Strongly Minimal MapReduce Algorithms: A TeraSort Case Study

Daniel Xia, Michael Simpson, Venkatesh Srinivasan, Alex Thomo

University of Victoria, Canada
`daniel.f.xia@gmail.com`, {`simpsonm,srinivas,thomo`}`@uvic.ca`

Abstract. MapReduce is a widely used parallel computing paradigm for the big data realm on the scale of terabytes and higher. The introduction of *minimal* MapReduce algorithms promised efficiency in load balancing among participating machines by ensuring that *partition skew* (where some machines end up processing a significantly larger fraction of the input than other machines) is prevented. Despite minimal MapReduce algorithms guarantee of load-balancing within constant multiplicative factors, the constants are relatively large which severely diminishes the theoretical appeal for true efficiency at scale.
We introduce the notion of *strongly minimal* MapReduce algorithms that provide strong guarantees of parallelization up to a small additive factor that diminishes with an increasing number of machines. We show that a strongly minimal MapReduce algorithm exists for sorting; this leads to strongly minimal algorithms for several fundamental database algorithms and operations that crucially rely on sorting as a primitive. Our techniques are general and apply beyond the analysis of strongly minimal MapReduce algorithms; we show that given a sufficiently high, but still realistic, sampling rate, the approximate partitions obtained from a particular sampling strategy are almost as good as the partitions produced by an ideal partitioning.

Keywords: distributed sorting, minimal MapReduce algorithms, Sample-Partition problem

1 Introduction

Data is being generated at an increasing pace that leads to an enormous volume being created and stored each year. As a result, there has been a strong push towards *big data analytics* as industry and governments around the world aim to keep pace with the explosion of information. This has led database organizations to build massive parallel computing platforms that rely upon huge numbers of commodity machines. Among these platforms, *MapReduce* has emerged as the popular choice after years of improvement and advancement.

At a high level, MapReduce algorithms instruct how these machines can perform a given task collaboratively. Typically, the input data is distributed across the machines and the algorithm executes in *rounds* made up of *map* and *reduce* phases. The *map* phase prepares data to be exchanged to other machines and the *reduce* phase has the machines perform isolated computations on its local storage. Rounds proceed until the given task is complete after a *reduce* phase. Ideally, MapReduce algorithms should aim for the minimization of space, CPU, I/O, and network costs for each machine as well as even load balancing. Despite these principles guiding the design of MapReduce algorithms, most previous work has relied upon heuristic approaches or been driven by an experimental performance basis where less emphasis is placed on enforcing rigorous constraints on these performance metrics.

Tao et al. [24] introduced *minimal* MapReduce algorithms that promise efficiency in multiple aspects simultaneously. The notion of *minimal* MapReduce algorithms bounds the storage space and the amount of information sent over the network of each machine to be optimal up to a constant multiplicative factor. In addition, a constraint is placed that the algorithm terminates in a constant number of rounds and that the algorithm achieves a speedup of a factor t when using t machines in parallel. One of the benefits of designing minimal algorithms includes guaranteeing that *partition skew* (where some machines end up processing a significantly larger fraction of the input than other machines) is prevented. Tao et al. [24] conclude that TeraSort[1], the state-of-the-art MapReduce sorting algorithm, is load-balanced within constant multiplicative factors and thus satisfies their minimality definition. However, the multiplicative factor in the proof is relatively large, up to 32; such a factor severely diminishes the theoretical appeal due to a large imbalance in the workload of machines. Our work is motivated by the question of whether the notion of minimal MapReduce algorithms can be strengthened even further?

The main contributions of our work are as follows: (1) We propose a strengthening of the notion of minimality by introducing an additional requirement called *balanced partition* that restricts the number of objects processed by each machine to be evenly balanced up to an additive factor that diminishes with the number of machines. We say such MapReduce algorithms are *strongly* minimal. (2) As a core result, we prove that sorting, which is a backbone primitive for many algorithms and operations in databases and beyond, has a strongly minimal algorithm, i.e. the workload is evenly distributed across machines up to an additive term rather than a multiplicative factor. This is important because in large data centers of similar machines having an overloaded machine is a critical bottleneck (referred to by practitioners as "the curse of the last reducer", see [23]).

More specifically, we aim for a more accurate analysis of the performance of TeraSort. We give a series of bounds which describes the trade-off between the number of machines and the partition skew (hence the worst case maximum workload on a single machine). We conclude that a larger number of available machines allows for more even partitions. In particular, we show that as the

[1] We use the name TeraSort in this paper to refer to the Sample-Partition-Sort *paradigm* for distributed sorting. We would like to emphasize that the version inplemented in Hadoop (a popular MapReduce implementation), while following the general paradigm, is not minimal (please see Section 3).

number of machines grows, the partition skew approaches its optimal value with high probability.

Our techniques are general and apply beyond TeraSort; we show that given a sufficiently high, but still realistic, sampling rate, the approximate partitions obtained from a particular sampling strategy are almost as good as the partitions produced by an ideal partitioning. The sampling strategy we analyze, self-sampling, has appealing probabilistic properties that we are able to leverage through a more extensible probabilistic method that is capable of generating a series of tight bounds for partition evenness. We use a new and refined analysis technique by an interesting bucketing argument that allows *overlapping* buckets in contrast to the non-overlapping scheme of [24].

1.1 Strongly Minimal MapReduce Algorithms

Instead of mappers and reducers, we refer to the workers as *machines*. To accommodate the statelessness of mappers and reducers, we assume that unmentioned data in the algorithm is "carried forward" implicitly to the reducer with the same index in the next round.

Let A be the input for the underlying problem, n be the number of objects in A, and t be the number of machines in the network. Define $m = n/t$ as the number of objects per machine when A is evenly distributed across the machines. Then, as defined in [24], a minimal MapReduce algorithm for a problem on A has the following properties:

- *Minimal footprint:* each machine uses $O(m)$ storage at all times.
- *Bounded net-traffic:* every machine sends and receives at most $O(m)$ words of information over the network in each round.
- *Constant round:* the algorithm terminates after a constant number of rounds.
- *Optimal computation:* the algorithm achieves a speedup of t when using t machines in parallel. Precisely, each machine performs $O(T_{seq}/t)$ work in total over all rounds where T_{seq} is the time required by a *fixed* algorithm $\mathcal{A}$ to solve the problem on a single machine.

Now, we introduce the notion of a *strongly* minimal MapReduce algorithm which strengthens the minimality conditions above by adding an additional condition that we refer to as *balanced partition*. A strongly minimal MapReduce algorithm for a problem on A has the following property:

- *Balanced partition:* each machine processes $m(1 + o(1))$ objects.

Remark; Here, $o(1)$ denotes a lower order term that is independent of the input and goes to 0 as t grows. Note that strong minimality implies that, as t grows, the number of objects processed by each machine approaches the optimal value. As a consequence, the hidden constant in footprint and net-traffic in strongly minimal MapReduce algorithms is substantially smaller than 32, typically close to 1.

In the following sections, we prove that TeraSort is strongly minimal by showing that the *balanced partition* condition is satisfied using an interesting bucketing argument.

2 Related Work

The existing investigation on MapReduce can be broadly classified into two categories: (1) a focus on improving the internal working of the framework, and (2) developing novel MapReduce algorithms to solve interesting problems. On the framework implementation side there has been a variety of work that typically focuses on performing well on a subset of the minimality conditions. These range from specialized methods to rectify skewness [11, 14, 16] to optimizing the network traffic by keeping relevant data at the same machine [6, 12]. On the algorithms side, there has been extensive work dedicated to developing MapReduce algorithms for important database problems [4, 1, 18, 19, 26], graph processing [25, 23, 2, 13, 17], and statistical analysis [5, 21, 9, 7, 10].

Tao et al. [24] justify theoretically the good performance of TeraSort [20] (the state-of-the-art MapReduce sorting algorithm) observed in practice which inspired the new definition of minimal MapReduce algorithms. Their goal is accomplished by specifying how to set a crucial parameter of TeraSort that ensures minimality. Designing minimal algorithms is highly sought after since a minimal algorithm excels on all the minimality conditions simultaneously. Often, it is easy to perform well on certain aspects, while failing on others. Furthermore, Tao et. al. [24] point out that even a minimal algorithm can benefit from clever optimization at the system level, and the minimality property may considerably simplify such optimizations. For instance, as the minimality requirements already guarantee good load balancing in storage, computation, and communication, there would be less skewness to deserve specialized optimization.

Studying the minimality of MapReduce algorithms is similar in goal to other models of theoretical parallel computing. We give two such examples now. Karloff et al. [13] put forth the notion of $\mathcal{MRC}$, a class of MapReduce algorithms computable by a MapReduce system characterized by a certain amount of resources. Further, class $\mathcal{MRC}^i$ runs in $O(\log^i n)$ rounds and $\mathcal{MRC}$ is defined by the union of $\mathcal{MRC}^i$ over i. When the algorithm is randomized, it must output the correct answer with probability at least 3/4. The deterministic subset of $\mathcal{MRC}$ is called $\mathcal{DMRC}$. Note, not all algorithms in $\mathcal{MRC}$ are efficient; rather it only offers to characterize them. One would expect efficient algorithms in $\mathcal{MRC}^0$ and $\mathcal{MRC}^1$ since they consist of constant rounds and logarithmic rounds, respectively. [13] shows that a variety of problems have solutions in $\mathcal{MRC}^0$ and $\mathcal{MRC}^1$, such as finding an MST in dense graphs, frequency moments, and undirected *s*-*t* connectivity. Recent work following the $\mathcal{MRC}$ model include [3, 22, 15].

Massive, unordered, distributed (MUD) is a class of MapReduce algorithms proposed by [8] to compute a *distributed* stream. The MUD algorithms consist of three components: a *local function*, an *aggregator*, and *post-processing*. The algorithm designer must ensure that the overall output is independent of the order of application of the aggregator. A connection between MUD algorithms and the MapReduce framework can be established where the local function can be implemented by mappers, and the independence of the post-processing and the order of application implies that we can divide and conquer the output of the local function in a series of rounds. From this construction, it is easy to see that MUD algorithms can be computed very efficiently in a MapReduce system and mostly independent of the underlying computing capability.

3 Sorting with MapReduce

For sorting, the input is a set A of n objects drawn from an ordered domain. Suppose that t machines store A and are indexed from 1 to t, namely $\mathcal{M}_1, ..., \mathcal{M}_t$. A parallel algorithm that solves the sorting problem should terminate with all the objects distributed across the t machines in a (total) sorted fashion. That is, for each machine $\mathcal{M}_i$, the objects that end up in $\mathcal{M}_i$ are in sorted order. Further, this implies that all objects in $\mathcal{M}_i$ precede those in $\mathcal{M}_j$ for all $1 \leq i < j \leq t$.

It is well known that sorting can be solved in $O(n \log n)$ time on a single machine, while there has been a substantial amount of progress on sorting in parallel. *TeraSort* is the state-of-the-art MapReduce algorithm for sorting and the work of [24] proves that it is minimal when a crucial parameter of the algorithm is set appropriately.

3.1 Sampling and Partitioning

Sampling and partitioning form the central idea behind TeraSort. TeraSort conceptually consists of three steps: Sample, Partition, and Sort. First, the algorithm extracts a random sample set from the input and then computes t partition elements from the sample. The partition elements, referred to as *boundary elements*, divide A into t partitions. In the second round, each machine receives all the elements from a distinct partition and sorts them locally using an apriori fixed algorithm $\mathcal{A}$. As the performance of TeraSort is sensitive to the quality of the partition, it is worth examining potential sampling strategies. We describe two such sampling strategies below.

In the current implementation of TeraSort included in Hadoop, the sample is created by reading a elements in total from b locations which are evenly spread across the input dataset. a and b are configurable by users. At each location, $\frac{a}{b}$ elements are read. No guarantee exists that such sampling scheme yields good partitioning. In fact, there are bad cases for every a, b in which the partitions are extremely unbalanced. When the sample comprises elements concentrated in a few small ranges, it may lead to uneven buckets.

Tao et al. [24] discuss the strategy of *self-sampling*, where each element is selected into the sample independently with the same probability. Self-sampling is a good fit for the MapReduce framework as mappers are assumed to have no other knowledge than the input item currently being processed. As we will show later, self-sampling has very appealing probabilistic properties and it achieves asymptotically optimal evenness with high probability. [24] also report the experimental results of another strategy: sampling without replacement. The results are promising and comparable to self-sampling: the unevenness remains low when the sample size is no less than the expected size of self-sampling, regardless of how much it exceeds the latter. However, there is no further investigation on how the evenness is affected by the layout of the input dataset, nor is any bound provided for all input layouts.

In this paper, we focus on analyzing TeraSort with self-sampling. However, our techniques are general and apply beyond TeraSort; we show that given a sufficiently high, but still realistic, sampling rate, the approximate partitions obtained from self-sampling are almost as good as the partitions produced by an ideal partitioning.

3.2 Even Partitions

In TeraSort, we use the notion of an ordered even t-partition which divides a set as evenly as possible.

Definition 1 (ordered t-partition). *An ordered t-partition divides an ordered set of n elements into t partitions. Elements of partition i are smaller than those of partition j for all $i < j$. The first element of every partition except for the first one is called a t-partition element as they describe the partitions in full.*

Definition 2 (ordered even t-partition). *An ordered even t-partition is an ordered t-partition in which the sizes of the partitions differ by at most* 1.

An ordered even partition always exists for any dataset. In fact, we may construct one in the following way. Let $n = ts_1 + s_2$ where $s_1 = \left\lfloor \frac{n}{t} \right\rfloor$. The indices of the partition elements are $d_j = d_{j-1} + (s_1 + 1)$ for $1 \leq j \leq s_2$ and $d_j = d_{j-1} + s_1$ for $s_2 \leq j \leq t$ with $d_0 = s_1$.

3.3 TeraSort

Recall, TeraSort consists of three steps: Sample, Partition, and Sort. First, the algorithm extracts a random sample set from the input and then computes t partition elements from the sample. The partition elements, referred to as *boundary elements*, divide A into t partitions. In the second round, each machine receives all the elements from a distinct partition and sorts them locally using a fixed algorithm $\mathcal{A}$. Importantly, the construction of the sample is crucial to efficiency since the partition elements may be insufficiently scattered among the input leading to partition skew in the second round. On the other hand, while it usually implies better partitioning, large samples could incur expensive overheads. We measure the *unevenness* of the partitions in TeraSort as a ratio of the maximum partition size to the optimal size m.

Tao et. al. [24] conclude that TeraSort is load-balanced within constant multiplicative factors and thus satisfies their minimality definition. However, the multiplicative factor in the proof is relatively large (16 to 32). Moreover, the proof itself does not extend to substantially smaller bounds. In this work, we seek a more accurate description of the performance of TeraSort. We give a series of bounds which describes the trade-off between the number of machines and the evenness of the partition (hence the worst case maximum workload on a single machine). We conclude that a larger number of available machines allows for more even partitions. In particular, we show that as t grows, the evenness approaches exactly m with high probability.

Initially, the n elements are distributed evenly across the machines, each storing m or $m+1$ elements. Parameterized by $\rho \in (0, 1]$, TeraSort runs as follows:

Map 1

Each element is selected into the sample S with probability ρ.

Reduce 1

S is sent to $\mathcal{M}_1$. $\mathcal{M}_1$ uses $\mathcal{A}$ to compute an ordered even t-partition of S made up of b_i, $i = 1, ..., t-1$. Each b_i is a *boundary element.*

Map 2

(Assume that b_i's have been broadcast to all machines.) Element x is sent to $\mathcal{M}_i$ if $b_{i-1} \leq x < b_i$, where $b_0 := -\infty$ and $b_n := +\infty$.

Reduce 2

On each machine, sort elements locally using $\mathcal{A}$.

In [24], it was shown that TeraSort is minimal when $\rho = \frac{1}{m} \ln nt$ using a detailed analysis of the *minimum footprint* and *bounded net-traffic* conditions. Note that the broadcast assumption in the algorithm may incur a network outflow of size $O(t^2)$ (or $O(t)$ depending on the size of message) at $\mathcal{M}_1$, which would make TeraSort non-minimal when t^2 is no longer $O(m)$. However, in practice the broadcast can be implemented in Hadoop as $\mathcal{M}_1$ writing to a shared file which is then read by all machines. This way, the broadcast cost is evenly distributed among machines. This is an approach that [24] follows as well. Furthermore, [24] shows in Section 3.3 that this is not a restrictive constraint because it can be overcome by additional techniques. Finally, the experimental analysis provided in [24] for *pure TeraSort* (their implementation of TeraSort with $\rho = \frac{1}{m} \ln nt$) exhibits very even partitions. Specifically, it can be observed that the load balancing ratio does not exceed a factor of 2 across all the datasets considered. Therefore, our results can be viewed as giving a sound theoretical explanation for the experimental observations in [24].

In the following sections, we prove that TeraSort is *strongly* minimal by showing that the *balanced partition* condition is satisfied, using an interesting bucketing argument.

4 A New Proof of TeraSort's Minimality

In this section we give a new proof of the results in [24] that prove the minimality of TeraSort, but using a different and a more extensible probabilistic method that is capable of proving that TeraSort is strongly minimal.

4.1 Probability Tools

Chernoff bounds restrict from above the tail probability of sums of independent Bernoulli random variables. There are several forms/variants of Chernoff bounds of similar restrictive power. In this work, we use the following form:

$$\Pr\left[\sum_{i=1}^{n} X_i > (1+\delta)\mu\right] \le \exp\left\{-\frac{\delta^2\mu}{\delta+2}\right\},\ \delta > 0$$
$$\Pr\left[\sum_{i=1}^{n} X_i < (1-\delta)\mu\right] \le \exp\left\{-\frac{\delta^2\mu}{2}\right\},\ 0 < \delta < 1$$

where the X_i's are independent and $X_i = 1, 0$ with probability $p_i, 1 - p_i$ respectively. The mean is $\mu = \sum_{i=1}^{n} p_i$.

4.2 Minimality

The minimality of TeraSort is equivalent to the following claims.

Claim 1. *In Map 1:* $|S| = O(m)$

Claim 2. *In Reduce 2: every machine ends up with* $O(m)$ *elements*

Claim 1 limits the size of the sample S and thus the amount of traffic that machines send and receive in the first round. Claim 2 limits the machine sizes and the network input in the second round, since the map phase of round 2 never violates minimality as long as every machine holds $O(m)$ elements at the beginning of the algorithm. In the following we show Claims 1 and 2 hold with high probability (at least $1 - O(\frac{1}{n})$). It is straightforward to show Claim 1.

Lemma 1. $\Pr[|S| > kn\rho] \le \left(\frac{1}{nt}\right)^t$ *when* $\rho \ge \frac{1}{m}\ln nt$ *and* $k \ge 3$.

Proof. $|S|$ is the sum of n Bernoulli random variables of probability ρ; $\mathbb{E}[|S|] = n\rho$. By Chernoff bounds, we have

$$\Pr[|S| > kn\rho] \le \exp\left\{-\tfrac{(k-1)^2}{k+1}n\rho\right\} \tag{1}$$

For the lemma to hold, we require that the exponent on the RHS be bounded above by $\left(\frac{1}{nt}\right)^t$. To show this, we take the minimal values satisfying the inequalities given in the lemma statement: $\rho = \frac{1}{m}\ln nt$ and $k = 3$. It is easy to verify that under these parameter settings the RHS is $\exp\left\{-n \cdot \frac{t}{n}\ln nt\right\} = \left(\frac{1}{nt}\right)^t$.

Theorem 1 (Claim 1). *By setting* $\rho \ge \frac{1}{m}\ln nt$ *and assuming* $m \ge t\ln nt$*, Claim 1 holds with probability* $1 - \left(\frac{1}{nt}\right)^t$.

Proof. By Lemma 1,

$$\Pr\left[|S| > 3m\right] \le \Pr\left[|S| > 3t\ln nt\right] \le \Pr\left[|S| > 3n\rho\right] \le \left(\frac{1}{nt}\right)^t$$

The event that Claim 1 holds is given by complement of the above. Therefore the claim holds with high probability.

In reality, typically $m \gg t$, namely, the memory size of a machine is significantly greater than the number of machines. More specifically, m is at the order of at least 10^6 (this is using only a few megabytes per machine), while t is at the order of 10^4 or lower. Therefore, $m \ge n\rho = t\ln(nt)$ is a (very) reasonable assumption, which explains why TeraSort has excellent efficiency in practice.

Next, we present our approach to the proof of Claim 2 in the form of a few interesting lemmas. First, we formulate a problem closely related to Claim 2.

Problem 1 (Sample-Partition). Let A denote a set of n elements from an ordered universe; a_j denotes the $(j+1)$-th smallest element in A. Construct a sample $S \subseteq A$ by independently picking each element with probability ρ. Let $b_1, b_2, ..., b_{t-1} \in S$ be the ordered even t-partition elements of S. **Question:** how evenly do the b_i's partition A?

Problem 1 captures the probabilistic structure of TeraSort. Clearly, the set of the elements on $\mathcal{M}_i$ in Reduce 2 is exactly $A \cap [b_{i-1}, b_i)$, independent of how the input dataset is spread across machines at the outset. An answer to Problem 1 that the partitions are all $O(m)$ in size proves Claim 2.

The approach of [24] is to suppose we have an ordered partition of A and refer to every partition as a *bucket*. It is easy to observe that if every bucket contains a boundary element, then the distance between any two adjacent boundary elements is less than the sum of the sizes of the buckets in which they exist. This observation will lead to Claim 2 if we additionally ensure that buckets are $O(m)$ in size.

We make a stronger observation: if we allow buckets to overlap with one another, we have a promise of shorter distances between adjacent boundary elements. Formally, consider an ordered even t-partition of A. Let $d(i)$ be the index in A of the i-th smallest partition element; and manually set $d(0) := 1$. Our notion of a bucket is defined by the intervals $I_j := [a_{d(j)}, a_{d(j)+lm})$. The variable $l \ge 0$ controls the length of the interval. The I_j's are well defined for all $j \ge 0$ with $d(j) + lm \le n - 1$. We cover the largest few elements with one additional interval $[a_{n-lm}, a_n]$. Notice that the intervals form a cover of A if $l \ge 1$ and under this condition there can be at most t intervals.

Lemma 2. *If every interval I_j has at least one boundary element, then no two boundary elements are more than $(l+1)m$ away from each other.*

Proof. We prove the contrapositive. Suppose that $|[b_i, b_{i+1}]| > (l+1)m$ for some $i \in \{1, t-1\}$, then there exists an interval $I_j \subseteq [b_i, b_{i+1}]$ which contains no boundary element.

Consider we start at b_i and walk towards larger elements in steps of size $(l+1)m$. Since $|[b_i, b_{i+1}]| > (l+1)m$, we must have not met another boundary elements yet. The interval after the one containing b_i starts at most m away from b_i (due to the spacing of intervals), so its end cannot pass the current element. Therefore, this interval contains no boundary element.

Now, we only need to put a ceiling on the probability that some interval contains no boundary element to conclude Claim 2. This is shown in Lemmas 3 and 4. First, Lemma 3 considers a more generalized notion in which we consider an *arbitrary* subset of A instead of the intervals of sequential elements considered in the definition of buckets.

Lemma 3. *Fix an arbitrary subset $B(x)$ of size x of A. Then $B(x) \cap S$ denotes the set of sampled elements in $B(x)$. With $\rho \geq \frac{1}{m} \ln nt$ and $l \geq 7$,*

$$\Pr\left[|B(lm) \cap S| < \frac{|S|}{t}\right] \leq \frac{1}{nt} + \left(\frac{1}{nt}\right)^t \tag{2}$$

Proof. Condition on the event $|S| > 3n\rho$ and decompose the probability as follows,

$$\begin{aligned}\Pr\left[|B(lm) \cap S| < \frac{|S|}{t}\right] &\leq \Pr\left[|B(lm) \cap S| < \frac{|S|}{t} \text{ and } |S| \leq 3n\rho\right] \\ &\quad + \Pr\left[|B(lm) \cap S| < \frac{|S|}{t} \text{ and } |S| > 3n\rho\right] \\ &\leq \Pr\left[|B(lm) \cap S| < \frac{3n\rho}{t}\right] + \Pr\left[|S| > 3n\rho\right]\end{aligned}$$

By Lemma 1 we can bound the second term in the last line above. Then, we apply Chernoff bounds to the first term of the same line noting that $\mathbb{E}\left[|B(lm) \cap S|\right] = \frac{lm}{n} \cdot n\rho = lm\rho$. We show an upper bound on the RHS by taking the minimal value satisfying the inequality given in the lemma statement: $l = 7$.

$$\begin{aligned}\Pr\left[|B(lm) \cap S| < \frac{|S|}{t}\right] &\leq \exp\left\{-\frac{(l-3)^2}{2l} m\rho\right\} + \left(\frac{1}{nt}\right)^t \\ &\leq \frac{1}{nt} + \left(\frac{1}{nt}\right)^t\end{aligned}$$

This bound proves the lemma.

Now, we are ready to bound the probability that a bucket does not cover a boundary element.

Lemma 4. *For any $0 \leq j \leq t-1$,*

$$\Pr\left[I_j \text{ has no boundary element}\right] \leq O\left(\frac{1}{n}\right) \tag{3}$$

Proof. In A, if a block of consecutive ordered elements contains no boundary element, then it must contribute no more than $\left\lceil \frac{|S|}{t} \right\rceil$ to the sample S (since a boundary element is taken every $\left\lceil \frac{|S|}{t} \right\rceil$ consecutive samples). Then, Lemma 3 gives us,

$$\begin{aligned}\Pr\left[I_j \text{ has no boundary element}\right] &\le \Pr\left[|B(lm)\cap S| < \left\lceil \frac{|S|}{t} \right\rceil\right] \\ &= \Pr\left[|B(lm)\cap S| < \frac{|S|}{t}\right] \\ &\le \frac{1}{nt} + \left(\frac{1}{nt}\right)^t\end{aligned}$$

By the union bound, with probability $1 - O\left(\frac{1}{n}\right)$ every interval covers at least one boundary element given $l \ge 7$.

As a result, we have that no bucket can fall between two consecutive boundary elements, hence every $A \cap [b_{i-1}, b_i)$ can contain objects in at most 2 buckets. So, by Lemma 2 and setting $l \ge 7$ we have that $\Pr\left[|A \cap [b_i, b_{i+1}]| \ge 8m\right] \le O\left(\frac{1}{n}\right)$. Finally, we have the following theorem,

Theorem 2 (Claim 2). *By setting $\rho \ge \frac{1}{m}\ln nt$, Claim 2 holds with probability at least $1 - O\left(\frac{1}{n}\right)$.*

5 Proof of TeraSort's Strong Minimality

A natural question is whether a fixed constant factor on m is the farthest we could go with the tools in hand. In other words, can we achieve better guarantees of evenness? Towards this goal, we consider the problem of Sample-Partition as a simple, generic routine potentially used for a variety of problems and so it is worthwhile in pushing the bound further.

Our new analysis is based on a re-examination of the proof of Lemmas 3 and 4. We focus on tighter bounds in Claim 2 (the evenness of partition), though as shown later the choice of parameters also ensures Claim 1.

Please note that in the statement of the following theorem, k and l are free parameters, for which we will later choose appropriate values that will satisfy conditions (4).

Theorem 3. *Given t and $\rho \ge \frac{1}{m}\ln nt$, for any choice of k and l satisfying the constraints below,*

$$\begin{cases}(k-1)^2(t-l) \ge (k+1) \\ ((t-1)l-(t-l)k)^2 \ge 2l \\ (t-1)l > (t-l)k \\ k,t > 1,\ l > 0\end{cases} \tag{4}$$

we have,

$$\Pr\left[|B(lm)\cap S| < \frac{|S|}{t}\right] \le \frac{2}{nt} \tag{5}$$

Proof. Let Y_j be the indicator random variable representing whether element $a_j \in A$ is sampled into S. Let $W(x)$ denote the sum of x independent $Bernoulli(\rho)$ random variables. Note, we require that there is no dependency between the random variables underlying two $W(\cdot)$ expressions.

$$\begin{aligned}
\Pr\left[|B(lm)\cap S| < \frac{|S|}{t}\right] &= \Pr\left[\sum_{j:a_j\in B(lm)} Y_j < \frac{1}{t}\sum_{j=0}^{n-1} Y_j\right] \\
&= \Pr\left[(t-1)\sum_{j:a_j\in B(lm)} Y_j < \sum_{j:a_j\notin B(lm)} Y_j\right] \\
&= \Pr\left[(t-1)W(lm) < W(n-lm)\right] \qquad (6)
\end{aligned}$$

To establish an upper bound on (6), we use the same decoupling technique as in Lemma 3 again. Let,

$$\begin{aligned}
W_1 &= W(lm) \\
W_2 &= W(n-lm)
\end{aligned}$$

we have,

$$\begin{aligned}
\Pr\left[(t-1)W(lm) < W(n-lm)\right] &\le \Pr\left[W_1 < \frac{W_2}{t-1},\ W_2 \le k(n-lm)\rho\right] \\
&\quad + \Pr\left[W_1 < \frac{W_2}{t-1},\ W_2 > k(n-lm)\rho\right] \\
&\le \Pr\left[W_1 < \frac{k(n-lm)\rho}{t-1}\right] + \Pr\left[W_2 > k(n-lm)\rho\right]
\end{aligned}$$

By Chernoff bounds, given $k > 1$ and $(t-1)l > (t-l)k$, we get

$$\Pr\left[W_2 > k(n-lm)\rho\right] \le \exp\left\{-\frac{(k-1)^2}{k+1}(t-l)m\rho\right\} \tag{7}$$

$$\Pr\left[W_1 < \frac{k(n-lm)\rho}{t-1}\right] \le \exp\left\{-\frac{lm\rho}{2}\left(1-\frac{(t-l)k}{(t-1)l}\right)^2\right\} \tag{8}$$

Next, we can bound the probability of each of W_1 and W_2 independently. We make the initial observation that $\rho \ge \frac{1}{m}\ln nt$ implies $m\rho \ge \ln nt$.

First, using the observation above, the RHS of (7)

$$\exp\left\{-\frac{(k-1)^2}{k+1}(t-l)m\rho\right\} \le \exp\left\{-\frac{(k-1)^2}{k+1}(t-l)\ln nt\right\} \tag{9}$$

Therefore, the probability that $W_2 > k(n-lm)\rho$ is less than $\frac{1}{nt}$ exactly when $\frac{(k-1)^2}{k+1}(t-l) \geq 1$, or $(k-1)^2(t-l) \geq (k+1)$. The resulting inequality yields the second set of constraints from (4).

Second, again using the observation above, the RHS of (8)

$$\exp\left\{-\frac{lm\rho}{2}\left(1-\frac{(t-l)k}{(t-1)l}\right)^2\right\} \leq \exp\left\{-\frac{l}{2}\left(1-\frac{(t-l)k}{(t-1)l}\right)^2 \ln nt\right\} \tag{10}$$

Therefore, the probability that $W_1 < \frac{k(n-lm)\rho}{t-1}$ is less than $\frac{1}{nt}$ exactly when $\frac{l}{2}\left(1-\frac{(t-l)k}{(t-1)l}\right)^2 \geq 1$, or $((t-1)l-(t-l)k)^2 \geq 2l$. The resulting inequality yields the third set of constraints from (4). Finally, we enforce the sanity conditions given in the third inequality from (4) to ensure that the RHS of (8) remains positive.

We remark that although Theorem 3 focuses on Claim 2, the system of equations given by (4) also ensures that Claim 1 holds, because $(k-1)^2(t-l) \geq (k+1)$ is a stronger condition than that required for Claim 1 to hold: $(k-1)^2 t \geq (k+1)$ (see Eq. 1).

Theorem 3 describes a family of bounds for $(l+1)m$ given admissible values for l, t and k. The choices of l and t dictates the trade-off between the evenness and number of partitions. For a fixed k, as t increases, lower l is accessible and therefore a greater number of partitions implies better evenness. First, we present the following corollary.

Corollary 1. *Let $0 < \epsilon < \frac{1}{2}$ be a parameter and set $k := 1 + \frac{1}{t^\epsilon}$ and $l := 1 + \frac{2}{t^\epsilon}$. Given $\rho \geq \frac{1}{m}\ln nt$, it holds that*

$$\Pr\left[|B(lm) \cap S| < \frac{|S|}{t}\right] \leq \frac{2}{nt} \tag{11}$$

It can be verified that the chosen values for k and l satisfy the system of equations given by (4) for $0 < \epsilon < \frac{1}{2}$ and t large enough. Furthermore, the system of equations given by (4) is always satisfied for large enough t given fixed ϵ, k and l. Therefore, we obtain arbitrarily strong evenness as long as t is allowed to be sufficiently large. Given our chosen value of l and Corollary 1 we arrive at the following.

Corollary 2. *With probability at least $1 - O(\frac{1}{n})$, the size of every partition is less than $(2 + \frac{2}{t^\epsilon})m$ where $0 < \epsilon < \frac{1}{2}$.*

In fact, it can be shown that for $\epsilon = \frac{1}{4}$, we can satisfy the system of equations given by (4) when $t > 10$. By combining all of above, we arrive at Corollary 3.

Corollary 3. *Given $\rho \geq \frac{1}{m}\ln nt$ and $t > 10$,*

$$\Pr\left[|A \cap [b_i, b_{i+1}]| > \left(2 + \frac{2}{\sqrt[4]{t}}\right)m\right] \leq O\left(\frac{1}{n}\right) \tag{12}$$

for $0 \leq i \leq t-1$ and where $b_0 := -\infty$ and $b_t := +\infty$.

Finally, we observe that a stronger version of Theorem 1 (Claim 1) exists by noting that plugging $k := 1 + \frac{1}{\sqrt[4]{t}}$ into Lemma 1 yields an analog to Theorem 1 that bounds the probability that $|S| > \left(1 + \frac{1}{\sqrt[4]{t}}\right) m$ to at most $O\left(\frac{1}{n}\right)$.

5.1 Tightening the Bound

Given the memory available in modern machines and typical values for t, we observe that probabilities of failure to produce strongly even partitions on the order of $O(\frac{1}{m})$ are negligible. This is reasonable since current main memories of computation nodes are on the order of gigabytes. If we consider m (the average workload) to be safely within the capacity of main memory (e.g. m is on the order of 2^{20}), then $\frac{1}{m}$ is very small.

Thus, we are able to further restrict the size of the intervals which corresponds to partitions that are more even. The trick is to construct the intervals $\{ I_j \}_j$ by placing the left endpoint of the interval $j+1$ a distance $\frac{m}{t}$ away from the left endpoint of interval j (see Figure 1). As a result,

- Lemma 2 has a stronger form: no two adjacent boundary elements are more than $(l + \frac{1}{t})$ away from each other.
- When we apply union bound with the at most t^2 intervals, we obtain a probability of failure no larger than $O(\frac{1}{nt}) \times t^2 = O(\frac{1}{m})$.

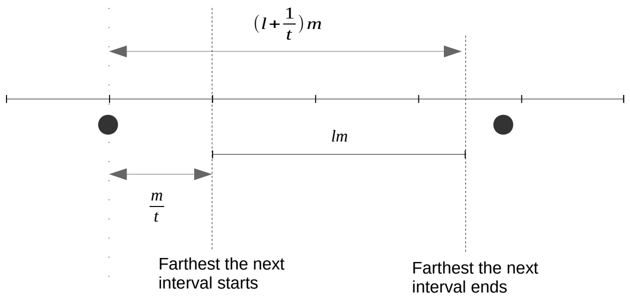
$(l+\frac{1}{t})m$
lm
$\frac{m}{t}$
Farthest the next interval starts
Farthest the next interval ends

Fig. 1. With $\frac{m}{t}$ spacing, Lemma 2 is strengthened to ensure adjacent boundary elements are within $(l + \frac{1}{t})$ of each other with high probability.

This argument applies to every bound shown in the previous section. We phrase the counterpart of Corollary 3 as an example.

Theorem 4. *Given $\rho \geq \frac{1}{m} \ln nt$ and $t > 10$,*

$$\Pr\left[|A \cap [b_i, b_{i+1}]| > \left(1 + \frac{2}{\sqrt[4]{t}} + \frac{1}{t}\right) m\right] \leq O\left(\frac{1}{m}\right) \tag{13}$$

for $0 \leq i \leq t-1$ and where $b_0 := -\infty$ and $b_t := +\infty$.

As a result, we see that as the number of machines increases, the evenness of the partitions approach its optimal value m with high probability.

Remark: Strongly Minimal Algorithms for Databases. As mentioned in the introduction, Tao et al. [24] show how a minimal algorithm for sorting leads to minimal algorithms for other database problems by using a single additional round after sorting. The problems considered include *ranking*, *group-by*, *semi-join*, and *2D skyline*. As a result of our analysis for TeraSort, these problems have strongly minimal MapReduce algorithms.

6 Conclusions

Despite the great variety of algorithms developed for MapReduce, few are able to achieve the ideal goal of parallelization: balanced workload across the participating machines, network traffic bounded by the total input data size, and a speedup over sequential algorithms linear in the number of machines available.

In this paper we introduce the new notion of *strongly minimal* MapReduce algorithms. Our definition strengthens the minimality criteria of minimal MapReduce algorithms as defined in [24]. Precisely, strongly minimal algorithms have partitions that approach the optimal evenness value of m as the number of machines t grows with high probability. We prove that the popular parallel sorting paradigm, TeraSort, is strongly minimal under the self-sampling strategy with a sampling rate $\rho = \frac{1}{m} \ln nt$. Additionally, this leads to strongly minimal algorithms that settle an array of important database problems.

Finally, our techniques are general and apply beyond the analysis of strongly minimal MapReduce algorithms to any setting that fits the Sample-Partition problem model; we show that given a sufficiently high sampling rate the approximate partitions obtained from self-sampling as the number of partitions increases approach the partition sizes obtained from an ideal partitioning with high probability. We believe that the refined bucketing arguments we use in our analysis are of independent interest and are likely to have other applications.

In future work, our goal is to continue this line of research and identify other fundamental problems that have strongly minimal algorithms.

References

1. F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Transactions on Knowledge and Data Engineering*, 23(9):1282–1298, 2011.
2. B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.
3. S. Behnezhad, M. Derakhshan, M. Hajiaghayi, and R. M. Karp. Massively parallel symmetry breaking on sparse graphs: Mis and maximal matching. *arXiv preprint arXiv:1807.06701*, 2018.

4. S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 975–986. ACM, 2010.
5. A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.
6. M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson. Cohadoop: flexible data placement and its exploitation in hadoop. *Proceedings of the VLDB Endowment*, 4(9):575–585, 2011.
7. A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689. ACM, 2011.
8. J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. *ACM Transactions on Algorithms (TALG)*, 6(4):66, 2010.
9. R. L. Ferreira Cordeiro, C. Traina Junior, A. J. Machado Traina, J. López, U. Kang, and C. Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 690–698. ACM, 2011.
10. A. Ghoting, P. Kambadur, E. Pednault, and R. Kannan. Nimble: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 334–342. ACM, 2011.
11. B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In *2012 IEEE 28th International Conference on Data Engineering*, pages 522–533. IEEE, 2012.
12. Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1199–1208. IEEE, 2011.
13. H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. SIAM, 2010.
14. L. Kolb, A. Thor, and E. Rahm. Load balancing for mapreduce-based entity resolution. In *2012 IEEE 28th international conference on data engineering*, pages 618–629. IEEE, 2012.
15. R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing (TOPC)*, 2(3):14, 2015.
16. Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 25–36. ACM, 2012.
17. S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 85–94. ACM, 2011.
18. Y. Lin, D. Agrawal, C. Chen, B. C. Ooi, and S. Wu. Llama: leveraging columnar storage for scalable join processing in the mapreduce framework. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 961–972. ACM, 2011.

19. A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 949–960. ACM, 2011.
20. O. OMalley. Terabyte sort on apache hadoop. *Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop. pdf,(May)*, pages 1–3, 2008.
21. B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437, 2009.
22. T. Roughgarden, S. Vassilvitskii, and J. R. Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *Journal of the ACM (JACM)*, 65(6):41, 2018.
23. S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*, pages 607–614. ACM, 2011.
24. Y. Tao, W. Lin, and X. Xiao. Minimal mapreduce algorithms. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 529–540. ACM, 2013.
25. C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 837–846. ACM, 2009.
26. X. Zhang, L. Chen, and M. Wang. Efficient multi-way theta-join processing using mapreduce. *Proceedings of the VLDB Endowment*, 5(11):1184–1195, 2012.