COSC2637 Big Data Processing Assignment 2018

Dzmitry Kakaruk
School of Science / Master of IT
RMIT
Melbourne, AU
s3668804@student.rmit.edu.au

Patrick Jacob
School of Science / Master of IT
RMIT
Melbourne, AU
s3622259@student.rmit.edu.au

Abstract—**This assignment analyzes the performance of different implementations of optimizations for MapReduce Algorithms including stripes, pairs, combiner, partitioner and local aggregation. This MapReduce tasks are performed on two different publically available datasets building a co-occurrence matrix. The results show a significant favor in processing speed for one implementation .**

Keywords—hadoop, big data, co-occurrence, stripes, pairs, combiner, partitioner, local aggregation

I. Introduction

As part of the 2018 Big Data Processing curriculum the major assignment is to analyse the publicly available "big" data using MapReduce. We define three main dimensions of Big Data high -velocity, -volume, -variety. We decided on an natural language processing related Big Data Task which is analyzing co-occurrence of words in large body of text. This technique allows to identify frequency of word pairs within certain defined distance. It is used in sentiment or notion analysis of opinions (e.g. customer reviews). We have through AWS S3 a large corpus of text such as the amazon reviews available which fit perfectly to our assignment specification. To process the amount of text available we will implement MapReduce with several optimizations including stripes and pairs and deploy to AWS Servers with Hadoop, which are accessible to us for use within scope of this project. While this analysis meets the quality of big data in terms of variety and volume. We will focus on mostly velocity of processing as dependent variable in the analysis. We are expecting that with the different experiments and implementations we will see significant differences in processing performance. Main question of this task is which implementation / optimization is the fastest algorithm for co-occurrence processing. Secondary question is which algorithm delivers the more space efficient results.

II. Experiments

To achieve a meaningful result for this assignment we decided to process the co-occurrence MapReduce program on two text corpus:

1. The common crawl with two cases -
   a. One case with one archive with 116.7 MB compressed data
   b. One case with 10 archives together accounting for 1158 MB
2. Secondary data set the UK Amazon Customer Reviews.

Each of them having their own size, dataformats, structure and content. Each data set will be processed using the same basic MapReduce algorithms and following the same test steps and setting so that a comparability is ensured. Only differentiating factor will be the inherited structure and size. See attached test tables (Attachment 1) for an general idea of our tests. We will run the MapReduce over the data corpus with 7 different implementation (Stripes, Pairs etc). We will test all cases with basie two data nodes default setup and with boosted to four data notes. Additionally we test all cases with 1 and 5 for reducers. This means our experiment will have a complexity of 3 (datasets) x 7 (optimization) x 2 (cluster data notes setups) x 2 (reducer setups) = 84 test cases.

1. Test Scenario with 5 reducers and 2 data notes

Parameter			**Result in days for small common crawl data set**		
Optimization	***# Nodes***	***# Reducer***	**Map Phase**	**Reduce Phase**	**Total**
Pairs (plain)	2	5	0.24	0.17	0.41
Pairs Combiner	2	5	0.58	0.14	0.72
Pairs Partitioner	2	5	0.26	0.16	0.41
Local Aggregation - in mapper (local)	2	5	0.21	0.06	0.27
Local Aggregation - in mapper (global)	2	5	0.23	0.18	0.41
Stripes (plain)	2	5	0.05	0.10	0.15
Stripes Combiner	2	5	0.18	0.16	0.34

As displayed on Table 1, we decided on incremental steps on the cluster size to better visualise the variances in each step between the other confounding parameters

The results should be sufficient to report on any significant performance variance.

A. Measuring

For this experiments we read the CPU processing time as per "Counter" print of the specific job in Hue excludes any waiting times and provides a more exact reading. Even though measured in milliseconds for better readability we calculate it to hours. Second measurement is size of MapReduce result.

B. Common Crawl

The Common Crawl is a "text" corpus that contains petabytes of raw website, meta data and text extractions which have been collected since 2011. The common crawl project saves their files in different formats [1]:

- WARC files store the raw website data
- WAT files store computed metadata WARC files.
- WET files extracted plaintext from the WARC files

For our analysis we are looking only into the WET File archives. We won't analyze the total of 7 years of data, only 2 subsets it. From the one bucket we are looking into the achieve: CC-MAIN-20180918130631-20180918150631-00000 with a size of 116.7 MB and do a complete test run. The second test run will be performed over the CC-MAIN-20180918130631-20180918150631-00000 until -00009. This accounts 10 files that sum up to 1158 MB. Our aim is to be able to see differences in processing between a singular file or several files. To have comparable data we choose this taking advantage of the fact that per each archive file only one mapper is assigned. In these test runs the efficiency of parallelisation should visualize itself.

C. *Amazon Customer Reviews*

Amazon is known for it's webshop, which allows each product to be reviewed. These collected reviews from 1995 until 2015 are available as tab separated file (tsv) in several languages [2]. We are only looking at the english UK reviews from that time period which are saved in the amazon_reviews_multilingual_UK_v1_00. This dataset has a size of 333.2MB and is suited for our co-occurrence analysis as it contains highly opinionated review data and is within the application of occurrence. We know from looking into the dataset that this is more condensed data and expect it to take longer for processing than the more sparse common crawl data.

III. Implementation

We decided to implement all our MapReduce Programs in Java 8 using maven for dependency control based our conclusion of research on mapreduce supported programming languages [4]. We use the common crawl word count example as architectural foundation. For parsing the different file formats we implemented dedicated methods for reading them. Following the requirements of the FAQ - Option 1 of the assignment we implemented the following optimisations:

A. *Pairs*

The pairs implementation is a straightforward implementation in which each mapper takes a sentence and generates each pair a count. The reducer than sums the pair count up. We expect it not to be the most efficient implementation as the amount of pairs to be shuffled and sorted is higher than in over comparable optimizations. Additionally in the reduce phase it needs to hold state across reducers which decreases the ability to quickly process relative frequencies [6].

B. *Stripes*

The other common implementation is called stripes, while less straightforward than pairs it can make better use of combiners and decreases the sorting and shuffling. The idea is to use associative arrays instead of intermediate key value pairs. Which mean for each key all values and they cooruance count are saved in an array and during the reduce phase all associative arrays with the same key are summed up per each element [6]

C. *Combiners*

Combiners are optional optimization steps. Their purpose is to summarize the intermediate output of a specific mapper and basically do a small scale version of the reducers task on the mapper side. The advantage is that it increases performance of reducer by decrease the amount of data the reducer has to process. But the map reduce can't depend on the combiners execution as it might actually not happen [5]. We implemented it for both Pairs and Stripes approach.

D. *Partitioner - only Pairs*

Like Combiners, Partitioner are an optional and additional optimization which are placed between mappers and reducers. Their task is to break the intermediate (after mapper process) key space and group key value pairs to assign them to the reducer (usually using a hash function) [6]. The amount of partitioners has to the same as the reducers. This way an even distribution of map out over reducers is achieved (based on the keys) [7]. For implementation we used the alphabet as key for distribution.

E. *Local Aggregation - only Pairs*

Local *Aggregation* is implemented as in-mapper combiner. The aim is to reduce the amount of intermediate key value pairs. The difference is, that this process is done in memory to increase performance (as in memory processes are faster than disk executed ones) and to be certain that this combining like task happens. During implementation and execution attention needs to be set on memory allocation, as this task can break under memory space limitations. [6] [8]. In our approach we did two versions. One is "global" which means, using hashmap on class level, it lives as long a the mapper or unless reached flush level threshold. In contrast the "local" version has the hashmap in the map method, thus it lives as long as the record is being processed.

IV. Results

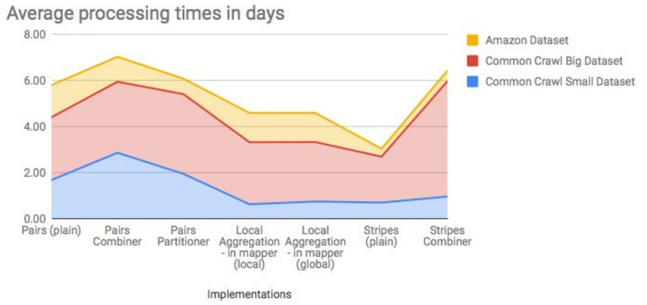
An general overview of the results that came out of the experiments

A. *Performance*

Processing

In average the pairs is less efficient than stripes implementations. While various optimisations improve the pairs do improve, stripes (plain) remains the most performant implementation. While we following the recommendation of Vangjee[8] to implement a combiner of stripes we can't confirm that it improves the processing time - see Picture 1. This is confirmed for all datasets used.

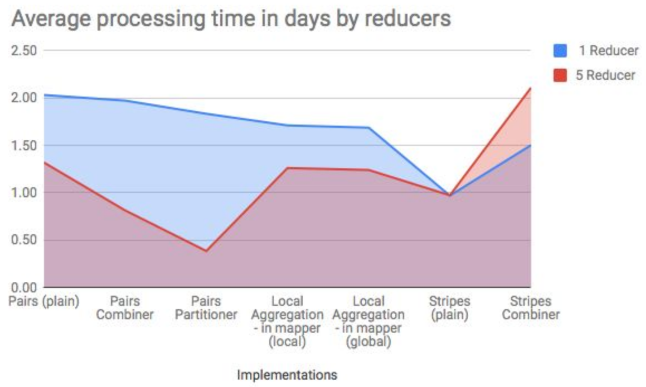
Diagram 1 - average processing time for all implementations



On the pairs optimisations we see different behaviours. E.g. combiners have a negative effect on the smaller common crawl dataset, on the bigger one it shows slight improvement and in the amazon dataset it improves significantly. See Attachment 1 - diagrams/tables 2 - 4. We assume that this is due to the fact that the amazon dataset wasn't compressed like common crawl wet files thus letting it take advantage of breaking it up in multiple files and using the combining functionality. On the other hand local aggregation seems to show the best results for the singular file data set of the common crawl data and only to a lesser extent for the other data sets used as seen in attachment 1 - diagrams/tables 2 - 4. Which throws the question if the combiner was called in the smaller common crawl dataset. In terms for the different implementation for "local" and "global" in mapping combining the "global" one seems to be slightly more efficient (except for one case using 5 reducers unboosted on small common crawl dataset).

As for the reducers increase, it seems in general (except for stripes combiner) it does improve the overall performance - see Diagram 2

Diagram 2 - average processing time for all implementations by reducer

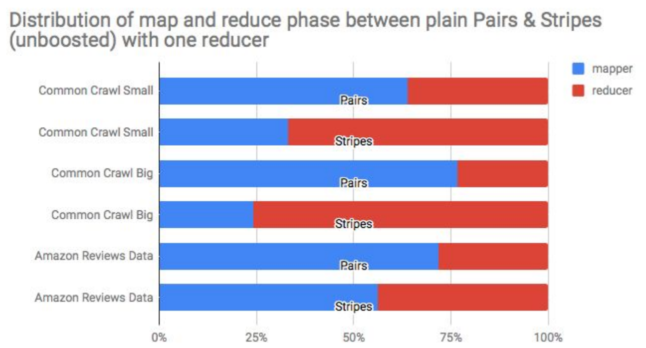


According to data collection shown on diagrams/tables 2 - 4 boosting 2 more nodes doesn't improve the performance of processing in all cases. It shows that in case of common crawl plain, partitioner and stripes with combiner implementation don't improve on having a boost. This is against our expectations as we thought that due to the compressed data there shouldn't be a difference.

Distribution between Map and Reduce Phase

A significant difference between the distribution of time used for processing a mapreduce job is shown between pairs and stripes as per Diagram 3. While pairs implementation usually is heavy on the Map Phase with a split of around 60 % - 70 % and 40 % - 30 % percent on the reduce phase. Opposed to that the Stripes implementation is closer to have 20 % - 30 % of the processing time allocated for mapping and the rest for reducing.

Diagram 3 - average processing time for all implementations by reducer



The distribution of processing time doesn't change between using either 1 or 5 reducers as shown in Appendix 1 - Table/Diagram 5 & 6.

Result file

Lastly there are differences in the output between various implementations. We have seen that Pairs is a more voluminous implementation and uses more space, often three times as much as the result files of stripes implementation. Besides that, the size of the always remains the same across the pairs implementation and corresponds to the amount of reducers. The difference here is the partitioner implementation has an uneven distribution. This is caused by the key that is set us, which is the dividing the alphabet in 3 groups using it against the first letter of a word as sorting key.

B. Limitations

We have encountered several restrictions that have had influence on our Experiments and the Plans.

1. During implementation we have seen that only one mapper can be assigned to one common crawl file., due to assigning one cpu kernel to one uncompressing and mapping process of the Wet file. This in practice has left our processing having a bottleneck, since not all resources available where used.
2. The provided AWS Cluster Environment had several issues by itself. Often we encountered that performance dropped, bugs appeared and terminating and restarting cluster was necessary. Additionally the cluster resources and memory where capped - and shared among students, thus leaving the performance at peak times dropping or memory running out.

Acknowledgment

We want to express our gratitude to the teaching team of the RMIT course Big Data Processing - 2018 for providing examples and guidance, which we extensively use throughout this assignment and the MapReduce implementation.

References

[1] CommonCrawl Foundation, 2018, So you're ready to get started., accessed 25.09.2018. <http://commoncrawl.org/the-data/get-started/>

[2] Amazon, 2018, *Amazon Customer Reviews Dataset,* accessed 25.09.2018. <https://registry.opendata.aws/amazon-reviews/>

[3] Amazon, 2018, Linguistic data of 32k film s*ubtitles with IMDb meta-data,* accessed 25.09.2018, <https://registry.opendata.aws/imdb/>

[4] Techvidvan Team, Battle: Apache Spark vs Hadoop MapReduce, TechVidvan, accessed 26.09.2018 <https://techvidvan.com/tutorials/apache-spark-vs-hadoop-mapreduce/>

[5] Bd hd team, *Hadoop Combiner Introduction, Working & Advantages*, TechVidvan, accessed 26.09.2018 <https://techvidvan.com/tutorials/hadoop-combiner-introduction-working-advantages/>

[6] Jimmy Lin , Chris Dyer, *Data-Intensive Text Processing with MapReduce*, Morgan and Claypool Publishers, 2010

[7] DataFlair Team , *Hadoop Partitioner – Internals of MapReduce Partitioner,* accessed 26.09.2018 <https://data-flair.training/blogs/hadoop-partitioner-tutorial/>

[8] Vangjee, *The "In-Mapper Combing" Design Pattern for Map/ Reduce Programming in Java,* accessed 26.09.2018, <https://vangjee.wordpress.com/2012/03/07/the-in-mapper-combining-design-pattern-for-mapreduce-programming/

Appendix 1

Table 1

	Average processing time in days		
Implementations	Common Crawl Small Dataset	Common Crawl Big Dataset	Amazon Dataset
Pairs (plain)	1.68	2.73	1.40
Pairs Combiner	2.86	3.08	1.09
Pairs Partitioner	1.95	3.46	0.67
Local Aggregation - in mapper (local)	0.63	2.69	1.27
Local Aggregation - in mapper (global)	0.75	2.58	1.26
Stripes (plain)	0.70	1.99	0.34
Stripes Combiner	0.96	5.01	0.46

Diagram 1

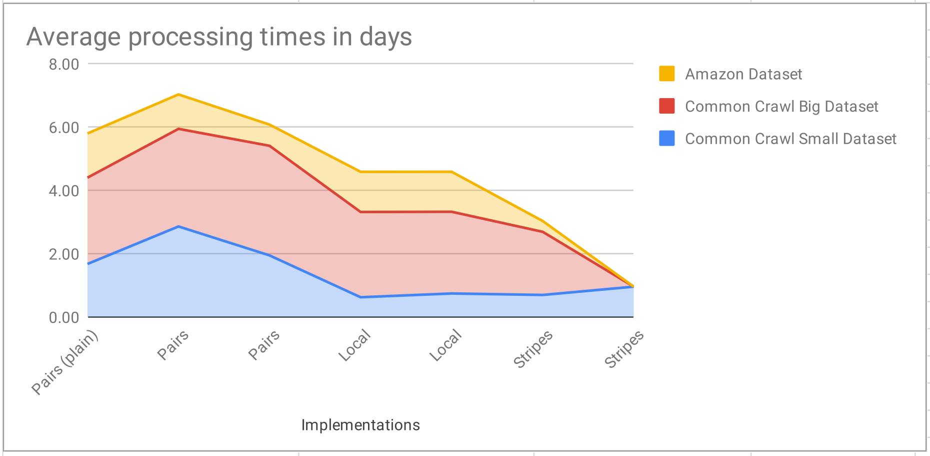


Table 2

	Common Crawl Dataset - 116 MB - 1 File (in days)			
Algorithm Implementation	1 reducer unboosted	1 reducer boosted	5 reducer unboosted	5 reducer boosted
Pairs (plain)	0.40	0.46	0.41	0.41
Pairs Combiner	0.80	0.74	0.72	0.60
Pairs Partitioner	0.49	0.46	0.41	0.59
Local Aggregation - in mapper (local)	0.36		0.27	
Local Aggregation - in mapper (global)	0.33		0.41	
Stripes (plain)	0.15	0.22	0.15	0.18
Stripes Combiner	0.62	0.00	0.34	

Diagram 2

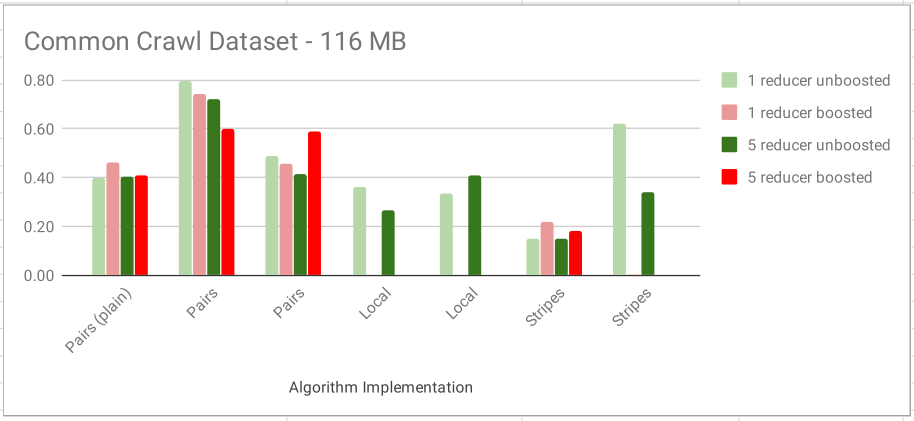


Table 3

	Common Crawl Dataset - 1158 MB - 10 Files (in days)			
Algorithm Implementation	1 reducer unboosted	1 reducer boosted	5 reducer unboosted	5 reducer boosted
Pairs (plain)	3.4	0.1	3.2	4.3
Pairs Combiner	3.2		3.0	
Pairs Partitioner	3.5		3.4	
Local Aggregation - in mapper (local)	2.8		2.6	

Local Aggregation - in mapper (global)	2.7		2.5	
Stripes (plain)	2.0	2.4	1.8	1.7
Stripes Combiner	4.1	5.9		

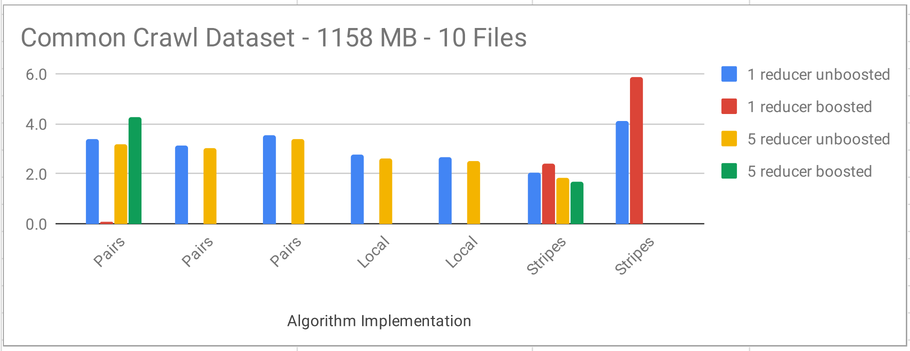


Table 4

Amazon Reviews Data (333.2 MB / 820.3 MB uncompressed - 1 File) (in days)				
Algorithm Implementation	1 reducer unboosted	1 reducer boosted	5 reducer unboosted	5 reducer boosted
Pairs (plain)	1.4		1.4	
Pairs Combiner	1.1		1.1	
Pairs Partitioner	1.2		0.1	
Local Aggregation - in mapper (local)	1.3		1.3	
Local Aggregation - in mapper (global)	1.3		1.2	
Stripes (plain)	0.3		0.3	
Stripes Combiner	0.5		0.4	

Diagram 4

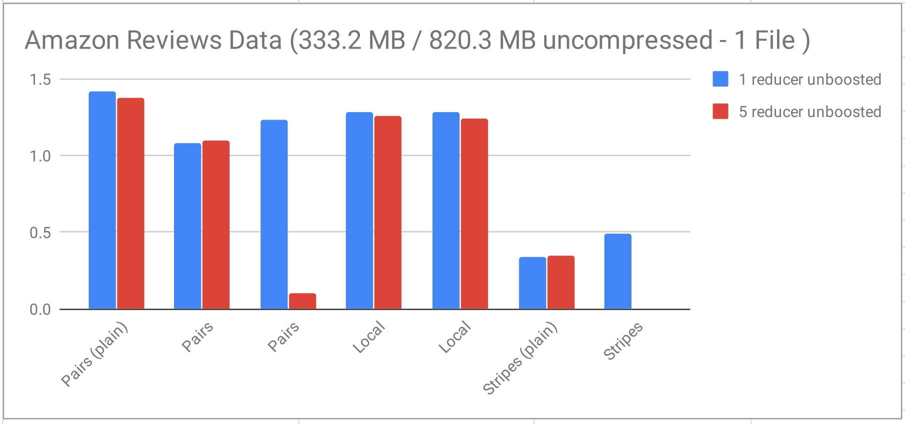
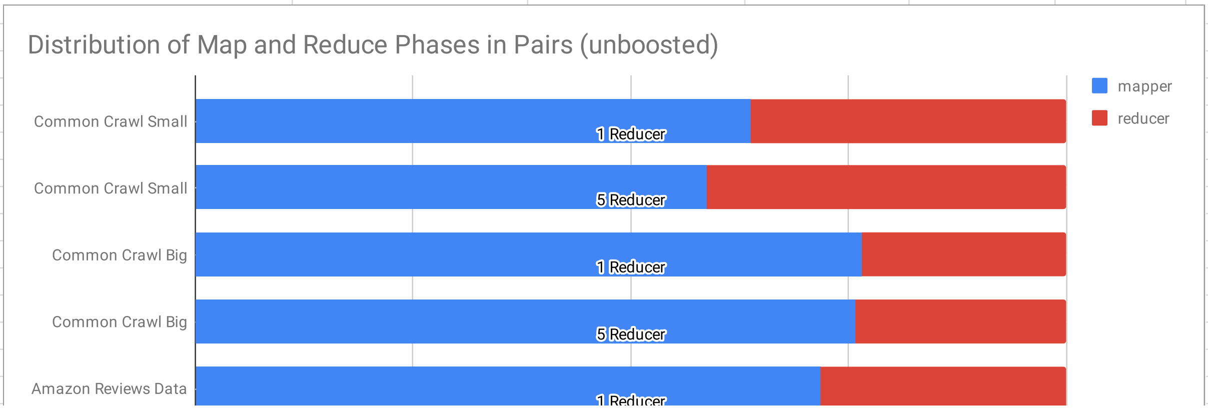


Table 5 - Distribution processing time between Map and Reduce Phase in Pairs Implementation

Dataset	Reducer	Mapper in days	Reducer in days
Common Crawl Small	1	64%	36%
Common Crawl Small	5	59%	41%
Common Crawl Big	1	77%	23%
Common Crawl Big	5	76%	24%
Amazon Reviews Data	1	72%	28%
Amazon Reviews Data	5	72%	28%

Diagram 5



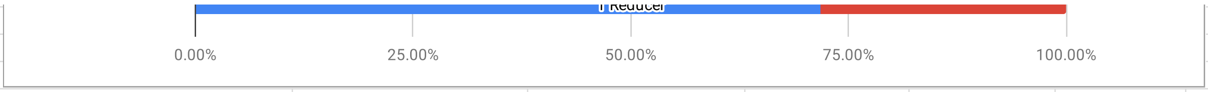


Table 6 - Distribution processing time between Map and Reduce Phase in Pairs Implementation

Dataset	Reducer	Mapper in days	Reducer in days
Common Crawl Small	1	33%	67%
Common Crawl Small	5	33%	67%
Common Crawl Big	1	24%	76%
Common Crawl Big	5	27%	73%
Amazon Reviews Data	1	56%	44%
Amazon Reviews Data	5	56%	44%

Diagram 6

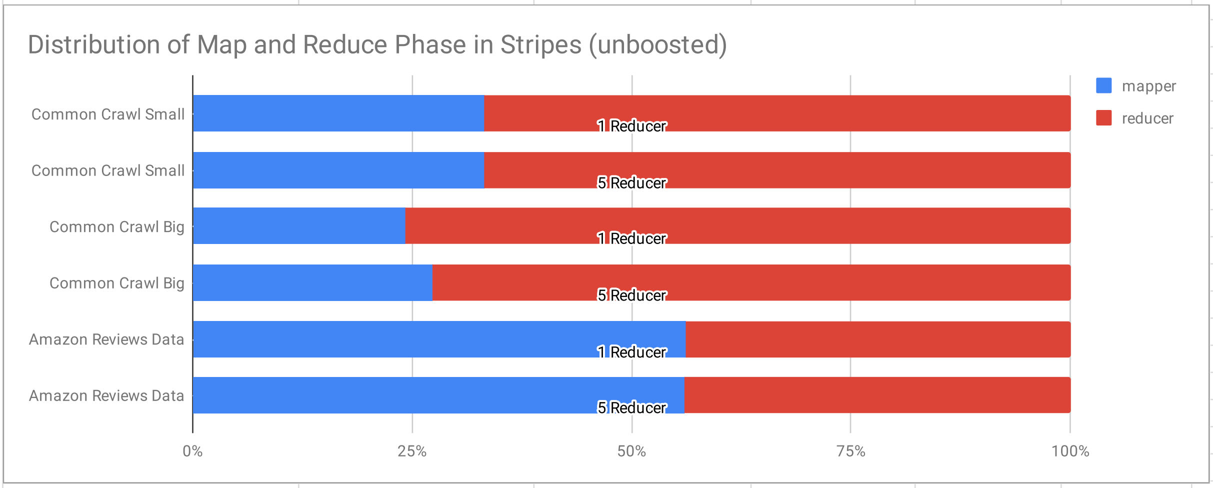

Distribution of Map and Reduce Phase in Stripes (unboosted)

Table 7 - Distribution of map and reduce phase between plain Pairs & Stripes (unboosted) with one reducer

Dataset	Implementation	Mapper in days	Reducer in days
Common Crawl Small	Pairs	64%	36%
Common Crawl Small	Stripes	33%	67%
Common Crawl Big	Pairs	77%	23%
Common Crawl Big	Stripes	24%	76%
Amazon Reviews Data	Pairs	72%	28%
Amazon Reviews Data	Stripes	56%	44%

Diagram 7

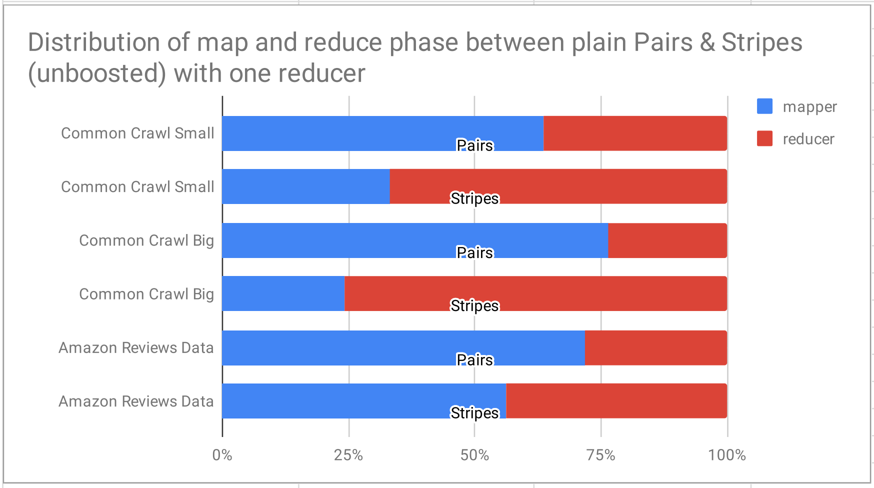

Distribution of map and reduce phase between plain Pairs & Stripes (unboosted) with one reducer

Table 8 - Distribution of map and reduce phase between plain Pairs & Stripes (unboosted) with 5 reducer

Implementation	Implementation	mapper	reducer
Common Crawl Small	Pairs	59%	41%
Common Crawl Small	Stripes	33%	67%
Common Crawl Big	Pairs	76%	24%
Common Crawl Big	Stripes	27%	73%
Amazon Reviews Data	Pairs	72%	28%

Amazon Reviews Data	Stripes	56%	44%

Diagram 8

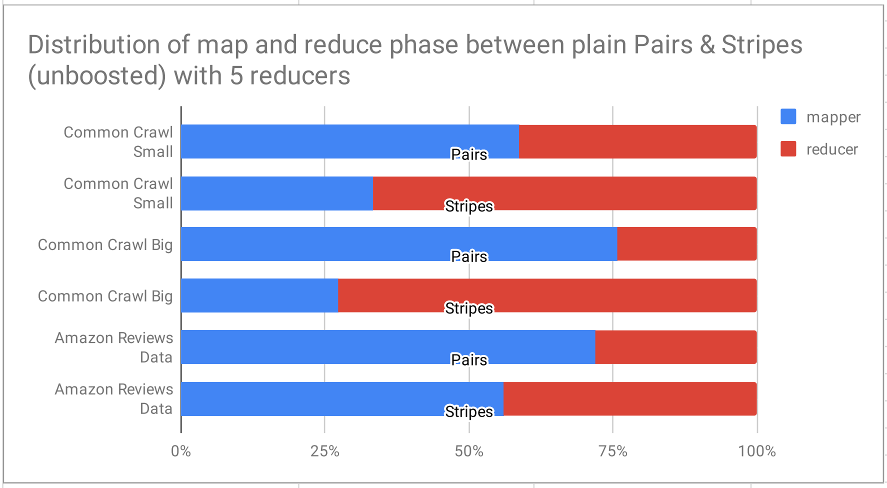
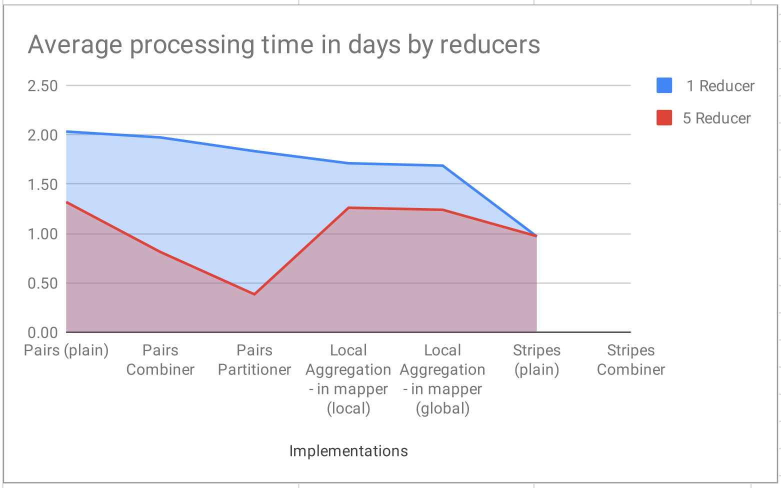


Table 9 - Average processing time in days by reducers

Implementations	1 Reducer	5 Reducer
Pairs (plain)	2.03	1.32
Pairs Combiner	1.97	0.81
Pairs Partitioner	1.83	0.38
Local Aggregation - in mapper (local)	1.71	1.26
Local Aggregation - in mapper (global)	1.69	1.24
Stripes (plain)	0.97	0.97
Stripes Combiner	1.50	2.11

Diagram 9



Appendix 2

Common Crawl Data (116.7 Mb - 1 file)

Algorithm Implementation	Cluster	Reducers	Map Phase (ms)	Map Phase (decimal days)	Map Phase %	Reduce Phase (ms)	Reduce Phase (decimal days)
Pairs (plain)	2 (basic)	1	1321230	0.25	63.72%	752320	0.15
Pairs Combiner	2 (basic)	1	2376450	0.46	57.49%	1757580	0.34
Pairs Partitioner	2 (basic)	1	1752440	0.34	69.31%	775910	0.15
Local Aggregation - in mapper (local)	2 (basic)	1	1150490	0.22	61.49%	720530	0.14
Local Aggregation - in mapper (global)	2 (basic)	1	1016900	0.20	58.60%	718360	0.14
Stripes (plain)	2 (basic)	1	258200	0.05	33.22%	519090	0.10
Stripes Combiner	2 (basic)	1	1929590	0.37	59.90%	1291790	0.25
Pairs (plain)	4	1	1354840	0.26	56.57%	1040160	0.20
Pairs Combiner	4	1	2276070	0.44	58.98%	1582900	0.31
Pairs Partitioner	4	1	1352020	0.26	57.23%	1010370	0.19
Local Aggregation - in mapper (local)	4	1		0.00			0.00
Local Aggregation - in mapper (global)	4	1		0.00			0.00
Stripes (plain)	4	1	320720	0.06	28.18%	817510	0.16
Stripes Combiner	4	1		0.00			0.00
Pairs (plain)	2	5	1234070	0.24	58.74%	866820	0.17
Pairs Combiner	2	5	2999430	0.58	80.18%	741360	0.14
Pairs Partitioner	2	5	1333640	0.26	62.11%	813640	0.16
Local Aggregation - in mapper (local)	2	5	1063330	0.21	77.38%	310810	0.06
Local Aggregation - in mapper (global)	2	5	1176320	0.23	55.25%	952650	0.18
Stripes (plain)	2	5	258200	0.05	33.22%	519090	0.10
Stripes Combiner	2	5	928130	0.18	52.81%	829430	0.16
Pairs (plain)	4	5	1240980	0.24	58.37%	885180	0.17
Pairs Combiner	4	5	2187660	0.42	70.51%	915050	0.18
Pairs Partitioner	4	5	2165200	0.42	70.80%	892810	0.17
Local Aggregation - in mapper (local)	4	5		0.00			0.00
Local Aggregation - in mapper (global)	4	5		0.00			0.00
Stripes (plain)	4	5	352650	0.07	37.98%	575870	0.11
Stripes Combiner	4	5		0.00			0.00

Common Crawl Data (1158 MB - 10 Files)

Algorithm Implementation	Cluster	Reducers	Map Phase	Map Phase (decimal days)	Map Phase %	Reduce Phase	Reduce Phase (decimal days)
Pairs (plain)	2 (basic)	1	13416430	2.59	76.54%	4111630	0.79
Pairs Combiner	2 (basic)	1	13857300	2.67	84.68%	2506320	0.48
Pairs Partitioner	2 (basic)	1	13417900	2.59	73.39%	4866110	0.94
Local Aggregation - in mapper (local)	2 (basic)	1	10965660	2.12	75.86%	3489570	0.67
Local Aggregation - in mapper (global)	2 (basic)	1	10522530	2.03	76.16%	3294420	0.64
Stripes (plain)	2 (basic)	1	2553850	0.49	24.25%	7979340	1.54
Stripes Combiner	2 (basic)	1	10595320	2.04	49.48%	10816070	2.09
Pairs (plain)	4	1	86368	0.02	26.32%	241780	0.05
Pairs Combiner	4	1		0.00			0.00
Pairs Partitioner	4	1		0.00			0.00
Local Aggregation - in mapper (local)	4	1		0.00			0.00
Local Aggregation - in mapper (global)	4	1		0.00			0.00
Stripes (plain)	4	1	2573750	0.50	20.45%	10010480	1.93
Stripes Combiner	4	1	24967980	4.82	0.8164633518	5612670	1.08
Pairs (plain)	2	5	12475500	2.41	75.72%	4000920	0.77
Pairs Combiner	2	5	13046490	2.52	83.59%	2562140	0.49
Pairs Partitioner	2	5	13245840	2.56	75.20%	4368950	0.84
Local Aggregation - in mapper (local)	2	5	10183080	1.96	75.61%	3284230	0.63
Local Aggregation - in mapper (global)	2	5	9735100	1.88	75.21%	3208960	0.62
Stripes (plain)	2	5	2585600	0.50	27.31%	6882790	1.33
Stripes Combiner	2	5					
Pairs (plain)	4	5	14019810	2.70	63.19%	8168140	1.58
Pairs Combiner	4	5		0.00			
Pairs Partitioner	4	5		0.00			
Local Aggregation - in mapper (local)	4	5		0.00			
Local Aggregation - in mapper (global)	4	5		0.00			
Stripes (plain)	4	5	2564420	0.49	29.28%	6193740	1.19
Stripes Combiner	4	5					

Amazon Reviews Data (333.2 MB / 820.3 MB uncompre

Algorithm Implementation	Cluster	Reducers	Map Phase	Map Phase (decimal days)	Map Phase %	Reduce Phase	Reduce Phase (decimal days)
Pairs (plain)	2 (basic)	1	5279220	1.02	71.81%	2072020	0.40
Pairs Combiner	2 (basic)	1	5158970	1.00	92.10%	442430	0.09
Pairs Partitioner	2 (basic)	1	5203500	1.00	81.27%	1199570	0.23
Local Aggregation - in mapper (local)	2 (basic)	1	5050920	0.97	76.15%	1581560	0.31
Local Aggregation - in mapper (global)	2 (basic)	1	4855710	0.94	72.78%	1815670	0.35
Stripes (plain)	2 (basic)	1	992150	0.19	56.18%	773750	0.15
Stripes Combiner	2 (basic)	1	2126870	0.41	83.90%	408160	0.08
Pairs (plain)	4	1	5292330			2059540	
Pairs Combiner	4	1					
Pairs Partitioner	4	1					
Local Aggregation - in mapper (local)	4	1					
Local Aggregation - in mapper (global)	4	1					
Stripes (plain)	4	1					
Stripes Combiner	4	1					
Pairs (plain)	2	5					
Pairs Combiner	2	5	5118680	0.99	71.81%	2009800	0.39
Pairs Partitioner	2	5	4989320	0.96	88.00%	680610	0.13
Local Aggregation - in mapper (local)	2	5	5178648	1.00	967.09%	593411	0.11
Local Aggregation - in mapper (global)	2	5	4646550	0.90	71.14%	1885050	0.36
Stripes (plain)	2	5	4720380	0.91	73.48%	1703400	0.33
Stripes Combiner	2	5	997200	0.19	56.04%	782180	0.15
Pairs (plain)	4	5					
Pairs Combiner	4	5					
Pairs Partitioner	4	5					
Local Aggregation - in mapper (local)	4	5					
Local Aggregation - in mapper (global)	4	5					
Stripes (plain)	4	5					
Stripes Combiner	4	5					

Appendix 2

Algorithm Implementation	Cluster	Reducers	Reduce Phase %	Total (ms)	Total (decimal days)	result file size (GB)
Pairs (plain)	2 (basic)	1	36.28%	2073550	0.40	1.25
Pairs Combiner	2 (basic)	1	42.51%	4134030	0.80	1.25
Pairs Partitioner	2 (basic)	1	30.69%	2528350	0.49	1.25
Local Aggregation - in mapper (local)	2 (basic)	1	38.51%	1871020	0.36	1.25
Local Aggregation - in mapper (global)	2 (basic)	1	41.40%	1735260	0.33	1.25
Stripes (plain)	2 (basic)	1	66.78%	777290	0.15	0.39
Stripes Combiner	2 (basic)	1	40.10%	3221380	0.62	0.39
Pairs (plain)	4	1	43.43%	2395000	0.46	1.25
Pairs Combiner	4	1	41.02%	3858970	0.74	1.25
Pairs Partitioner	4	1	42.77%	2362390	0.46	1.25
Local Aggregation - in mapper (local)	4	1				
Local Aggregation - in mapper (global)	4	1				
Stripes (plain)	4	1	71.82%	1138230	0.22	0.39
Stripes Combiner	4	1			0.00	
Pairs (plain)	2	5	41.26%	2100890	0.41	1.25
Pairs Combiner	2	5	19.82%	3740790	0.72	1.25
Pairs Partitioner	2	5	37.89%	2147280	0.41	1.25
Local Aggregation - in mapper (local)	2	5	22.62%	1374140	0.27	1.25
Local Aggregation - in mapper (global)	2	5	44.75%	2128970	0.41	1.25
Stripes (plain)	2	5	66.78%	777290	0.15	0.39
Stripes Combiner	2	5	47.19%	1757560	0.34	0.39
Pairs (plain)	4	5	41.63%	2126160	0.41	1.25
Pairs Combiner	4	5	29.49%	3102710	0.60	1.25
Pairs Partitioner	4	5	29.20%	3058010	0.59	1.25
Local Aggregation - in mapper (local)	4	5				
Local Aggregation - in mapper (global)	4	5				
Stripes (plain)	4	5	22.46%	928520	0.18	0.39
Stripes Combiner	4	5				

Algorithm Implementation	Cluster	Reducers	Reduce Phase %	Total	Total (decimal days)	result file size (GB)
Pairs (plain)	2 (basic)	1	23.46%	17528060	3.38	7.1
Pairs Combiner	2 (basic)	1	15.32%	16363620	3.16	7.1
Pairs Partitioner	2 (basic)	1	26.61%	18284010	3.53	7.1
Local Aggregation - in mapper (local)	2 (basic)	1	24.14%	14455230	2.79	7.1
Local Aggregation - in mapper (global)	2 (basic)	1	23.84%	13816950	2.67	7.1
Stripes (plain)	2 (basic)	1	75.75%	10533190	2.03	2.2
Stripes Combiner	2 (basic)	1	50.52%	21411390	4.13	2.2
Pairs (plain)	4	1	73.68%	328148	0.06	7.1
Pairs Combiner	4	1				
Pairs Partitioner	4	1				
Local Aggregation - in mapper (local)	4	1				
Local Aggregation - in mapper (global)	4	1				
Stripes (plain)	4	1	79.55%	12584230	2.43	2.2
Stripes Combiner	4	1	0.1835366482	30580650	5.90	2.2
Pairs (plain)	2	5	24.28%	16476420	3.18	7.1
Pairs Combiner	2	5	16.41%	15608630	3.01	7.1
Pairs Partitioner	2	5	24.80%	17614790	3.40	7.1
Local Aggregation - in mapper (local)	2	5	24.39%	13467310	2.60	7.1
Local Aggregation - in mapper (global)	2	5	24.79%	12944060	2.50	7.1
Stripes (plain)	2	5	72.69%	9468390	1.83	2.2
Stripes Combiner	2	5				
Pairs (plain)	4	5	36.81%	22187950	4.28	
Pairs Combiner	4	5				
Pairs Partitioner	4	5				
Local Aggregation - in mapper (local)	4	5				
Local Aggregation - in mapper (global)	4	5				
Stripes (plain)	4	5	70.72%	8758160	1.69	2.2
Stripes Combiner	4	5				

essed - 1 File)

Algorithm Implementation	Cluster	Reducers	Reduce Phase %	Total	Total (decimal days)	result file size (GB)	
Pairs (plain)	2 (basic)	1	28.19%	7351240	1.42	1.3	
Pairs Combiner	2 (basic)	1	7.90%	5601400	1.08	1.3	
Pairs Partitioner	2 (basic)	1	18.73%	6403070	1.24	1.3	
Local Aggregation - in mapper (local)	2 (basic)	1	23.85%	6632480	1.28	1.3	
Local Aggregation - in mapper (global)	2 (basic)	1	27.22%	6671380	1.29	1.3	
Stripes (plain)	2 (basic)	1	43.82%	1765900	0.34	0.3785	
Stripes Combiner	2 (basic)	1	16.10%	2535030	0.49	0.3785	
Pairs (plain)	4	1		7351870			
Pairs Combiner	4	1					
Pairs Partitioner	4	1					
Local Aggregation - in mapper (local)	4	1					
Local Aggregation - in mapper (global)	4	1					
Stripes (plain)	4	1					
Stripes Combiner	4	1					
Pairs (plain)	2	5					
Pairs Combiner	2	5	28.19%	7128480	1.38	1.30	
Pairs Partitioner	2	5	12.00%	5669930	1.09	1.30	
Local Aggregation - in mapper (local)	2	5	110.82%	535486	0.10	1.30	
Local Aggregation - in mapper (global)	2	5	28.86%	6531600	1.26	1.30	260.2 x 5
Stripes (plain)	2	5	26.52%	6423780	1.24	1.30	260.2 x 5
Stripes Combiner	2	5	43.96%	1779380	0.34	0.38	75.2, 75.8, 76.2, 81.6, 69.7
Pairs (plain)	4	5					
Pairs Combiner	4	5					
Pairs Partitioner	4	5					
Local Aggregation - in mapper (local)	4	5					
Local Aggregation - in mapper (global)	4	5					
Stripes (plain)	4	5					
Stripes Combiner	4	5					