Using Statistics for Computing Joins with MapReduce

Theresa Csar[1], Reinhard Pichler[1], Emanuel Sallinger[1], and Vadim Savenkov[2]

[1] Vienna University of Technology
{csar, pichler, sallinger}@dbai.tuwien.ac.at
[2] Vienna University of Economy and Business (WU)
vadim.savenkov@wu.ac.at

1 Introduction

The MapReduce model has been designed to cope with ever-growing amounts of data [4]. It has been successfully applied to various computational problems. In recent years, multiple MapReduce algorithms have also been developed for computing joins – one of the fundamental problems in managing and querying data.

The main optimization goals of these algorithms for distributing the computation tasks to the available reducers are the replication rate and the maximum load of the reducers. The HyperCube algorithm of Afrati and Ullman [1] minimizes the former by considering only the size of the involved tables. This algorithm was later enhanced by Beame et al. [3] to minimize the latter by taking into account also so-called "heavy hitters" (i.e., attribute values that occur particularly often). However, in contrast to most state-of-the-art database management systems, more elaborate statistics on the distribution of data values have not been used for optimization purposes so far.

Recently, several approaches for handling skew in the computation of joins have been proposed, improving the partitioning of the data using histograms or varying a cost model [6, 7], but there is still ample room for enhancements and optimization. In [5] a survey of recent approaches for dealing with the weaknesses and limitations of the MapReduce model can be found.

The goal of this paper is to study the potential benefit of using more fine-grained statistics on the distribution of data values in MapReduce algorithms for join computation. To this end, we investigate the performance of known algorithms [1, 3] in the presence of skewed data, and extend them by utilizing data statistics. We compare the original algorithms with a modified one that makes use of additional statistical measures. Our initial study shows that our approach can indeed improve existing methods.

2 Preliminaries

A MapReduce join computation consists of three basic phases. First, in the *Map-Phase*, a key-value is assigned to every tuple. In the *Shuffle-Phase*, the tuples are distributed among the reduce tasks (also called *reducers*) according to their key-values. In the final *Reduce-Phase*, each reducer performs the join on all its tuples. The theoretical foundations of MapReduce query processing have been laid among others by [1–3], based on the HyperCube algorithm outlined below.

The HyperCube Algorithm. Key-values for tuples are formed by concatenating the hashes of the join attributes. Consider the triangle join $R(A, B) \bowtie S(B, C) \bowtie T(C, A)$ in which all attributes A, B and C are join attributes. Key-values are triples (a_i, b_i, c_i) obtained by the respective hash functions h_a, h_b and h_c. A tuple is sent to all reducers that may have join candidates for it. For instance, the tuple $R(a_1, b_1)$ is sent to the reducers identified by keys of the form $(h_a(a_1), h_b(b_1), *)$ where $*$ matches any value in the range of h_c. We take $[1, a], [1, b]$ and $[1, c]$ to be the ranges of the respective hash functions h_a, h_b and h_c. The size the range is called the *share* of the attribute. The respective shares are thus a, b and c, and the total number of reducers equals the product of the shares: $k = abc$.

An important measure for the performance of the HyperCube algorithm is the replication rate. For instance, each R-tuple $R(a_i, b_i)$ is replicated c times, since it is sent to the reducers responsible for the keys $(h(a_i), h(b_i), 1), \ldots, (h(a_i), h(b_i), c)$. The replication rate for the triangle join is $rc + sa + tb$, where r, s and t are the sizes of the tables. In [1], shares are chosen in order to minimize the replication rate. The solution for the shares for the triangle query in the model of [1] is $a = \sqrt[3]{\frac{krt}{s^2}}$, $b = \sqrt[3]{\frac{krs}{t^2}}$ and $c = \sqrt[3]{\frac{kst}{r^2}}$. For the four-atom chain query $R(A, B) \bowtie S(B, C) \bowtie T(C, D) \bowtie U(D, E)$, the solutions for the shares are $b = d\sqrt{\frac{rs}{tu}}$, $c = \sqrt{\frac{st}{ru}}$ and $d = \sqrt{\frac{ku}{s}}$.

In [3], the shares are chosen to minimize the maximum load per reducer, that is, the maximum number of tuples sent to a single reducer. The shares are calculated as the solution to a linear program. In contrast to [1], the method in [3] also addresses the problem of skew by treating heavy hitters separately. Also the expected load and maximum load per server is analyzed in [3], and a lower bound for the maximum load per server is given.

3 An Empirical Study and the Need for Statistics

The goal of our study is to compare the performance of HyperCube-based algorithms. To this end, we investigate how the shares chosen by such methods influence the workload distributions among the reduce tasks, in particular the maximum load. The analysis was performed on two well-studied types of queries, namely the triangle query $(R(A, B) \bowtie S(B, C) \bowtie T(C, A))$ and the chain query of length four $(R(A, B) \bowtie S(B, C) \bowtie T(C, D) \bowtie U(D, E))$. In both cases, there are three join attributes.

Methods. Apart from known methods for computing shares, namely [1] (which we shall call `AU`) and [3] (which we shall call `BKS`), we next introduce baseline methods as well as weighted variants of `AU` that take into account additional statistics. To facilitate a fair comparison, the shares produced by each method are normalized in the following way: they are rounded to integer values in such a way that the product of the shares is as close as possible to the fixed number of reduce tasks k. Shares have to be at least 1 and at most k. For the naive method, we define shares `naive` $= (\sqrt[3]{k}, \sqrt[3]{k}, \sqrt[3]{k})$. The worst-case we identified, `worst` $= (k, 1, 1)$, will be omitted from charts to keep the differences between other methods visible. The nearly-worst-case methods we consider are defined as `share1` $= (2\sqrt[3]{k}, \frac{1}{2}\sqrt[3]{k}, \sqrt[3]{k})$ and `share2` $= (\sqrt{k}, \sqrt{k}, 1)$, respectively.

Weigthed `AU`. The shares computed using `AU` (or `BKS`) depend only on the sizes of the tables, but not on other statistics indicating, e.g., the degree of skew. A simple way to detect a distribution with high variability is using the *standard deviation*. The more elaborate *gini-coefficient* is a measure for the variability of a distribution. For an observation X with possible values $x_1, \ldots, x_n$ and relative frequencies $p_1, \ldots, p_n$, the gini-coefficient is $\sum_{i=1}^{n} p_i^2$. A gini-coefficient close to 1 means that the values are very unevenly distributed.

We define our method `SD` as a variant of `AU` with the following modification: Assume that a table T has size t and attributes A and B. Instead of t, we give to `AU` the weighted value $t \cdot sd(T.A) \cdot sd(T.B)$, where sd denotes the standard deviation of the attribute values. The `Gini` method is defined analogously. Finally, the variant `SD2` of `SD` is defined by normalizing the standard deviation relative to the maximum attribute value, i.e., $sd2(T.A) := sd(T.A) \,/\, max(T.A)$. Note that standard deviation is only defined for numeric values (and our test scenarios use only numeric values). We leave the study of similar variations of `BKS` for future work.

Test Methodology. The experimental study was implemented using the programming language `R` (`http://cran.r-project.org/`). The goal of our study is to compute the work loads of all reducers, and derive in particular the maximum load and various other statistics based on the loads. Thus, we only implement the *Map-Phase* of the MapReduce process. To this end we compute the loads of the reducers and our presented statistics.

As databases for our test scenarios, we use randomly generated data sets where attributes are generated according to a variety of different distributions. For each such database, all methods (`AU`, `BKS`, ...) are applied 1000 times to the input tables to compute the shares. In each round, other (randomly generated) hash functions are used. Performing 1000 repetitions is done to be able to isolate the effect of the method (in particular, the chosen shares) from the effect of the exact hash function that is used.

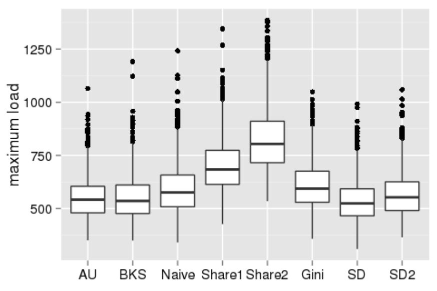

Fig. 1: Triangle query – maximum loads

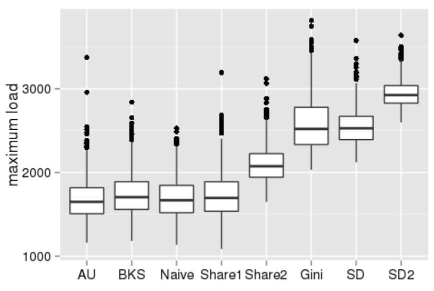

Fig. 2: Chain query – maximum loads

Triangle Query. For the triangle query, we first look at a sample database generated using the methodology described above. The number of reduce tasks used is 150. The resulting maximum load at the reduce tasks can be seen in Fig. 1, where it can be observed that all reasonable methods (i.e., all methods besides the nearly-worst-case ones) do not show any significant difference in the performance based on the the maximum loads. When observing the variance and the gini-coefficient of the loads, a similar picture arises. This is surprising, since the assigned shares differ a lot (see Table 1a). As expected, `AU` yields the lowest replication rate.

method	a	b	c	replication rate
AU	3	6	8	4.72
BKS	3	6	8	4.72
Naive	5	6	5	4.91
Gini	6	5	5	5.55
SD	3	5	10	4.82
SD2	2	6	11	4.73
share1	10	3	5	7.18
share2	8	9	2	7.18
worst	150	1	1	82.3

(a) First triangle query

method	b	c	d	replication rate
AU	8	3	6	13.3
BKS	11	2	7	13.5
Naive	5	6	5	14.5
Gini	48	1	3	25.9
SD	1	50	3	33.0
SD2	7	21	1	41.6
share1	10	3	5	14.4
share2	8	9	2	23.3
worst	150	1	1	75.6

(b) Chain query

Table 1: Shares and replication rates for the triangle and chain queries.

Triangle Query for Highly Skewed Data. For highly skewed data, we show that the `AU` and `BKS` methods do not always yield optimal maximum load. Indeed, the maximum load produced by `AU` and `BKS` exceeds the value obtained with the `SD` by more than 30%. We illustrate this by an example.

We consider the database instance D given in Fig. 3 and let the maximum number of reducers be 64. Table R contains 1040 distinct tuples (a_i, b_i) and the tables S and T contain groups of 16 $c_{i,j}$ values associated to the same a_i or b_j value, which sums up to 1040 values per table as well.

In total, few R tuples take part in triangles, but those that take part have 16 $c_{i,j}$ values that form triangles with them. Such a situation would be typical in a company where, say, few employees are department heads, but those who are department heads have a number of employees they are responsible for.

We calculated the shares, the resulting replication rate and maximum load for the discussed methods. Both `AU` and `BKS` yield shares $(4, 4, 4)$. Applying the pigeonhole principle, one can show that this leads to the load of $65 + 16 + 16 = 97$ tuples for *most* reducers as a lower bound. A much better maximum load (66 tuples for one reducer and 33 for the rest) could have been obtained using shares $(8, 8, 1)$. The suboptimal result of `AU` and `BKS` is due to taking only table sizes into account, whereas the `SD` method yields the solution $(8, 8, 1)$. An important observation here is that the actual performance of each method depends heavily on the concrete hash function and that "usual" hash functions based on integer division may by far miss the optimum.

R	
A	**B**
a_1	b_1
$\vdots$	$\vdots$
a_{1040}	b_{1040}

S	
A	**C**
a_1	$c_{1,1}$
a_1	$c_{1,2}$
$\vdots$	$\vdots$
a_1	$c_{1,16}$
$\vdots$	$\vdots$
a_{65}	$c_{65,1}$
a_{65}	$c_{65,2}$
$\vdots$	$\vdots$
a_{65}	$c_{65,16}$

T	
C	**B**
$c_{1,1}$	b_1
$c_{1,2}$	b_1
$\vdots$	$\vdots$
$c_{1,16}$	b_1
$\vdots$	$\vdots$
$c_{65,1}$	b_{65}
$c_{65,2}$	b_{65}
$\vdots$	$\vdots$
$c_{65,16}$	b_{65}

Fig. 3: Database instance D.

Chain Query. For the chain query, a random database is constructed according to the methodology outlined earlier. Again, our tests are performed for 150 reduce tasks. Interestingly, the resulting maximum loads are much higher for `share2`, `Gini`, `SD`, and `SD2` than for the other methods (see Fig. 2). The high maximum load in case of `Gini`, `SD`, and `SD2` suggests that some fine-tuning of the weights caused by the data statistics is needed. On the positive side, it turns out that the loads resulting from the `Gini`, `SD`, and `SD2` methods are distributed more evenly among the reducers than with `AU` and `BKS`, as can be seen in Fig. 5. The high variability in the median (Fig. 4), especially for the `Gini` method, again underlines that the choice of the hash function is crucial.

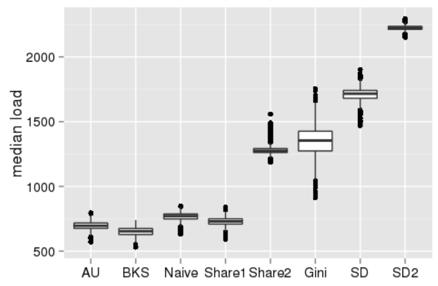

Fig. 4: Chain query – median of loads

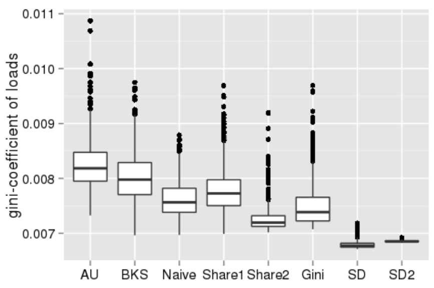

Fig. 5: Chain query – gini of loads

4 Conclusion

We have initiated the comparative study of methods for computing joins using MapReduce. We have seen that current methods perform relatively well compared to baseline and adapted methods. However, we have also seen that data-dependent statistics provide much potential for further improvement of these algorithms, which needs to be further explored. In particular, if we aim at a uniform distribution of computation tasks among the available reducers, taking into account additional statistical measures such as standard deviation or gini coefficient seems inevitable. Another important lesson learned from our investigation is the importance and difficulty of choosing an optimal hash function: even if the shares are – in theory – "optimal" for a certain criterion (such as maximum load), it is highly non-trivial to actually attain this optimum by choosing the "right" hash function. Current MapReduce research thus also has to be extended towards optimizing the hash function. Beyond that, we want to investigate the tradeoff between the cost of computing statistics and the gain provided by these statistics.

Acknowledgements. This work was supported by the Austrian Science Fund projects (FWF):P25207-N23 and (FWF):Y698.

References

1. Afrati, F.N., Ullman, J.D.: Optimizing multiway joins in a map-reduce environment. IEEE Trans. Knowl. Data Eng. 23(9), 1282–1298 (2011)
2. Beame, P., Koutris, P., Suciu, D.: Communication steps for parallel query processing. In: Proc. PODS 2013. pp. 273–284. ACM (2013)
3. Beame, P., Koutris, P., Suciu, D.: Skew in parallel query processing. In: Proc. PODS 2014. pp. 212–223. ACM (2014)
4. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
5. Doulkeridis, C., Nørvåg, K.: A survey of large-scale analytical query processing in mapreduce. The VLDB Journal 23(3), 355–380 (2014)
6. Gufler, B., Augsten, N., Reiser, A., Kemper, A.: Load balancing in mapreduce based on scalable cardinality estimates. In: Proc. ICDE 2012. pp. 522–533 (2012)
7. Okcan, A., Riedewald, M.: Processing theta-joins using mapreduce. In: Proc. SIGMOD 2011. pp. 949–960. ACM (2011)