. Let $f_i(b_j)$ denote the relative frequency of join attribute value b_j in relation R_i. Given the relative frequencies of the join attribute values $b_1, b_2, \ldots, b_m$, the join selectivity of $R_1 \bowtie R_2$ is equal to [5]

$$\mu = \sum_{b_j \in D} \prod_{i=1}^{2} f_i(b_j) = \sum_{b_j \in D} f_1(b_j) f_2(b_j) \tag{1}$$

Since $\mu = \frac{|R_1 \bowtie R_2|}{|R_1 \times R_2|}$ and the size of the result set of the cross product $R_1 \times R_2$ is equal to the product $|R_1||R_2|$, the cardinality of the result set associated with the join operation $R_1 \bowtie R_2$ is determined by the magnitude of the join selectivity.

By extending the previous analysis, the join selectivity μ can be considered as the probability of the event that two randomly picked tuples, belonging to the relations R_1 and R_2 respectively, join on the same join attribute value. Based on this observation an analytical formula concerning the size of the result set of the chain join (which is one of the most common form of the join operation) is proven. Especially we state that the join selectivity of the chain join, denoted by $R = \bowtie_{i=1}^{k} R_i(A_{i-1}, A_i)$, is equal to the product of the selectivities $\mu_{i,i+1}$ of the constituent binary operation $R_i(A_{i-1}, A_i) \bowtie R_{i+1}(A_i, A_{i+1})$ under a certain condition of independence. In our notation, we omit to include attributes in the relations that do not participate in the join process. Formally, we have the following

Lemma *Given that the values of the join attributes A_i in a chain join of k relations are independent of each other, the overall join selectivity of the chain join, denoted by μ, is equal to the product of the selectivities of the constituent binary join operations, i.e., $\mu = \prod_{i=1}^{k-1} \mu_{i,i+1}$.*

Proof: We define a pair of random variables $(\mathsf{X}_i, \mathsf{Y}_i)$ for every relation R_i, where $i = 2, \ldots, k - 1$. Specifically, the random variable X_i corresponds to the join attribute $R_i.A_i$ and it is defined as the function $\mathsf{X}_i(t) : \Omega_i \to \mathbb{N}_{\mathsf{X}_i}$, where Ω_i is the set of the tuples in the relation R_i. $\mathbb{N}_{\mathsf{X}_i}$ stands for the set $\{0, 1, \ldots, |D_{A_i}| - 1\}$, where D_{A_i} is the domain of the join attribute A_i. In other words, $\mathbb{N}_{\mathsf{X}_i}$ defines an enumeration of the values of the join attribute A_i, in such a way that there is a one-to-one correspondence between the values of the set D_{A_i} and $\mathbb{N}_{\mathsf{X}_i}$. Similarly, the random variable $\mathsf{Y}_i(t) : \Omega_i \to \mathbb{N}_{\mathsf{Y}_i}$ corresponds to the join attribute A_{i+1}, where $\mathbb{N}_{\mathsf{Y}_i}$ represents the set $\{0, 1, \ldots, |D_{A_{i+1}}| - 1\}$.

As for the edge relations R_1 and R_k, only the random variables Y_1 and X_k are defined, since the attributes $R_1.A_0$ and $R_k.A_k$ do not participate in the join process.

Let $\mathcal{R}$ denote the event of the join process. Then we have that

$$p(\mathcal{R}) = p\big(\mathsf{Y}_1 = \mathsf{X}_2 \wedge \mathsf{Y}_2 = \mathsf{X}_3 \wedge \ldots \wedge \mathsf{Y}_{k-1} = \mathsf{X}_k\big)$$

By assumption, the random variables are independent of each other. Thus, it is valid to say that

$$p(\mathcal{R}) = \prod_{i=1}^{k-1} p(\mathsf{Y}_i = \mathsf{X}_{i+1})$$

Moreover, $p(\mathsf{Y}_i = \mathsf{X}_{i+1})$ represents the probability of the event that two randomly picked tuples from relations R_i and R_{i+1} agree on their values of the join attribute A_i. Since it holds that $p(\mathsf{Y}_i = \mathsf{X}_{i+1}) = \mu_{i,i+1}$, the lemma follows. $\qquad\square$

As a direct consequence of the previous lemma, the cardinality of the result set associated with the join operation $R = \bowtie_{i=1}^{k} R_i(A_{i-1}, A_i)$ is given by the formula

$$|R| = \Big(\prod_{i=1}^{k-1} \mu_{i,i+1}\Big) \cdot \Big(\prod_{j=1}^{k} |R_j|\Big)$$

3.1 Homogeneous Input Relations

Firstly, we examine the natural join of two homogeneous relations $R_1(A, B) \bowtie R_2(B, C)$ in the context of the join product skew effect. In the case of the homogeneous relations the distribution of the join attribute values b_i is the same for both input relations R_1 and R_2. That is, there exists a distribution f such that $f_1(b) = f_2(b) = f(b)$ for any $b \in D$. In this setting, the distribution f is skewed when there are join attribute values $b_i, b_j \in D$ such that $f(b_i) \gg f(b_j)$.

The join attribute values with the same relative frequency f_k defines the *frequency class* $C_k = \{b \in D \mid f(b) = f_k\}$.

Thus, the domain D of the join attribute B is disjointly separated into classes of different frequencies. This separation can be represented with a two level tree, called *frequency tree*. The nodes of the first level correspond to classes of different frequencies. The kth node of the first level is labeled with C_k. The descendant leaves of the labeled node C_k correspond to the join attributes belonging to class C_k. Each leaf is labeled with the value of one of the join attributes of the class corresponding to the parent node. The following picture depicts the structure of a simple frequency tree for join operation $R_1 \bowtie R_2$ assuming that $D = \{b_1, \ldots, b_6\}$ is separated into four frequency classes $C_1, \ldots, C_4$.
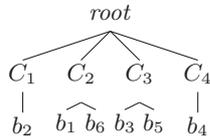
Fig. 1. The frequency tree for $R_1 \bowtie R_2$.

The number of produced joined tuples for a given class C_k is equal to $|C_k| f_k^2 |R_1||R_2|$ since $f_k|R_1|$ tuples of relation R_1 matches with $f_k|R_2|$ tuples of relation R_2 on any join attribute value $b \in C_k$. Let N be the number of the database processors participating in the computation of the join operation. Since only the join product skew effect is considered, the workload associated with each node is determined by the size of the partial result set that is computed locally. In order the workload of the join operation to be evenly apportioned on the N database processors, each node should produce approximately $\left(\frac{\sum_{k=1}^{K} |C_k| f_k^2}{N} \right) |R_1||R_2|$ number of joined tuples, where K denotes the number of frequency classes. In terms of the frequency classes, this is equivalent to an appropriate assignment of either entire or subset of frequency class(es) to each database processors in order to achieve the nearly even distribution of the workload. This assignment can be represented by the selection of some internal nodes and leaves in the frequency tree. By construction, the selection of an internal node in the frequency tree amounts to the exclusive assignment of the corresponding frequency class to some database processor. Thus, this database processor will join tuples from the relations R_1 and R_2 whose join attribute value belongs to the selected class. Finally, to guarantee the integrity of the final result set, the sequence of selections must span all the leaves of the frequency tree.

3.2   Heterogeneous Input Relations

We extend the previous analysis in the case of heterogenous input relations. The join attribute values are distributed to the input relations $R_1(A, B)$ and $R_2(B, C)$ according to the data distributions f_1 and f_2, respectively. In general, it holds that the relative frequencies of any join attribute value $b \in D$ are different in the relations R_1 and R_2, i.e., $f_1(b) \neq f_2(b)$ for any $b \in D$. The above are depicted in table 1.

The number of joined tuples corresponding to the join attribute value $b \in D$ is rendered by the product $f_1(b) f_2(b)$. Thus, the join product skew happens when $f_1(b_i) f_2(b_i) \gg f_2(b_j) f_2(b_j)$ for some $b_i, b_j \in D$. This means that the workload of the join process for the database processor, to which the tuples with join attribute value equal to b_i have been shipped at the redistribution phase, will be disproportional compared with the respective workload of another database processor. Similarly to section 3.1, the classes $C_k = \{b \in D \mid f_1(b) f_2(b) = f_k\}$ disjointly partition the join attribute values.

Alternatively, it is possible the definition of classes of ranges of frequencies according to the schema $C_k = \{b \in D \mid f_{k-1} \leq f_1(x) f_2(x) < f_k\}$ (range partitioning in the frequency level).

The "primary-key-to-foreign-key" join consists a special case of heterogeneity where in one of the two relation, say R_1, two different tuples always have different values in the attribute B. This attribute is called primary key and its each value $b \in D$ uniquely identifies a tuple in relation R_1. As to relation R_2, attribute B, called foreign key, matches the primary key of the referenced relation R_1. In this setting, which is very common in practice, we have that $f_1(b_i) = \frac{1}{m}$ for

any $b_i \in D$, and in general $f_2(b_i) \neq \frac{1}{m}$ with $f_2(b_i) > 0$. The join product skew happens when $f_2(b_i) \gg f_2(b_j)$ for some $b_i, b_j \in D$, since $f_1(b_i) = f_1(b_j)$. Thus, the separation of the join attribute values into disjoint frequency classes can be defined with respect to the data distribution f_2, i.e., $C_k = \{x \in D \mid f_2(x) = f_k\}$.

Join Attribute Values	R_1	R_2
b_1	$f_1(b_1)$	$f_2(b_1)$
$\ldots$	$\ldots$	$\ldots$
b_m	$f_1(b_m)$	$f_2(b_m)$

Table 1. Relative frequencies of the join attribute values.

4   Algorithm HJPS

In this section, we propose an algorithm, called HJPS, that alleviates the join product skew effect. The algorithm deals with the case of the binary join operation $R(A, B) \bowtie S(B, C)$ in which the join predicate is $R.B = S.B$.

Let $D = \{b_1, b_2, ..., b_m\}$ be the domain of values associated with the join attribute B. We denote by $|R_{b_i}|$ $(|S_{b_i}|)$ the number of tuples of the relation R (respectively S) with join attribute value equal to b_i, where $b_i \in D$. The algorithm considers that the quantities $|R_{b_i}|$, $|S_{b_i}|$ for every $b_i \in D$ are known in advance by either previously collected or sampled statistics. We also denote by n the number of the database processors. In our setting, all the database processors are supposed to have identical configuration.

As it has been mentioned earlier, the number of the needed computations for the evaluation of the join operation, that identifies the total processing cost (TPC), is determined by the sum of products of the number of tuples in both relations that have the same join attribute values. This means that TPC is expressed by the equation

$$TPC = \sum_{b_i \in D} |R_{b_i}| * |S_{b_i}|$$

In the context of the parallel execution of the join operator, the ideal workload assigned to each processor, denoted by pwl, is defined as the approximate number of the joined tuples that it should produce in order not to experience the join product skew effect. Obviously, it holds that that $pwl = TPC/n$.

HJPS determines whether or not a join attribute value $b_i \in D$ is skewed by the number of the processors dedicated to the production of the joined tuples

corresponding to this value. To be more specific, the quotient of the division of the number of joined tuples associated with the join attribute value b_i (which is equal to $|R_{b_i}| * |S_{b_i}|$) by pwl gives the number of the processors needed to handle this attribute value. In the case that the result of the division, denoted by vwl_{b_i}, exceeds the value of two, the algorithm considers the join attribute value as skewed. The latter is inserted into a set of values, denoted by SK.

Let $SK = \{b_{a_1}, b_{a_2}, b_{a_3}, ..., b_{a_l}\}$ be the set of the skewed values. The algorithm iterates over the set SK. In particular, for the value b_{a_1}, suppose that the number of the needed processors is equal to $vwl_{b_{a_1}}$. The algorithm takes a decision based on the number of tuples with join attribute value b_{a_1} in relations R and S. If $|R_{b_{a_1}}| > |S_{b_{a_1}}|$, the tuples of the relation R are redistributed to the first $vwl_{b_{a_1}}$ processors while all the tuples from the second relation are duplicated to all of the $vwl_{b_{a_1}}$ processors. In order to decide which of the $vwl_{b_{a_1}}$ processors is going to receive a tuple of the relation R with join attribute value b_{a_1}, the algorithm applies a hash function on a set of attributes. On the contrary, if it holds that $|R_{b_{a_1}}| < |S_{b_{a_1}}|$, all the tuples from the relation R with join attribute value equal to b_{a_1} are duplicated to all of the $vwl_{b_{a_1}}$ processors while the tuples of the relation S are distributed to all of the $vwl_{b_{a_1}}$ processors according to a hash function. The same procedure takes place for the rest skewed values. The remaining tuples are redistributed to the rest processors according to a hash function on the join attribute. A Pseudocode of the algorithm is given below.

5   Conclusion and Future Work

We address the problem of join product skew in the context of the PDBMS. In our analysis, the apriori knowledge of the distribution of the join attribute values has been taken for granted. We concentrated on the case of partitioned parallelism, according to which the join operator to be parallelized is split into many independent operators each working on a part of data. We introduced the notion of frequency classes and we examined its application in the general cases of homogeneous and heterogeneous input relations. Furthermore, an heuristic algorithmic called HJPS is proposed to handle join product skew. The proposed algorithm identifies the skew elements and assigns a specific number of processors to each of them. Given a skewed join attribute value, the number of dedicated processors is determined by the process cost for computing the join for this attribute value, and by the workload that a processor can afford.

We are looking at generalizing our analysis with frequency classes at multiway joins. In this direction we have proven the lemma of section 3 which is about the chain join of k relations. Furthermore, other types of multiway join operations, e.g., star join, cyclic join, are going to be studied in the perspective of the data skew effect and under the context of frequency classes. Finally, in a future work we will examine the case of multiway joins supposing that no statistical information about the distribution of the join attribute values is given in advance.

Algorithm HJPS (* Handling Join Product Skew *)

Input: t_{r_i} tuples of relations R and t_{r_j} tuples of relations S, N number of processors.

Output: correspondence of tuple to processor

Consider the join attribute value is the set:

$D = \{b_1, b_2, ..., b_m\}$

(* compute all frequencies for every join attribute value inD *)

for $j := b_1$ **to** b_m **do**

    calculate the frequencies f_{R_j}, f_{S_j};

$TPC = \sum_{b_i \in D} |R_{b_i}| * |S_{b_i}|$ (*TCP the total process cost*)

$pwl = TPC/N$

(*pwl the process cost of each processor*)

$vwl_{b_i} = |R_{b_i}| * |S_{b_i}|$;

(*vwl$_{b_i}$ the process cost for each join attribute value b_i)

$pn_{b_i} = vwl_{b_i}/pwl$;

(*pn$_{b_i}$ ideal number of processors for the join attribute value b_i*)

if $(pn_{b_i} >= 2)$ consider b_i a skewed value;

Let $SK = \{b_{a_1}, b_{a_2}, b_{a_3}, ..., b_{a_l}\}$ be the set of skewed values

for $i := a_1$ **to** a_l **do**

    if $|R_{b_{a_1}}| > |S_{b_{a_1}}|$

        distribute every t_{r_i} to the next vn_{b_i} processors;

(*for distribution use a hash function to a set of attributes*)

        send every t_{s_i} to the next vn_{b_i} processors;

    else

        distribute every t_{s_i} to the next vn_{b_i} processors;

        send every t_{r_i} to the next vn_{b_i} processors;

assign rest tuples from both relations to the rest processors;

(*for assignment HJPS applies a hash function to the join attribute *)

References

1. K. Alsabti and S. Ranka. Skew-insensitive parallel algorithms for relational join. In *HIPC '98: Proceedings of the Fifth International Conference on High Performance Computing*, page 367, Washington, DC, USA, 1998. IEEE Computer Society.
2. Mostafa Bamha and Gaétan Hains. Frequency-adaptive join for shared nothing machines. pages 227–241, 2001.
3. David J. DeWitt and Jim Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, 35(6):85–98, 1992.
4. David J. DeWitt, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *18th International Conference on VLDB, Vancouver, Canada, Proceedings*, pages 27–40. Morgan Kaufmann, 1992.
5. Peter J. Haas, Jeffrey F. Naughton, and Arun N. Swami. On the relative cost of sampling for join selectivity estimation. In *PODS '94: Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 14–24, New York, NY, USA, 1994. ACM.
6. Lilian Harada and Masaru Kitsuregawa. Dynamic join product skew handling for hash-joins in shared-nothing database systems. In *Database Systems for Advanced*

Applications '95, Proceedings of the 4th International Conference on DASFAA, Singapore, 1995, volume 5 of *Advanced Database Research and Development Series*, pages 246–255.

7. M. Seetha Lakshmi and Philip S. Yu. Effectiveness of parallel joins. *IEEE Trans. Knowl. Data Eng.*, 2(4):410–424, 1990.
8. Manish Mehta and David J. DeWitt. Data placement in shared-nothing parallel database systems. *VLDB J.*, 6(1):53–72, 1997.
9. Christopher B. Walton, Alfred G. Dale, and Roy M. Jenevein. A taxonomy and performance model of data skew effects in parallel joins. In *17th International Conference on VLDB, 1991, Barcelona, Catalonia, Spain, Proceedings*, pages 537–548. Morgan Kaufmann, 1991.
10. Yu Xu and Pekka Kostamaa. Efficient outer join data skew handling in parallel dbms. *PVLDB*, 2(2):1390–1396, 2009.
11. Yu Xu, Pekka Kostamaa, Xin Zhou, and Liang Chen. Handling data skew in parallel joins in shared-nothing systems. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1043–1052, New York, NY, USA, 2008. ACM.
12. Z. Xiaofang and M.E. Orlowska. Handling data skew in parallel hash join computation using two-phase scheduling. In *Algorithms and Architectures for Parallel Processing*, pages 527 – 536. IEEE Computer Society, 1995.
13. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.