Towards minimal algorithms for big data analytics with spreadsheets

Jacek Sroka, Artur Leśniewski, Mirosław Kowaluk, Krzysztof Stencel, Jerzy Tyszkiewicz
Institute of Informatics, University of Warsaw,
Banacha 2, 02097 Warszawa, Poland
{j.sroka,a.lesniewski,m.kowaluk,k.stencel,j.tyszkiewicz}@mimuw.edu.pl

ABSTRACT

The motivation for our research is the need for a simple and accessible method to process large datasets available to the public. We aim to significantly lower the technological barrier, which currently prevents average users from analyzing big datasets available as CSV files. Spreadsheets are perfectly suited for this task, being the most popular and accessible data analysis tool. We present ideally balanced algorithms for specifying MapReduce computations of high number of range queries. With our algorithms, it will be possible to automatically perform analysis defined with a spreadsheet on data of size exceeding the dimensions of the spreadsheet grid by orders of magnitude.

Keywords

Big data, spreadsheet, MapReduce, Hadoop, minimal algorithms, range queries

1. INTRODUCTION

We describe an algorithmic problem with strong practical motivation and design an ideally balanced algorithms for MapReduce for it. The motivation is the need for a simple and accessible method to process large datasets available to the public. It is a new practice that a growing number of large datasets are made available for transparency reasons and to enable new applications for the data by scientists, industry and community. [3] presents a survey of 9000 open data sets from 20 cities from North America. Although those large datasets are already available and new ones are added regularly, the authors note "there is a great need for scalable and automatic techniques to process and integrate the data". At the time of that publication the size of the studied datasets was tens of gigabytes and was reported to quickly increase.

Handling of the data of such a volume is not a problem for data scientists or companies hiring professional programmers, but it is beyond the processing capabilities of majority of people who do not have advanced computer science skills. Professionals usually analyze data of moderate size with spreadsheet extensions like PowerPivot or languages like R and Python. When the data is too big for a single computer yet the problem is "embarrassingly parallel" HPC Services for Excel can be used. More general computation may be distributed to a cluster of computers with frameworks like Apache Hadoop, Spark, or Apache Pig. Effective programming in those frameworks requires expert knowledge. This effectively limits the accessibility of the published datasets to selected professionals and those who can afford hiring them.

Our ultimate goal is to create a tool, which translates spreadsheets of regular structure into semantically equivalent MapReduce computations, so that the user may create a spreadsheet on a sample of the data and automatically translate it into MapReduce and execute on a dataset of size exceeding what Excel or other spreadsheets can process. Such a tool would be based on vanilla spreadsheet formulas not requiring the user to learn any extensions nor being limited to only a chosen paid spreadsheet tool. It is estimated that spreadsheet programmers outnumber the programmers of all other languages together [4]. Therefore achieving our goal would empower a huge number of new users to use and benefit from the power of analyzing Big data.

Our present contribution towards this goal is an algorithm to answer bulk range queries. It follows the idea of a range tree construction. However, it works on data that does not fit into memory of a single machine. Furthermore, it allows answering queries in bulk, i.e., there can be comparably as many queries as data points. This algorithm facilitates distributing the computation of the most expressive and interesting of popular spreadsheet functions, namely `COUNTIFS`. This function is used to count rows in the data, which satisfy constraints expressed by comparisons to specified values. It subsumes numerous other spreadsheet functions and allows expressing frequently occurring, practical and non-trivial computations. We demonstrate examples in Section 2.

The algorithm we present is perfectly balanced and does not suffer from skew. Our techniques can be adapted to other highly expressive spreadsheet functions like `SUMIFS`.

2. USE CASE

2.1 Particular example

Let us assume that someone wants to analyze the New York City Taxi Trip dataset [1] (NYCTT dataset). As is typically the case this dataset is available in the form of multiple compressed CSV files, each of (uncompressed) size between 1 GB and 2 GB, which covers taxi trips from one month.

As the use case for this paper we assume that the user wants to start with *data cleaning*, which is an important preliminary step in any serious data analysis. We are looking for a particular kind of anomalies, related to timestamps: pick-ups and drop-offs of passengers in the middle of another trip of the same cab. This task

BeyondMR'17 May 19, 2017, Chicago, IL, USA
© 2017 ACM. ISBN 978-1-4503-5019-8/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3070607.3073005

needs only three fields of the dataset records: `medallion`, which is the cab's id, and pickup timestamp and drop-off timestamp.

We target an average user who might attempt to use spreadsheet for this task. Due to the size of the files, he/she can only process the records from the first three days of May 2013. The formulas shown below are taken from row i in the spreadsheet, assuming the other rows are filled with their copies with the references adjusted automatically by the spreadsheet tool.

Columns `A` and `B` and `C` contain the input data. Column `D` uses the formula:
`=Ci<Bi`
to detect if the present trip ends before it starts. Columns `E` and `F` contain two very similar formulas
`=COUNTIFS(A:A,Ai,D:D,FALSE,B:B,">"&Bi,B:B,"<"&Ci)`
`=COUNTIFS(A:A,Ai,D:D,FALSE,C:C,">"&Bi,C:C,"<"&Ci)`
which produce the counts of other trips by cab with the same medallion as the present one, passing the test in column `D` and whose pickup (drop-off, resp.) time is strictly between the pick-up time and drop-off time of the present trip. These are the formulas which perform the data cleaning, and a non-zero result indicates an anomaly. The test discovers such anomalies: e.g., rows 947994 and 951814 of May 2013 dataset contain two overlapping rides of taxi cab 2013000062 on May 2nd.

This is a reasonable solution for a person who does not have advanced programming experience and skills. And, indeed, this spreadsheet solves the data cleaning problem. However, it is limited by the spreadsheet size limitation, and is extremely slow (see Section 4).

The algorithm presented in this paper is an important step towards automatic translation of such spreadsheets into a MapReduce program to process the full dataset on a large cluster of computers. The spreadsheet contains a huge number of mutually independent, very similar formulas, which should be evaluated in parallel. Our MapReduce implementation of `COUNTIFS` achieves this using a single, distributed data structure shared by evaluations of all formulas, to make it efficient.

2.2 Spreadsheet translations

In general, we postulate that spreadsheets can be used by end users as a tool to specify data processing algorithms. Then such spreadsheets should be automatically translated into MapReduce programs representing the same data transformation and executed in Hadoop or a similar framework, increasing the users' data processing abilities by orders of magnitude. In order to achieve this goal it is necessary to be able to translate the most frequent spreadsheet functions. `COUNTIFS` and its close relative `SUMIFS` are known to be quite frequent in two prominent spreadsheet corpora: EUSES and ENRON. In both corpora these functions are among 15 most frequently used ones [2]. Moreover, they are definitely among the most expressive ones on the list, in addition to lookup functions. However, our algorithm is also crucial in translating simpler functions on the list, such us `SUM` (which is the most frequent one in both corpora), `SUBTOTAL` and `AVERAGE`. Consider such a formula whose argument is a range with relative addressing, e.g. `SUM(A$1:A1)`. If this formula is copied down the spreadsheet, the formula in row i will become `SUM(A$1:Ai)`. Now it is indeed the sum over an interval, whose ends are expressed in terms of the row number. This results in a similar algorithmic problem as for `COUNTIFS`.

We need parallel algorithms to evaluate whole columns of formulas, resulting from copying a single, initial formula from the first row. We assume that all columns used as arguments of a function have been materialized, and now we must evaluate the whole subsequent column of formulas, with `COUNTIFS` computation in each row.

3. ALGORITHMS

While distributing computations in MapReduce one needs to take into account numerous concerns. Firstly, the machines in the cluster need to be able to handle the work assigned to them without running out of memory. Secondly, as the data transfer is slow, the algorithm needs to limit the amount of data transferred over the network during the shuffling phase and between multiple Map-Reduce iterations. Thirdly, the algorithm must guarantee that all the nodes finish at approximately the same time. If it is not the case, we say that there is *skew*. The article [5] discusses methods for constructing *minimal* Map-Reduce algorithms with guarantees for optimal load balancing of the: amount of memory used by each node, amount of data transferred from and to each node, and computation done by each node. The first of those guarantees also protects workers from running out of memory. The algorithms we develop are minimal in the sense of [5]. We find this crucial as spreadsheet users are not likely to remedy the skew by themselves, or even to diagnose why their computation is slow.

We introduce our algorithms on examples. The presentation proceeds incrementally. In Section 3.1 we start with the simplest variants of the operator and discuss an idea for naïve algorithm and its limitations. Then, in Sections 3.2 and 3.3, we proceed with a idea of more advanced IMP algorithms that is not susceptible to skew, and at the same time cover more advanced syntax of the operator. Finally, in Section 3.3 we present our MIMP algorithm for even more general case.

3.1 The naïve `COUNTIFS` algorithm

In the simplest version we assume the formula `=COUNTIFS(A:A,Bi)` is to be placed in the cell `Ci` for $i = 1, 2, 3, \ldots,$ using the spreadsheet fill mechanism. In each row i the formula counts the number of rows in the range `A:A`, i.e., the first column, that are equal to the value in cell `Bi`. The naïve approach to implement this computation in MapReduce is to partition the values in column `A`, emit them with some marker, and at the same time partition the values in column `B` and emit them with their row numbers. Then in each group the marked values are counted, producing the result for the matching row.

Although simple, this solution is susceptible to skewed inputs, e.g., if the considered values represent the days of the week, then some days will probably appear significantly more often than others, potentially causing some nodes to run out of memory. Furthermore, only seven reducers in total will be used, limiting the amount of parallelism.

A quick fix for the simplest case is to use combiners, i.e., to count the number of all `A` values in map processes, and from each map process emit to each reducer only a single value — the aggregated `A` count. This takes care of the data skew in column `A` and limits the amount of data sent over the network. However, the data skew in column `B` is not addressed in this way.

3.2 Interval Multiquery Processor

`COUNTIFS` can also be used with inequalities and multiple conditions, where the naïve partitioning by value is not sufficient. For example, the formula `=COUNTIFS(B:B,">"&Bi,B:B,"<"&Ci, B:B,">="&Di)` counts for each row i the number of rows where at the same time the value in column `B` is larger than value in the cell `Bi`, the value in column `B` is smaller than the value in cell `Ci`, and the value in column `B` is larger or equal than the value in cell `Di`. We start with the algorithm for the case where all conditions refer to the same column, i.e., B in this case. Multiple conditions referring to different columns are covered in Subsection 3.3.

The algorithm we present here, called Interval Multiquery Processor (IMP), is perfectly balanced and transfers only a linear amount of data. In particular, our solution guarantees that the total work of all nodes is the same as the work done by an optimal centralized algorithm, and that the work is divided evenly. Each node sends and receives a comparable share of the data in each MapReduce round. In fact our algorithm is *minimal* in the sense of [5] which guarantees that the computation is balanced and free of skew.

It is easy to see that for each row i, the conditions define an interval, which can be open or closed on any end, or unbounded on one end. In a degenerate case the interval contains exactly one point or is even empty. Thus for a row with conditions representing a non-empty interval we need to consider at most two values: the right-hand-side value (with condition being either greater or greater-or-equal) and the left-hand-side value (with condition either smaller or smaller-or-equal). If one of these values is missing, then the interval is unbounded on that side. The distributed solution mimics a centralized one, where we sort the input based on the values in column B. Then for each row we find the positions of interval ends. Such an algorithm is clearly linearithmic.

Sorting in MapReduce can be implemented efficiently by redefining the partitioning function so that some reducer gets all the smallest values, another the next smallest values and so on. The reducers sort their shares, so that concatenating the outputs of successive reducers gives the result. The article [5] presents how to sample the data to determine the bounds for the partitioning, so that with probability of at least $1 - O(\frac{1}{n})$ the algorithm is minimal. In this case it means that the data is divided evenly between the reducers, while the transfer overhead for sampling is low. To deal with the data skew we order equal values based on their position in the input.

It is now enough to compute the number of data (non-end) elements smaller[1] (or equal) than each interval end and the total number of the data elements. Since in MapReduce we cannot easily perform linear amount of position searches for the interval ends, we sort them together with the dataset (interval ends are sorted behind data elements). In the last MapReduce phase, mappers operate on the sorted outputs of the previous reducers. They compute locally the required counts but using only their local values. They also compute locally the total number of data (non-end) elements in their share and maximum of their share and emit them as a tuple. In the final reduce phase, every reducer gets the pairs of total value and maximum over all mappers, and its share of range ends with their local counts. This information is enough to compute the grand total counts for each interval end in the share. Those values in turn suffice to compute the element counts for each range.

3.3 Multidimensional Interval Multiquery Processor

For a more general case where the conditions in COUNTIFS operator refer to different columns we propose another algorithm called Multidimensional Interval Multiquery Processor (MIMP). Such formulas are much less frequent in practical scenarios, but a realistic example is present in our use case:

`=COUNTIFS(A:A,Ai,D:D,FALSE,B:B,">"&Bi,B:B,"<"&Ci)` (*)

It counts the number of rows where at the same time the value in column A is equal to the value in Ai, the value in column D is FALSE, the value in column B is larger that the value in Bi, and the value in column B is smaller than the value in Ci.

If we treat the input rows for (*) as points in 2-D space, then a single row is essentially a query counting points located in the intersection of appropriate half-spaces (half-planes). However, each row is such a query. Therefore, if the row count is n, then we have n points and n queries. In general, the bulk algorithm we present in MapReduce is inspired by the classic range tree data structure, used for answering single counting queries of the kind we need. Yet, our idea does not require having one machine with memory sufficiently large for storing such a tree structure. We do not create this data structure explicitly, but imitate it by appropriate labelling of our data and query points.

A specific feature of our algorithm is that the number of queries which must be processed is of the same order as the number of data elements. In order to manage this situation, we combine data structure creation and query computation into a common process, in which query points are treated similarly to the data points.

The IMP algorithm is not suitable here as we cannot sort the input based on the Ai and Bi values at the same time. The problem of computing queries for this form of COUNTIFS can be abstracted in the following way. Assume k-dimensional space. The input is composed of two multisets D and Q. Multiset D consists of n *data points* $\langle a, \vec{x}\rangle = \langle a, x_1, \ldots, x_k\rangle$, while Q contains m *query points* $\langle b, \vec{y}\rangle = \langle b, y_1, \ldots, y_k\rangle$, where $\vec{x}, \vec{y} \in \mathbb{R}^k$. a and b are labels of queries and points. The labels represent locations of the value in a range tree with all values.

Note that we present here a more general algorithm than the one needed for (*). The labels, apart from serving the internal needs of the algorithm, correspond to possible equality conditions in a COUNTIFS instance (for (*) they would represent the condition imposed on the value in column D). Thus, the label will be also composed from the values needed for equality aggregation. Next, we present the algorithm for counting, but it will work, after obvious and simple modifications, for any associative and commutative operation, such as maximum, minimum, sum, product, disjunction, conjunction, exclusive or, etc.

Our algorithm simultaneously serves m queries on n points in total time $O((n + m)\log^{k-1}(n + m))$ and transfers data of total size $O((n + m)\log^{k-1}(n + m))$.

Henceforth we use *d-points* for data points and *q-points* for query points. D_a and Q_a are the multisets of all d-points (q-points, resp.) with label a.

Furthermore, we put $A_a = D_a \cup Q_a$ and $A = D \cup Q$.

The required output is the multiset of tuples

$$\{\{\langle a, \vec{y}, |\{\{\langle a, \vec{x}\rangle \in D_a \ : \ \vec{y} < \vec{x}\}\}|\rangle \ : \ \langle a, \vec{y}\rangle \in Q_a\}\}.$$

This is the number of points in the half-space intersection after partitioning the input set using labels. In plain words, for each q-point in Q_a we want to get the count of all d-points in D_a, whose spatial component is coordinate-wise larger than that of the q-point. Thus we restrict our presentation to one-sided conditions. The method for combinations of one-sided and two-sided conditions is conceptually very similar, but has more details.

Case $\mathbf{k = 1}$: This case is handled by a slight modification of the IMP algorithm, presented in the previous section. The only new element are the labels, which are easy to accommodate. We can also allow for the number of queries and data points to vary independently. The time taken by the algorithm is linear in $m+n$ (with high probability).

Case $\mathbf{k > 1}$: First A is sorted by: the label (most important), the first coordinate (less important) and q- and d- type (least important; d-points come before q-points). For each label a we rank

[1] We could also use counts of larger values.

the points with this label. For each point $p = \langle a, \vec{x} \rangle \in A_a$ we describe the location of this point in a range tree of points A_a and express it as a binary word w_p of length $\lceil \log |A_a| \rceil$. This requires two rounds of minimal MapReduce.

Then the mappers emit the following tuples:

- $\langle \langle a, v \rangle, x_2, \ldots, x_k \rangle$ for all $p = \langle a, x_1, x_2, \ldots, x_k \rangle \in D_a$ and all binary words v such that word $v1$ is a prefix of w_p;
- $\langle \langle a, v \rangle, x_2, \ldots, x_k \rangle$ for all $p = \langle a, x_1, x_2, \ldots, x_k \rangle \in Q_a$ and all binary words v such that word $v0$ is a prefix of w_p.

The first subset of the tuples uses binary labels to mark the locations of d-points in the range tree. The second set uses labels to spread the processing of the q-points over the tree and combine them with the d-points which must be counted to yield the answer to that query.

Lemma 1. *For each $d = \langle a, x_1, x_2, \ldots, x_k \rangle \in D_a$ and each $q = \langle a, y_1, y_2, \ldots, y_k \rangle \in Q_a$ such that $y_1 < x_1$, there is precisely one v such that $\langle \langle a, v \rangle, x_2, \ldots, x_k \rangle$ and $\langle \langle a, v \rangle, y_2, \ldots, y_k \rangle$ are simultaneously emitted.*

If $d = \langle a, x_1, x_2, \ldots, x_k \rangle \in D_a$ and $q = \langle a, y_1, y_2, \ldots, y_k \rangle \in Q_a$ such that $y_1 \geq x_1$, then there is no v such that $\langle \langle a, v \rangle, x_2, \ldots, x_k \rangle$ and $\langle \langle a, v \rangle, y_2, \ldots, y_k \rangle$ are simultaneously emitted.

Proof. Lexicographic order of w_d and w_q agrees with the order of d and q w.r.t. their first spatial components. Hence if $x_1 > y_1$, then $w_d > w_q$ lexicographically, so for their longest common prefix v, we get that $v1$ is a prefix of w_d and $v0$ is a prefix of w_q. For this particular v both required tuples are emitted, and they are both emitted for no other word.

If $x_1 \leq y_1$, then $w_d < w_q$ lexicographically (recall that a d-point precedes a q-point in the sorting, if their first spatial components are equal), so for no common prefix v of w_d and w_q, the next letter following v is 1 in w_q and 0 in w_d. So in this case it is impossible that both $\langle \langle a, v \rangle, x_2, \ldots, x_k \rangle$ and $\langle \langle a, v \rangle, y_2, \ldots, y_k \rangle$ are simultaneously emitted. □

Corollary 1. *For each $\langle a, y_1, y_2, \ldots, y_k \rangle \in Q_a$ holds*

$$|\{\{\langle a, x_1, x_2, \ldots, x_k \rangle \in D_a \ : \ \vec{x} < \vec{y}\}\}| = \sum_v |\{\{\langle \langle a, v \rangle, x_2, \ldots, x_k \rangle \in D_{\langle a, v \rangle} \ : \ (x_2, \ldots, x_k) < (y_2, \ldots, y_k)\}\}| \quad (1)$$

Corollary 1 shows that the problem is reduced to a smaller $k-1$ dimensional one in the following way. We run the algorithm for $k-1$ on all created tuples. The returned tuples are then combined using minimal MapReduce methods to compute Group By and the sum based on [5], according to formula (1). This produces the final result of the algorithm.

Since we have a correct algorithm for the one-dimensional case, the construction is finished.

4. TESTS

We undertook tests of the algorithms and implementation. We used the example from Section 2 as the source of spreadsheet formulas to translate. We evaluated them by means of a MapReduce computation using our algorithm. The test environment was Hadoop 2.6.0, running on a cluster of 16 Linux machines with Intel Xeon E3-1220 3.1GHz CPU, 8 GB of RAM and HDD 500 GB, connected by 100 Mb/s Ethernet.

Platform	MR 4	MR 8	MR 16
time [min:sec]	16:41	14:14	11:18

Table 1: Execution times for the complete 15,285,050 rows of NYCTT May 2013 dataset.

The test was conducted on the complete data from May 2013, which consisted of 15,285,050 lines of data, roughly 15 times more than what Excel can process. We used the above cluster of computers, with varying the number of active machines. The results are shown in Table 1.

For comparison purpose, only, we tested also the initial Excel spreadsheet. We processed 1024 thousand rows (Excel 2013 64 bit under Windows 10 64 bit, machine with Intel Xeon CPU @2.20 GHz and 8 GB RAM). It turns out that the formulas, as evaluated by Excel, yield a quadratic algorithm. This amount of rows, maximal possible in Excel, covers about 50 hours of May 2013 data, and requires more than 11 hours to process in Excel. A user with advanced knowledge about spreadsheet programming, algorithmic skills and knowledge about the characteristics of this particular dataset, can rewrite this spreadsheet to perform the same task much faster, but cannot overcome the maximal data size limitation.

5. SUMMARY AND FURTHER RESEARCH

In this paper we presented a minimal algorithm for distributing the spreadsheet `COUNTIFS` function with MapReduce. Such effort is necessary to bring large scale analytics to the masses as spreadsheets are the most accessible data analytics tools available for people with no sophisticated programming skills. The algorithm presented in this paper covers the most popular syntax of `COUNTIFS` and our techniques can be easily adapted to other popular spreadsheet functions like `SUM`. The minimal algorithm guarantees that the computation is balanced. The problem of supporting the full syntax of `COUNTIFS` is left for a future work due to the space limitations of this extended abstract.

Acknowledgements: Jacek Sroka was sponsored by National Science Centre based on decision DEC-2012/07/D/ST6/02492 and Mirosław Kowaluk based on decision DEC-2014/13/B/ST6/00770.

6. REFERENCES

[1] B. Donovan and D. B. Work. New York City Taxi Trip Data (2010-2013) 1.0. http://dx.doi.org/10.13012/J8PN93H8, 2014.

[2] B. Jansen. Enron versus EUSES: A comparison of two spreadsheet corpora. *CoRR*, abs/1503.04055, 2015.

[3] B. Luciano, P. Kien, S. Claudio, V. M. R., and F. Juliana. Structured open urban data: Understanding the landscape. *Big Data*, 2(3):144–154, 2014.

[4] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 207–214. IEEE, 2005.

[5] Y. Tao, W. Lin, and X. Xiao. Minimal MapReduce algorithms. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 529–540, New York, NY, USA, 2013. ACM.