B. Mitschang et al. (Hrsg.): BTW 2017 – Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 163

Computational Social Choice in the Clouds

Theresa Csar[1], Martin Lackner[2], Reinhard Pichler[3] and Emanuel Sallinger[4]

Abstract: In the era of big data we are concerned with solving computational problems on huge datasets. To handle huge datasets in cloud systems dedicated programming frameworks are used, among which MapReduce is the most widely employed. It is an important issue in many application areas to design parallel algorithms which can be executed efficiently on cloud systems and can cope with big data. In computational social choice we are concerned with computational questions of joint decision making based on preference data. The question of how to handle huge preference datasets has not yet received much attention. In this report we summarize our recent work on designing and evaluating algorithms for winner determination in huge elections using the MapReduce framework.

Keywords: Computational Social Choice, Cloud Computing, MapReduce, Parallel Computing, Distributed Computing

1 Introduction

Especially with the rise of e-business platforms, huge preference datasets are becoming more and more common. Some interesting examples are the comparison of results of search engines, preferences in online movie or book stores, surveys among a large number of people, etc. Computing winner sets over such preferences is an important topic in computational social choice. Yet the handling of huge sets of preference data has not received much attention. Often these large datasets are deployed and analysed in cloud systems. In such cloud systems MapReduce [DG08] is the most widely used framework for data intensive parallel computation.

In [Cs17] algorithms for winner determination in huge elections are proposed, analysed and experimentally evaluated. In this report we summarize our recent work in [Cs17] and explain one MapReduce algorithm for winner determination in more detail. We provide a short introduction to MapReduce, methods of social choice and winner determination and at the end give possible directions for future work.

[1] TU Wien, Abteilung für Datenbanken und Artificial Intelligence, 1040 Wien, csar@dbai.tuwien.ac.at
[2] University of Oxford, Department of Computer Science, Oxford, martin.lackner@cs.ox.ac.uk
[3] TU Wien, Abteilung für Datenbanken und Artificial Intelligence, 1040 Wien, pichler@dbai.tuwien.ac.at
[4] University of Oxford, Department of Computer Science, Oxford, emanuel.sallinger@cs.ox.ac.uk

2 Preliminaries

2.1 MapReduce

The challenges of handling huge data sets in the cloud is an issue in many research areas, ranging from core database tasks such as computing joins [AU10, BKS13] to data intensive computations in general [Af13]. MapReduce, and its open source implementation Hadoop, are widely used frameworks in cloud computing. MapReduce was first introduced by Google [DG08]. The MapReduce model is divided into three phases: map, shuffle and reduce. In the map phase the input data is read and key-value pairs are produced. In the shuffle phase all key-value pairs are assigned to a corresponding reduce task with the same key. The reduce tasks then perform simple calculations on the received values. The reduce tasks cannot communicate among each other and write their results back to the distributed file system after the computation is finished. If a computation has to be done in several MapReduce rounds, the data is read again in each map phase and is written to the distributed file system after the reduce phase.

In the theoretical analysis of MapReduce algorithms the following performance characteristics are of interest: total communication cost, wall clock time, replication rate [AU10] and the number of MapReduce rounds. These performance characteristics describe the communication cost and the computation time in more detail:

Communication Cost. We analyse two measures of communication cost: the total communication cost and the replication rate. The total communication cost refers to the total number of values transmitted by the map and reduce tasks, whereas the replication rate is a relative measure of communication cost. It is defined as the ratio of the input size to the reduce phase and the input size to the map phase.

Computation Time. There are two important measures to describe computation time. First one can consider the number of MapReduce rounds needed to complete the total computation. Since MapReduce is typically not optimized for iterative computations, each additional MapReduce round takes time for writing and reading the data from/to disks. The second measure is wall clock time, which refers to the maximum time needed for the parallel execution.

2.2 Computational Social Choice: Winner Determination

A central problem in computational social choice is winner determination. Given a set of candidates C and a list of votes, the goal is to compute the set of all winners according to a given voting rule. The input is a set of votes, and the votes (preferences) can be given as partial or total orders (e.g., $\{a > c > d > b, a > e\}$).

Preferences can be aggregated to a weak and/or strict dominance matrix. The (weak/strict) dominance matrix D is a $(0,1)$-matrix, where $D[a,b] = 1$ means that a (weakly/strictly) dominates b. A candidate a dominates b, if there are more votes preferring a over b.

In Winner Determination we are concerned with selecting a subset of all candidates as winners. There are many different approaches for selecting such a winner set. A seemingly straight forward method is selecting the *Condorcet winner*, which is a single candidate that dominates all other candidates in a pairwise comparison. Unfortunately such a winner does not necessarily exist. Many other methods for selecting subsets of candidates as winners have been developed. The computational complexity of some of them is studied in [BFH09], one of them is the Smith set. Due to space reasons we choose the Smith set as the only method outlined in this report, since the algorithm illustrates the overall idea of MapReduce very well. The Smith set is defined as the smallest possible set of candidates that dominate all outside candidates. If there exists a Condorcet Winner, the Condorcet Winner is the only candidate in the Smith set. The proposed algorithm for computing the Smith set works very well with MapReduce and is explained in the next section.

3 Computational Social Choice in the Clouds

In [Cs17] we develop parallel algorithms for winner determination in huge elections using MapReduce. In particular, we present parallel algorithms for computing the *Schwartz Set*, *Smith Set*, *Copeland Set* and *dominance graphs*. Furthermore, we show that some methods for winner determination are unlikely to be parallelizable, e.g. we show that the single-transferable vote (STV) rule is P-complete and thus a highly parallelizable algorithm may not be hoped for. In this report the computation of the Smith Set is explained in more detail. The algorithm is based on non-trivial observations of [BFH09] and fits very well into the MapReduce framework. It is noteworthy that preference data sets can be large in two dimensions (many votes and/or many candidates) and both scenarios can be found in real world data. A closer inspection of algorithms from [Cs17] show that a large number of voters is far easier to handle than a large number of candidates; hence the focus of the paper is on the more challenging case of data sets with many candidates.

All algorithms presented in [Cs17] use the MapReduce framework for parallelization. Some performance characteristics of the proposed algorithms have been studied theoretically and experiments have been performed. In the theoretical analysis we focused on the following parameters: total communication cost, wall clock time, replication rate and the number of MapReduce rounds (see Section 2.1).

3.1 MapReduce Algorithm for computing the Smith Set

This is a brief sketch of the MapReduce algorithm for computing the Smith set proposed in [Cs17]. A candidate a is in the Smith set if for every candidate b there is a path from a to b in the weak dominance graph [BFH09]. A naive algorithm for computing the Smith set would thus first compute the transitive closure of the weak dominance graph. This would take logarithmically many MapReduce rounds. We can do much better by applying the following property: Let $D^k_{\succeq}(v)$ denote the set of vertices reachable from vertex v by a path of length $\leq k$ in the weak dominance graph. [BFH09] show that in the weak dominance

graph a vertex t is not reachable from a vertex s if and only if there exists a vertex v such that $D^2_{\succeq}(v) = D^3_{\succeq}(v)$, $s \in D^2_{\succeq}(v)$, and $t \notin D^2_{\succeq}(v)$. This characteristic is put to value in our MapReduce algorithm for computing the Smith set.

Each vertex is saved and processed as a list of three sets: **old** contains the set of vertices that are already known to be reached from a, **new** is the set with newly found vertices that can be reached from a and **reachedBy** contains all vertices known to reach a.

Map Phase. The vertices are read from the input file and for each vertex the following key-value pairs are produced: For every vertex r in *reachedBy* as key, the whole set *new* is emitted with mode 'new' as value. The resulting key-value pair is: (key=r,value= (set *new*, mode='new')). Likewise, for every vertex n in *new* the pair (key=n,value= (set *reachedBy*, mode='reachedBy')) is emitted.

Reduce Phase. Each reduce task is responsible for one candidate/vertex and combines the received values. The outputs are the corresponding vertices with the sets *new*, *old* and *reachedBy*. The set *reachedBy* is the union of all input values with mode 'reachedBy'; the same applies to the set *old*. The set *new* must not contain any values that are already listed in *old*, therefore we first create the set *new* as the union of all received values with mode 'new', but then we set $new = new - old$.

The map and reduce phase are done twice and then an additional round for post-processing is needed to identify the candidates contained in the Smith set. The algorithm takes three MapReduce rounds in total.

The proposed MapReduce algorithm for computing the Smith set of a data set with m candidates has a replication rate of $rr \leq 2m+1$ and the number of keys is m, where m is the number of candidates. The wall clock time is $\leq 6m^2 + 8m$ and the total communication cost is $\leq 6m^3 + 8m^2$ [Cs17].

4 Future Work

Real World Data: Identify Social Choice Problems in the Clouds. An important next step will be to evaluate the algorithms on real world data, since the experiments in [Cs17] where performed solely on synthetic datasets. Other problems in computational social choice that can make use of such parallel algorithms have to be identified. It will be important to identify what preference data sets are already available in the cloud.

Alternatives to MapReduce/Hadoop. MapReduce and its Open Source implementation Apache Hadoop are widely used for parallel computation. However MapReduce is typically not ideal for iterative computations. There are already many new adaptations and extensions available. In [Le16] an experimental comparison of iterative MapReduce frameworks has been performed. As expected Hadoop performs worse for iterative computations, since Hadoop/MapReduce is optimized for batch processing. Spark outperforms all other tested frameworks. In comparison to Hadoop, Spark [Za10] loads all data into memory and organizes it as Resilient Distributed Datasets (RDDs), while Hadoop reads the data

in junks from the distributed file system. Spark also allows for more generic data flows and is not limited to the strict MapReduce API [Le16]. An interesting topic is to analyze for which problem settings Spark outperforms Hadoop and the other way around.

5 Conclusion

We have made a first step to take computational social choice to the cloud. Yet, more work has yet to be done. This report provides an overview of recent work and possible future research directions. The challenges of cloud computing start from dataset management in a distributed file system, choosing a fitting parallel framework and finally lead to designing algorithms to efficiently solve computation problems. All these challenges have to be overcome to take computational social choice to the cloud.

References

[Af13] Afrati, Foto N.; Sarma, Anish Das; Salihoglu, Semih; Ullman, Jeffrey D.: Upper and Lower Bounds on the Cost of a Map-Reduce Computation. Proceedings of the VLDB Endowment, 6(4):277–288, 2013.

[AU10] Afrati, Foto N.; Ullman, Jeffrey D.: Optimizing joins in a map-reduce environment. In: EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, March 22-26, 2010, Proceedings. volume 426 of ACM International Conference Proceeding Series. ACM, pp. 99–110, 2010.

[BFH09] Brandt, Felix; Fischer, Felix; Harrenstein, Paul: The computational complexity of choice sets. Mathematical Logic Quarterly, 55(4):444–459, 2009.

[BKS13] Beame, Paul; Koutris, Paraschos; Suciu, Dan: Communication steps for parallel query processing. In: Proc. PODS 2013. ACM, pp. 273–284, 2013.

[Cs17] Csar, Theresa; Lackner, Martin; Pichler, Reinhard; Sallinger, Emanuel: Winner Determination in Huge Elections with MapReduce. In: Proceedings of AAAI-17. AAAI Press, 2017. To appear.

[DG08] Dean, Jeffrey; Ghemawat, Sanjay: MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

[Le16] Lee, Haejoon; Kang, Minseo; Youn, Sun-Bum; Lee, Jae-Gil; Kwon, YongChul: An Experimental Comparison of Iterative MapReduce Frameworks. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM, pp. 2089–2094, 2016.

[Za10] Zaharia, Matei; Chowdhury, Mosharaf; Franklin, Michael J; Shenker, Scott; Stoica, Ion: Spark: cluster computing with working sets. HotCloud, 10:10–10, 2010.

Acknowledgments: This work was supported by the Austrian Science Fund projects (FWF):P25207-N23, (FWF):P25518-N23 and (FWF):Y698, by the Vienna Science and Technology Fund (WWTF) through project ICT12-015, by the European Research Council (ERC) under grant number 639945 (ACCORD) and by the EPSRC programme grant EP/M025268/1.