Distributed Data Processing

Data & Web Science, MSc.
Aristotle University of Thessaloniki

Name:	Periklis Fregkos	Lazaros Gogos
Student Number:	203	211
Research Centre:	AUTh	
Research Project Title:	Matching Bounds for the All-Pairs Map-Reduce Problem	
Supervisor:	Anastasios Gounaris	

1 Project Summary

The aim of this project is to solve the All-Pairs problem using the Map-Reduce framework. We evaluate different partitioning strategies and utilize the findings from Afrati & Ullman (2013) to achieve the lower bound for this problem.

Moreover, we investigate the optimal replication rate to minimize network traffic while maintaining parallelism. A mathematical framework is applied to determine the minimal replication rate for specific input sizes and a given reducer capacity.

The code of this project is publicly accessible at https://github.com/fregkos/pyspark-all-sets.

1.1 Problem setup

We define as d the total amount of input data, q as each reducer's maximum capacity (the number of data it can process at once) and as p a prime number, which satisfies certain conditions and aids in optimal partitioning of the data. Let r denote the replication rate, or how much data is shuffled across the network. For us, the replication rate is what defines the communication cost, which is what needs to be minimized.

1.2 Dataset preparation

We prepare our dataset using an iterator which is consumed and converted to a list, before it is converted to an RDD[1]. The purpose of that is to simulate an actual dataset, while allowing us to test with variable data lengths. In our experiments, we made sure that the size of the dataset fits in our main memory, while the master node executes the creation of the dataset before proceeding to other operations. We account for this constant time in our results.

1.3 Experiments' environment

In order to run our experiments, we set up a set of Docker containers, under Docker Engine, using Docker Compose and py-spark[2] to handle data processing in a distributed way, simulating the technology that would be used in a distributed setup, solidifying our results.

The operating systems used in the experiments are Debian and Fedora, on an Intel i5 4660 16GB DDR3 machine and an Intel i5 1240P 16GB DDR4 machine respectively.

We create a simple py-spark script that initializes an accumulator and creates an RDD for a range of numbers, indicating the indexes of the data we need to match all pairs with.

To follow the Map-Reduce framework, key-value pairs are emitted by a mapping function, and these key-value pairs are processed by workers in a fashion similar to how a reducer would process them. Therefore, a key consists of i, j pairs of the d data, and a value, denoted by R_i, is typically of some significant size. A reducer's job is to receive and combine the R_i, R_j based on the i, j keys and output whether some condition is met. For

[1]Resilient Distributed Dataset or RDD is a fault-tolerant collection of elements that can be operated on in parallel.
[2]https://github.com/apache/spark

example, in the context of testing drug combinations, the output would be whether two drugs, i and j, have some effect when combined with each other.

For the rest of this report, we mainly focus on the amount of shuffles that take place when different configurations are used for solving this problem, and avoid measuring the exact traffic on the network, as this proved to be futile; measuring traffic on a task running locally does not produce valuable results.

1.4 Key findings

Our key findings include:

- The simplest approach is to measure the amount of shuffles that occur when single pairs are emitted from the mappers to the reducers.
- The naive approach of dividing the data into $g = 2 \times \frac{d}{q}$ groups, which results in replication rates $r = 2 \times \frac{d}{q} - 1$.
- A more optimal approach involves finding a prime p such that $p^2 \mid d$, $q = \frac{d}{p}$, and $group\_size = \frac{d}{p^2}$. This leads to an optimal replication rate of $r = \frac{d}{q} + 1$.

This work demonstrates how careful partitioning can significantly reduce network traffic while maintaining computational efficiency. The results can possibly have a significant effect on distributed data processing systems and large-scale parallel computing.

2 Naive approaches

Let us consider the following configuration: $d = 5000$ (total data), $q = 1000$ (maximum reducer memory/capacity). There are two ways this problem can be handled. One way would be to have each mapper output 1-to-1 combinations of the d input data. Another option would be to group data with one another and utilize the reducers' computational capability. Therefore, each group can have at most $\frac{1000}{2} = 500$ elements, so that at any point two groups can fit in a reducer, so they can be compared with one another. Hence, we assess the following configurations:

Having each mapper emit i, j keys with a value of R_i, and each reducer receive the i, j key and the R_i, R_j values, outputting whether the values meet some predefined condition. This would result in $d \times (d-1) = 5000 \times 4999 = 24995000$ or approximately d^2 shuffles of the data, from the mappers to the reducers. To generalize this, we can write this as having a replication rate of $r = 2 \times d/q - 1 = 2 \times 5000/2 - 1 = 4999$, with $q = 2$ since each group contains 1 .

Should we opt for grouping the data before sending them to the reducers, we would have the following shuffle results.

Each reducer requires the $\frac{d}{g}$ members of two different groups to be compared, thus $q = 2 \times \frac{d}{g} \Leftrightarrow g = 2 \times \frac{d}{q}$. Let us assess the following configurations and examine their effects.

2.1 Approach 1

Let us choose to use a reducer size of $q = 1000$.

- $g = 2 \times \frac{d}{q} \Rightarrow g = 2 \times \frac{5000}{1000} = 10$ **pairs** of groups
- Therefore, $\frac{d}{g} = \frac{5000}{10} = 500$ items per group
- $2 \times 500 = 1000$ elements per *reducer* $\equiv q = 1000$
- This would be an *edge case* scenario, as **no more than** 1000 elements could fit in a reducer at once in any given moment. The reducers would be maximally utilized.

2.2 Approach 2

We can assess a different size for a reducer, in order to examine if it can achieve a lower replication rate. Let us choose to use a reducer size of $q/2 = 500$.

- $g = 2 \times \frac{d}{q} \Rightarrow g = 2 \times \frac{5000}{500} = 20$ **pairs** of groups
- Therefore, $\frac{d}{g} = \frac{5000}{20} = 250$ items per group
- $2 \times 250 = 500$ elements per *reducer* $\equiv q/2 = 500 < q = 1000$
- This would be a *feasible* scenario and a setting which results in under-utilization, as 500 elements are half of what could fit in a reducer at once in any given moment.

2.3 Approach 3

Let us choose to use a reducer size of $q/10 = 100$.

- $g = 2 \times \frac{d}{q} \Rightarrow g = 2 \times \frac{5000}{100} = 100$ **pairs** of groups
- Therefore, $\frac{d}{g} = \frac{5000}{100} = 50$ items per group
- $2 \times 50 = 100$ elements per *reducer* $\equiv q/10 = 100 \ll q = 1000$
- This would be *feasible* configuration, though resulting in under-utilizing the computational resources, as 100 elements could fit in a reducer at once in any given moment, but utilize only $\frac{100}{1000} = 10\%$ of the reducer's capacity. We would need 9 more groups to utilize full capacity.

The above grouping methods would result in the following replication rates (respectively):

- $r = 2 \times \frac{d}{q} - 1 = 2 \times \frac{5000}{1000} - 1 = 10 - 1 = 9$
- $r = 2 \times \frac{d}{q} - 1 = 2 \times \frac{5000}{500} - 1 = 20 - 1 = 19$
- $r = 2 \times \frac{d}{q} - 1 = 2 \times \frac{5000}{100} - 1 = 100 - 1 = 99$

We can safely state that the less we utilize the reducer's memory, the worse the replication rate gets. However, according to Afrati & Ullman (Theorem 2.1), we know that the **best** possible lower bound approximates $r = \lceil \frac{d-1}{q-1} \rceil = \lceil \frac{5000-1}{1000-1} \rceil = 6$.

Consequently, this begs the question, what is the optimal number of groups, so that the replication rate is minimized? Based on Afrati & Ullman (2013), it is known that $r = \frac{d}{q} + 1$ is the lowest bound.

3 Proposed approach

Firstly, we need to find a prime p which satisfies the following conditions:

1. p^2 divides d
2. $q = \frac{d}{p}$
3. $group\_size = \frac{d}{p^2}$

Let us consider the following example. If that prime is $p = 5$, our total data d is 5000 in total and the maximum capacity of each reducer q is 1000, then:

1. $groups = p^2 \Rightarrow 5^2 = 25$
   $5^2 | 5000 \Rightarrow 5000 \bmod 25 = 0 \Rightarrow True$
2. $q = \frac{d}{p} \Rightarrow q = \frac{5000}{5} = 1000$, which respects our maximum capacity requirement of q.
3. $group\_size = \frac{d}{p^2} \Rightarrow \frac{5000}{25} = 200$ elements per group

Therefore, each group must contain 200 elements, meaning each reducer receives $\frac{q}{group\_size} = \frac{1000}{200} = 5$ groups per reducer, thus maximizing reducer usage and parallelization, achieving the lower bound for the all-pairs problem.

3.1 Lower bounds

Based on the findings of the paper (Theorem 3.1), $r = \frac{d}{q} + 1$ since the above conditions are satisfied.

Ergo, the replication rate becomes: $r = \frac{d}{q} + 1 = \frac{5000}{1000} + 1 = 6$, Q.E.D. (quod erat demonstrandum).

3.2 Measuring network traffic

This work only includes measurements on a theoretical level. However, the related github repository contains the necessary tools for the interested reader in order to measure actual network traffic. Namely, the said tools are Grafana, ContentAdvisor and Prometheus, with which network traffic can be easily monitored. It is worth mentioning that our tries at shuffling data were futile in local settings, because Spark operates with internal buffers, thus making the actual traffic measurements inconclusive.

4 Observations and results

Although truly no shuffling takes place, the impact on performance is easily discernible. Emitting single pairs takes the longest time, as each reducer must wait to receive a pair before proceeding with processing. The grouping of pairs produces better results. Using the proposed solution results in the smallest replication rate possible.

For the following results, we calculate the amount of data shuffled based on the replication rate using $r \times d$.

Method	Elements per group	# of groups	Data shuffled
Approach 1	1	2	24995K
Approach 2	100	2	495K
Approach 3	500	2	45K
Proposed approach	200	5	**30**K

Table 1: Comparison of multiple methods evaluating them on replication rate of each approach *(lower is better)*. The naive approach (Approach 1) emits single pairs to each reducer. Approach 2 emits pairs with 100 elements per group to each reducer, improving the replication rate. Approach 3 utilizes the reducers' size to the full, achieving a significant 2 orders of magnitude improvement. Finally, the Proposed approach achieves the lowest possible bound for the all pairs problem. The *# of groups* column refers to the number of groups that are sent to a reducer at any given time.

Our explanation is that the single pairs method will emit more pairs of groups than the other methods. We should point out that more pairs do not imply better parallelism. As the overhead that accompanies the network traffic, it defeats the purpose of the actual calculation. Hence, this method holds the worst performance as the overhead aggregated is non-negligible.

Focusing on the grouping methods that both utilize a percentage of the total allocated capacity per worker, we observe that the naive hypothesis of using Approach 3 is actually more efficient, with a replication rate relatively close to the proposed method. Whereas using Approach 2, even though it reduces the total data shuffled (compared to emitting single pairs), the replication rate is mediocre, because it fails to utilize the reducers' full capacity.

As we would expect, the proposed method is the most efficient; even better than the grouping methods, having the best replication rate.

5 Comments and future work

Depending on the actual task at hand, the solution to these problems should be crafted after carefully investigating the limiting factor for the calculation. Ideally, one should try to approach a solution that balances execution time and replication rate. In practice, such experiments involve optimal mapping of the input data to a specific set of reducers, to achieve maximum reducer usage and minimal replication rate.

Most of the time, the communication cost would be the limiting factor. Additionally, we should not forget that completion time is an important variable in a feasible solution of a problem.

References

Afrati, F., & Ullman, J. 2013, in Proceedings of the 17th International Database Engineering & Applications Symposium, IDEAS '13 (New York, NY, USA: Association for Computing Machinery), 3–4, doi: 10.1145/2513591.2513663