Upper and Lower Bounds on the Cost of a Map-Reduce Computation[*]

Foto N. Afrati[†], Anish Das Sarma[♯], Semih Salihoglu[‡], Jeffrey D. Ullman[‡]
[†]National Technical University of Athens, [♯]Google Research, [‡]Stanford University
afrati@softlab.ece.ntua.gr, anish.dassarma@gmail.com, semih@cs.stanford.edu, ullman@gmail.com

ABSTRACT

In this paper we study the tradeoff between parallelism and communication cost in a map-reduce computation. For any problem that is not "embarrassingly parallel," the finer we partition the work of the reducers so that more parallelism can be extracted, the greater will be the total communication between mappers and reducers. We introduce a model of problems that can be solved in a single round of map-reduce computation. This model enables a generic recipe for discovering lower bounds on communication cost as a function of the maximum number of inputs that can be assigned to one reducer. We use the model to analyze the tradeoff for three problems: finding pairs of strings at Hamming distance d, finding triangles and other patterns in a larger graph, and matrix multiplication. For finding strings of Hamming distance 1, we have upper and lower bounds that match exactly. For triangles and many other graphs, we have upper and lower bounds that are the same to within a constant factor. For the problem of matrix multiplication, we have matching upper and lower bounds for one-round map-reduce algorithms. We are also able to explore two-round map-reduce algorithms for matrix multiplication and show that these never have more communication, for a given reducer size, than the best one-round algorithm, and often have significantly less.

1. INTRODUCTION

We assume the reader is familiar with map-reduce [9] and its open-souce implementation Hadoop [27]. A brief summary can be found in Chapter 2 of [22]. There have been many custom solutions using a single round of map-reduce for specific problems, e.g., performing fuzzy joins [3, 26], clustering [8], graph analyses [2, 24], multiway join [1], and so on. Here, we develop techniques for analyzing problems of this type and optimizing the performance on any distributed computing environment by explicitly studying an inherent trade-off between *communication cost* and *parallelism*.

1.1 Communication and Parallelism for Map-Reduce

This paper offers a model that helps us analyze how suited problems are to a map-reduce solution. We focus on two parameters that represent the tradeoff involved in designing map-reduce algorithms.

First is the amount of communication between the map phase and the reduce phase. Often, but not always, the cost of communication is the dominant cost of a map-reduce algorithm. To represent the communication cost, we define and study *replication rate*. The replication rate of any map-reduce algorithm is the average number of key-value pairs that the mappers create from each input.

The second parameter is the "reducer size." A *reducer*, in the sense we use the term in this paper, is a reduce-key (one of the keys that can appear in the output of the mappers) together with its list of associated values, as would be delivered to a reduce-worker. *Reducer size* is the upper bound on how long the list of values can be. For example, we may want to limit a reducer to no more input than can be processed in main memory. A reduce-worker may be assigned many reduce-keys and works on them one at a time. The total computation cost of the reducers is the sum over all keys (or "reducers") of the computation cost of processing all the values associated with that key.[1]

Limiting reducer size also enables more parallelism. Small reducer sizes force us to redesign the notion of a "key" in order to allow more, smaller reducers, and thus allow more parallelism if enough compute nodes are available.

1.2 How the Tradeoff Can Be Used

Suppose we have determined that the best algorithms for a problem have replication rate r and reducer size q, where $r = f(q)$ for some function q. Look ahead to Fig. 1 for an example of what such a function f might look like. In particular, be aware that $f(q)$ usually grows as q shrinks. When we try to solve an instance of this problem on a particular cluster, we must determine the true costs of execution. For example, if we are running on EC2 [6], we pay particular amounts for communication and for rental of virtual processors. The communication cost is proportional to r; the constant of proportionality depends on the rate EC2 charges for communication and the size of our data. The cost of renting processors is some function of q.

EXAMPLE 1.1. *If the reducer must compare all pairs of its inputs (e.g., consider the Hamming-distance-based similarity join discussed later in Example 2.3), then the work at each reducer is $O(q^2)$, and the number of reducers is inversely proportional to q,*

[*]The work of Foto Afrati and Jeff Ullman was 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 Edu- cation and Lifelong Learning, under the program THALES.

Proceedings of the VLDB Endowment, Vol. 6, No. 4
Copyright 2013 VLDB Endowment 2150-8097/13/02... $ 10.00.

[1] Computation cost at the mappers is not treated separately, but is incorporated into the communication cost.

so the total processor cost is proportional to q. That is, the cost of solving this instance of our problem is $ar + bq$ for some constants a and b. Since $r = f(q)$, the cost is $af(q) + bq$. We find the value of q that minimizes this expression. That value tells us which of the algorithms lying along the curve $r = f(q)$ should be selected for this job.[2]

If we were concerned more with wall-clock time than with total computation cost, then we might add a term representing the execution time for a single reducer. In this hypothetical example, the time to compare $\binom{q}{2}$ pairs is $O(q^2)$, so we would minimize a function of the form $af(q) + bq + cq^2$.

Different problems will have different functions $r = f(q)$, and they will also have different functions of q that measure the computation cost. This function may not be the linear or quadratic functions suggested in Example 1.1. However:

- Deducing the proper function of q to represent the computation cost is not harder than analyzing, theoretically or experimentally, the running time of the serial algorithm that implements the reduce-function.

1.3 Outline of the Paper

There may be many ways to solve nontrivial problems in a single round of map-reduce. The more parallelism you want, the more communication overhead you face due to having to replicate inputs to many reducers. In this paper:

- We offer a simple model of how inputs and outputs are related. We show how our model can capture a varied set of problems (Section 2).
- We define the fundamental tradeoff between
  a) Reducer size: the maximum number of inputs that one reducer can receive, and
  b) Replication rate, or average number of key-value pairs to which each input is mapped by the mappers.
- We study three well-known problems: *Hamming Distance* (Section 3), *triangle finding* (Section 4) and some generalizations (Section 5), and *matrix multiplication* (Section 6). In each case there is a lower bound on the replication rate that grows as reducer size shrinks (and therefore as the parallelism grows). Moreover, we present algorithms that match these lower bounds for various reducer sizes.

Due to space constraints, complete proofs of some technical results are omitted from the submission, and are available in our online technical report [4].

1.4 Related Work

The present paper is the first work that addresses the tradeoff between reducer size and communication cost in one round map-reduce computations. In [19], the optimization of theta-join implementation by map-reduce was considered from a point of view similar to what we propose here. This paper considers only one special case of our model, where each output depends on only two inputs. An inherent trade-off between *communication cost* and *parallelism* has been studied in different contexts, e.g., pipelined parallelism [13]; we study this trade-off for one round map-reduce jobs.

The model of [15] proposes that a map-reduce algorithm should limit the input size of any reducer to be asymptotically smaller than the total amount of input. This idea is appropriate for eliminating trivial algorithms that really do all the work serially in one reducer and thus limits consideration to algorithms that we might think of as truly parallel.

[21] study the problem of matrix multiplication in map-reduce. They derive lower bounds on the number of map-reduce rounds required to multiply two dense or two sparse matrices as a function of different reducer sizes and cumulative memory available in the cluster.

Map-reduce differs from previous parallel-computation models (e.g., PRAM) in that it interleaves sequential and parallel computation. Thus the essential constraint on map-reduce comes not so much from the demand for parallelism, but from the limit on how much input we can expect a reducer to handle and how costly communication among processors is. For instance, if the input is small enough, then the optimal choice is to run everything at one compute node thus minimizing communication, regardless of the asymptotics of your algorithm.

There has been a lot of interest in handling skewed data in map-reduce (e.g., [17, 16]). The work closer to our setting is [16] where the authors propose a slight modification to the map-reduce computational framework to allow for small amount of communication among the mappers in order to decide how to handle skewed data. Handling skewed data is not the focus of our paper, but the need to deal with skewed data, will require alternative algorithms.

Our model for describing problems is closely related to the notion of data provenance [25]. There has also been some work [14, 20] on provenance in the context of distributed workflows, including map-reduce workflows.

2. THE MODEL

The model is simple yet powerful: We can develop some quite interesting and realistic insights into the range of possible map-reduce algorithms for a problem. For our purposes, a *problem* consists of:

1. Sets of *inputs* and *outputs*.
2. A *mapping* from outputs to sets of inputs. The intent is that each output depends on only the set of inputs it is mapped to.

There are two non-obvious points about this model:

- Inputs and outputs are hypothetical, in the sense that they are all the possible inputs or outputs that might be present in an instance of the problem. Any *instance* of the problem will have a subset of the inputs. We assume that an output is never made unless at least one of its inputs is present, and in many problems, we only want to make the output if *all* of its associated inputs are present.
- We need to limit ourselves to finite sets of inputs and outputs. Thus, a finite domain or domains from which inputs are constructed is essential, and a "problem" is really a family of problems, one for each choice of finite domain(s). We also require that there be a finite set of outputs associated with each choice of input domain(s). The values that these outputs can take may be a function of the inputs on which each output depends, and we do not need to specify the domain for the output in advance. Example 2.4 illustrates how the outputs can compute a function of their associated inputs.

2.1 Examples of Problems

In this section we offer several examples of common map-reduce problems and how they are modeled.

EXAMPLE 2.1. Natural join *of two relations $R(A, B)$ and $S(B, C)$. The inputs are tuples in R or S, and the outputs are tuples with schema (A, B, C). To make this problem finite, we need*

[2] Note that typically, $f(q)$ is monotonically decreasing in q, so there is a minimum at some finite value of q.

to assume finite domains for attributes A, B, and C; say there are N_A, N_B, and N_C members of these domains, respectively.

Then there are $N_A N_B N_C$ outputs, each corresponding to a triple (a, b, c). Each output is mapped to a set of two inputs. One is the tuple $R(a, b)$ from relation R and the other is the tuple $S(b, c)$ from relation S. The number of inputs is $N_A N_B + N_B N_C$.

Notice that in an instance of the join problem, not all the inputs will be present. That is, the relations R and S will be subsets of all the possible tuples, and the output will be those triples (a, b, c) such that both $R(a, b)$ and $S(b, c)$ are actually present in the input instance.

EXAMPLE 2.2. Finding triangles. *We are given a graph as input and want to find all triples of nodes such that in the graph there are edges between each pair of these three nodes. To model this problem, we need to assume a domain of size N for the nodes of the input graph. An output is thus a set of three nodes, and an input is a set of two nodes. The output $\{u, v, w\}$ is mapped to the set of three inputs $\{u, v\}$, $\{u, w\}$, and $\{v, w\}$. Notice that, unlike the previous and next examples, here, an output is a set of more than two inputs. In an instance of the triangles problem, some of the possible edges will be present, and the outputs produced will be those such that all three edges to which the output is mapped are present.*

EXAMPLE 2.3. Hamming distance 1. *The inputs are binary strings, and since domains must be finite, we shall assume that these strings have a fixed length b. There are thus 2^b inputs. The outputs are pairs of inputs that are at Hamming distance 1; that is, the inputs differ in exactly one bit. Hence there are $(b/2)2^b$ outputs, since each of the 2^b inputs is Hamming distance 1 from exactly b other inputs – those that differ in exactly one of the b bits. However, that observation counts every pair of inputs at distance 1 twice, which is why we must divide by 2.*

EXAMPLE 2.4. Grouping and aggregation. *This example illustrates how to deal with a problem where the outputs are more than "yes" or "no" responses to whether a given set of inputs exists. Here, each output depends on a large set of possible inputs, and the result of an output is calculated from those of its associated inputs that actually appear in the data set. Suppose we have a relation $R(A, B)$ and we want to implement group-by-and-sum:*

```
SELECT A, SUM(B)
FROM R
GROUP BY A;
```

We must assume finite domains for A and B. An output is a value of A, say a, chosen from the finite domain of A-values, together with the sum of all the B-values. This output is associated with a large set of inputs: all tuples with A-value a and any B-value from the finite domain of B. In any instance of this problem, we do not expect that all these tuples will be present for a given A-value, a, but (unlike the previous examples) as long as at least one of them is present there will be an output for this value a.

2.2 Mapping Schemas

In our discussion, we shall use the convention that q is the maximum number of inputs that can be sent to any one reducer.

A *mapping schema* for a given problem, with a given value of q, is an assignment of a set of inputs to each reducer, subject to the constraints that:

1. No reducer is assigned more than q inputs.
2. For every output, there is (at least) one reducer that is assigned all of the inputs for that output. We say such a reducer *covers* the output. This reducer need not be unique, and it is, of course, permitted that these same inputs are assigned also to other reducers.

The figure of merit for a mapping schema with a given reducer size q is the replication rate, which we defined to be the average number of reducers to which an input is mapped by that schema. Suppose that for a certain algorithm, the ith reducer is assigned $q_i \leq q$ inputs, and let I be the number of different inputs. Then the replication rate r for this algorithm is

$$r = \sum_{i=1}^{p} q_i / I$$

EXAMPLE 2.5. *To see one subtlety of the model, consider the canonical example of a map-reduce algorithm: word-count. In the standard formulation, inputs are documents, and the outputs are pairs consisting of a word w and a count of the number of times w appears among all the documents. The standard algorithm works as follows. The map function takes a document, breaks it into words, and for each word w, it generates a key-value pair $(w, 1)$. There is one reducer for each key (i.e., for each word), and the reduce-function sums the 1's in the list of values it is given for a word and thus computes the count for that word.*

It looks like there is a great deal of replication, because each input results in as many key-value pairs as there are words. However, this view is deceptive. We could just as well have thought of the inputs as the word occurrences themselves, and then each word occurrence results in exactly one key-value pair. That is, the replication rate is 1, independent of the limit q on reducer size.[3] *Since the replication rate is identically 1, there is no tradeoff at all between q and replication rate; i.e., the word-count problem is embarrassingly parallel, as we knew all along.*

We want to derive upper and lower bounds on the minimum possible r, as a function of q, for various problems, thus demonstrating the tradeoff between high parallelism (many small reducers, so q is small) and low overhead (total communication cost – measured by the replication rate).

2.3 Independence of Inputs in the Mappers

When we calculate bounds on the replication rate we pretend that we have an instance of the problem where all inputs over the given domain are present. This actually captures the nature of map-reduce computation. Normally, in a mapper, a map function turns input objects into key-value pairs independently, without knowing what else is in the input. Thus, we can take the assumption that the mapping schema assigns inputs to processors without reference to what inputs are actually present. Consequently, the replication rate r we calculate represents the expected communication if we multiply it by the number of inputs actually present, so r is a good measure of the communication cost incurred by any instance of the problem.

Further to this point, recall that q counts the number of potential inputs in a reducer, regardless of which inputs are actually present for an instance of the problem. However, on the assumption that inputs are chosen independently with fixed probability, we can expect the number of actual inputs at a reducer to be q times that probability, and there is a vanishingly small chance of significant deviation

[3] Technically, if q is smaller than the number of occurrences of a particular word, then this algorithm will not work at all. But there is little reason to chose a q that small.

for large q. If we know the probability of an input being present in the data is x, and we can tolerate q_1 real inputs at a reducer, then we can use $q = q_1/x$ to account for the fact that not all inputs will actually be present.

2.4 The Recipe for Lower Bounds

While upper bounds on r for all problems are derived using constructive algorithms, there is a generic technique for deriving lower bounds. Before proceeding to concrete lower bounds, we outline in this section the recipe that we use to derive all the lower bounds used in this paper.

1. **Deriving $g(q)$:** First, find an upper bound, $g(q)$, on the number of outputs a reducer can cover if q is the number of inputs it is given.
2. **Number of Inputs and Outputs:** Count the total numbers of inputs $|I|$ and outputs $|O|$.
3. **The Inequality:** Assume there are p reducers, each receiving $q_i \leq q$ inputs and covering $g(q_i)$ outputs. Together they cover all the outputs. That is:

$$\sum_{i=1}^{p} g(q_i) \geq |O| \tag{1}$$

4. **Replication Rate:** Manipulate the inequality from Equation 1 to get a lower bound on the replication rate, which is $\sum_{i=1}^{p} q_i/|I|$.

Note that the last step above may require clever manipulation to factor out the replication rate. We have noticed that the following "trick" is effective in Step (4) for all problems considered in this paper. First, arrange to isolate a single factor q_i from $g(q_i)$; that is:

$$\sum_{i=1}^{p} g(q_i) \geq |O| \Rightarrow \sum_{i=1}^{p} q_i \frac{g(q_i)}{q_i} \geq |O| \tag{2}$$

Assuming $\frac{g(q_i)}{q_i}$ is monotonically increasing in q_i, we can use the fact that $\forall q_i : q_i \leq q$ to obtain from Equation 2:

$$\sum_{i=1}^{p} q_i \frac{g(q)}{q} \geq |O| \tag{3}$$

Now, divide both sides of Equation 3 by the input size, to get a formula with the replication rate on the left:

$$r = \frac{\sum_{i=1}^{p} q_i}{|I|} \geq \frac{q|O|}{g(q)|I|} \tag{4}$$

Equation 4 gives us a lower bound on r. Thus, in summary, given a particular problem, we derive our lower bounds in this paper as follows:

- Suppose the instance of the problem has $|I|$ inputs and $|O|$ outputs.
- We find an upper bound, $g(q)$, on the number of outputs any q inputs can generate.
- If $g(q)/q$ is monotonically increasing in q then we can compute the replication rate using our recipe.
- Suppose the maximum number of inputs any reducer can take is q. Then the replication rate is $r \geq \frac{q|O|}{g(q)|I|}$.

2.5 Our Results

We summarize our results in two tables.

Table 1 gives the lower bounds for each problem we obtain. The table enumerates for each problem the total number of inputs $|I|$, number of outputs $|O|$, the upper bound $g(q)$ on the number of outputs q inputs can generate for each problem, and the lower bound we derived.

Table 2 gives the upper bound on the replication rate for each problem. In several cases our upper bounds are derived using multiple constructive algorithms, giving different upper bounds depending on the input parameters. Therefore, Table 2 only gives a representative upper bound for each problem, with a forward reference to the section in which more detailed results are present.

3. HAMMING DISTANCE 1

We begin with the tightest result we can offer. For the problem of finding pairs of bit strings of length b that are at Hamming distance 1, we have a lower bound on the replication rate r as a function of q, the maximum number of inputs assigned to a reducer. This bound is essentially best possible, as we shall point to a number of mapping schemas that solve the problem and have exactly the replication rate stated in the lower bound.

3.1 Bounding the Number of Outputs

As described in Section 2.4, our first task is to develop a tight upper bound on the number of outputs that can be covered by a reducer of size q.

LEMMA 3.1. *For the Hamming-distance-1 problem, a reducer of size q can cover no more than $(q/2)\log_2 q$ outputs.*

The proof of this result, obtained by induction on the length b of strings, is presented in our online technical report [4].

3.2 Lower Bound for Hamming Distance 1

We can use Lemma 3.1 to get a lower bound on the replication rate as a function of q, the maximum number of inputs at a reducer.

THEOREM 3.2. *For the Hamming-distance-1 problem with inputs of length b, the replication rate r is at least $b/\log_2 q$.*

PROOF. Suppose there are p reducers, and the ith reducer has $q_i \leq q$ inputs. We apply our four step recipe described in Section 2.4:

- **Deriving $g(q)$:** Recall that $g(q)$ is the maximum number of outputs a reducer can cover with q inputs. By Lemma 3.1, $g(q) = (q/2)\log_2 q$
- **Number of Inputs and Outputs:** There are 2^b bitstrings of length b. The total number of outputs is $(b/2)2^b$. Therefore $|I| = 2^b$ and $|O| = (b/2)2^b$.
- $\sum_{i=1}^{p} g(q_i) \geq |O|$ **Inequality:** Substituting for $g(q_i)$ and $|O|$ from above:

$$\sum_{i=1}^{p} \frac{q_i}{2}\log_2 q_i \geq \frac{b}{2}2^b \tag{5}$$

- **Replication Rate:** Finally we employ the manipulation trick from Section 2.4, where we arrange the terms of this inequality so that the left side is the replication rate. Recall we must separate a factor q_i from other factors involving q_i by replacing all other occurrences of q_i on the left by the upper bound q. That is, we replace $\log_2 q_i$ by $\log_2 q$ on the left of Equation 5. Since doing so can only increase the left side, the inequality continues to hold:

$$\sum_{i=1}^{p} \frac{q_i}{2}\log_2 q \geq \frac{b}{2}2^b \tag{6}$$

The replication rate is $r = \sum_{i=1}^{p} q_i/|I| = \sum_{i=1}^{p} q_i/2^b$. We can move factors in Equation 6 to get a lower bound on

Problem	$\lvert I\rvert$	$\lvert O\rvert$	$g(q)$	Lower bound on r
Hamming-Distance-1, b-bit strings	2^b	$\frac{b2^b}{2}$	$\frac{q\log_2 q}{2}$ (Section 3.1)	$\frac{b}{\log_2 q}$ (Section 3.2)
Triangle-Finding, n nodes	$\frac{n^2}{2}$	$\frac{n^3}{6}$	$\frac{\sqrt{2}}{3}q^{\frac{3}{2}}$ (Section 4.1)	$\frac{n}{\sqrt{2q}}$ (Section 4.1)
Sample Graphs (size s nodes) in Alon Class in graph of m edges, n nodes	$\binom{n}{2}$ or m	n^s	$q^{s/2}$ (Section 5.2)	$(\frac{n}{\sqrt{q}})^{s-2}$ or $(\sqrt{\frac{m}{q}})^{s-2}$ (Sections 5.2 and 5.3)
2-Paths in n-node graph	$\binom{n}{2}$	$\frac{n^3}{2}$	$\binom{q}{2}$ (Section 5.4.1)	$\frac{2n}{q}$ (Section 5.4.1)
Multiway Join: N bin. rels, m vars., Dom. n, parameter ρ from [7]	$N\binom{n}{2}$	$\binom{n}{m}$	q^ρ ([7])	$\frac{n^{m-2}}{q^{\rho-1}}$ (Section 5.5.1)
$n \times n$ **Matrix Multiplication**	$2n^2$	n^2	$\frac{q^2}{4n^2}$ (Section 6.1)	$\frac{2n^2}{q}$ (Section 6.1)

Table 1: Lower bound on replication rate r for various problems in terms of number of inputs $|I|$, number of outputs $|O|$, and maximum number of inputs per reducer q.

Problem	Upper bound on r
Hamming-Distance-1 b-bit strings	$\frac{b}{\log_2 q}$ (Section 3.3)
Triangle-Finding, n nodes	$\mathcal{O}(\frac{n}{\sqrt{2q}})$ (Section 4.2 and [2, 24])
Sample Graphs (size s nodes) in Alon Class in graph of m edges, n nodes	$\mathcal{O}((\sqrt{\frac{m}{q}})^{s-2})$ (Result from [2])
2-Paths in n-node graph	$\mathcal{O}(\frac{2n}{q})$ (Section 5.4.2)
Multiway Join: N rels, m vars., Dom. n (Section 5.5.2)	**Chain join:** $(n/\sqrt{q})^{N-1}$ **Star join:** fact, dim. sizes f, d_0: $\frac{Nd_0(Nd_0/q)^{N-1}}{f+Nd_0}$
$n \times n$ **Matrix Multiplication**	$\frac{2n^2}{q}$ for $q \geq 2n^2$ (Section 6.2 and [18])

Table 2: Representative upper bound on the replication rate r for each problem considered in this paper. This table only presents a representative upper bound, with a forward reference to the section that derives all upper bounds with constructive algorithms for each problem.

$r = \sum_{i=1}^{p} q_i/2^b \geq b/\log_2 q$, which is exactly the statement of the theorem.

□

3.3 Upper Bound for Hamming Distance 1

There are a number of algorithms for finding pairs at Hamming distance 1 that match the lower bound of Theorem 3.2. First, suppose $q = 2$; that is, every reducer gets exactly 2 inputs, and is therefore responsible for at most one output. Theorem 3.2 says the replication rate r must be at least $b/\log_2 2 = b$. But in this case, every input string w of length b must be sent to exactly b reducers – the reducers corresponding to the pairs consisting of w and one of the b inputs that are Hamming distance 1 from w.

There is another simple case at the other extreme. If $q = 2^b$, then we need only one reducer, which gets all the inputs. In that case, $r = 1$. But Theorem 3.2 says that r must be at least $b/\log_2(2^b) = 1$.

In [3], there is an algorithm called Splitting that, for the case of Hamming distance 1 uses $2^{1+b/2}$ reducers, for some even b. Half of these reducers, or $2^{b/2}$ reducers correspond to the $2^{b/2}$ possible bit strings that may be the first half of an input string. Call these *Group I reducers*. The second half of the reducers correspond to the $2^{b/2}$ bit strings that may be the second half of an input. Call these *Group II reducers*. Thus, each bit string of length $b/2$ corresponds to two different reducers.

An input w of length b is sent to 2 reducers: the Group-I reducer that corresponds to its first $b/2$ bits, and the Group-II reducer that corresponds to its last $b/2$ bits. Thus, each input is assigned to two reducers, and the replication rate is 2. That also matches the lower bound of $b/\log_2(2^{b/2}) = b/(b/2) = 2$. It is easy to observe that every pair of inputs at distance 1 is sent to some reducer in common. These inputs must either agree in the first half of their bits, in which case they are sent to the same Group-I reducer, or they agree on the last half of their bits, in which case they are sent to the same Group-II reducer.

We can generalize the Splitting Algorithm so that for any $c > 2$ such that c divides b evenly, we can have reducer size $2^{b/c}$ and replication rate c. Note that for reducer size $2^{b/c}$, the lower bound on the replication rate is exactly $b/\log_2(2^{b/c}) = c$. We split each bit string w into c segments, $w_1w_2\cdots w_c$, each of length b/c. We will have c groups of reducers, numbered 1 through c. There will be $2^{b-b/c}$ reducers in each group, corresponding to each of the $2^{b-b/c}$ bit strings of length $b - b/c$. For $i = 1, ..., c$, we map w to the Group-i reducer that corresponds to bit string $w_1\cdots w_{i-1}w_{i+1}\cdots w_c$, that is, w with the ith substring w_i removed. Thus, each input is sent to c reducers, one in each of the c groups, and the replication rate is c. Finally, we need to argue that the mapping schema solves the problem. Any two strings u and v at Hamming distance 1 will disagree in only one of the c segments of length b/c, and will agree in every other segment. If they disagree in their ith segments, then they will be sent to the same Group-i reducer, because we map them to the Group-i reducers ignoring the values in their ith segments. Thus, this Group-i reducer will cover the output pair (u, v).

Figure 1 illustrates what we know. The hyperbola is the lower bound. Known algorithms that match the lower bound on replication rate are shown with dots.

3.4 An Algorithm for Large q

The lower bound in Fig. 1 is matched for many values of q as long as $\log_2 q \leq b/2$. However, what happens between $b/2$ and b is less clear. Surely $r \leq 2$ for that entire range. In this subsection

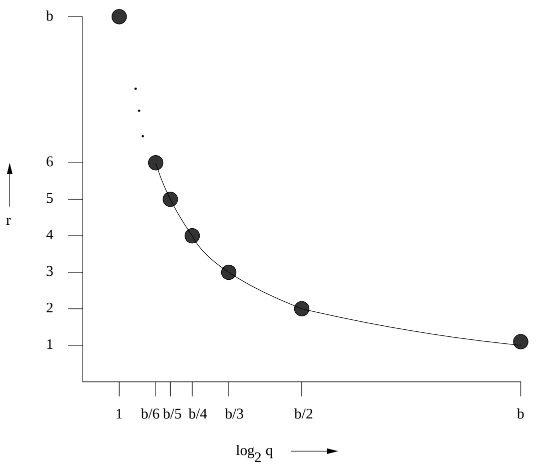

Figure 1: Known algorithms matching the lower bound on replication rate

and the next we shall show that there are algorithms for $\log_2 q$ near b with replication rates strictly less than 2.

There is a family of algorithms that use reducers with large input – q well above $2^{b/2}$, but lower that 2^b. The simplest version of these algorithms divides bit strings of length b into left and right halves of length $b/2$ and organizes them by weights, as suggested by Fig. 2. The *weight* of a bit string is the number of 1's in that string. In detail, for some k, which we assume divides $b/2$, we partition the weights into $b/(2k)$ groups, each with k consecutive weights. Thus, the first group is weights 0 through $k-1$, the second is weights k through $2k-1$, and so on. The last group has an extra weight, $b/2$, and consists of weights $\frac{b}{2}-k$ through $b/2$.

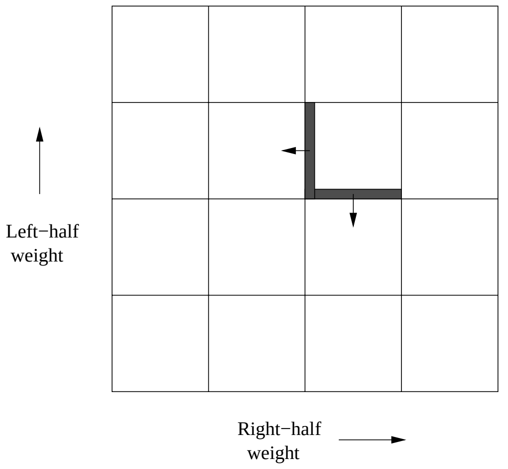

Figure 2: Partitioning by weight. Only the border weights need to be replicated

There are $(\frac{b}{2k})^2$ reducers; each corresponds to a range of weights for the first half and a range of weights for the second half. A string is assigned to reducer (i,j), for $i,j = 1,2,\ldots,b/2k$ if the left half of the string has weight in the range $(i-1)k$ through $ik-1$ and the right half of the string has weight in the range $(j-1)k$ through $jk-1$.

Consider two bit strings w_0 and w_1 of length b that differ in exactly one bit . Suppose the bit in which they differ is in the left half, and suppose that w_1 has a 1 in that bit. Finally, let w_1 be assigned to reducer R. Then unless the weight of the left half of w_1 is the lowest weight for the left half that is assigned to reducer R, w_0 will also be at R, and therefore R will cover the pair $\{w_0, w_1\}$. However, if the weight of w_1 in its left half is the lowest possible left-half weight for R, then w_0 will be assigned to the reducer with the same range for the right half, but the next lower range for the left half. Therefore, to make sure that w_0 and w_1 share a reducer, we need to replicate w_1 at the neighboring reducer that handles w_0. The same problem occurs if w_0 and w_1 differ in the right half, so any string whose right half has the lowest possible weight in its range also has to be replicated at a neighboring reducer. We suggested in Fig. 2 how the strings with weights at the two lower borders of the ranges for a reducer need to be replicated at a neighboring reducer.

Now, let us analyze the situation, including the maximum number q of inputs assigned to a reducer, and the replication rate. For the bound on q, note that the vast majority of the bit strings of length n have weight close to $n/2$. The number of bit strings of weight exactly $n/2$ is $\binom{n}{n/2}$. Stirling's approximation [10] gives us $2^n/\sqrt{2\pi n}$ for this quantity. That is, one in $O(\sqrt{n})$ of the strings have the average weight.

If we partition strings as suggested by Fig. 2, then the most populous $k \times k$ cell, the one that contains strings with weight $b/4$ in the first half and also weight $b/4$ in the second half, will have no more than

$$k^2\Big(\frac{2^{b/2}}{\sqrt{2\pi(b/2)}}\Big)^2 = \frac{k^2 2^b}{\pi b}$$

strings assigned.[4] If k is a constant, then in terms of the horizontal axis in Fig. 1, this algorithm has $\log_2 q$ equal to $b - \log_2 b$ plus or minus a constant. It is thus very close to the right end, but not exactly at the right end.

For the replication rate of the algorithm, if k is a constant, then within any cell there is only a small ratio of variation, among all pairs (i,j) assigned to that cells, of the numbers of strings with weights i and j in the left and right halves, respectively. Moreover, when we look at the total number of strings in the borders of all the cells, the differences average out, so the total number of replicated strings is very close to $(2k)/k^2 = 2/k$. That is, a string is replicated if either its left half has a weight divisible by k or its right half does. Note that strings in the lower-left corner of a cell are replicated twice, strings of the other $2k-2$ points on the border are replicated once, and the majority of strings are not replicated at all. We conclude that the replication rate is $1 + \frac{2}{k}$.

3.5 Generalization to d Dimensions

The algorithm of Section 3.4 can be generalized from 2 dimensions to d dimensions. Break bit strings of length b into d pieces of length b/d, where we assume d divides b. Each string of length b can thus be assigned to a cell in a d-dimensional hypercube, based on the weights of each of its d pieces. Assume that each cell has side k in each dimension, where k is a constant that divides b/d.

[4] Note that many of the cells have many fewer strings assigned, and in fact a large fraction of the strings have weights within $\sqrt{b}$ of $b/4$ in both their left halves and right halves. In the best implementation, we would combine the cells with relatively small population at a single compute node, in order to equalize the work at each node.

The most populous cell will be the one that contains strings where each of its d pieces has weight $b/(2d)$. Again using Stirling's approximation, the number of strings assigned to this cell is

$$k^d\Big(\frac{2^{b/d}}{\sqrt{2\pi b/d}}\Big)^d = \frac{k^d 2^b}{b^{d/2}(2\pi/d)^{d/2}}$$

On the assumption that k is constant, the value of $\log_2 q$ is

$$b - (d/2)\log_2 b$$

plus or minus a constant.

To compute the replication rate, observe that every point on each of the d faces of the hypercube that are at the low ends of their dimension must be replicated. The number of points on one face is k^{d-1}, so the sum of the volumes of the faces is dk^{d-1}. The entire volume of a cell is k^d, so the fraction of points that are replicated is d/k, and the replication rate is $1+d/k$. Technically, we must prove that the points on the border of a cell have, on average, the same number of strings as other points in the cell. As in Section 3.4, the border points in any dimension are those whose corresponding substring has a weight divisible by k. As long as k is much smaller than b/d, this number is close to $1/k$th of all the strings of that length.

3.6 Larger Hamming Distances

Unfortunately, the analysis for Hamming distance 1 does not generalize easily to higher distances. To see why, consider Hamming distance 2. While for Hamming distance 1 we learned that there is an $O(q \log q)$ upper bound on the number of outputs covered by a reducer with q inputs, for distance 2 this bound is much higher: $\Omega(q^2)$, at least for small q. That prevents us from getting a good lower bound on replication rate.

The $\Omega(q^2)$ bound comes from an algorithm from [3] called "Ball-2" that creates one reducer for each string of length b. For distance 2, this algorithm assigns to the reducer for string s all those strings at distance 1 from s. Notice that all distinct strings at distance 1 from s are distance 2 from each other. Thus, each reducer covers $\binom{b}{2}$ outputs. Since $q = b$, each reducer covers $\binom{q}{2}$, or about $b^2/2$ outputs.

On the other hand, we can generalize the upper bound of Section 3.3 to distance d. We divide the b bits of input strings into k equal-length pieces. A reducer corresponds to a choice of d of the k pieces to delete and a bit string of length $b(1-d/k)$ corresponding to the $k-d$ pieces of a string that are not deleted.. An input s is sent to $\binom{k}{d}$ reducers – those corresponding to the strings we obtain by deleting d of the k segments of string s. Thus, the replication rate is approximately $k^d/d!$, assuming k is much larger than d. Again using Stirling's approximation for the factorial, this replication rate is approximately $r = (ek/d)^d$.

4. TRIANGLE FINDING

We shall now consider the problem of finding triangles, introduced in Example 2.2. We shall first derive a lower bound assuming that all possible edges in the data graph can be present. That assumption follows our model, since we assume every possible output can be made, and every possible input could be present. However, applications of triangle-finding, such as in analysis of communities in social networks are generally applied to large but sparse graphs. As a result, we shall continue the analysis by showing how to adjust the bound q on reducer size to take into account the fact that most inputs will not be present. When we make this adjustment, we see that the lower bound we get matches, to within a constant factor, the upper bound obtained from known algorithms.

4.1 Lower and Upper Bound for Finding Triangles

Recall that, as described in Example 2.2, the inputs are the possible edges of a graph, and the outputs are the triples of edges that form a triangle. Suppose n is the number of nodes of the input graph. Following the recipe from Section 2.4:

- **Deriving $g(q)$:** We claim a reducer with q inputs can cover at most $\frac{\sqrt{2}}{3}q^{3/2}$ outputs (triangles), which happens when the reducer is sent all the edges among a set of $k = \sqrt{2q}$ nodes. This point was proved, to within an order of magnitude in [24], who in turn credit the thesis of Schank [23].[5] Suppose we assign to a reducer all the edges among a set of k nodes. Then there are $\binom{k}{2}$ edges assigned to this reducer, or approximately $k^2/2$ edges. Since this quantity is q, we have $k = \sqrt{2q}$. The number of triangles among k nodes is $\binom{k}{3}$, or approximately $k^3/6$ outputs. In terms of q, the upper bound on the number of outputs is $\frac{\sqrt{2}}{3}q^{3/2}$.
- **Number of Inputs and Outputs:** The number of inputs is $\binom{n}{2}$ or approximately $n^2/2$. The number of outputs is $\binom{n}{3}$, or approximately $n^3/6$.
- **$\sum_{i=1}^{p} g(q_i) \geq |O|$ Inequality:** So using the formulas from (1) and (2), if there are p reducers each with $\leq q$ inputs:

$$\sum_{i=1}^{p} \frac{\sqrt{2}}{3} q_i^{3/2} \geq n^3/6 \tag{7}$$

We can replace a factor of $\sqrt{q_i}$ on the left of Equation 7 by $\sqrt{q}$, since $q \geq q_i$, and then move that factor to the denominator of the right side. Thus,

$$\sum_{i=1}^{p} \frac{\sqrt{2}}{3} q_i \geq n^3/6\sqrt{q} \tag{8}$$

- **Replication Rate:** The replication rate is $\sum_{i=1}^{p} q_i$ divided by the number of inputs, which is $n^2/2$ from (1). We can manipulate Equation 8 as per the trick in Section 2.4 to get

$$r = \frac{2\sum_{i=1}^{p} q_i}{n^2} \geq \frac{n}{\sqrt{2q}}$$

Upper Bound: There are known algorithms that, to within a constant factor, match the lower bound on replication rate. See [24] and [2]. These algorithms are stated in terms of the number of edges, m, rather than the number of nodes, n. However, for the case $m = \binom{n}{2}$, which is what we assume when we consider all possible edges and triangles, these algorithms do in fact imply a replication rate that is $O(n/\sqrt{q})$. We shall next consider how to modify the analysis on the assumption that the true input will consist of m randomly chosen edges.

4.2 Analysis for Sparse Data Graphs

The lower bound $r = \Omega(n/\sqrt{q})$ holds on the assumption that all edges are actually present in the input. But as we pointed out, commonly the data graph to which a triangle-finding algorithm is applied is sparse. We shall show that, with essentially the same limitation q on the number of edges that any reducer must deal with, the lower bound on replication rate can be transformed to $r = \Omega(\sqrt{m/q})$.

[5] What is actually proved is that among q edges, you can form at most $O(q^{3/2})$ triangles. However, picking a set of nodes and all edges among them will match this upper bound.

Suppose the data graph has m of the possible $\binom{n}{2}$ edges, and that these edges are chosen randomly. Then if we want no more than an expected value of q for the number of edges input to any one reducer, we can actually assign a "target" $q_t = qn(n-1)/2m$ of the possible edges to one reducer and know that the expected number of edges that will actually arrive will be q.

We already know from Section 4.1 that if we assign at most q_t of the $\binom{n}{2}$ possible edges to any reducer, then the replication rate r is $\Omega(n/\sqrt{q_t})$. But on the assumption that only m edges are truly present in the input, q_t is $O(qn^2/m)$, from which we can conclude

$$r = \Omega(n/\sqrt{qn^2/m}) = \Omega(\sqrt{m/q})$$

This lower bound is met (to within a constant factor) by the algorithms of [24] and [2] when we measure reducer size in terms of the number of edges m (as these papers do), rather than in terms of the number of possible edges $\binom{n}{2}$. There is a natural concern that a random selection of the edges will cause more than q actual edges to be assigned to some of the reducers. However, we are only claiming bounds to within a constant factor, and by lowering the target q_t by, say, a factor of 2, we can make the probability that one or more reducers will get more than q actual edges as low as we like for large n and m.

5. FINDING INSTANCES OF OTHER GRAPHS

The analysis of Section 4 extends to any *sample graph* whose instances we want to find in a larger *data graph*. For each problem of this type, the sample graph is fixed, while the data graph is the input. Previously, we looked only at the triangle as a sample graph, but we could similarly search for cycles of some length greater than 3, or for complete graphs of a certain size, or any other small graph whose instances in the data graph we wanted to find.

5.1 The Alon Class of Sample Graphs

In [5], Noga Alon analyzed the maximum number of occurrences of a sample graph that could occur in a data graph of n nodes and m edges. In particular, he defined a class of graphs, which we shall call the *Alon class* of sample graphs. These graphs have the property that we can partition the nodes into disjoint sets, such that the subgraph induced by each partition is either:

- A single edge between two nodes, or
- Contains an odd-length Hamiltonian cycle.

The sample graph may have any other edges as well. The Alon class is very rich. Every cycle, every graph with a perfect matching, and every complete graph is in the Alon class. Paths of odd length are also in the Alon class, since we may use alternating edges along the path as a decomposition. However, paths of even length are not in the Alon class, since there are no cycles of any length, and the odd number of nodes cannot be partitioned into disjoint edges.

5.2 Lower Bound for the Alon Class

The key result from [5] that we need is that for any sample graph S in the Alon class, if S has s nodes, then the number of instances of S in a graph of m edges is $O(m^{s/2})$. So if the ith reducer has q_i inputs, the number of instances of S that it can find is $O(q_i^{s/2})$. But if all edges are present, the number of instances of S is $\Omega(n^s)$. Note the number of instances need not be exactly n^s, since there may be symmetries in S as we saw for the case of the triangle. However, there are surely at least $n^s/s!$ distinct sets of nodes that form the sample graph S.

Now, we can repeat the analysis we did for the triangle. If there are p reducers, and the ith reducer has q_i inputs, then

$$\sum_{i=1}^{p} q_i^{s/2} = \Omega(n^s)$$

If q is an upper bound on q_i, we can write the above as

$$\sum_{i=1}^{p} q_i q^{(s/2)-1} = \Omega(n^s)$$

The number of inputs is $\binom{n}{2}$. Thus, the replication rate r is

$$r = \frac{\sum_{i=1}^{p} q_i}{\binom{n}{2}} = \Omega(n^{s-2}/q^{(s-2)/2}) = \Omega\big((n/\sqrt{q})^{s-2}\big)$$

5.3 Bounds in Terms of Edges

As we did for triangles, we can scale q up by a factor of n^2/m if we assume that the actual data is m out of the $\binom{n}{2}$ possible edges. If we do so, the lower bound on r becomes

$$r = \Omega\Big(\big(n/\sqrt{(qn^2/m)}\big)^{s-2}\Big) = \Omega\big((\sqrt{m/q})^{s-2}\big)$$

The algorithm given in [2] matches this lower bound, to within a constant factor.

5.4 Paths of Length Two

The analysis for sample graphs not in the Alon class is harder, and we shall not try to give a general rule. However, to see the problems that arise, we will look at the simplest non-Alon graph: the path of length 2 (*2-paths*). The problem of finding 2-paths is similar, although not identical to, the problem of computing a natural self join

$$E(A, B) \bowtie E(B, C)$$

The difference is that the edge relation E contains sets of two nodes, rather than ordered pairs. That is, if a tuple (u, v) is in E, when finding 2-paths we can treat it as (v, u), even if the latter tuple is not found in E.

5.4.1 Lower Bound

We again follow the recipe from Section 2.4 to obtain a lower bound:

- **Deriving $g(q)$:** Any two distinct edges can be combined to form at most one 2-path. Thus, the number of outputs (2-paths) covered by this reducer is at most $\binom{q}{2}$ or approximately $q^2/2$.
- **Number of Inputs and Outputs:** $|I|$ is $\binom{n}{2}$ or approximately $n^2/2$. For counting $|O|$, observe that there are $\binom{n}{3}$ sets of three nodes, and any three nodes can form a 2-path in three ways. That is, any of the three nodes can be chosen to be the middle node. Thus, $|O|$ is $3\binom{n}{3}$, or approximately $3n^3/6 = n^3/2$.
- **$\sum_{i=1}^{p} g(q_i) \geq |O|$ Inequality:** Using the formulas from (1) and (2), if there are p reducers each with $\leq q$ inputs:

$$\sum_{i=1}^{p} q_i^2/2 \geq n^3/2 \tag{9}$$

Replacing a factor of q_i by q on the left:

$$\sum_{i=1}^{p} (q_i)(q/2) \geq n^3/2 \tag{10}$$

- **Replication Rate:** We rearrange terms in Equation 10 to make the left side equal to $\sum_{i=1}^{p} q_i$ divided by $|I| = n^2/2$.

$$r = \frac{\sum_{i=1}^{p} q_i}{n^2/2} \geq 2n/q$$

This lower bound on replication rate is unlike those we have seen before. For small q it makes sense, but for $q > 2n$ it is less than 1, which is useless. Rather, it should be replaced by the trivial lower bound $r \geq 1$ for large n. Once we make this replacement, the bound is tight for an infinite number of pairs of q and n. If $q = n^2/2$, then we can send all edges to one reducer and do the work there, so $r = 1$ is correct.

5.4.2 *Upper Bound*

If $q = n$, then we can have one reducer for each node. We send the edge (a, b) to the reducers for its two nodes a and b. The replication rate is thus 2, which agrees with the lower bound. The reducer for node u receives all edges consisting of u and another node, so it can put them together in all possible ways and produce all 2-paths that have u as the middle node.

If $q < n$, we have to divide the task of producing the 2-paths with middle node u among several different reducers. That means every pair of edges with u as one end has to be assigned to some reducer in common. Suppose for convenience that k^2 divides n. Suppose h is a hash function that divides the n nodes into k equal-sized buckets. The reducers will correspond to pairs $[u, \{i, j\}]$, where u is a node (intended to be the node in the middle of the 2-path), and i and j are bucket numbers in the range $1, 2, \ldots, k$. There are thus $n\binom{k}{2}$ or approximately $nk^2/2$ reducers.

Let (a, b) be an edge. We send this edge to the $2(k-1)$ reducers $[b, \{h(a), *\}]$ and $[a, \{*, h(b)\}]$, where $*$ denotes any bucket number from 1 to k other than the other bucket number in the set. We claim that any 2-path is covered by at least one reducer. In particular, look at the reducer $[u, \{i, j\}]$. This reducer covers all 2-paths $v - u - w$ such that $h(v)$ and $h(w)$ are each either i or j. Note that if $h(v) = h(w)$, then many reducers will cover this 2-path, and we want only one to produce it. So we let the reducer $[u, \{i, j\}]$ produce the 2-path $v - u - w$ if either

- One of $h(v)$ and $h(w)$ is i and the other is j, or
- $h(v) = h(w) = i$ and $j = i+1$ modulo k (i.e., $j = i+1 \leq k$ or $i = k$ and $j = 1$).

Each reducer receives $q = 2n/k$ edges, and as mentioned, the replication rate r is $2(k-1)$, or approximately $2k$. Since $2n/q = k$, the lower bound is approximately half what this algorithm achieves. Thus, to within a constant factor, the upper and lower bounds match for small q as well as for large q (where both bounds are between 1 and 2).

5.5 Multiway Join

We begin by looking at the join of several binary relations. We can think of this extension as looking for sample graphs in a data graph with labeled edges; the relation names are the edge labels. Suppose n is the number of nodes of the data graph. The inputs are the possible edges of a graph, and the outputs are the sets of s edges that make the body of the multiway join true (i.e., the s labeled edges of the sample graph). We assume also that the multiway join seen as a Datalog rule, (or as a hypergraph) uses m variables (m attributes/nodes in the hypergraph equivalently).

5.5.1 *A Lower Bound for Multiway Join*

Following the recipe from Section 2.4:

- **Deriving $g(q)$:** According to [7] when we have q inputs in a multiway join, then we can have at most $g(q) = q^\rho$ outputs where ρ is the *fractional edge cover number*, a parameter that depends on properties of the hypergraph associated with the specific multiway join. E.g., if all hyperedges have the same number of attributes/nodes and there are ρ_1 edges that cover all the nodes exactly once, and moreover this is the minimum number of edges with this property, then $\rho = \rho_1$. Otherwise, ρ comes from the solution of a linear program that is associated to the hypergraph (see also Subsection 5.5.3). From here on, we drop constant factors, but do not use the implied big-oh notation, for simplicity.
- **Number of Inputs and Outputs:** $|I| = s\binom{n}{2}$ or on the order of n^2. $|O| = \binom{n}{m}$ or on the order of n^m. Note that m here is a constant, so in big-oh calculations we can drop the factor $1/m!$ when approximating binomial coefficients.
- $\sum_{i=1}^{p} g(q_i) \geq |O|$ **Inequality:** Replacing for $g(q)$ and $|O|$ from above:

$$\Sigma_{i=1}^{p} q_i^\rho \geq n^m \tag{11}$$

We can replace a factor of $q_i^{\rho-1}$ on the left of Equation 11 by $q^{\rho-1}$, since $q \geq q_i$, and then move that factor to the denominator of the right side. Thus,

$$\Sigma_{i=1}^{p} q_i \geq n^m/q^{\rho-1} \tag{12}$$

- **Replication Rate:** The replication rate is $\sum_{i=1}^{p} q_i$ divided by the number of inputs, which is n^m from (1). We can manipulate Equation 12 as per the trick in Section 2.4 to get

$$r = \frac{\sum_{i=1}^{p} q_i}{n^2} \geq \frac{n^{m-2}}{q^{\rho-1}}$$

Below we discuss in detail some algorithms from the literature [1] that offer upper bounds that match or are close to the lower bound for cases of multiway join.

5.5.2 *Compare Upper and Lower Bounds for Cases of Multiway Join*

Chains of odd number of relations. Suppose we have N relations in the chain and N is an odd positive integer. Then, let us compute more carefully the above lower bound. We have now $m = N + 1$ and $\rho = (N + 1)/2$. Hence the lower bound is:

$$r \geq \frac{n^{N-1}}{q^{(N+1)/2-1}} = (n/\sqrt{q})^{N-1}$$

Observe that a chain of odd number of relations with all relations same size has the same lower bound as the lower bound we found for finding patterns graphs that are contained in Alon's class. This is to be expected since finding a pattern graph can be viewed as computing a multiway join[2].

For the upper bound we use the results in [1]. This paper computes the communication cost for when we have p reducers (denoted k in [1]), each relation has size R and there are N relations in the join, hence the total input size is $|I| = RN$. In [1] the expression that gives the communication cost is given as the sum of N terms, each a product of "shares" for some of the attributes; a *share* for an attribute is the number of ways the values for that attribute are to be hashed. The reducers correspond to vectors of a hash value for each attribute apppearing in any relation schema. When we optimize the shares for each of the variables, we get communication cost per input (hence replication rate) to be equal to: $r = p^{\frac{N-1}{N+1}}$.

After similar arithmetic manipulations as in previous sections, we get an upper bound on the replication rate to be:

$$r = (n/\sqrt{q})^{N+1}$$

If $\sqrt{q}$ is equal to n then the two bounds trivially match. If q is equal to n then the two bounds differ by a factor of n; in this case the lower bound is $n^{(N-1)/2}$, thus for large N the difference between the two bounds is getting smaller as a ratio of the factor n over $n^{(N-1)/2}$. The case for even number of relations in a chain query is similar.

Star joins. A star join joins a central *fact table* with several *dimension tables*. It is expected that the fact table is very large while the dimension tables are smaller. Suppose the size of the fact table is f, and all dimension tables have the same size d_0. Then according to [1], in order to minimize the communication cost, the share for the attributes not in the fact table is 1 (i.e., their values are not used when naming the reducers), while the share for each attribute in the fact table is $d_0p^{1/N}/d_0 = p^{1/N}$; here N is the number of dimension tables and p is the number of reducers. We assume, moreover, that dimension tables pairwise do not share attributes. Thus, using the communication cost as computed in [1] and dividing it by $f + Nd_0$ we get the replication rate:

$$r = \frac{f + Nd_0p^{\frac{N-1}{N}}}{f + Nd_0}$$

However, since the star join has this special property that the fact table is much larger than the dimension tables, it is more interesting to compute the upper and lower bounds on the replication rate ignoring the fact table and assuming q is the maximum amount of data that a reducer can hold only from the dimension tables.

In that case, the upper bound on replication rate r is

$$r = \frac{Nd_0p^{\frac{N-1}{N}}}{Nd_0} = p^{\frac{N-1}{N}}$$

Setting $rNd_0 = pq$ we find $p = rNd_0/q$ and $r = (rNd_0/q)^{\frac{N-1}{N}}$, hence

$$r = (Nd_0/q)^{N-1} = (|I|/q)^{N-1}$$

For the lower bound we have:

- Input to each reducer: $q = |I|r/p$
- Output from each reducer: $q^N = (|I|r/p)^N$
- Total output: $|I|^N$
- The total output must be less than or equal to the output from each reducer multiplied by the number p of reducers, hence: $p(|I|r/p)^N \geq |I|^N$
- We replace p from the first bullet in terms of q and get:

$$r \geq (|I|/q)^{N-1}$$

Thus the upper and lower bounds match. This proves that the algorithm of [1] for hashing tuples to the reducers is optimal. More details on our results for star joins including a fact table are in our online technical report [4].

General case of multiway join It will be interesting to compare the lower bound with the upper bound as implied by the algorithm in [1] for replication rate for the general case of multiway join. However a closed form is not given in [1], hence it will require a computation executed by a program. Such a comparison will give insight as to which cases a better algorithm is needed (e.g., for the chain joins above) and for which cases the algorithm in [1] is optimal (e.g., the star joins). An interesting remark is that although the algorithm that gives the upper bound and the computation of the lower bound start from completely irrelevant premises, the formulas in both bounds have the same form. I.e., the replication rate for the upper bound in the general case is a summation of terms in the form $\beta p^{1-\alpha_i/s_i}$ where α_i depends on the arity of the relations in the join, and s_i depends on the number of attributes and β depends on how the sizes of the relations compare to each other, but neither of these dependencies are in closed form.

5.5.3 Output Size for Multiway Join – General Case

For the general case, we can apply the same technique to get lower bounds on the replication rate, only we need to know how to compute a bound on the size of the output of any multiway join. We explain here how to compute a tight bound as offered in [7, 12].

Let J be a multiway join and let $G(J)$ be the corresponding hypergraph. Thus the nodes of the hypergraph are the attributes in the query and the edges of the hypergraph correspond to the relational atoms in the query. For each edge e of $G(J)$ we have a variable x_e. Let $E(attr)$ be the set of edges that include attribute $attr$. We form the following linear program:

A number of inequalities, one inequality for each attribute/node $attr$ of the hypergraph.

$$\Sigma_{e \in E(attr)} x_e \geq 1$$

We optimize on:

$$\text{minimize}\Sigma_{e \in G(J)} x_e$$

The solution to this program is called an optimal *fractional edge cover* of the query hypergraph and the cost of an optimal solution is the fractional edge cover number ρ. It can be shown [12] that there is always a solution whose values are rational and of bit-length polynomial in the size of the query. Fractional edge covers can be used to give an upper bound on the size $|O|$ of the output of the query. Let $|R_e|$ be the size of the relation that corresponds to the edge e of the hypergraph $G(J)$.

$$|O| \leq \Pi_{e \in G(J)} |R_e|^{x_e}$$

6. MATRIX MULTIPLICATION

We shall now take up the common application of matrix multiplication. That is, we suppose we have $n \times n$ matrices $R = [r_{ij}]$ and $S = [s_{jk}]$ and we wish to form their product $T = [t_{ik}]$, where $t_{ik} = \sum_{j=1}^{n} r_{ij}s_{jk}$. This problem introduces a number of ideas not present in the previous examples. First, each output depends on many inputs, rather than just two or three. In particular, the output t_{ik} depends on an entire row of R and and entire column of S, that is, $2n$ inputs, as suggested by Fig. 3.

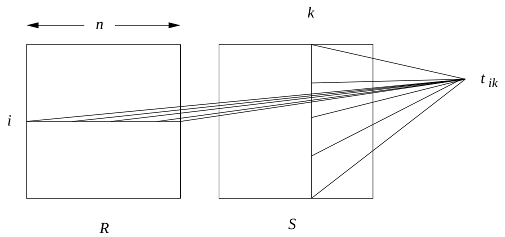

Figure 3: Input/output relationship for the matrix-multiplication problem

There is also an interesting structure to the way outputs are related to inputs, and we can exploit that structure. Finally, the fact that sum is associative and commutative lets us explore methods that use two interrelated rounds of map-reduce. Surprisingly, we discover that two-round methods are never worse than one-round methods, and can be considerably better.

6.1 The Lower Bound on Replication Rate

- **Deriving $g(q)$:** Suppose a reducer covers the outputs t_{14} and t_{23}. Then all of rows 1 and 2 of R are input to that reducer, and all of columns 4 and 3 of S are also inputs. Thus, this reducer also covers outputs t_{13} and t_{24}. As a result, the set of outputs covered by any reducer form a "rectangle," in the sense that there is some set of rows $i_1, i_2, \ldots i_w$ of R and some set of columns $k_1, k_2, \ldots, k_h$ of S that are input to the reducer, and the reducer covers all outputs $t_{i_u k_v}$, where $1 \leq u \leq w$ and $1 \leq v \leq h$.

  We can assume this reducer has no other inputs, since if an input to a reducer is not part of a whole row of R or column of S, it cannot be used in any output made by the reducer. Thus, the number of inputs to this reducer is $n(w + h)$, which must be less than or equal to q, the upper bound on the number of inputs to a reducer. As the total number of outputs covered is gh, it is easy to show that for a given q, the number of outputs is maximized when the rectangle is a square; that is, $w = h = q/(2n)$. In this case, the number of outputs covered by the reducer is $g(q) = q^2/(4n^2)$.
- **Number of Inputs and Outputs:** There are two matrices each of size n^2. Therefore $|I| = 2n^2$ and $|O| = n^2$.
- $\sum_{i=1}^{p} g(q_i) \geq |O|$ **Inequality:** Substituting for $g(q_i)$ and $|O|$:

$$\sum_{i=1}^{p} \frac{q_i^2}{4n^2} \geq n^2$$

- **Replication Rate:** We first leave one factor of q_i on the left as is, and replace the other factor q_i by q. Then, we manipulate the inequality so the expression on the left is the replication rate and obtain:

$$r = \sum_{i=1}^{p} \frac{q_i}{2n^2} \geq \frac{2n^2}{q}$$

6.2 Matching Upper Bound on Replication Rate

The lower bound $r \geq 2n^2/q$ can be matched by an upper bound for a wide range of q's. If $q \geq 2n^2$, then the entire job can be done by one reducer, and if $q < 2n$, then no reducer can get enough input to compute even one output. Between these ranges, we can match the lower bound by giving each reducer a set of rows of R and an equal number of columns of S.

The technique of computing the result of a matrix multiplication by tiling the output by squares is very old indeed [18]. In the map-reduce model, that is correct if a single round of map-reduce is used, but, as we shall see in Section 6.3, not quite correct for two-phase matrix multiplication, where the minimum cost occurs when the matrices are tiled with rectangles of aspect ratio 2:1.

Let s be an integer that divides n, and let $q = 2sn$. Partition the rows of R into n/s groups of s rows, and do the same for the columns of S. There is one reducer for each pair (G, H) consisting of a group G of R's rows and a group H of S's columns. This reducer has $q = 2sn$ inputs, and can produce all the outputs t_{ik} such that i is one of the rows in group G and k is one of the columns in the group H. Since every pair (i, k) has i in some group for R and has k in some group for S, every element of the product matrix T will be produced by exactly one reducer.

The replication rate for each input element is the number of groups with which its group is paired. That number is $r = n/s$, since both R and S are partitioned into this number of groups. Since $q = 2sn$, and thus $s = q/(2n)$, we have that $r = 2n^2/q$, exactly matching the lower bound on r.

6.3 Matrix Multiplication Using Two Phases

There is another strategy for perfoming matrix multiplication using two map-reduce jobs. As we shall see, this method always beats the one-phase method. An interesting aspect of our analysis is that, while tiling by squares works best for the one-phase algorithm,

- For the two-phase algorithm, the least cost occurs when the matrices are tiled with rectangles that have aspect ratio 2:1.

We assume that we are multiplying the same $n \times n$ matrices R and S as previously in this section.

1. In the first phase, we compute $x_{ijk} = r_{ij}s_{jk}$ for each i, j, and k between 1 and n. We sum the x_{ijk}'s at a given reducer if they share common values of i and k, thus producing a *partial sum* for the pair (i, k).
2. In the second phase, the partial sum for each pair (i, k) is sent from each reducer that has computed at least one x_{ijk} for some j to a reducer of the second phase whose responsibility to to sum all these partial sums and thus compute t_{ik}.

Figure 4 suggests what the mappers and reducers of the two phases do.

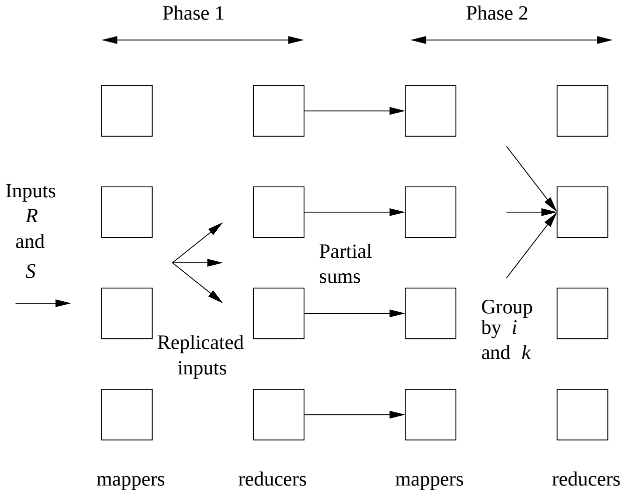

Figure 4: The two-phase method of matrix multiplication

The second phase is embarrassingly parallel, since each partial sum contributes to only one output. However, the first phase requires careful organization. To begin, it is not sufficient to compute only the replication rate of the first phase, since there is significant communication in the second phase. The number of partial sums could be as large as n^3 and thus dominate the communication cost. We optimize on the number r of rows, number s of columns and number t of partial-sums terms. Interestingly we find that although $s = r$ as in the one-phase algorithm, $t = s/2$ (proof in [4]). We thus calculate the total communication involved in both phases, and obtain a total communication of:

$$\frac{2n^3}{\sqrt{q}} + \frac{n^3}{\sqrt{q}/2} = \frac{4n^3}{\sqrt{q}}$$

On the other hand, the total communication for the optimum one-phase method described in Section 6.2 is the replication rate times the number of inputs, or

$$(2n^2/q) \times 2n^2 = 4n^4/q$$

For what values of q does the one-phase method use less communication than the two-phase method? Whenever

$$\frac{4n^4}{q} < \frac{4n^3}{\sqrt{q}}$$

or $q > n^2$. That is, for any number of reducers except 1, the two-phase method uses less communication than the one-phase method, and for small q the two-phase approach uses a lot less communication. There are other costs besides communication, of course, but since both methods perform the same arithmetic operations the same number of times, the communication difference is decisive. Interestingly an independent study in [11] provides experimental evidence that validates our theoretical analysis, even without the optimization we introduced in our two-phase method. Experiments in [11] show that indeed the one-phase method only outperforms the two-phase method in cases the replication rate of the one-phase method is not large. Although the communication cost of the one-phase method is always the smallest, the overhead that is incurred by the two-phase MapReduce is compensated when the difference between the communication costs of the two methods is small and thus, the one-phase method may be preferable in such cases (e.g., when one of the matrices is small enough to fit in one reducer). Finally, for the two-phase method, a rather straightforward observation is that we can use a similar function as the "combine" function in many reducers of the first phase if they are colocated (in same or close-by physical device) to produce partial sums, thus reducing further the communication cost.

7. SUMMARY

This paper has attempted to set a new direction for the study of optimal map-reduce algorithms. We introduced a simple model for map-reduce algorithms, enabling us to study their performance across a spectrum of possible computing clusters and computing-cluster properties such as communication speed and main-memory size. We identified *replication rate* and *reducer input size* as two parameters representing the communication cost and compute-node capabilities, respectively, and we demonstrated that for a wide variety of problems these two parameters are related by a precise tradeoff formula. These problems include finding bit strings at a fixed Hamming distance, finding triangles and other fixed sample graphs in a larger data graph, computing multway joins, and matrix multiplication.

7.1 Open Problems

The analyses done in this paper for several problems of interest should be carried out for many other problems. Discovering the tradeoff for Hamming distances greater than 1 seems hard. Analogous investigations are warranted for other kinds of similarity joins besides those based on Hamming distance. One question that arises naturally is how closely the general lower bound on multiway joins derived in this paper matches the general upper bounds in [1]? Since there is no closed formula for either upper or lower bounds in the general case, this question seems to need nontrivial arguments in order to be answered.

Another interesting direction is to explore whether it is possible to analyze algorithms taking two or more rounds of map-reduce along the lines of Section 6.3. A possible first place to look is at SQL statements that require two phases of map-reduce, e.g., joins followed by aggregations.

8. REFERENCES

[1] F. Afrati and J. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Transactions on Knowledge and Data Engineering*, 23(9):1282–1298, 2011.

[2] F. N. Afrati, D. Fotakis, and J. D. Ullman. Enumerating subgraph instances using map-reduce. Technical report, Stanford InfoLab, January 2012. Available at http://ilpubs.stanford.edu:8090/1020/.

[3] F. N. Afrati, A. D. Sarma, D. Menestrina, A. Parameswaran, and J. D. Ullman. Fuzzy joins using mapreduce. In *ICDE*, 2012.

[4] F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *CoRR*, abs/1206.4377, 2012.

[5] N. Alon. On the number of subgraphs of prescribed type of graphs with a given number of edges. *Israel Journal of Mathematics*, 38(1-2):116–130, 1981.

[6] Amazon. *Amazon Elastic Compute Cloud (Amazon EC2)*. Amazon Inc., 2008.

[7] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748, 2008.

[8] R. L. F. Cordeiro, C. T. Jr., A. J. M. Traina, J. López, U. Kang, and C. Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *KDD*, 2011.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[10] W. Feller. *An Introduction to Probability Theory and Its Applications*. Vol. 1, 3rd ed. New York: Wiley, 1968. (Stirling's Formula in Section 2.9).

[11] A. Ghoting, R. Krishnamurthy, E. P. D. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *ICDE*, pages 231–242, 2011.

[12] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[13] W. Hasan and R. Motwani. Optimization algorithms for exploiting the parallelism-communication tradeoff in pipelined parallelism. In *VLDB*, 1994.

[14] R. Ikeda, H. Park, and J. Widom. Provenance for generalized map and reduce workflows. In *CIDR*, 2011.

[15] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *SODA*, pages 938–948, 2010.

[16] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, pages 223–234, 2011.

[17] Y. Kwon, M. Balazinska, B. Howe, and J. A. Rolia. Skewtune: mitigating skew in mapreduce applications. In *SIGMOD Conference*, pages 25–36, 2012.

[18] A. C. McKellar and E. G. Coffman. Organizing matrices and matrix operations for paged memory systems. *Commun. ACM*, 12(3):153–165, 1969.

[19] A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. In *SIGMOD Conference*, pages 949–960, 2011.

[20] C. Olston and B. Reed. Inspector gadget: A framework for custom monitoring and debugging of distributed dataflows. In *Proc. of VLDB*, 2011.

[21] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-round tradeoffs for mapreduce computations. *CoRR*, abs/1111.2228, 2011.

[22] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press. Also available on-line at i.stanford.edu/~ullman/mmds.html, 2011.

[23] T. Schank. *Algorithmic Aspects of Triangle-Based Network*. University of Karlsruhe (TH), 2007.

[24] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, 2011.

[25] W.-C. Tan. Provenance in Databases: Past, Current, and Future. *IEEE Data Engineering Bulletin*, 2008.

[26] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD Conference*, 2010.

[27] T. White. *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (2. ed.)*. O'Reilly, 2011.