Assignment Problems of Different-Sized Inputs in MapReduce

FOTO AFRATI, National Technical University of Athens, Greece
SHLOMI DOLEV, EPHRAIM KORACH, and SHANTANU SHARMA, Ben-Gurion University, Israel
JEFFREY D. ULLMAN, Stanford University, USA

A MapReduce algorithm can be described by a *mapping schema*, which assigns inputs to a set of reducers, such that for each required output there exists a reducer that receives all the inputs that participate in the computation of this output. Reducers have a capacity, which limits the sets of inputs that they can be assigned. However, individual inputs may vary in terms of size. We consider, for the first time, mapping schemas where input sizes are part of the considerations and restrictions. One of the significant parameters to optimize in any MapReduce job is communication cost between the map and reduce phases. The communication cost can be optimized by minimizing the number of copies of inputs sent to the reducers. The communication cost is closely related to the number of reducers of constrained capacity that are used to accommodate appropriately the inputs, so that the requirement of how the inputs must meet in a reducer is satisfied. In this work, we consider a family of problems where it is required that each input meets with each other input in at least one reducer. We also consider a slightly different family of problems in which, each input of a list, X, is required to meet each input of another list, Y, in at least one reducer. We prove that finding an optimal mapping schema for these families of problems is NP-hard, and present a bin-packing-based approximation algorithm for finding a near optimal mapping schema.

Categories and Subject Descriptors: H.2.4 [**Systems**]: Parallel Databases; H.2.4 [**Systems**]: Distributed Databases; C.2.4 [**Distributed Systems**]: Distributed Databases

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Distributed computing, mapping schema, MapReduce algorithms, reducer capacity, and reducer capacity and communication cost tradeoff

1. INTRODUCTION

MapReduce [Dean and Ghemawat 2004] is a programming system used for parallel processing of large-scale data. It has two phases, the *map phase* and the *reduce phase*. The given input data is processed by the map phase that applies a user-defined map function to produce intermediate data (of the form $\langle key, value \rangle$). Intermediate data is, then, processed by the reduce phase that applies a user-defined reduce function to keys and their associated values. The final output is provided by the reduce phase. A detailed description of MapReduce can be found in Chapter 2 of [Leskovec et al. 2014].

This paper is accepted in **ACM Transactions on Knowledge Discovery from Data (TKDD), August 2016.** Preliminary versions of this paper have appeared in the proceeding of DISC 2014 and BeyondMR 2015 [Afrati et al. 2015].
This work of F. Afrati is supported by the project Handling Uncertainty in Data Intensive Applications, co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program "Education and Lifelong Learning," under the program THALES. This work of S. Dolev is partially supported by Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant number 428/11), Cabarnit Cyber Security MAGNET Consortium, and Ministry of Science and Technology, Infrastructure Research in the Field of Advanced Computing and Cyber Security.
Author's addresses: F. Afrati, School of Electrical and Computing Engineering, National Technical University of Athens, Greece (e-mail: afrati@softlab.ece.ntua.gr), S. Dolev, Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel (e-mail: dolev@cs.bgu.ac.il), E. Korach, Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Israel (e-mail: korach@bgu.ac.il), S. Sharma, Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel (e-mail: sharmas@cs.bgu.ac.il), J.D. Ullman, Department of Computer Science, Stanford University, USA (e-mail: ullman@cs.stanford.edu).
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2016 ACM 1539-9087/2016/08-ARTA $15.00
DOI:http://dx.doi.org/10.1145/0000000.0000000

arXiv:submit/1699814 [cs.DB] 20 Oct 2016

Communication Cost and Reducer Capacity. An important performance measure for MapReduce algorithms is the amount of data transferred from the *mappers* (the processes that implement the map function) to the *reducers* (the processes that implement the reduce function). This is called the *communication cost*. The minimum communication cost is, of course, the size of the desired inputs that provide the final output, since we need to transfer all these inputs from the mappers to the reducers at least once. However, we may need to transfer the same input to several reducers, thus increasing the communication cost.

Depending on various factors of our setting, each reducer may process a larger or smaller amount of data. The amount of data each reducer processes however affects the wall clock time of our algorithms and the degree of parallelization. If we send all data in one reducer, then we have low communication (equal to the size of the data) but we have low degree of parallelization, and thus the wall clock time increases. Thus, the maximum amount of data a reducer can hold is a constraint when we build our algorithm.

Reducer capacity. We define *reducer capacity* to be the upper bound on the sum of the sizes of the $values$ that are assigned to the reducer. For example, we may choose the reducer capacity to be the size of the main memory of the processor on which the reducer runs or we may arbitrarily set a low reducer capacity if we want high parallelization. We always assume in this paper that all the reducers have an identical capacity, denoted by q.

There are various works in the field of MapReduce algorithms design (*e.g.*, [Karloff et al. 2010; Ullman 2012; Afrati et al. 2013; Goodrich 2010; Pietracaprina et al. 2012; Afrati and Ullman 2013]) that investigate problems and/or build algorithms with minimum communication cost when the reducer size is bounded by the number of inputs that a reducer is allowed to hold. In this paper, we consider for the first time problems where each input may have a different size and the *reducer capacity* is an upper bound on the sum of the sizes of the inputs in a reducer. Here, we investigate the problem where each input is required to meet in a reducer with any other input. We give now some examples where this problem may appear in practice.

Motivating Examples. We present three examples.

Example 1.1. *Computing common friends.* An input is a list of friends. We have such lists for m persons. Each pair of lists of friends corresponds to one output, which will show us the common friends of the respective persons. Thus, it is mandatory that lists of friends of every two persons are compared. Specifically, the problem is: a list $F = \{f_1, f_2, \ldots, f_m\}$ of m friends is given, and each pair of elements $\langle f_i, f_j \rangle$ corresponds to one output, common friends of persons i and j; see Figure 1.

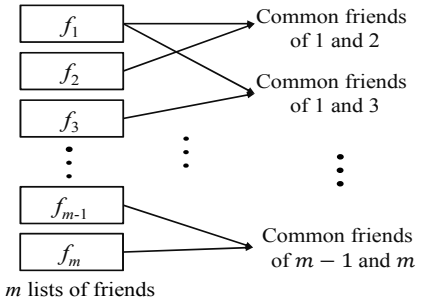

Fig. 1: Computing common friends example.

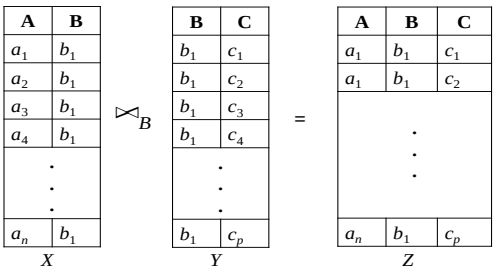

Fig. 2: Skew join example for a heavy hitter, b_1.

Example 1.2. *Similarity-join.* Similarity-join is an example of the *A2A mapping schema problem* that can be used to find the similarity between any two inputs, *e.g.*, Web pages or documents. A set of m inputs (*e.g.*, Web pages) $WP = \{wp_1, wp_2, \ldots, wp_m\}$, a similarity

function $sim(x, y)$, and a similarity threshold t are given, and each pair of inputs $\langle wp_x, wp_y \rangle$ corresponds to one output such that $sim(wp_x, wp_y) \geq t$.

It is necessary to compare all-pairs of inputs when the similarity measure is sufficiently complex that shortcuts like locality-sensitive hashing are not available. Therefore, it is mandatory that every two inputs (Web pages) of the given input set (WP) are compared. The similarity-join is useful in various applications, mentioned in [Bayardo et al. 2007], *e.g.*, near-duplicate document detection, collaborative filtering, and query refinement for Web search.

Example 1.3. *The drug-interaction problem.* The drug-interaction problem is given in [Ullman 2012], where a list of inputs consists of 6,500 drugs and a drug i holds information about the medical history of patients who had taken the drug i. The objective is to find pairs of drugs that had particular side effects. In order to achieve the objective, it is mandatory that each pair of drugs is compared.

Example 1.4. *Skew join of two relations $X(A, B)$ and $Y(B, C)$.* The join of relations $X(A, B)$ and $Y(B, C)$, where the joining attribute is B, provides output tuples $\langle a, b, c \rangle$, where (a, b) is in A and (b, c) is in C. One or both of the relations X and Y may have a large number of tuples with an identical B-value. A value of the joining attribute B that occurs many times is known as a *heavy hitter*. In skew join of $X(A, B)$ and $Y(B, C)$, all the tuples of both the relations with an identical heavy hitter should appear together to provide the output tuples.

In Figure 2, b_1 is considered as a heavy hitter; hence, it is required that all the tuples of $X(A, B)$ and $Y(B, C)$ with the heavy hitter, $B = b_1$, should appear together to provide the desired output tuples, $\langle a, b_1, c \rangle$ ($a \in A, b_1 \in B, c \in C$), which depend on exactly two inputs.

Problem Statements. We define two problems where exactly two inputs are required for computing an output:

All-to-All problem.. In the *all-to-all* (*A2A*) problem, a list of inputs is given, and each pair of inputs corresponds to one output.

X-to-Y problem.. In the *X-to-Y* (*X2Y*) problem, two disjoint lists X and Y are given, and each pair of elements $\langle x_i, y_j \rangle$, where $x_i \in X, y_j \in Y, \forall i, j$, of the lists X and Y corresponds to one output.

Computing common friends on a social networking site, and the drug-interaction problem are examples of A2A problems. Skew join is an example of a X2Y problem.

A *mapping schema* defines a MapReduce algorithm. A *mapping schema* assigns input to reducers, so that no reducer exceeds the reducer capacity and all pairs of inputs (in A2A problem) or all pairs of X-to-Y inputs (in X2Y problem) meet in the same reducer.[1]

The communication cost, is a significant factor in the performance of a MapReduce algorithm. The communication cost comes with a tradeoff in the degree of parallelism, as we mentioned. A mapping schema is *optimal* if there is no other mapping schema with a lower communication cost. In this paper, we investigate how to construct optimal mapping schemas or good approximations of them.

Outline of Paper and Our Contribution. In this paper, we investigate the problem of finding an optimal or near optimal mapping schema for the case we have inputs of different sizes.

— In Section 2, we warm up to the problem with discussing how the tradeoffs appear.
— In Section 3, we prove that finding an optimal mapping schema is intractable.
— In Section 4, we present preliminary results and present one of our techniques to obtain near optimal mapping schemas. The technique is to do bin-packing first and collect inputs in bins, then treat bins as inputs, possibly all of equal size.

[1]For more general problems, we are given the graph which defines which pairs of inputs should meet in the same reducer to solve the problem and this is what the mapping schema should achieve – but we do not consider such problems here.

— In Section 5, we present algorithms to construct optimal mapping schemas in certain cases where the inputs are all of equal size.
— In Sections 6 and 8, we combine bin-packing and algorithmic techniques from Section 5 to build algorithms that construct mapping schemas that are good approximations to the optimal. For each algorithm, we argue in the end how good an approximation this is.
— In Section 7, we extend the idea presented in Section 5.3 for equal size inputs.
— In Sections 6 and 8, we only considered the case when there is no input of size $> \frac{q}{2}$ (remember we denote with q the reducer capacity). Thus in Section 9, we investigate the case where there is an input of size $> \frac{q}{2}$. We mainly use similar techniques as in Section 4.
— So far we have investigated the A2A problem. In Section 10, we take the X2Y problem to provide algorithms for this too.

Related Work. MapReduce was introduced by Dean and Ghemawat in 2004 [Dean and Ghemawat 2004]. Karloff et al. [Karloff et al. 2010] presents a model for comparing MapReduce with the Parallel Random Access Machine (PRAM) model and states that a large class of PRAM algorithms can be simulated by MapReduce. However, parallel and sequential computations (used in MapReduce) differentiate MapReduce and PRAM model. Another model considers the efficiency of MapReduce algorithms in terms of algorithm's running time, suggested in [Goodrich 2010]. The author simulates PRAM algorithms by MapReduce and defines memory-bound for MapReduce algorithms in terms of reducer I/O sizes for each round and each reducer.

Following [Karloff et al. 2010; Goodrich 2010], a filtering technique for MapReduce is suggested in [Lattanzi et al. 2011]. This technique removes some of nonessential data and results in fewer rounds than in both the previous stated models [Karloff et al. 2010; Goodrich 2010]. Essentially, the models, in [Karloff et al. 2010; Goodrich 2010; Lattanzi et al. 2011], provide a way to simulate a large family of PRAM algorithms by MapReduce.

Afrati et al. [Afrati et al. 2013] presents a model for MapReduce algorithms where an output depends on two inputs, and shows a tradeoff between the communication cost and parallelism. In [Afrati and Ullman 2013], the authors consider a case where each pair of inputs produces an output and present an upper bound that meets the lower bound on the communication cost as a function of the number of inputs sent to a reducer. However, both in [Afrati et al. 2013] and [Afrati and Ullman 2013] the authors regard the reducer capacity in terms of the number of inputs (assuming each input is of an identical size) sent to a reducer.

Our setting is closely related to the settings given by Afrati et al. [Afrati et al. 2013], but we allow the input sizes to be different. To the best of our knowledge, we for the first time do not restrict the input sizes to be identical. Thus, we consider a more realistic settings for MapReduce algorithms that can be used in various practical scenarios.

2. MAPPING SCHEMA AND TRADEOFFS

Our system setting is an extension of the standard system setting [Afrati et al. 2013] for MapReduce algorithms, where we consider, for the first time, inputs of different sizes. In this section, we provide formal definitions and some examples to show the tradeoff between communication cost and degree of parallelization.

Mapping Schema. A mapping schema is an assignment of the set of inputs to some given reducers so that the following two constraints are satisfied:

— A reducer is assigned inputs whose sum of the sizes is less than or equal to the reducer capacity q.
— For each output, we must assign its corresponding inputs to at least one reducer in common.

A mapping schema is optimal when the communication cost is minimum. The number of reducers we use often is minimal for an optimal mapping schema but this may not always be the case. It is desirable to minimize the number of reducers too. We offer insight about communication cost and number of reducers uses in Examples 2.1 and 2.2.

Tradeoffs. The following tradeoffs appear in MapReduce algorithms and in particular in our setting:

— A tradeoff between the reducer capacity and the number of reducers. For example, large reducer capacity allows the use of a smaller number of reducers.
— A tradeoff between the reducer capacity and the parallelism. For example, if we want to achieve a high degree of parallelism, we set low reducer capacity.
— A tradeoff between the reducer capacity and the communication cost. For example, in the case reducer capacity is equal to the total size of the data then we can use one reducer and have minimum communication (of course, this goes at the expense of parallelization).

In the subsequent subsections, we present the A2A mapping schema problem and the X2Y mapping schema problem with fitting examples and explain the tradeoffs.

2.1. The *A2A Mapping Schema Problem*

An instance of the *A2A mapping schema problem* consists of a list of m inputs whose input size list is $W = \{w_1, w_2, \ldots, w_m\}$ and a set of z identical reducers of capacity q. A solution to the *A2A mapping schema problem* assigns every pair of inputs to at least one reducer in common, without exceeding q at any reducer.

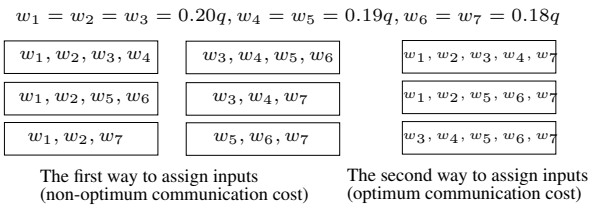

Fig. 3: An example to the *A2A mapping schema problem*.

Example 2.1. We are given a list of seven inputs $I = \{i_1, i_2, \ldots, i_7\}$ whose size list is $W = \{0.20q, 0.20q, 0.20q, 0.19q, 0.19q, 0.18q, 0.18q\}$ and reducers of capacity q. In Figure 3, we show two different ways that we can assign the inputs to reducers. The best we can do to minimize the communication cost is to use three reducers. However, there is less parallelism at the reduce phase as compared to when we use six reducers. Observe that when we use six reducers, then all reducers have a lighter load, since each reducer may have capacity less than $0.8q$.

The communication cost for the second case (3 reducers) is approximately $3q$, whereas for the first case (6 reducers) it is approximately $4.2q$. Thus, in tradeoff, in the 3-reducers case we have low communication cost but also lower degree of parallelization, whereas in the 6-reducers case we have high parallelization at the expense of the communication cost.

2.2. The *X2Y Mapping Schema Problem*

An instance of the *X2Y mapping schema problem* consists of two disjoint lists X and Y and a set of identical reducers of capacity q. The inputs of the list X are of sizes $w_1, w_2, \ldots, w_m$, and the inputs of the list Y are of sizes $w'_1, w'_2, \ldots, w'_n$. A solution to the *X2Y mapping schema problem* assigns every two inputs, the first from one list, X, and the second from the other list, Y, to at least one reducer in common, without exceeding q at any reducer.

Example 2.2. We are given two lists, X of 12 inputs, and Y of 4 inputs (see Figure 4) and reducers of capacity q. We show that we can assign each input of the list X with each input of the list Y in two ways. In order to minimize the communication cost, the best way is to use 12 reducers. Note that we cannot obtain a solution for the given inputs using less than 12 reducers. However, the use of 12 reducers results in less parallelism at the reduce phase as compared to when we use 16 reducers.

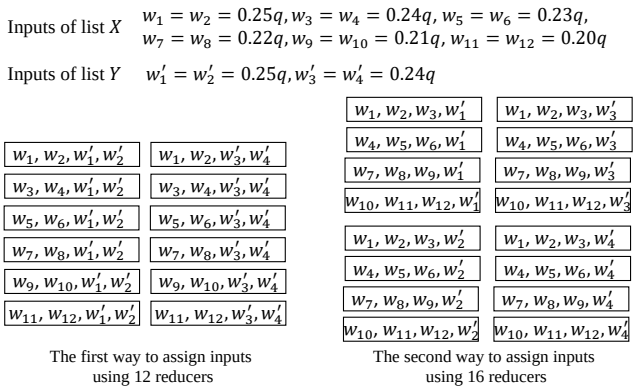

Fig. 4: An example to the *X2Y mapping schema problem*.

— In this paper, we assume we have made a decision on the degree of parallelization we want (by setting the reducer capacity q).

3. INTRACTABILITY OF FINDING A MAPPING SCHEMA

In this section, we will show that the *A2A* and the *X2Y* mapping schema problems do not possess a polynomial solution. In other words, we will show that the assignment of *two required inputs* to the minimum number of identical-capacity reducers to find solutions to the *A2A* and the *X2Y* mapping schema problems cannot be achieved in polynomial time.

3.1. NP-hardness of the *A2A Mapping Schema Problem*

A list of inputs $I = \{i_1, i_2, \ldots, i_m\}$ whose input size list is $W = \{w_1, w_2, \ldots, w_m\}$ and a set of identical reducers $R = \{r_1, r_2, \ldots, r_z\}$, are an input instance to the *A2A mapping schema problem*. The *A2A mapping schema problem* is a decision problem that asks whether or not there exists a mapping schema for the given input instance such that every input, i_x, is assigned with every other input, i_y, to at least one reducer in common. An answer to the *A2A mapping schema problem* will be "yes," if for each pair of inputs ($\langle i_x, i_y \rangle$), there is at least one reducer that holds them.

In this section, we prove that the *A2A mapping schema problem* is NP-hard in the case of $z > 2$ identical reducers. In addition, we prove that the *A2A mapping schema problem* has a polynomial solution to one and two reducers.

If there is only one reducer, then the answer is "yes" if and only if the sum of the input sizes $\sum_{i=1}^{m} w_i$ is at most q. On the other hand, if $q < \sum_{i=1}^{m} w_i$, then the answer is "no." In case of two reducers, if a single reducer is not able to accommodate all the given inputs, then there must be at least one input that is assigned to only one of the reducers, and hence, this input is not paired with all the other inputs. In that case, the answer is "no." Therefore, we achieve a polynomial solution to the *A2A mapping schema problem* for one and two identical-capacity reducers.

We now consider the case of $z > 2$ and prove that the *A2A mapping schema problem* for $z > 2$ reducers is at least as hard as the partition problem.

THEOREM 3.1. *The problem of finding whether a mapping schema of m inputs of different input sizes exists, where every two inputs are assigned to at least one of $z \geq 3$ identical-capacity reducers, is NP-hard.*

The proof appears in Appendix A.

3.2. NP-hardness of the *X2Y Mapping Schema Problem*

Two lists of inputs, $X = \{i_1, i_2, \ldots, i_m\}$ whose input size list is $W_x = \{w_1, w_2, \ldots, w_m\}$ and $Y = \{i'_1, i'_2, \ldots, i'_n\}$ whose input size list is $W_y = \{w'_1, w'_2, \ldots, w'_n\}$, and a set of identical reducers $R = \{r_1, r_2, \ldots, r_z\}$ are an input instance to the *X2Y mapping schema problem*. The *X2Y mapping schema problem* is a decision problem that asks whether or not there exists a mapping schema for the given input instance such that each input of the list X is assigned with each input of the list Y to at least one reducer in common. An answer to the *X2Y mapping schema problem* will be "yes," if for each pair of inputs, the first from X and the second from Y, there is at least one reducer that has both those inputs.

The *X2Y mapping schema problem* has a polynomial solution for the case of a single reducer. If there is only one reducer, then the answer is "yes" if and only if the sum of the input sizes $\sum_{i=1}^{m} w_i + \sum_{i=1}^{n} w'_i$ is at most q. On the other hand, if $q < \sum_{i=1}^{m} w_i + \sum_{i=1}^{n} w'_i$, then the answer is "no." Next, we will prove that the *X2Y mapping schema problem* is an NP-hard problem for $z > 1$ identical reducers.

THEOREM 3.2. *The problem of finding whether a mapping schema of m and n inputs of different input sizes that belongs to list X and list Y, respectively, exists, where every two inputs, the first from X and the second from Y, are assigned to at least one of $z \geq 2$ identical-capacity reducers, is NP-hard.*

The proof appears in Appendix A.

4. APPROXIMATION ALGORITHMS: PRELIMINARY RESULTS

Since the *A2A Mapping Schema Problem* is NP-hard, we start looking at special cases and developing approximation algorithm to solve it. We propose several approximation algorithms for the *A2A mapping schema problem* that are based on bin-packing algorithms, selection of a prime number p, and division of inputs into two sets based on their sizes.

Each algorithm takes the number of inputs, their sizes, and the reducer capacity (see Table II). The approximation algorithms have two cases depending on the sizes of the inputs, as follows:

(1) Input sizes are upper bounded by $\frac{q}{2}$.
(2) One input is of size, say w_i, greater than $\frac{q}{2}$, but less than q, and all the other inputs have size less than or equal to $q - w_i$. In this case most of the communication cost comes from having to pair the large input with every other input.

Of course, if the two largest inputs are greater than the given reducer capacity q, then there is no solution to the *A2A mapping schema problem* because these two inputs cannot be assigned to a single reducer in common.

Parameters for analysis. We analyze our approximation algorithms on the following parameters of the mapping schema created by those algorithms:

(1) *Number of reducers*. This is the number of reducers used by the mapping schema to send all inputs to.
(2) *The communication cost*, c. The communication cost is defined to be the sum of all the bits that are required, according to the mapping schema, to transfer from the map phase to the reduce phase.

Table I summarizes all the results in this paper. Before describing the algorithms, we look at lower bounds for the above parameters as they are expressed in terms of the reducer capacity q and sum of sizes of all inputs s.

THEOREM 4.1. (LOWER BOUNDS ON THE COMMUNICATION COST AND NUMBER OF REDUCERS) *For a list of inputs and a given reducer capacity q, the communication cost and the number of reducers, for the A2A mapping schema problem, are at least $\frac{s^2}{q}$ and $\frac{s^2}{q^2}$, respectively, where s is the sum of all the input sizes.*

Cases	Theorems	Communication cost	Approximation ratio
The **lower bounds** for the *A2A mapping schema problem*			
Different-sized inputs	4.1	$\frac{s^2}{q}$	
Equal-sized inputs	5.1	$m\lfloor\frac{m-1}{q-1}\rfloor$	
The **lower bounds** for the *X2Y mapping schema problem*			
Different-sized inputs	10.1	$\frac{2\cdot sum_x\cdot sum_y}{q}$	
Optimal algorithms for the *A2A mapping schema problem* (* equal-sized inputs)			
Algorithm for reducer capacity $q=2$	5.6	$m(m-1)$	optimal
Algorithm for reducer capacity $q=3$	5.6	$m\lfloor\frac{m-1}{2}\rfloor$	optimal
The *AU* method: When q is a prime number	5.6	$m\lfloor\frac{m-1}{q-1}\rfloor$	optimal
Non-optimal algorithms for the *A2A mapping schema problem* and their **upper bounds**			
Bin-packing-based algorithm, not including an input of size $>\frac{q}{2}$	4.5	$\frac{4s^2}{q}$	$\frac{1}{4}$
Algorithm 1	6.3	$\frac{q}{2k}\lceil\frac{sk}{q(k-1)}\rceil(\lceil\frac{sk}{q(k-1)}\rceil-1)$	$1/k-1$
Algorithm 2: The first extension of the *AU method*	7.1	$qp(p+1)+z'$	$q/(q+1)$
Algorithm 3: The second extension of the *AU method*	7.5	$q^2\times(q(q+1))^{l-1}$	$(q^l-1)/q(q-1)(q+1)^{l-1}$
Bin-packing-based algorithm considering an input of size $>\frac{q}{2}$	9.1	$(m-1)\cdot q+\frac{4s^2}{q}$	$\frac{s^2}{mq^2}$
A non-optimal algorithm for the *X2Y mapping schema problem* and their **upper bounds**			
Bin-packing-based algorithm, $q=2b$	10.2	$\frac{4\cdot sum_x\cdot sum_y}{b}$	$\frac{1}{4}$
Approximation ratio. The ratio between the optimal communication cost and the communication cost obtained from an algorithm. Notations: s: sum of all the input sizes. q: the reducer capacity. m: the number of inputs. sum_x: sum of input sizes of the list X. sum_y: sum of input sizes of the list Y. p: the nearest prime number to q. $l>2$. $k>1$.			

Table I: The bounds for heuristics for the *A2A* and the *X2Y* mapping schema problems.

PROOF. Since an input i is replicated to at least $\lfloor\frac{s-w_i}{q-w_i}\rfloor$ reducers, the communication cost for the input i is $w_i\times\lfloor\frac{s-w_i}{q-w_i}\rfloor$. Hence, the communication cost for all the inputs will be at least $\sum_{i=1}^{m} w_i\frac{s-w_i}{q-w_i}$. Since $s\geq q$, we can conclude $\frac{s-w_i}{q-w_i}\geq\frac{s}{q}$. Thus, the communication cost is at least $\sum_{i=1}^{m} w_i\frac{s}{q}=\frac{s^2}{q}$.

Since the communication cost, the number of bits to be assigned to reducers, is at least $\frac{s^2}{q}$, and a reducer can hold inputs whose sum of the sizes is at most q, the number of reducers must be at least $\frac{s^2}{q^2}$. □

4.1. Bin-packing-based Approximation

Our general strategy for building approximation algorithms is as follows: we use a known bin-packing algorithm to place the given m inputs to bins of size $\frac{q}{k}$, $k\geq 2$. Assume that we need x bins to place m inputs. Now, each of these bins is considered as a single input of size $\frac{q}{k}$ for our problem of finding an optimal mapping schema. Of course, the assumption is that all inputs are of size at most $\frac{q}{k}$, $k\geq 2$.

First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) [Coffman et al. 1997] are most notable bin-packing algorithms. FFD or BFD bin-packing algorithm ensures that all the bins (except only one bin) are at least half-full. There also exists a pseudo polynomial bin-packing algorithm, suggested by Karger and Scott [Karger and Scott 2008], that can place the m inputs in as few bins as possible of certain size.

Algorithms	Inputs
Non-optimal algorithms for the *A2A mapping schema problem*	
Bin-packing-based algorithm	Any number of inputs of any size
Algorithm 1	Any number of inputs of size at most $\frac{q}{k}$, $k > 3$
Algorithm 2: The first extension of the *AU method*	$p^2 + p \cdot l + l$, $p + l = q$, $l > 2$
Algorithm 3: The second extension of the *AU method*	q^l, $l > 2$ and q is a prime number
A non-optimal algorithm for the *X2Y mapping schema problem*	
Bin-packing-based algorithm, $> \frac{q}{2}$	Any number of inputs of any size
Notations: w_i and w_j: the two largest size inputs of a list. p: the nearest prime number to q. w_k: the largest input of a list X. w'_k: the largest input of a list Y.	

Table II: Reducer capacity and input constraints for different algorithms for the mapping schema problems.

Example 4.2. Let us discuss in more detail the case $k = 2$. In this case, since the reducer capacity is q, any two bins can be assigned to a single reducer. Hence, the approximation algorithm uses at most $\frac{x(x-1)}{2}$ reducers, where x is the number of bin; see Figure 5 for an example.

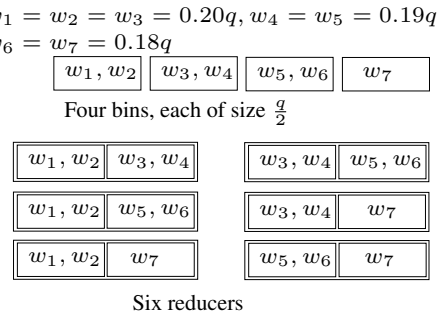

Fig. 5: Bin-packing-based approximation algorithm.

For this strategy a lower bound on communication cost depends also on k as follows:

THEOREM 4.3 (LOWER BOUND ON THE COMMUNICATION COST). *Let $q > 1$ be the reducer capacity, and let $\frac{q}{k}$, $k > 1$, is the bin size. Let the sum of the given inputs is s. The communication cost, for the A2A mapping schema problem, is at least $s\lfloor\frac{\frac{sk}{q}-1}{k-1}\rfloor$.*

PROOF. A bin can hold inputs whose sum of the sizes is at most $\frac{q}{k}$. Since the total sum of the sizes is s, it is required to divide the inputs into at least $x = \frac{sk}{q}$ bins. Now, each bin can be considered as an identical sized input.

Since a bin i is required to be sent to at least $\lfloor\frac{x-1}{k-1}\rfloor$ reducers (to be paired with all the other bins), the sum of the number of copies of (x) bins sent to reducers is at least $x\lfloor\frac{x-1}{k-1}\rfloor$. We need to multiply this by $\frac{q}{k}$ (the size of each bin) to find the communication cost. Thus, we have at least

$$x\left\lfloor\frac{x-1}{k-1}\right\rfloor\frac{q}{k} = \frac{sk}{q}\left\lfloor\frac{\frac{sk}{q}-1}{k-1}\right\rfloor\frac{q}{k} = s\left\lfloor\frac{\frac{sk}{q}-1}{k-1}\right\rfloor$$

communication cost. □

This communication cost in the above theorem, as expected, is larger than the one in Theorem 4.1, where no restriction in a specific strategy was taken into account.

Example 4.4. *Example for* $k = 2$. Let us apply our strategy to the case where $k = 2$, *i.e.*, we have the algorithm: (*i*) we do bin-packing to put the inputs in bins of size $\frac{q}{2}$; and (*ii*) we provide a mapping schema for assigning each pair of bins to at least one reducer. Such a schema is easy and has been discussed in the literature (*e.g.*, [Ullman 2012]).

FFD and BFD bin-packing algorithms provide an $\frac{11}{9} \cdot \text{OPT}$ approximation ratio [Johnson 1973], *i.e.*, if any optimal bin-packing algorithm needs OPT bins to place (m) inputs in the bins of a given size $\frac{q}{2}$, then FFD and BFD bin-packing algorithms always use at most $\frac{11}{9} \cdot \text{OPT}$ bins of an identical size (to place the given m inputs). Since we require at most $\frac{x(x-1)}{2}$ reducers for a solution to the *A2A mapping schema problem*, the algorithm requires at most $(\frac{11}{9} \cdot \text{OPT})^2/2$ reducers.

Note that, here in this case, OPT does not indicate the optimal number of reducers to assign m inputs that satisfy the *A2A mapping schema problem*; OPT indicates the optimal number of bins of size $\frac{q}{2}$ that are required to place m inputs.

The following theorem gives the upper bounds that this approximation algorithm achieves on the communication cost and the number of reducers.

THEOREM 4.5. (UPPER BOUNDS ON COMMUNICATION COST AND NUMBER OF REDUCERS FOR $k = 2$) *The above algorithm using a bin size $b = \frac{q}{2}$ where q is the reducer capacity achieves the following upper bounds: the number of reducers and the communication cost, for the A2A mapping schema problem, are at most $\frac{8s^2}{q^2}$, and at most $4\frac{s^2}{q}$, respectively, where s is the sum of all the input sizes.*

PROOF. A bin i can hold inputs whose sum of the sizes is at most b. Since the total sum of the sizes is s, it is required to divide the inputs into at least $\frac{s}{b}$ bins. Since the FFD or BFD bin-packing algorithm ensures that all the bins (except only one bin) are at least half-full, each bin of size $\frac{q}{2}$ has at least inputs whose sum of the sizes is at least $\frac{q}{4}$. Thus, all the inputs can be placed in at most $\frac{s}{q/4}$ bins of size $\frac{q}{2}$. Since each bin is considered as a single input, we can assign every two bins to a reducer, and hence, we require at most $\frac{8s^2}{q^2}$ reducers. Since each bin is replicated to at most $4\frac{s}{q}$ reducers, the communication cost is at most $\sum_{1 \le i \le m} w_i \times 4\frac{s}{q} = 4\frac{s^2}{q}$. □

5. EQUAL-SIZED INPUTS OPTIMAL ALGORITHMS

As we explained, looking at inputs of same size makes sense because we imagine the inputs are being bin-packed into bins of size $\frac{q}{k}$, for $k \ge 2$ (using bin-packing-based algorithm Section 4.1), and that once this is done, we can treat the bins themselves as things of unit size to be sent to the reducers. Thus, in this section, we will shift the notation so that all inputs are of unit size, and q is some small integer, *e.g.*, 3.

In this section, we provide optimal algorithms for $q = 2$ (in Section 5.1) and $q = 3$ (in Section 5.2). Afrati and Ullman [Afrati and Ullman 2013] provided an optimal algorithm for the *A2A mapping schema problem* where q is a prime number and the number of inputs is $m = q^2$. We extend this algorithm for $m = q^2 + q + 1$ inputs (in Section 5.3), and this extension also meets the lower bound on the communication cost. We will generalize these three algorithms in the Sections 6 and 7.

In this setting, by minimizing the number of reducers, we minimize communication, since each reducer is more-or-less filled to capacity. So we define

— $r(m, q)$ to be the minimum number of reducers of capacity q that can solve the all-pairs problem for m inputs.

The following theorem sets a lower bound on $r(m, q)$ and the communication cost for this setting.

THEOREM 5.1. (LOWER BOUNDS ON THE COMMUNICATION COST AND NUMBER OF REDUCERS) *For a given reducer capacity $q > 1$ and a list of m inputs, each input is of size one, the communication cost and the number of reducers ($r(m, q)$), for the A2A mapping schema problem, are at least $m\lfloor\frac{m-1}{q-1}\rfloor$ and at least $\lfloor\frac{m}{q}\rfloor\lfloor\frac{m-1}{q-1}\rfloor$, respectively.*

PROOF. Since an input i is required to be sent to at least $\lfloor\frac{m-1}{q-1}\rfloor$ reducers, the sum of the number of copies of (m) inputs sent to reducers is at least $m\lfloor\frac{m-1}{q-1}\rfloor$, which result in at least $m\lfloor\frac{m-1}{q-1}\rfloor$ communication cost.

There are at least $m\lfloor\frac{m-1}{q-1}\rfloor$ number of copies of (m) inputs to be sent to reducers and a reducer can hold at most q inputs; hence, $r(m, q) \geq \lfloor\frac{m}{q}\rfloor\lfloor\frac{m-1}{q-1}\rfloor$. □

5.1. Reducer Capacity $q = 2$

Here, we offer a recursive algorithm and show that this algorithm does not only obtain the bound $r(m, 2) \leq \frac{m(m-1)}{2}$, but it does so in a way that divides the reducers into $m - 1$ "teams" of $\frac{m}{2}$ reducers, where each team has exactly one occurrence of each input. We will use these properties of the output of this algorithm to build an algorithm for $q = 3$ in the next subsection.

The recursive algorithm. We are given a list A of m inputs. The intention is to have all pairs of inputs from list A partitioned into $m - 1$ teams with each team containing exactly $\frac{m}{2}$ pairs and each input appearing exactly once within a team. Hence, we will use $\frac{m(m-1)}{2}$ reducers for assigning pairs of each input.

We split A into two sublists A_1 and A_2 of size $\frac{m}{2}$ each. Suppose, we have the $\frac{m}{2} - 1$ teams for a list of size $\frac{m}{2}$. We will take the $\frac{m}{2} - 1$ teams of A_1, the $\frac{m}{2} - 1$ teams of A_2 and "mix them up" in a rather elaborate way to form the $m - 1$ teams for A:

Let the teams for A_1 and A_2 be $\{g_1, g_2, g_3, \ldots, g_{\frac{m}{2}}\}$ and $\{h_1, h_2, h_3, \ldots, h_{\frac{m}{2}}\}$ respectively. We will form two kind of teams, teams of kind I and teams of kind II as follows:

Teams of kind I. We will form $\frac{m}{2}$ teams of kind I by taking one input from A_1 and one input from A_2. For example, the first team for A is $\{(g_1, h_1), (g_2, h_2), (g_3, h_3), \ldots, (g_{\frac{m}{2}}, h_{\frac{m}{2}})\}$, the second team for A is $\{(g_1, h_2), (g_2, h_3), (g_3, h_4), \ldots, (g_{\frac{m}{2}}, h_1)\}$, and so on.

Teams of kind II. We will form the remaining $\frac{m}{2} - 1$ teams having $\frac{m}{2}$ reducers in each. In teams of kind I each pair (reducer) contains only inputs from one of the lists A_1 or A_2. Now we produce pairs, with each pair having both inputs from A_1 or A_2. In order to do that, we divide recursively divide A_1 into two sublists and perform the operation what we performed in the team of kind I. The same procedure is recursively implemented on A_2.

Example 5.2. For $m = 8$, we form 7 teams. First we form teams of kind I. We divide 8 inputs into two lists A_1 and A_2. After that, we take one input from A_1 and one input from A_2, and create 4 teams, see Figure 6. Now, we recursively follow the same rule on each sublist, A_1 and A_2, and create 3 remaining teams of kind II, see Figure 6.

1,5	1,6	1,7	1,8	1,3	1,4	1,2
2,6	2,7	2,8	2,5	2,4	2,3	3,4
3,7	3,8	3,5	3,6	5,7	5,8	5,6
4,8	4,5	4,6	4,7	6,8	6,7	7,8
Team 1	Team 2	Team 3	Team 4	Team 5	Team 6	Team 7
	Teams of kind I				Teams of kind II	

Fig. 6: The teams for $m = 8$ and $q = 2$.

Actually in Figure 7, the teams for this example are shown in non-bold face fonts (two in each triplet in Figure 7, notice that they are from 1-8) in teams 1 through 7 in Figure 7.

The following theorem is easy to prove.

THEOREM 5.3. *In each team an input appears only once. In each team all inputs appear. There are* $m-1$ *teams which is the minimum possible. Hence this is an optimal mapping scheme that assigns inputs to reducers.*

This works if the number of inputs is a power of two. We can use known techniques to make it work with good approximation in general.

5.2. Reducer Capacity $q = 3$

Here, we present an algorithm that constructs an optimal mapping schema for $q = 3$. Our recursive algorithm starts by taking the mapping schema constructed in previous subsection for $q = 2$. We showed there that for $q = 2$, we can not only obtain the bound $r(m, 2) \leq \frac{m(m-1)}{2}$, but that we can do so in a way that divides the reducers into $m-1$ teams of $\frac{m}{2}$ reducers in each team, where each team has exactly one occurrence of each input.

Now, we split m inputs into two disjoint sets: set A and set B. Suppose $m = 2n-1$. Set A has n inputs and set B has $n-1$ inputs. We start with the n inputs in set A, and create $n-1$ teams of $\frac{n}{2}$ reducers, each reducer getting two of the n inputs in A, by following the algorithm given in Section 5.1. Next, we add to all reducers in one team another input from set B. *I.e.*, in a certain team we add to all $\frac{n}{2}$ reducers of this team a certain input from set B, and thus, we form a triplet for each reducer.

Since there are $n-1$ teams, we can handle another $n-1$ inputs. This is the start of a solution for $q = 3$ and $m = 2n-1$ inputs. To complete the solution, we add the reducers for solving the problem for the $n-1$ inputs of the set B. That leads to the following recurrence

$$r(m,3) = \frac{n(n-1)}{2} + r(n-1,3), \text{ where } m = 2n-1$$
$$r(3,3) = 1$$

We solve the recurrence for m a power of 2, and it exactly matches the lower bound of $r(m,3) = \frac{m(m-1)}{6}$. Moreover, notice that we can prove that this case is optimal either by proving that $r(m,3) = m(m-1)/6$ (as we did above) or by observing that every pair of inputs meets exactly in one reducer. This is easy to prove. Hence the following theorem:

THEOREM 5.4. *This algorithm constructs an optimal mapping schema for the reducer capacity* 3.

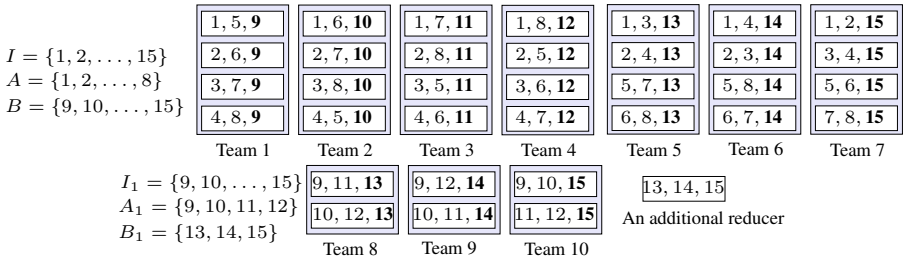

Fig. 7: An example of a mapping schema for $q = 3$ and $m = 15$.

Example 5.5. An example is shown in Figure 7. We explained how this figure is constructed for $q = 2$ (the non-bold entries). Now we use the algorithm just presented here to construct the 35 ($= 15 \times \frac{14}{6}$) reducers. We explain below in detail how we construct these 35 reducers.

We are given 15 inputs ($I = \{1, 2, \ldots, 15\}$). We create two sets, namely A of $y = 8$ inputs and B of $x = 7$ inputs, and arrange $(y-1) \times \lceil \frac{y}{2} \rceil = 28$ reducers in the form of 7 *teams* of 4 reducers in each team. These 7 teams assign each input of the set A with all other inputs of the set A and all the inputs in the set B as follows. We pair every two inputs of the set A and assign them to exactly one of 28 reducers as we explained in Section 5.1. Once every pair of $y = 8$ inputs of the set A is assigned to exactly one of 28 reducers, then we assign the i^{th} input of the set B to all the four reducers of $(i-8)^{th}$ team. Thus, *e.g.*, input 10 is assigned to the four reducers of Team 2.

Now these 28 reducers have seen that each pair of inputs from set A meet in at least one reducer and each pair of inputs, one from A and one from B meet in at least one reducer. Thus, it remains to build more reducers so that each pair of inputs (both) from set B meet. According to the recursion we explained, we break set B into sets A_1 and B_1, of size 4 and 3 respectively, and we apply our method again. In particular, we create two sets, $A_1 = \{9, 10, 11, 12\}$ of $y_1 = 4$ inputs and $B_1 = \{13, 14, 15\}$ of $x_1 = 3$. Then, we arrange $(y_1 - 1) \times \lceil \frac{y_1}{2} \rceil = 6$ reducers in the form of 3 *teams* of 2 reducers in each team. We assign each pair of inputs of the set A_1 to these 6 reducers, and then i^{th} input of the set B_1 to all the two reducers of a team, see Team 8 to Team 10.

The last team is constructed so that all inputs in B_1 meet at the same reducers (since B_1 has only 3 elements and 3 is the size of a reducer, one reducer suffices for this to happen).

Open problem. Now the interesting observation is that if we can argue that the resulting reducers can be divided into $\frac{m-1}{2}$ teams of $\frac{m}{3}$ reducers each (with each team having one occurrence of each input), then we can extend the idea to $q = 4$, and perhaps higher.

5.3. When q or $q-1$ is a prime number

An algorithm to provide a mapping schema for the reducer capacity q, where q is a prime number, and $m = q^2$ inputs is suggested by Afrati and Ullman in [Afrati and Ullman 2013]. This method meets the lower bounds on the communication cost. We call this algorithm the *AU method*. For the sake of completeness, we provide an overview of the *AU method*. Interested readers may refer to [Afrati and Ullman 2013].

The *AU method*. We divide the m inputs into q^2 equal-sized *subsets* (each with $\frac{m}{q^2}$ inputs) that are arranged in a $Q = q \times q$ square. The subsets in row i and column j are represented by $S_{i,j}$, where $0 \leq i < q$ and $0 \leq j < q$.

We now organize $q(q+1)$ reducers in the form of $q+1$ *teams* of q *players* (or reducers) in each team. Note that sum of sizes of the inputs in each row and column of the Q square is exactly q.

The teams are arranged from 0 to q, and the reducers are arranged from 0 to $q-1$. We first arrange inputs to the team q. Since the sum of the sizes in each column of the P square is q, we place one column of the P square to one reducer of the team q. Now we place the inputs to the remaining teams. We use modulo operation for the assignment of each subset to each team. The subset $S_{i,j}$ is assigned to a reducer r of each team t, $0 \leq t < q$, such that $(i + tj) modulo\ q = r$. An example for $q = 3$ and $m = 9$ is given in Figure 8.

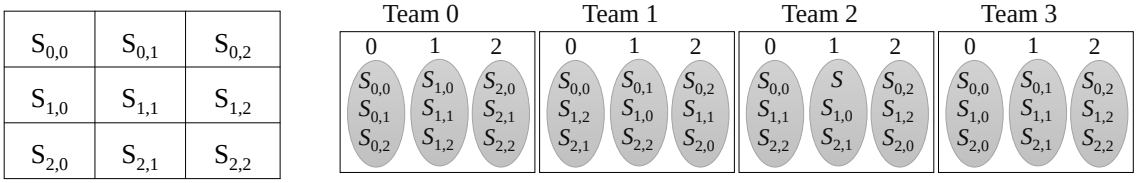

Fig. 8: The *AU method* for the reducer capacity $p = 3$ and $m = 9$.

Total required reducers. The *AU method* uses $q(q+1)$ reducers, which are organized in the form of $q+1$ teams of q reducers in each team, and the communication cost is $q^2(q+1)$.

A simple extension of the *AU method*. Now, we can extend the *AU method* as follows: we can add $q+1$ additional inputs, add one to each reducer and add one more reducer that has the $q+1$ new inputs. That gives us reducers of size $q = q+1$ and $m = q^2+q+1$, or $r(q^2+q+1, q+1) = q(q+1)+1 = q^2+q+1$. If you substitute $m = q^2+q+1$ and $p = p+1$, you can check that this also meets the bound of $r = \frac{m(m-1)}{q(q-1)}$. In Figure 9, we show a mapping schema for this extension to the *AU method* for $q = 4$ and $m = 14$.

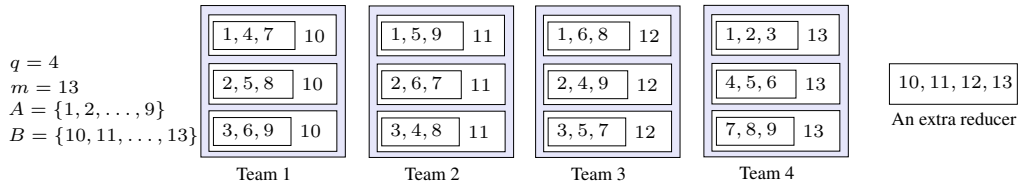

Fig. 9: An optimum mapping schema for $q = 4$ and $m = 14$ by extending the *AU method*.

In conclusion, in this section we have shown the following:

THEOREM 5.6. *We can construct optimal mapping schemas for the following cases:*

(*1*) $q = 2$.
(*2*) $q = 3$.
(*3*) q *being a prime number and* $m = q^2$.
(*4*) $q-1$ *being a prime number and* $m = (q-1)^2 + q$, *where* q *is the reducer capacity and* m *is the number of inputs.*

Open problem: Can we generalize the last idea to get optimal schemas for more cases?

Approximation Algorithms for the *A2A Mapping Schemas Problem*. We can use the optimal mapping schemas of Section 5 to construct good approximation of mappings schemas in many cases. The general techniques, we will use in this section move along the following dimensions/ideas:

— Assuming that there are no inputs of size greater than $\frac{q}{k}$, construct bins of size $\frac{q}{k}$, and treat each of the bins as a single input of size 1 and assume the reducer capacity is k. Then apply one of the optimal techniques of Section 5 to construct a mapping schema. These algorithms are presented in Sections 6 and 8.
— Getting inspiration from the methods developed (or only presented – in the case of the *AU method*) in Section 5.3, we extend the ideas to construct good approximation algorithms for inputs that are all of equal size (see Sections 7.1 and 7.2).

Thus, in Sections 6, 7, and 8, we will give several such techniques and show that some of them construct mapping schemas close to the optimal. To that end, we have already shown a schema based on bin-packing algorithms in Section 4.1.

6. GENERALIZING THE TECHNIQUE FOR THE REDUCER CAPACITY $Q > 3$ AND INPUTS OF SIZE $\leq Q/K$, $K > 3$

In this section, we will generalize the algorithm for $q = 3$ given in Section 5.2 and present an algorithm (Algorithm 1) for inputs of size less than or equal to $\frac{q}{k}$ and $k > 3$. For simplicity, we assume that k divides q evenly throughout this section.

6.1. Algorithm 1A

We divide Algorithm 1 into two parts based on the value of k as even or odd. Algorithm 1A considers that k is an odd number. Pseudocode of Algorithm 1A is given in Appendix B. Algorithm 1A works as follows:

First places all the given inputs, say m', to some bins, say m, each of size $\frac{q}{k}$, $k > 3$ is an odd number. Thus, a reducer can hold an odd number of bins. After placing all the m' inputs to m bins, we can treat each of the m bins as a single input of size one and the reducer capacity to be k. Now, it is easy to turn the problem to a case similar to the case of $q = 3$. Hence, we divide the m bins into two sets A and B, and follow a similar approach as given in Section 5.2.

Aside. Equivalently, we can consider q to be odd and the inputs to be of unit size. In what follows, we will continue to use q, which is an odd number, as the reducer capacity and assume all inputs (that are actually bins containing inputs) are of unit size.

Example 6.1. If $q = 30$ and $k = 5$, then we can pack given inputs to some bins of size 6. Hence, a reducer can hold 5 bins. Equivalently, we may consider each of the bins as a single input of size 1 and $q = 5$.

For understanding of Algorithm 1A, an example for $q = 5$ is presented in Figure 10, where we obtain $m = 23$ bins (that are considered as 23 unit-sized inputs) after implementing a bin-packing algorithm to given inputs.

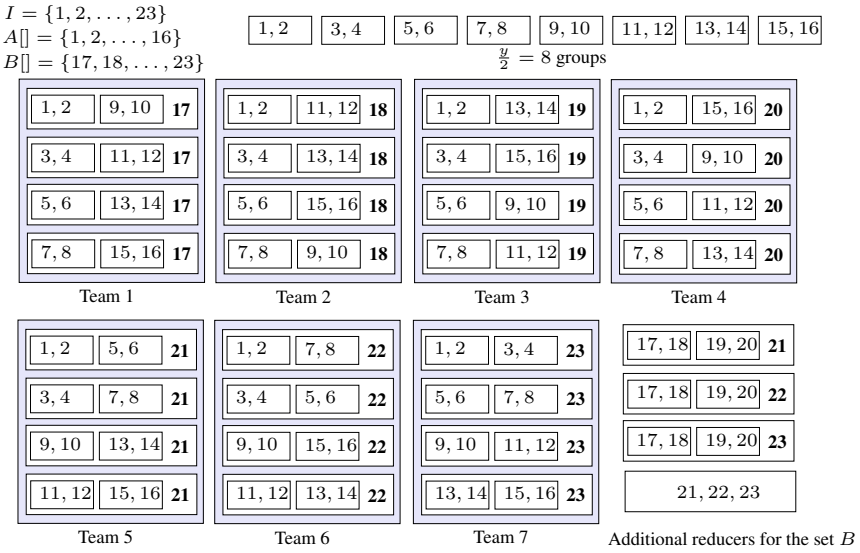

Fig. 10: Algorithm 1A – an example of a mapping schema for $q = 5$ and 23 bins.

Algorithm 1A consists of six steps as follows:

(1) *Implement a bin-packing algorithm*: Implement a bin-packing algorithm to place all the given m' inputs to bins of size $\frac{q}{k}$, where $k > 3$ is an odd number and the size of all the inputs is less than or equal to $\frac{q}{k}$. Let m bins are obtained, and now each of the bins is considered as a single input.

(2) *Division of bins (or inputs) to two sets, A and B*: Divide m inputs into two sets A and B of size $y = \lfloor \frac{q}{2} \rfloor (\lfloor \frac{2m}{q+1} \rfloor + 1)$ and $x = m - y$, respectively.

(3) *Grouping of inputs of the set A*: Group the y inputs into $u = \lceil \frac{y}{q - \lceil q/2 \rceil} \rceil$ disjoint groups, where each group holds $\lceil \frac{q-1}{2} \rceil$ inputs. (We consider each of the u $(= \lceil \frac{y}{q - \lceil q/2 \rceil} \rceil)$ disjoint

groups as a single input that we call the *derived input*. By making u disjoint groups[2] (or derived inputs) of y inputs of the set A, we turn the case of any odd value of q to a case where a reducer can hold only three inputs, the first two inputs are pairs of the derived inputs and the third input is from the set B.)

(4) *Assigning groups (inputs of the set A) to some reducers*: Organize $(u-1) \times \lceil \frac{u}{2} \rceil$ reducers in the form of $u-1$ teams of $\lceil \frac{u}{2} \rceil$ reducers in each team. Assign every two groups to one of $(u-1) \times \lceil \frac{u}{2} \rceil$ reducers. To do so, we will prove the following Lemma 6.2.

LEMMA 6.2. *Let q be the reducer capacity. Let the size of an input is $\lceil \frac{q-1}{2} \rceil$. Each pair of $u = 2^i$, $i > 0$, inputs can be assigned to $2^i - 1$ teams of 2^{i-1} reducers in each team.*[3]

(5) *Assigning inputs of the set B) to the reducers*: Once every pair of the derived inputs are assigned, then assign i^{th} input of the set B to all the reducers of i^{th} team.

(6) *Use previous steps on the inputs of the set B*: Apply (the above mentioned) steps 1-4 on the set B until there is a solution to the *A2A mapping schema problem* for the x inputs.

THEOREM 6.3 (THE COMMUNICATION COST OBTAINED USING ALGORITHM 1). *For a given reducer capacity $q > 1$, $k > 3$, and a list of m inputs whose sum of sizes is s, the communication cost, for the A2A mapping schema problem, is at most $\frac{q}{2k} \lceil \frac{sk}{q(k-1)} \rceil (\lceil \frac{sk}{q(k-1)} \rceil - 1)$.*

PROOF. Since the FFD or BFD bin-packing algorithm ensures that all the bins (except only one bin) are at least half-full, each bin of size $\frac{q}{k}$ has at least inputs whose sum of the sizes is at least $\frac{q}{k/2}$. Thus, all the inputs can be placed in at most $x = s/(q/(k/2)) = \frac{sk}{2q}$ bins of size $\frac{q}{k}$. Now, each bin can be considered as an identical sized input.

According to the construction given in Algorithm 1A, there are at most $g = \lceil \frac{2x}{k-1} \rceil$ groups (derived inputs) of the given x bins. In order to assign each pair of the derived inputs, each derived input is required to assign to at most $g-1$ reducers. In addition, the size of each input (bin) is $\frac{q}{k}$, therefore we have at most

$$\frac{q}{k} \times g(g-1)/2 = \frac{q}{k} \times \left\lceil \frac{2x}{k-1} \right\rceil \left(\left\lceil \frac{2x}{k-1} \right\rceil - 1 \right)/2$$

$$= \frac{q}{2k} \times \left\lceil \frac{sk}{q(k-1)} \right\rceil \left(\left\lceil \frac{sk/q}{k-1} \right\rceil - 1 \right) > \frac{4s^2}{q}$$

communication cost. □

Algorithm correctness. The algorithm correctness appears in Appendix B.

Approximation factor. The optimal communication cost (from Theorem 4.3) is $s\lfloor (\frac{sk}{q} - 1)/k - 1 \rfloor \approx \frac{s^2}{q} \cdot \frac{k}{k-1}$ and the communication cost of the algorithm (from Theorem 6.3) is $\frac{q}{2k} \lceil \frac{sk}{q(k-1)} \rceil (\lceil \frac{sk}{q(k-1)} \rceil - 1) \approx s^2 k/q(k-1)^2$. Thus, the ratio between the optimal communication and the communication of our mapping schema is approximately $\frac{1}{k-1}$.

[2]We suppose that u is a power of 2. In case u is not a power of 2 and $u > q$, we add dummy inputs each of size $\lceil \frac{q-1}{2} \rceil$ so that u becomes a power of 2. Consider that we require d dummy inputs. If groups of inputs of the set B each of size $\lceil \frac{q-1}{2} \rceil$ are less than equal to d dummy inputs, then we use inputs of the set B in place of dummy inputs, and the set B will be empty.

[3]The proof appears in Appendix A.

6.2. Algorithm 1B

For the sake of completeness, we include the pseudocode of the algorithm for handling the case when k is an even number. We call it Algorithm 1B and pseudocode is given in Appendix C. In this algorithm, we are given m' inputs of size less than or equal to $\frac{q}{k}$ and $k \geq 4$ is an even number.

Similar to Algorithm 1A, Algorithm 1B first places all the m' inputs to m bins, each of size $\frac{q}{k}$, $k > 2$ is an even number. Thus, a reducer can hold an even number of bins. After placing all the m' inputs to m bins, we can treat each of the m bins as a single input of size one and the reducer capacity to be k. Now, we easily turn this problem to a case similar to the case of $q = 2$. Hence, we divide the m bins into two set A and B, and follow a similar approach as given in Section 5.1.

Example 6.4. If $q = 30$ and $k = 6$, then we can pack given inputs to some bins of size 5. Hence, a reducer can hold 6 bins. Equivalently, we may consider each of the bins as a single input of size 1 and $q = 6$.

Note. Algorithms 1A and 1B are based on a fact that how do we pack inputs in a *well* manner to bins of even or odd size. To understand this point, consider $q = 30$ and $m' = 46$. For simplicity, we assume that all the inputs are of size three. Now, consider $k = 5$, so we will use 23 bins each of size 6 and apply Algorithm 1A. On the other, consider $k = 6$, so we will use 46 bins each of size 5 and apply Algorithm 1B.

7. GENERALIZING THE *AU METHOD*

In this section, we extend the *AU method* (Section 5.3) to handle more than q^2 inputs, when q is a prime number, Algorithms 3 and 4. Recall that the *AU method* can assign each pair of q^2 inputs to reducers of capacity q. We provide two extensions: (*i*) take $m = p^2 + p \cdot l + l$ identical-sized inputs and assign these inputs to reducers of capacity $p + l = q$, where p is the nearest prime number to q, in Section 7.1, and (*ii*) take $m = q^l$ inputs, where $l > 2$, and assign inputs to reducers of capacity q, in Section 7.2.

7.1. When we consider the nearest prime to q

We provide an extension to the *AU method* that handles $m = p^2 + p \cdot l + l$ identical-sized inputs and assigns them to reducers of capacity $p + l = q$, where p is the nearest prime number to q. We call it the *first extension to the AU method* (Algorithm 2).

Algorithm 2: The *First Extension of the AU method*. We extend the *AU method* by increasing the reducer capacity and the number of inputs. Consider that the *AU method* assigns p^2 identical-sized inputs to reducers of capacity p, where p is a prime number. We add $l(p + 1)$ inputs and increase the reducer capacity to $p + l$ $(= q)$.

In other words, m identical-sized inputs and the reducer capacity q are given. We select a prime number, say p, that is near most to q such that $p + l = q$ and $p^2 + l(p + 1) \leq m$. Also, we divide the m inputs into two disjoint sets A and B, where A holds at most p^2 inputs and B holds at most $l(p + 1)$ inputs.

Algorithm 2 consists of six steps, where m inputs and the reducer capacity q are inputs to Algorithm 2, as follows:

(1) Divide the given m inputs into two disjoint sets A of $y = p^2$ inputs and B of $x = m - y$ inputs, where p is the nearest prime number to q such that $p + l = q$ and $p^2 + l(p + 1) \leq m$.
(2) Perform the *AU method* on the inputs of the set A by placing y inputs to $p + 1$ teams of p bins in each team, where the size of each bin is p.
(3) Organize $p(p+1)$ reducers in the form of $p+1$ teams of p reducers in each teams, and assign j^{th} bin of i^{th} team of bins to j^{th} reducer of i^{th} team of reducers.
(4) Group the x inputs of the set B into $u = \lceil \frac{x}{q-p} \rceil$ disjoint groups.
(5) Assign i^{th} group to all the reducers of i^{th} team.

(6) Use Algorithm 1A or Algorithm 1B to make each pair of inputs of the set B, depending on the case of the value of q, which is either an odd or an even number, respectively.

Note that when we perform the above mentioned step 3, we assign each pair of inputs of the set A to $p(p+1)$ reducers, and such an assignment uses p capacity of each reducer. Now, each of $p(p+1)$ reducers has $q-p$ remaining capacity that is used to assign i^{th} group of inputs of the set B. In this manner, all the inputs of the set A are assigned with all the m inputs.

Algorithm correctness. The algorithm correctness appears in Appendix D.

THEOREM 7.1 (THE COMMUNICATION COST OBTAINED USING ALGORITHM 2). *Algorithm 2 requires at most $p(p+1)+z$ reducers, where $z = \frac{2l^2(p+1)^2}{q^2}$, and results in at most $qp(p+1)+z'$ communication cost, where $z' = \frac{2l^2(p+1)^2}{q}$, q is the reducer capacity, and p is the nearest prime number to q.*

When $l = q-p$ equals to one, we have provided an extension of the *AU method* in Section 5.3, and in this case, we have an optimum mapping schema for q and $m = q^2+q+1$ inputs.

PROOF. In case of $l > 1$, a single reducer cannot be used to assign all the inputs of the set B. Since Algorithm 2 is based on the *AU method*, Algorithm 1A, and Algorithm 1B, we always use at most $p(p+1)+z$ reducers, where z ($= \frac{2l^2(p+1)^2}{q^2}$) reducers are used to assign each pair of inputs of the set B based on Algorithms 1A or 1B (for the value of z, the reader may refer to Theorem 11 of the technical report [Afrati et al. 2015]). Thus, the communication cost is at most $qp(p+1)+z'$, where z' ($= \frac{2l^2(p+1)^2}{q}$) is the maximum communication cost required by Algorithm 1A or 1B for assigning $(p+1)l$ inputs of the set B. □

Approximation factor. The optimal communication cost using the *AU method* is $q^2(q+1)$. Thus, the difference between the communication of our mapping schema ($q^2(q+1)+z'$, when assuming p is equal to q) and the optimal communication is z'. We can see two cases, as follows:

(1) When q is large. Consider that q is greater than square or cube of the maximum difference between any two prime numbers. In this case, z' will be very small, and we will get almost optimal ratio.
(2) When q is very small. In this case, then z' plays a role as follows: here, the number of inputs in the set B will be at most $(p+1)l < q^2$. Thus, the ratio becomes $q/(q+1)$.

7.2. For input size $m = q^l$ where q is a prime number

We also provide another extension to the *AU method* that handles $m = q^l$ identical-sized inputs and assigns them to reducers of capacity q, where q is a prime number and $l > 2$. We call it the *second extension to the AU method* (Algorithm 3).

Algorithm 3: The *Second Extension of the AU method.* The second extension to the *AU method* (Algorithm 3) handles a case when $m = q^l$, where $l > 2$ and q is a prime number. We present Algorithm 3 for $m = q^l$, $l > 2$, inputs and the reducer capacity q, where q is a prime number. Nevertheless, m inputs that are less than but close to q^l can also be handled by Algorithm 3 by adding dummy inputs such that $m = q^l$, $l > 2$.

Algorithm 3 consists of two phases, as follows:

The first phase: creation of a bottom up tree. Here, we present a simple example for the bottom-up tree's creation for $q = 3$ and $m = 3^4$; see Figure 11.

Example 7.2 (*Bottom-up tree creation*). A bottom-up tree for $m = q^l = 3^4$ identical-sized inputs and $q = 3$ is given in Figure 11. Here, we explain how we constructed it.

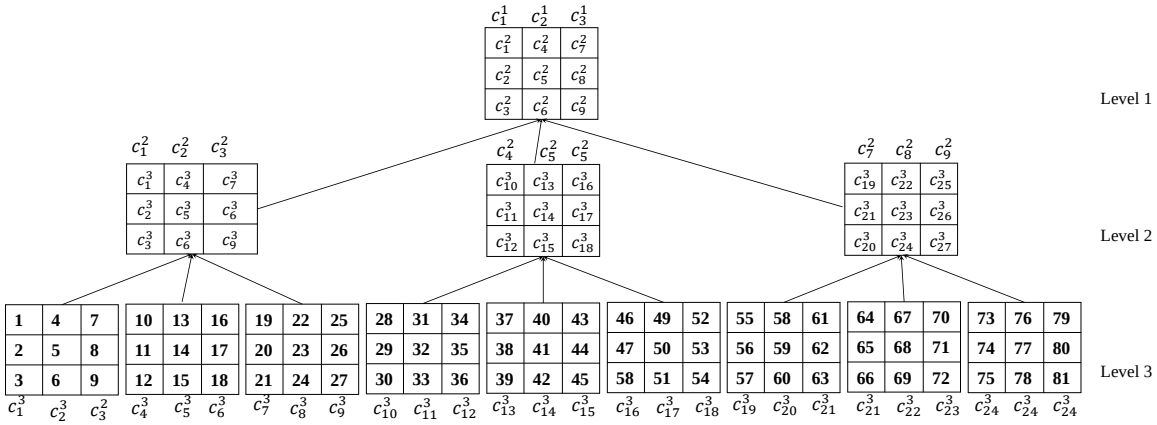

Fig. 11: The *second extension of the AU method* (Algorithm 3): Phase 1 – Creation of the bottom-up tree.

The height of the bottom up tree is $l-1$, and the last $(l-1)^{th}$ level has m inputs in the form of $\frac{m}{q^2}$ matrices of size $q \times q$. Note that we have $\frac{m}{q}$ columns at the last level, which holds m inputs; and these $\frac{m}{q}$ columns are called the *input columns*. We create the tree in bottom-up fashion, where $(l-2)^{th}$ level has $\frac{m}{q^3}$ matrices, whose each cell value refers to a *input column* of $(l-1)^{th}$ level. We use a notation to refer a column of i^{th} level by c^i_j, where j is column index. Note that each column, c^i_j, at level i holds q columns $(c^{i+1}_{(j-1)q+1}, c^{i+1}_{(j-1)q+2}, \ldots c^{i+1}_{jq})$ of $(i+1)^{th}$ level. In general, there are $\frac{m}{q^{l-i+2}}$ matrices at level i, whose each cell value, c^i_j, refers to a column, c^{i+1}_j, of $(i+1)^{th}$ level.

Following that the bottom-up tree for $m = 3^4$ identical-sized inputs and $q = 3$ has height 3. The last level ($(l-1)^{th} = 3^{rd}$) has 81 inputs in the form of $\frac{m}{q^2} = 9$ matrices of size 3×3. Note that we have $\frac{m}{q} = 24$ columns at 3^{rd} level; called the *input columns*. The $l-2 = 2^{ed}$ level has $\frac{m}{q^3} = 3$ matrices, whose each column, c^2_j, refers to $q = 3$ columns $(c^3_{(j-1)q+1}, c^3_{(j-1)q+2}, \ldots c^3_{jq})$ of 3^{rd} level. Further, the root node is at level 1, whose each column, c^1_j, refers to $q = 3$ columns $(c^2_{(j-1)q+1}, c^2_{(j-1)q+2}, \ldots c^2_{jq})$ of 2^{ed} level.

The second phase: creation of an assignment tree. The assignment tree is created in top-down fashion. Our objective is to assign each pair of inputs to a reducer, where inputs are arranged in the *input columns* of the bottom-up tree. If we can assign each pair of *input columns* (of the bottom-up tree) in the form of $(q \times q)$-sized matrices, then the implementation of the *AU method* on each such matrices results in an assignment of every pair of inputs to reducers. Hence, we try to make pairs of all the *input column*s by creating a tree called the *assignment tree*.

Here, we present a simple assignment tree for $m = 3^4$ and $q = 3$ (see Figure 12).

Example 7.3 (*Assignment tree creation*). The root node of the bottom-up tree becomes the root node the assignment tree. Recall that the root node of the bottom-up tree is a $q \times q$ matrix. First, consider the root node to understand the working of the *AU method* to create the assignment tree. Consider that each cell value of the root node matrix is of size one, and we have $(q+1)$ teams of q bins (of size q) in each team. Our objective to use the *AU method* on the root node matrix is to assign each pair of cell values ($\langle c^2_x, c^2_y \rangle$) in $q(q+1)$ bins that results in an assignment of every pair of cell values $\langle c^2_x, c^2_y \rangle$ at a bin.

Now, we create matrices by using these bins (the bins created by the *AU method*'s implementation on the root node) that are holding the indices of columns of the second level

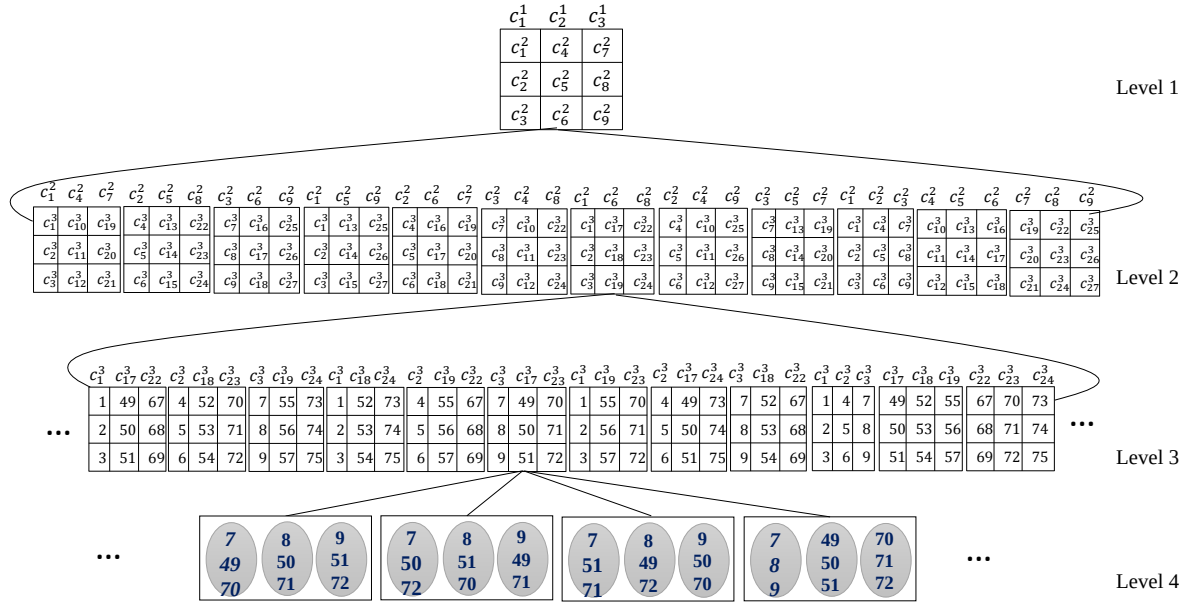

Fig. 12: The *second extension of the AU method* (Algorithm 3): Phase 2 – Creation of the assignment tree.

(c_x^2) of the bottom-up tree. We take each bin and its q indices $c_j^2, c_{j+1}^2, \ldots c_{j+q}^2$. We replace each c_j^2 with q columns as: $c_{(j-1)q+1}^3, c_{(j-1)q+2}^3, \ldots c_{jq}^3$ that results in $q(q+1)$ matrices of size $q \times q$, and these $q(q+1)$ matrices become child nodes of the root node. Now, we consider each such matrix separately and perform a similar operation as we did for the root node.

In this manner, the *AU method* creates $(q(q+1))^{i-1}$ child nodes (that are matrices of size $q \times q$) at i^{th} level of the assignment tree, and they create $(q(q+1))^i$ child nodes (matrices of size $q \times q$) at $(i+1)^{th}$ level of the assignment tree.

Recall that there are $\frac{m}{q}$ *input column*s at $(l-1)^{th}$ level of the bottom-up tree that hold the original m inputs. The implementation of the *AU method* on each node ($q \times q$-sized) matrix of $(l-2)^{th}$ level of the assignment tree assigns each pair of *input columns* at $(l-1)^{th}$ level of the assignment tree. Further the *AU method*'s implementation on each matrix of $(l-1)^{th}$ level assigns every pairs of the original inputs to $q^l \times (q+1)^{l-1}$ reducers at l^{th} level, which have reducers in the form of $(q(q+1))^{l-1}$ teams of q reducers in each team.

For $m = 3^4$ identical-sized inputs and $q = 3$, we take the root node of the bottom-up tree (Figure 11) that becomes the root node of the assignment tree. We implement the *AU method* on the root node and assign each pair of cell values (c_j^2, $1 \leq j \leq 9$) to a bin of size q. Each cell value of the bins (c_j^2) is then placed by $q = 3$ columns $c_{(j-1)q+1}^3, c_{(j-1)q+2}^3, \ldots c_{jq}^3$ that results in an assignment of each pair of columns of the second level of the bottom-up tree. For clarity, we are not showing bins. For the next 3^{rd} level, we again implement the *AU method* on all 12 matrices at 2^{nd} level and get 144 matrices at the third level. The matrices at 3^{rd} level are pairs of each *input columns* (of the bottom-up tree). The *AU method*'s implementation on each matrix of 3^{rd} level assigns each pair of original inputs to reducers. For clarity, we are only showing all the matrixes and teams at levels 3 and 4, respectively.

The assignment tree uses the root node of the bottom up tree, and we implement the *AU method* on the root node that results in $q(q+1)$ child nodes at level two. Each child node is a $q \times q$ matrix, and the columns of all the $q(q+1)$ matrices provide all-pairs of the cell values of the root node matrix. At level i, the assignment tree has $(q(q+1))^{i-1}$ nodes, see Figure 13. The height of the assignment tree is l, where $(l-1)^{th}$ level has all-pairs of *input column*s and l^{th} level has a solution to the *A2A mapping schema problem* for m inputs.

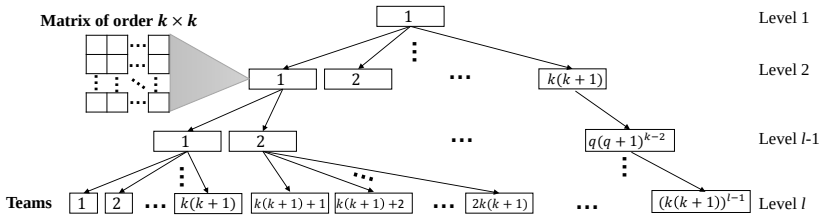

Fig. 13: An assignment tree created using Algorithm 3.

Algorithm correctness. Algorithm 3 satisfies the following Lemma 7.4:

LEMMA 7.4. *The height of the assignment tree is l, and l^{th} level of the assignment tree assigns each pairs of inputs to reducers.*

THEOREM 7.5 (THE COMMUNICATION COST OBTAINED USING ALGORITHM 3). *Algorithm 3 requires at most $q \times (q(q+1))^{l-1}$ reducers and results in at most $q^2 \times (q(q+1))^{l-1}$ communication cost, where q is the reducer capacity and $l > 2$.*

PROOF. For a given $m = q^l$, $l > 2$, the assignment tree has height l (Lemma 7.4), and (according to Algorithm 3) l^{th} level has $q \times (q(q+1))^{l-1}$ reducers providing an assignment of each pairs of inputs. Hence, Algorithm 3 uses $q(q(q+1))^{l-1}$ reducers, and the communication cost is at most $q^2 \times (q(q+1))^{l-1}$. □

Approximation factor. The optimal communication is $\frac{m(m-1)}{q-1}$ (see Theorem 5.1). Replacing m with q^l we get $q^l(q^l-1)/(q-1)$. Thus, the ratio between the optimal communication and the communication of our mapping schema is $(q^l-1)/q(q-1)(q+1)^{l-1}$. We can see two cases:

(1) When q is large. Then we drop the constant 1 and the ratio is approximately equal to $\frac{1}{q}$.
(2) When q is very small compared to q^l. Then the ratio is $q^l/q(q-1)(q+1)^{l-1}$.
For $q = 5$, the inverse of the ratio is approximately $(6/5)^{l-1}$. This is already acceptable for practical applications if we think that the size of data is 5^l, thus l may as well be $l = 9$, in which case this ratio is approximately 4.3. For $q = 2$ and $q = 3$ we already have optimal mappings schemas. Our conjecture is that there are optimal schemas for $q = 4$ and $q = 5$ even by using the techniques developed and presented here.

Open problem: In this section, we provided two algorithms for two different cases extending the *AU method*. However, this is an open problem of finding good approximation algorithms for the subcases that are not covered here.

8. A HYBRID ALGORITHM FOR THE *A2A MAPPING SCHEMA PROBLEM*

In the previous sections, we provide algorithms for different-sized and almost equal-sized inputs. The hybrid approach considers both different-sized and almost equal-sized inputs together. The objective of the hybrid approach is to place inputs to two different-sized bins, and then consider each of the bins as a single input.

Specifically, the hybrid approach uses the previously given algorithms (bin-packing-based approximation algorithm) and Algorithms 1A, 1B, 2, 3. We divide the given m inputs into two disjoint sets according to their input size, and then use the bin-packing-based approximation algorithm and Algorithms 1A, 1B, 2, or 3 depending on the size of inputs.

Algorithm 4. We divide m inputs into two sets A that holds the input i of size $\frac{q}{3} < w_i \leq \frac{q}{2}$, and B holds all the inputs of sizes less than or equal to $\frac{q}{3}$. Algorithm 4 consists of four steps, as follows:

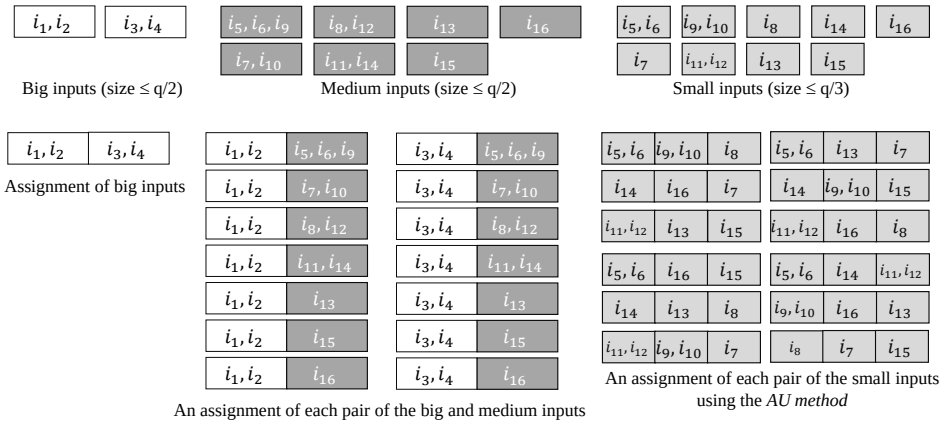

Fig. 14: An example to show the working of Algorithm 4. We are given 15 inputs, where inputs i_1 to i_4 are of sizes greater than $\frac{q}{3}$, and all the other inputs are of sizes less than or equal to $\frac{q}{3}$.

(1) Use the bin-packing-based approximation algorithm to place all the inputs of:
   (a) the set A to bins of size $\frac{q}{2}$, and each such bin is considered as a single input of size $\frac{q}{2}$ that we call the *big input*. Consider that x big inputs are obtained.
   (b) the set B twice, first to bins of size $\frac{q}{2}$, where each bin is considered as a single input of size $\frac{q}{2}$ that we call the *medium input*, and second, to bins of size $\frac{q}{3}$, where each bin is also considered as a single input of size $\frac{q}{3}$ that we call the *small input*. Consider that y medium and z small inputs are obtained.
(2) Use $\frac{x(x-1)}{2}$ reducers to assign each pair of big inputs.
(3) Use $x \times y$ reducers to assign each big input with each medium input.
(4) Use the *AU method*, Algorithm 1, 2, or 3 on the z small inputs, depending on the case, to assign each pair of small inputs.

We present an example to illustrate Algorithm 4 in Figure 14. Note that the use of $\frac{x(x-1)}{2}$ reducers assigns each pair of original inputs whose size between $\frac{q}{3}$ and $\frac{q}{2}$. Also by using $x \times y$ reducers, we assign each big input (or original inputs whose size is between $\frac{q}{3}$ and $\frac{q}{2}$) with each original input whose size is less than $\frac{q}{3}$. Further, the *AU method*, Algorithm 1, 2, or 3 assigns each pair of original inputs whose size is less than or equal to $\frac{q}{3}$.

Algorithm correctness. The algorithm correctness shows that every pair of inputs is assigned to reducers. Specifically, the algorithm correctness shows that each pair of the big inputs is assigned to reducers, each of the big inputs is assigned to reducers with each of the medium inputs, and each pair of the small inputs is assigned to reducers.

9. APPROXIMATION ALGORITHMS FOR THE *A2A MAPPING SCHEMA PROBLEM* WITH AN INPUT $> Q/2$

In this section, we consider the case of an input of size w_i, $\frac{q}{2} < w_i < q$; we call such an input as a *big input*. Note that if there are two big inputs, then they cannot be assigned to a single reducer, and hence, there is no solution to the *A2A mapping schema problem*. We assume m inputs of different sizes are given. There is a big input and all the remaining $m-1$ inputs, which we call the *small inputs*, have at most size $q - w_i$. We consider the following three cases in this section:

(1) The big input has size w_i, where $\frac{q}{2} < w_i \leq \frac{2q}{3}$,
(2) The big input has size w_i, where $\frac{2q}{3} < w_i \leq \frac{3q}{4}$,
(3) The big input has size w_i, where $\frac{3q}{4} < w_i < q$.

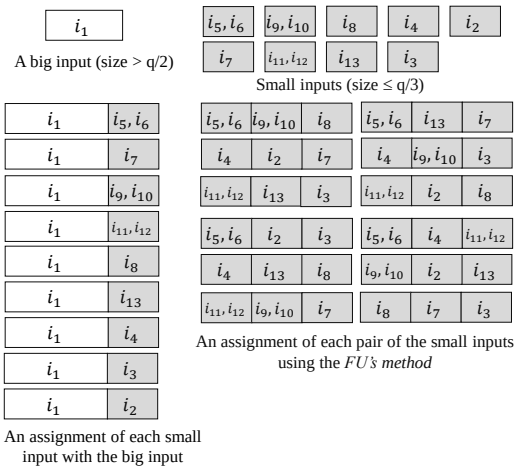

Fig. 15: An example to show an assignment of a big input of size $\frac{q}{2} < w_i \leq \frac{2q}{3}$ with all the remaining inputs of sizes less than or equal to $\frac{q}{3}$.

The communication cost is dominated by the big input. We consider three different cases of the big input to provide efficient algorithms in terms of the communication cost, where the first two cases can assign inputs to almost an optimal number of reducers, which results in almost minimum communication cost. We use the previously given algorithms (bin-packing-based approximation algorithm) and Algorithms 1-4 to provide a solution to the *A2A mapping schema problem* for the case of a big input.

A *simple solution* is to use FFD or BFD bin-packing algorithm to place the small inputs to bins of size $q - w_i$. Now, we consider each of the bins as a single input of size $q - w_i$. Let x bins are used. We assign each of the x bins to one reducer with a copy of the big input. Further, we assign the small inputs to bins of size $\frac{q}{2}$, and consider each of such bins as a single input of size $\frac{q}{2}$. Now, we can assign each pair of bins (each of size $\frac{q}{2}$) to reducers. In this manner, each pair of inputs is assigned to reducers.

The big input of size $\frac{q}{2} < w_i \leq \frac{2q}{3}$. In this case, we assume that the small inputs have at most $\frac{q}{3}$ size. We use First-Fit Decreasing (FFD) or Best-Fit Decreasing (BFD) bin-packing algorithm, the *AU method* (Section 5.3), and Algorithms 2, 3 (Section 7). We proceed as follows:

(1) First assign the big input with the small inputs.
   (a) Use a bin-packing algorithm to place the small inputs to bins of size $\frac{q}{3}$. Now, we consider each of the bins as a single input of size $\frac{q}{3}$.
   (b) Consider that x bins are used. Assign each of the bins to one reducer with a copy of the big input.
(2) Depending on the number of bins, we use the *AU method*, and Algorithms 2, 3 to assign each pair of the small inputs to reducers.

An example is given in Figure 15, where we place the small inputs to 9 bins of size $\frac{q}{3}$ and assign each of the bins to one reducer with a copy of the big input. Further, we implement the *AU method* on 9 bins to assign each pair of the small inputs.

The big input of size $\frac{2q}{3} < w_i \leq \frac{3q}{4}$. In this case, we assume that the small inputs have at most $\frac{q}{4}$ size. We use a bin-packing algorithm and Algorithms 1B (Sections 6). We proceed as follows:

(1) First assign the big input with the small inputs.
   (a) Use a bin-packing algorithm to place the small inputs to bins of size $\frac{q}{4}$.

(b) Consider that x bins are used. Assign each of the bins to one reducer with a copy of the big input.

(2) Depending on the number of bins, we use Algorithm 1B to assign each pair of small inputs.

The big input of size $\frac{3q}{4} < w_i < q$. In this case, we assume that the small inputs have at most $q - w_i$ size. In this case, we use a bin-packing algorithm and place the small inputs to bins of size $q - w_i$. We then place each of the bins to one reducer with a copy of the big input. Note that, we have not assigned each pair of small inputs. In order to assign each pair of small inputs, we use the bin-packing-based approximation algorithm (Section 4.1) or Algorithms 1-4 depending on size of the small inputs.

THEOREM 9.1 (UPPER BOUNDS FROM ALGORITHM). *For a list of m inputs where a big input, i, of size $\frac{q}{2} < w_i < q$ and for a given reducer capacity q, $q < s' < s$, an input is replicated to at most $m - 1$ reducers for the A2A mapping schema problem, and the number of reducers and the communication cost are at most $m - 1 + \frac{8s^2}{q^2}$ and $(m-1)q + \frac{4s^2}{q}$, respectively, where s' is the sum of all the input sizes except the size of the big input and s is the sum of all the input sizes.*

PROOF. The big input i can share a reducer with inputs whose sum of the sizes is at most $q - w_i$. In order to assign the input i with all the remaining $m - 1$ small inputs, it is required to assign a sublist of $m - 1$ inputs whose sum of the sizes is at most $q - w_i$. If all the small inputs are of size almost $q - w_i$, then a reducer can hold the big input and one of the small inputs. Hence, the big input is required to be sent to at most $m - 1$ reducers that results in at most $(m - 1)q$ communication cost.

Also, each pair of all the small inputs is assigned to reducers (by first placing them to bins of size $\frac{q}{2}$ using FFD or BFD bin-packing algorithm). The assignment of all the small inputs results in at most $\frac{8s'^2}{q^2} < \frac{8s^2}{q^2}$ reducers and at most $\frac{4s'^2}{q} < \frac{4s^2}{q}$ communication cost (Theorem 4.5). Thus, the number of reducers are at most $m - 1 + \frac{8s^2}{q^2}$ and the communication cost is at most $(m-1)q + \frac{4s^2}{q}$. □

Approximation factor. The optimal communication cost (from Theorem 4.1) is s^2/q and the communication cost of the algorithm (from Theorem 9.1) is $(m - 1)q + 4s^2/q$. Thus, the ratio between the optimal communication and the communication of our mapping schema is approximately $\frac{s^2}{mq^2}$.

10. AN APPROXIMATION ALGORITHM FOR THE *X2Y MAPPING SCHEMA PROBLEM*

We propose an approximation algorithm for the *X2Y mapping schema problem* that is based on bin-packing algorithms. Two lists, X of m inputs and Y of n inputs, are given. We assume that the sum of input sizes of the lists X, denoted by sum_x, and Y, denoted by sum_y, is greater than q. We analyze the algorithm on criteria (number of reducers and the communication cost) given in Section 4. We look at the lower bounds in Theorem 10.1, and Theorem 10.2 gives an upper bound from the algorithm. The bounds are given in Table I.

THEOREM 10.1. (LOWER BOUNDS ON THE COMMUNICATION COST AND NUMBER OF REDUCERS) *For a list X of m inputs, a list Y of n inputs, and a given reducer capacity q, the communication cost and the number of reducers, for the X2Y mapping schema problem, are at least $\frac{2 \cdot sum_x \cdot sum_y}{q}$ and $\frac{2 \cdot sum_x \cdot sum_y}{q^2}$, respectively, where q is the reducer capacity, sum_x is the sum of input sizes of the list X, and sum_y is the sum of input sizes of the list Y.*

PROOF. Since an input i of the list X and an input j of the list Y are replicated to at least $\frac{sum_y}{q}$ and $\frac{sum_x}{q}$ reducers, respectively, the communication cost for the inputs i and j

are $w_i \times \frac{sum_y}{q}$ and $w_j \times \frac{sum_x}{q}$, respectively. Hence, the communication cost will be at least $\sum_{i=1}^{m} w_i \frac{sum_y}{q} + \sum_{j=1}^{n} w_j \frac{sum_x}{q} = \frac{2 \cdot sum_x \cdot sum_y}{q}$.

Since the number of bits to be assigned to reducers is at least $\frac{2 \cdot sum_x \cdot sum_y}{q}$ and a reducer can hold inputs whose sum of the sizes is at most q, the number of reducers must be at least $\frac{2 \cdot sum_x \cdot sum_y}{q^2}$. □

Bin-packing-based approximation algorithm for the *X2Y mapping schema problem*. A solution to the *X2Y mapping schema problem* for different-sized inputs can be achieved using bin-packing algorithms. Let two lists X of m inputs and Y of n inputs are given. The algorithm will not work when a list holds an input of size w_i and the another list holds an input of size greater than $q - w_i$, because these inputs cannot be assigned to a single reducer in common. Let the size of the largest input, i, of the list X is w_i; hence, all the inputs of the list Y have at most size $q - w_i$. We place inputs of the list X to bins of size w_i, and let x bins are used to place m inputs. Also, we place inputs of the list Y to bins of size $q - w_i$, and let y bins are used to place n inputs. Now, we consider each of the bins as a single input, and a solution to the *X2Y mapping schema problem* is obtained by assigning each of the x bins with each of the y bins to reducers. In this manner, we require $x \cdot y$ reducers.

THEOREM 10.2 (UPPER BOUNDS FROM THE ALGORITHM). *For a bin size b, a given reducer capacity $q = 2b$, and with each input of lists X and Y being of size at most b, the number of reducers and the communication cost, for the X2Y mapping schema problem, are at most $\frac{4 \cdot sum_x \cdot sum_y}{b^2}$, and at most $\frac{4 \cdot sum_x \cdot sum_y}{b}$, respectively, where sum_x is the sum of input sizes of the list X, and sum_y is the sum of input sizes of the list Y.*

PROOF. A bin i can hold inputs whose sum of the sizes is at most b. Hence, it is required to divide inputs of the lists X and Y into at least $\frac{sum_x}{b}$ and $\frac{sum_y}{b}$ bins, respectively. Since the FFD or BFD bin-packing algorithm ensures that all the bins (except only one bin) are at least half-full, each bin of size b has at least inputs whose sum of the sizes is at least $\frac{b}{2}$. Thus, all the inputs of the lists X and Y can be placed in at most $\frac{sum_x}{b/2}$ and $\frac{sum_y}{b/2}$ bins of size b, respectively.

Let x (=$\frac{2 \cdot sum_x}{b}$) and y (=$\frac{2 \cdot sum_y}{b}$) bins are used to place inputs of the lists X and Y, respectively. Since each bin is considered as a single input, we can assign each of the x bins with each of the y bins at reducers, and hence, we require at most $\frac{4 \cdot sum_x \cdot sum_y}{b^2}$ reducers. Since each bin that is containing inputs of the list X (resp. Y) is replicated to at most $\frac{2 \cdot sum_y}{b}$ (resp. at most $\frac{2 \cdot sum_x}{b}$) reducers, the replication of individual inputs of the list X (resp. Y) is at most $\frac{2 \cdot sum_y}{b}$ (resp. at most $\frac{2 \cdot sum_x}{b}$) and the communication cost is at most $\sum_{1 \leq i \leq m} w_i \times \frac{2 \cdot sum_y}{b} + \sum_{1 \leq j \leq n} w_j \times \frac{2 \cdot sum_x}{b} = \frac{4 \cdot sum_x \cdot sum_y}{b}$. □

Approximation factor. The optimal communication is $\frac{2 \cdot sum_x \cdot sum_y}{q}$. Thus, the ratio between the optimal communication and the communication of our mapping schema is $\frac{1}{4}$.

11. CONCLUSION

Two new important practical aspects in the context of MapReduce, namely different-sized inputs and the reducer capacity, are introduced for the first time. The capacity of a reducer is defined in terms of the reducer's memory size. We note that processing time is typically proportional to the memory capacity. All reducers have an identical capacity, and any reducer cannot hold inputs whose input sizes are more than the reducer capacity. We demonstrated the importance of the capacity aspect by considering two common mapping schema problems of MapReduce, *A2A mapping schema problem* – every two inputs are required to be assigned to at least one common reducer – *X2Y mapping schema problem* – every two inputs, the first input from a list X and the second input from a list Y – is required to be assigned to at least

one common reducer. Unfortunately, it turned out that finding solutions to the *A2A* and the *X2Y* mapping schema problems that use the minimum number of reducers is not possible in polynomial time. On the positive side, we present near optimal approximation algorithms for the *A2A* and the *X2Y* mapping schema problems.

Mapping schemes for the case of reducers with different capacities are left for future research. Nevertheless, there exist a reduction to our proposed algorithms that may yield a reasonable performance in some cases. In particular, we can consider a common divisor of all the non-identical reducer capacity as a unit-sized reducer capacity. Then, we can follow our proposed algorithms to solve problems while regarding non-identical reducer capacities.

A. PROOFS OF THEOREMS 1, 2, AND LEMMA 1

Theorem 1 *The problem of finding whether a mapping schema of m inputs of different input sizes exists, where every two inputs are assigned to at least one of $z \geq 3$ identical-capacity reducers, is NP-hard.*

PROOF. The proof is by a reduction from the partition problem [Garey and Johnson 1979] that is a known NP-complete problem. The partition problem is defined as follows: given a set $I = \{i_1, i_2, \ldots, i_m\}$ of m positive integer numbers, it is required to find two disjoint subsets, $S_1 \subset I$ and $S_2 \subset I$, so that the sum of numbers in S_1 is equal to the sum of numbers in S_2, $S_1 \cap S_2 = \emptyset$, and $S_1 \cup S_2 = I$.

We are given m inputs whose input size list is $W = \{w_1, w_2, \ldots, w_m\}$, and the sum of the sizes is $s = \Sigma_{1 \leq i \leq m} w_i$. We add $z - 3$ additional inputs, $ai_1, ai_2, \ldots, ai_{z-3}$, each of size $\frac{s}{2}$. We call these new $z - 3$ ($ai_1, ai_2, \ldots, ai_{z-3}$) inputs the *medium inputs*. In addition, we add one more additional input, ai', of size $\frac{(z-2)s}{2}$ that we call the *big input*. Further, we assume that the reducer capacity is $\frac{(z-1)s}{2}$.

The proof proceeds in two steps: (*i*) we prove that in case the m original inputs can be partitioned, then all the inputs can be assigned to the z reducers such that every two inputs are assigned to at least one reducer, (*ii*) we prove that in case the mapping schema for all the inputs over the z reducers is successful, then there are two disjoint subsets S_1 and S_2 of the m original inputs that satisfy the partition requirements. We can assume that if the sum is not divisible by 2, then the answer to the partition problem is surely "no," so the reduction of the partition problem to the *A2A mapping schema problem* is trivial.

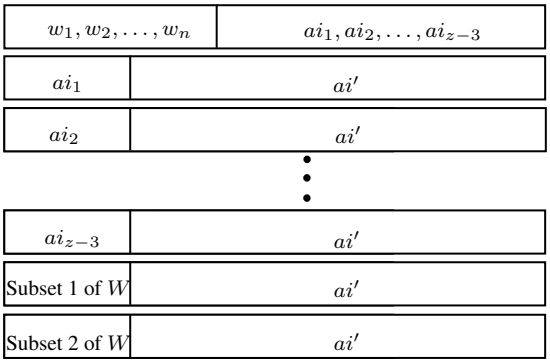

Fig. 16: Proof of NP-hardness of the *A2A mapping schema problem* for $z > 2$ identical-capacity reducers, Theorem 3.1.

We first show that if there are two disjoint subsets S_1 and S_2 of equal size of the m original inputs, then there must exist a solution to the *A2A mapping schema problem*. Recall that any of the reducers can hold a set of inputs whose sum of the sizes is at most $\frac{(z-1)s}{2}$, and the sum of the sizes of the new $z - 3$ medium inputs is exactly $\frac{(z-3)s}{2}$. Hence, all the m original inputs

$(i_1, i_2, \ldots, i_m)$ and a list of the $z-3$ medium inputs can be assigned to a single reducer (out of the z reducers), and this assignment uses $s + \frac{(z-3)s}{2}$ capacity, which is exactly the capacity of any reducer. Further, the big input, ai', of size $\frac{(z-2)s}{2}$ can share the same reducer with only one medium input ai_i (it could also share with original inputs). Thus, the big input, ai', and all the medium inputs are assigned to $z-3$ reducers (out of the remaining $z-1$ reducers). In addition, the remaining two reducers can be used for the following assignment: the first reducer is assigned the set S_1 and the big input, ai', and the second reducer is assigned the set S_2 and the big input, ai'. The above assignment is a solution to the *A2A mapping schema problem* for the given m original inputs, the $z-3$ medium inputs, and the big input using z reducers, see Figure 16.

Now, we show that a solution to the *A2A mapping schema problem* — for all the inputs over the z reducers — results in a partition of the m original inputs into two equal-sized blocks. We also show that in a solution to the *A2A mapping schema problem*, each of the m original inputs and every medium input, ai_i, are assigned to exactly two reducers, and the big input, ai', is assigned to exactly $z-1$ reducers. Recall that the total sum of the sizes is $s + \frac{(z-3)s}{2} + \frac{(z-2)s}{2} = \frac{(2z-3)s}{2}$.

Due to the reducer capacity of a single reducer, all the inputs cannot be assigned to a single reducer; only a subset of the inputs, whose sum of the sizes is at most $\frac{(z-1)s}{2}$, can be assigned to one reducer. Thus, each input is assigned to at least two reducers in order to be coupled with all the other inputs.

Moreover, the big input, ai', can share the same single reducer with only a sublist, S', whose sum of the sizes is at most $\frac{s}{2}$. Hence, the big input, ai', is required to be assigned to at least $z-3$ reducers in order to be paired with the medium inputs ai_i. Furthermore, the big input, ai', can share the same reducer with a sublist of the m original inputs whose sum of the sizes is at most $\frac{s}{2}$. This fact means that the big input, ai', must be assigned to two more reducers. On the other hand, all the medium inputs can share the same reducer with the original m inputs. Thus, here, the total reducer capacity occupied by all the inputs is $2 \times \Sigma_{1 \leq i \leq m} w_i + 2 \times \frac{(z-3)s}{2} + (z-1) \times \frac{(z-2)s}{2} = 2s + (z-3)s + \frac{(z-1)(z-2)s}{2} = \frac{(z-1)zs}{2}$, which is exactly the total capacity of all the z reducers. Thus, each of the m original inputs and each medium input ai_i cannot be assigned more than twice, and hence, each is assigned exactly twice. In addition, the big input, ai', is assigned to exactly $z-1$ reducers. This fact also shows that all the reducers are entirely filled with distinct inputs. Thus, a solution to the *A2A mapping schema problem* yields partitions of the m original inputs to S_1 and S_2 blocks, where the sum of the input sizes of any block is exactly $\frac{s}{2}$. Therefore, if there is a polynomial-time algorithm to construct the mapping schema, where every input is required to be paired with every other input, then the mapping schema finds the partitions of the m original inputs in polynomial time. □

Theorem 2 *The problem of finding whether a mapping schema of m and n inputs of different input sizes that belongs to list X and list Y, respectively, exists, where every two inputs, the first from X and the second from Y, are assigned to at least one of $z \geq 2$ identical-capacity reducers, is NP-hard.*

PROOF. The proof is by a reduction from the partition problem [Garey and Johnson 1979] that is a known NP-complete problem. We are given a list of inputs $I = \{i_1, i_2, \ldots, i_m\}$ whose input size list is $W = \{w_1, w_2, \ldots, w_m\}$, and the sum of the sizes is $s = \Sigma_{1 \leq i \leq m} w_i$. We add $z-2$ additional inputs, $ai_1, ai_2, \ldots, ai_{z-2}$, each of size $\frac{s}{2}$. We call these new $z-2$ $(ai_1, ai_2, \ldots, ai_{z-2})$ inputs the *big inputs*. In addition, we add one more additional input, ai', of size 1 that we call the *small input*. Further, we assume that the reducer capacity is $1 + \frac{s}{2}$. Now, the list I holds $m + z - 1$ inputs.

For the *X2Y mapping schema problem*, we consider m original inputs and the $z-2$ big inputs as a list X, and the small input as a list Y. A solution to the *X2Y mapping schema problem* assigns each of the m original inputs and each big input (of the list X) with the small input of the list Y.

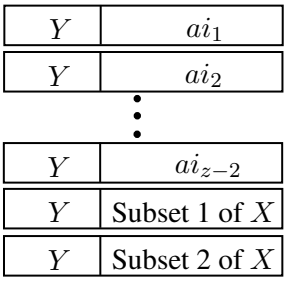

Fig. 17: Proof of NP-hardness of the *X2Y mapping schema problem* for $z > 1$ identical-capacity reducers, Theorem 3.2.

The proof proceeds in two steps: (*i*) we prove that in case the m original inputs can be partitioned, then all the m original inputs, the $z-2$ big inputs, and the small input can be assigned to the z reducers such that they satisfy the *X2Y mapping schema problem*, (*ii*) in case the *X2Y mapping schema problem* is successful, then there are two disjoint subsets, S_1 and S_2, of the m original inputs that satisfy the partition requirements.

We first show that if there are two disjoint subsets S_1 and S_2 of equal size of the m original inputs, then there must exist a solution to the *X2Y mapping schema problem*. Recall that any of the reducers can hold a set of inputs whose sum of sizes is at most $1+\frac{s}{2}$, and the sum of the sizes of the new $z-2$ big inputs is exactly $\frac{s}{2}$. Hence, the small input, ai', of size 1 and each big input, ai_i, can be assigned to $z-2$ reducers (out of the z reducers), and this assignment uses $1+\frac{s}{2}$ capacity, which is exactly the capacity of any reducer. In addition, the remaining two reducers can be used for the following assignment: the first remaining reducer is assigned the set S_1 and the small input, ai', and the second remaining reducer is assigned the remaining original inputs, S_2, and the small input, ai'. The above assignment is a solution to the *X2Y mapping schema problem* (for the given $m+z-2$ inputs of the list X and the one input of the list Y using z reducers, see Figure 17).

Now, we prove the second claim that a solution to the *X2Y mapping schema problem* results in a partition of the m original inputs into two equal-sized blocks. Recall that the total sum of the sizes is $s+\frac{(z-2)s}{2}+1=\frac{z\times s}{2}+1$.

Due to the reducer capacity of a single reducer, all the inputs cannot be assigned to a single reducer; only a sublist of the inputs, whose sum of the sizes is at most $1+\frac{s}{2}$, can be assigned to a single reducer. We show that the small input, ai', must be assigned to all the z reducers. The small input, ai', of size one can share the same single reducer with only a subset, S', whose sum of the sizes is at most $\frac{s}{2}$. Hence, the small input, ai', is required to be assigned to $z-2$ reducers (out of z reducers) in order to be paired with all the big inputs ai_i. and the remaining two reducers in order to be paired with all the m original inputs. This fact results in that a solution to the *X2Y mapping schema problem* yields partitions of the m original inputs to S_1 and S_2 blocks, where the sum of the input sizes of any block is exactly $\frac{s}{2}$. Therefore, if there is a polynomial-time algorithm to construct the mapping schema, where every input of one list is required to be paired with every other input of another list, then the mapping schema finds the partitions of the m original inputs in polynomial time. □

Lemma 1 *Let q be the reducer capacity. Let the size of an input is $\lceil\frac{q-1}{2}\rceil$. Each pair of $u=2^i$, $i>0$, inputs can be assigned to 2^i-1 teams of 2^{i-1} reducers in each team.*

PROOF. The proof is by induction on i.

Basis case. For $i=1$, we have $u=2$ inputs, and we can assign them to a team of one reducer of capacity q. Hence, Lemma 6.2 holds for ($i=1$) two inputs.

Inductive step. Assume that the inductive hypothesis — there is a solution for $u=2^{i-1}$ inputs, where all-pairs of $u=2^{i-1}$ inputs are assigned to $2^{i-1}-1$ teams of 2^{i-2} reducers in each team and have the team property (each team has one occurrence of each input, which we

will prove in algorithm correctness) — is true. Now, we can build a solution for $u = 2^i$ inputs, as follows:

(a) Divide $u = 2^i$ inputs into two groups of 2^{i-1} inputs in each group,
(b) Recursively create teams for each of the two groups,
(c) Create some of the teams for the 2^i inputs by combining the j^{th} team from the first group with the j^{th} team from the second group. Since by the inductive hypothesis we have a solution for $u = 2^{i-1}$ inputs, we can assign inputs of these two groups to $2 \cdot (2^{i-1} - 1)$ teams of 2^{i-2} reducers in each team. And, by combining j^{th}, where $j = 1, 2, \ldots, (2^{i-1} - 1)$, teams of each group, there are $2^{i-1} - 1$ teams of 2^{i-1} reducers in each team; see Teams 5-7 for 8 inputs in Figure 7.
(d) Create 2^{i-1} additional teams that pair the inputs from the first group with inputs from the second group. In each team, the j^{th} input from the first group is assigned to the j^{th} reducer. In the first team, the j^{th} input from the second group is also assigned to the j^{th} reducer. In subsequent teams, the assignments from the second group rotate, so in the t^{th} team, the j^{th} input from the second group is assigned to reducer $k + j - (2^{i-1} - 1)(modulo 2^{i-1})$; see Teams 1-4 for 8 inputs in Figure 7.

By steps (c) and (d), there are total $2^{i-1} - 1 + 2^{i-1} = 2^i - 1$ teams of 2^{i-1} reducers in each team, and these teams holds each pair of the $u = 2^i$ inputs. □

B. PSEUDOCODE AND CORRECTNESS OF ALGORITHM 1A

Algorithm 1A description. First, we divide m inputs (that are actually bins of size $\frac{q}{k}$, $k > 3$, after placing all the given m inputs to m' bins, each of size $\frac{q}{k}$) into two sets A of y inputs and B of x inputs. Then, we make $u = \left\lceil \frac{y}{q - \lceil q/2 \rceil} \right\rceil$ disjoint groups of y inputs of the set A such that each group holds $\frac{q-1}{2}$ inputs, lines 1, 2. (Now, each of the groups is considered as a single input that we call the *derived input*.) We do not show the addition of dummy inputs and assume that u is a power of 2. Function $2\_step\_odd\_q(lower, upper)$ recursively divides the derived inputs into two halves, line 4. Function $Assignment(lower, mid, upper)$ (line 9) pairs every two derived inputs and assigns them to the respective reducers (line 11). Each reducer of the last team is assigned using function $Last\_Team(groupA[])$, lines 15, 16.

Note that functions $2\_step\_odd\_q(lower, upper)$, $Assignment(lower, mid, upper)$, and $value\_b(lower, t, mid, upper)$ take two common parameters, namely $lower$ and $upper$ where $lower$ is the first derived input and $upper$ is the last derived input (*i.e.*, u^{th} group) at the time of the first call to functions, line 3. Once all-pairs of the derived inputs are assigned to reducers, line 11, function $Assign\_input\_from\_B(Team[])$ assigns i^{th} input of the set B to all the $\left\lceil \frac{u}{2} \right\rceil$ reducers of i^{th} team, lines 17, 18. After that, Algorithm 1A is invoked over inputs of the set B to assign each pair of the remaining inputs of the set B to reducers until every pair to the remaining inputs is assigned to reducers.

The algorithm correctness proves that every pair of inputs is assigned to reducers. Specifically, we prove that all those pairs of inputs, $\langle i, j \rangle$ and $\langle i', j' \rangle$, of the set A are assigned to a team whose $i \neq i'$ and $j \neq j'$ (Claim B.1). Then that all the inputs of the set A appear exactly once in each team (Claim B.2). We then prove that the set B holds $x \leq y - 1$ inputs, when $q = 3$ (Claim B.3). At last we conclude in Theorem B.4 that Algorithm 1A assigns each pair of inputs to reducers.

Note that we are proving all the above mentioned claims for $q = 3$; the cases for $q > 3$ can be generalized trivially where we make $u = \left\lceil \frac{y}{q - \lceil q/2 \rceil} \right\rceil$ derived inputs from y inputs of the set A (and assign in a manner that all the inputs of the A are paired with all the remaining $m - 1$ inputs).

CLAIM B.1. *Pairs of inputs $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i = i'$ or $j = j'$, of the set A are assigned to different teams.*

PROOF. First, consider $i = i'$ and $j \neq j'$, where $\langle i, j \rangle$ and $\langle i', j' \rangle$ must be assigned to two different teams. If $j \neq j'$, then both the j values may have an identical value of $lower$ and

Algorithm 1: Part A

Inputs: m: the number of bins obtained after placing all the given m' inputs (of size $\leq \frac{q}{k}$, $k > 3$ is an odd number) to bins each of size $\frac{q}{k}$,
q: the reducer capacity.
Variables:
A: A set A, where the total inputs in the set A is $y = \lfloor \frac{q}{2} \rfloor (\lfloor \frac{2m}{q+1} \rfloor + 1)$
B: A set B, where the total inputs in the B is $x = m - y$
$Team[i, j]$: represents teams of reducers, where index i indicates i^{th} team and index j indicates j^{th} reducer in i^{th} team. Consider $u = \lceil \frac{y}{q - \lceil q/2 \rceil} \rceil$. There are $u - 1$ teams of $v = \lceil \frac{u}{2} \rceil$ reducers in each team.
$groupA[]$: represents disjoint groups of inputs of the set A, where $groupA[i]$ indicates i^{th} group of $\lceil \frac{q-1}{2} \rceil$ inputs of the set A.

1 **Function** $create\_group(y)$ **begin**
2 **for** $i \leftarrow 1$ **to** u **do** $groupA[i] \leftarrow \langle i, i+1 \ldots, i + \frac{q-1}{2} - 1 \rangle, i \leftarrow i + \frac{q-1}{2}$
3 $2\_step\_odd\_q(1, u)$, $Last\_Team(groupA[])$, $Assign\_input\_from\_B(Team[])$
4 **Function** $2\_step\_odd\_q(lower, upper)$ **begin**
5 **if** $\lfloor \frac{upper - lower}{2} \rfloor < 1$ **then return**
6 **else**
7 $mid \leftarrow \lceil \frac{upper - lower}{2} \rceil$, $Assignment(lower, mid, upper)$
8 $2\_step\_odd\_q(lower, mid)$, $2\_step\_odd\_q(mid + 1, upper)$
9 **Function** $Assignment(lower, mid, upper)$ **begin**
10 **while** $mid > 1$ **do**
11 **foreach** $(a, t) \in [lower, lower + mid - 1] \times [0, mid - 1]$ **do**
 $Team\big[(u - 2 \cdot mid + 1) + t, a - \lfloor \frac{a-1}{mid} \rfloor \cdot \frac{mid}{2} \big] \leftarrow$
 $\langle groupA[a], groupA[value\_b(a, t, mid, upper)] \rangle$
12 **Function** $value\_b(a, t, mid, upper)$ **begin**
13 **if** $a + t + mid < upper + 1$ **then return** $(a + t + mid)$
14 **else if** $a + t + mid > upper$ **then return** $(a + t)$
15 **Function** $Last\_Team(lower, mid, upper)$ **begin**
16 **foreach** $i \in [1, v]$ **do** $Team[u - 1, i] \leftarrow groupA[2 \times i - 1], groupA[2 \times i]$
17 **Function** $Assign\_input\_from\_B(Team[])$ **begin**
18 **foreach** $(i, j) \in [1, u - 1] \times [1, v]$ **do** $Team[i, j] \leftarrow B[i]$

mid but they must have two different values of t (see lines 13, 14 of Algorithm 1A), where $j = lower + t + mid$ or $j = lower + t$. Thus, for two different values of j , we use two different values of t, say t_1 and t_2, that results in an assignment of $\langle i, j \rangle$ and $\langle i', j' \rangle$ to two different teams t_1 and t_2, (note that teams are also selected based on the value of t, $(y - 2 \cdot mid + 1) + t$, see line 11 of Algorithm 1A, where for $q = 3$, we have $u = y$). Suppose now that $i \neq i'$ and $j = j'$, where $\langle i, j \rangle$ and $\langle i', j' \rangle$ must be assigned to two different teams. In this case, we also have two different values of t, and hence, two different t values assign $\langle i, j \rangle$ and $\langle i', j' \rangle$ to two different teams ($(y - 2 \cdot mid + 1) + t$, line 11 of Algorithm 1A).

Hence, it is clear that pairs $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i \neq i'$ and $j \neq j'$, are assigned to a team. □

CLAIM B.2. *All the inputs of the set A appear exactly once in each team.*

PROOF. There are the same number of pairs of inputs of the set A and the number of reducers ($(y - 1)\lceil \frac{y}{2} \rceil$) that can provide a solution to the *A2A mapping schema problem* for the y inputs of the set A. Recall that $(y - 1)\lceil \frac{y}{2} \rceil$ reducers are arranged in the form of $(y - 1)$ teams

of $\lceil \frac{y}{2} \rceil$ reducers in each team, when $q = 3$. Note that if there is a input pair $\langle i, j \rangle$ in team t, then the team t cannot hold any pair that has either i or j in the remaining $\lceil \frac{y}{2} \rceil - 1$ reducers. For the given y inputs of the set A, there are at most $\lceil \frac{y}{2} \rceil$ disjoint pairs $\langle i_1, j_1 \rangle$, $\langle i_2, j_2 \rangle$, ..., $\langle i_{\lceil y/2 \rceil}, j_{\lceil y/2 \rceil} \rangle$ such that $i_1 \neq i_2 \neq \ldots \neq i_{\lceil y/2 \rceil} \neq j_1 \neq j_2 \neq \ldots \neq j_{\lceil y/2 \rceil}$. Hence, all y inputs of the set A are assigned to a team, where no input is assigned twice in a team. □

CLAIM B.3. *When the reducer capacity $q = 3$, the set B holds at most $x \leq y - 1$ inputs.*

PROOF. Since a pair of inputs of the set A requires at most $q - 1$ capacity of a reducer and each team holds all the inputs of the set A, an input from the set B can be assigned to all the reducers of the team. In this manner, all the inputs of the set A are also paired with an input of the set B. Since there are $y - 1$ teams and each team is assigned an input of the set B, the set B can hold at most $x \leq y - 1$ inputs. □

THEOREM B.4. *Algorithm 1A assigns each pair of the given inputs to at least one reducer in common.*

PROOF. We have $(y-1)\lceil \frac{y}{2} \rceil$ pairs of inputs of the set A of size $q-1$, and there are the same number of reducers; hence, each reducer can hold one input pair. Further, the remaining capacity of all the reducers of each team can be used to assign an input of B. Hence, all the inputs of A are paired with every other input and every input of B (as we proved in Claims B.2 and B.3). Following the fact that the inputs of the set A are paired with all the m inputs, the inputs of the set B is also paired by following a similar procedure on them. Thus, Algorithm 1A assigns each pair of the given m inputs to at least one reducer in common. □

C. PSEUDOCODE AND CORRECTNESS OF ALGORITHM 1B

Algorithm 1: Part B

Inputs: m: the number of bins obtained after placing all the given m' inputs (of size $\leq \frac{q}{k}$, $k \geq 4$ is an even number) to bins each of size $\frac{q}{k}$,
q: the reducer capacity.
Variables:
$Team[i, j]$: represents teams of reducers, where index i indicates i^{th} team and index j indicates j^{th} reducer in i^{th} team. Consider $u = \frac{2m}{q}$. There are $u - 1$ teams of $\lceil \frac{u}{2} \rceil$ reducers in each team.
$groupA[]$: represents disjoint groups of inputs of the set A, where $groupA[i]$ indicates i^{th} group of $\lceil \frac{q}{2} \rceil$ inputs of the set A.

```
1 Function create_group(m) begin
2     for i ← 1 to u do groupA[i] ← ⟨i, i + 1 ..., i + q/2 − 1⟩, i ← i + q/2
3     2_step_even_q(1, u), Last_Team(1, ⌈(u−1)/2⌉, u)
4 Function 2_step_even_q(lower, upper) begin
5     if ⌊(upper−lower)/2⌋ < 1 then return
6     else
7         mid ← ⌈(upper−lower)/2⌉, Assignment(lower, mid, upper)
8         2_step_even_q(lower, mid), 2_step_even_q(mid + 1, upper)
```

We show that every pair of inputs is assigned to reducers. Specifically, Algorithm 1B satisfies two claims, as follows:

CLAIM C.1. *Pairs of derived inputs $\langle i, j\rangle$ and $\langle i', j'\rangle$, where $i \neq i'$ or $j \neq j'$, are assigned to a team.*

CLAIM C.2. *All the given m inputs appear exactly once in each team.*

We do not prove Claims C.1 and C.2. Note that Claim C.1 follows Claims B.1, where Claims B.1 shows that all the pairs of inputs of the set A (in case $q = 3$) and all the pairs of derived inputs of the set A (in case $q > 3$) $\langle i, j\rangle$ and $\langle i', j'\rangle$, where $i \neq i'$ or $j \neq j'$ are assigned to a team. Also, Claim C.2 follows Claim B.2, where Claim B.2 shows that all the inputs of the set A appear in each team only once, while in case of Algorithm 1B the set A is considered as a set of m inputs.

THEOREM C.3. *Algorithm 1B assigns each pair of the given inputs to at least one reducer in common.*

PROOF. Since there are the same number of pairs of the derived inputs and the number of reducers, it is possible to assign one pair to each reducer that results in all-pairs of the m inputs. □

D. CORRECTNESS OF ALGORITHM 2

The correctness shows that all-pairs of inputs are assigned to reducers. Specifically, we show that each pair of inputs of the set A is assigned to $p(p + 1)$ reducers that use only p capacity of each reducer (Claims D.1 and D.2). Then, we prove that the set B holds $x \leq m - p^2$ inputs. At last we conclude that Algorithm 2 assigns each pair of inputs to reducers.

CLAIM D.1. *All the inputs of the set A are assigned to $p(p+1)$ reducers, and the assignment of the inputs of the set A uses only p capacity of each reducer.*

CLAIM D.2. *All the inputs of the set A appear in each team exactly once.*

We are not proving Claims D.1 and D.2 here. Claims D.1 and D.2 follow the correctness of the *AU method*; hence, all the inputs of the set A are placed to $p + 1$ teams of p bins (each of size q) in each team, and the assignment of each such bin only uses p capacity of each reducer. Further two bins cannot be assigned to a reducer because $2 \times p > q$. Claim D.2 also follows the correctness of the *AU method*, and hence, all the inputs of the set A appear only once in each team.

CLAIM D.3. *When the reducer capacity is q, the set B holds $x \leq m - p^2$ inputs, where p is the nearest prime number to q.*

PROOF. There are $p + 1$ teams of p reducers in each team, and inputs of the set A use $q - p$ capacity of each of the reducers. Hence, each reducer can hold $q - p$ additional unit-sized (almost identical-sized) inputs. Since inputs of the set A appear in each team (Claim D.2), an assignment of $q - p$ additional unit-sized inputs to all the reducers of a team provides pairs of all the inputs of the set A with additional inputs. In this manner, $p + 1$ teams, which hold p^2 inputs of the set A, can hold at most $(p + 1) \times (q - p)$ additional inputs. Since $p^2 < m \leq p^2 + (p + 1) \times (q - p)$, the set B can hold $x \leq m - p^2$ inputs. □

THEOREM D.4. *Algorithm 2 assigns each pair of inputs to reducers.*

We are not proving Theorem D.4 here. The proof of Theorem D.4 considers the fact that all the inputs of the set A are paired with each other using the *AU method*, and they are also paired with all the remaining inputs of the set B. Further, inputs of the set B will be paired with each other by using Algorithm 1A or 1B (Theorems B.4 or C.3).

REFERENCES

Foto Afrati, Shlomi Dolev, Ephraim Korach, Shantanu Sharma, and Jeffrey D. Ullman. 2015. Assignment of Different-Sized Inputs in MapReduce. In *2nd Workshop on Algorithms and Systems for MapReduce and Beyond (BeyondMR)*. 28–37. Also appears as a Brief Announcement in International Symposium on Distributed Computing (DISC), 2014, and as a technical report 14-05 at Department of Computer Science, Ben-Gurion University of the Negev.

Foto N. Afrati, Anish Das Sarma, Semih Salihoglu, and Jeffrey D. Ullman. 2013. Upper and Lower Bounds on the Cost of a Map-Reduce Computation. *PVLDB* 6, 4 (2013), 277–288. http://www.vldb.org/pvldb/vol6/p277-dassarma.pdf

Foto N. Afrati and Jeffrey D. Ullman. 2013. Matching bounds for the all-pairs MapReduce problem. In *17th International Database Engineering & Applications Symposium, IDEAS '13, Barcelona, Spain - October 09 - 11, 2013*. 3–4. DOI:http://dx.doi.org/10.1145/2513591.2513663

Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*. 131–140. DOI:http://dx.doi.org/10.1145/1242572.1242591

E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. 1997. Approximation algorithms for NP-hard problems. PWS Publishing Co., Chapter Approximation algorithms for bin packing: a survey, 46–93. http://dl.acm.org/citation.cfm?id=241938.241940

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*. 137–150. http://www.usenix.org/events/osdi04/tech/dean.html

M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Michael T. Goodrich. 2010. Simulating Parallel Algorithms in the MapReduce Framework with Applications to Parallel Computational Geometry. *CoRR* abs/1004.4708 (2010). http://arxiv.org/abs/1004.4708

David S Johnson. 1973. *Near-optimal bin packing algorithms*. Ph.D. Dissertation. Massachusetts Institute of Technology.

David R. Karger and Jacob Scott. 2008. Efficient Algorithms for Fixed-Precision Instances of Bin Packing and Euclidean TSP. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*. 104–117. DOI:http://dx.doi.org/10.1007/978-3-540-85363-3_9

Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*. 938–948. DOI:http://dx.doi.org/10.1137/1.9781611973075.76

Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. Filtering: a method for solving graph problems in MapReduce. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*. 85–94. DOI:http://dx.doi.org/10.1145/1989493.1989505

Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. 2014. *Mining of Massive Datasets, 2nd Ed*. Cambridge University Press.

Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. 2012. Space-round tradeoffs for MapReduce computations. In *International Conference on Supercomputing, ICS'12, Venice, Italy, June 25-29, 2012*. 235–244. DOI:http://dx.doi.org/10.1145/2304576.2304607

Jeffrey D. Ullman. 2012. Designing good MapReduce algorithms. *ACM Crossroads* 19, 1 (2012), 30–34. DOI:http://dx.doi.org/10.1145/2331042.2331053