Soft co-clustering in Map-Reduce using distributed Sparse Matrix Regression

Ioakim Perros

Submitted to the Department of Electronic and Computer Engineering
in partial fulfillment of the requirements for the

ECE Diploma Degree
Technical University of Crete, Greece

Thesis committee:
Advisor: Professor Minos Garofalakis
Assistant Professor Antonios Deligiannakis
Assistant Professor Michail G. Lagoudakis

October 2012

ABSTRACT

Modern technological and scientific advancements coupled with the massive increase in computing power and data storage capacity have given rise to a different paradigm of computer science. Often hailed as the Fourth Paradigm, it encompasses systems and techniques geared towards the analysis of and extraction of actionable knowledge from the abundance of generated data. Co-cluster analysis is an example of such useful knowledge extraction that describes the process of discovering concepts of a data matrix emerging from the correlations of row and column subsets.

This thesis studies the design and development of a co-clustering algorithm targeted for big data analysis. Our implementation is based on the Map-Reduce model and its open-source implementation, Hadoop. In contrast to earlier work, we approach Map-Reduce co-clustering through distributed optimization techniques exploiting the idea of Sparse Matrix Regression (SMR) which can define "soft" co-clusters capturing important correlations and masking noise in the input data. We give novel formulations of the different SMR subproblems in a parallelizable way over the Map-Reduce model of computation, and present an experimental study on real-life data that establishes the accuracy and scalability of our approach, as well as its ability to consistently discover new, robust concepts in large data collections.

ACKNOWLEDGMENTS

First, there are a lot of things for which I would like to thank my parents, Anastassia and Manoussos and my brother Iosif, but I have to thank them for the most important: for loving me the way they do.

Next, I would like to thank my advisor, professor Minos Garofalakis, for the trust he showed in me, his ideas and our discussions about my future steps.

I would also like to thank Evangelos Papalexakis, for greatly explaining many details of the method he developed and his aid as a whole. Furthermore, I would like to thank professor Athanasios Liavas for his willingness to explain optimization related material I had never encountered before.

Xenia Arapi is a person I would like to express my gratitude to, as well, for setting up HBase to our cluster, just for my work and for her availability to help if technical issues occured.

My dear Athena is the next person I would like to thank, for her continuous encouragement through this tough period. Without her support, the fulfillment of this thesis would literally not be possible.

Finally, I would like to thank my great friends, Lefteris, Nick P., Nikos K., Manolis, for all the fun, surprises, support and experiences we shared, through all these years.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 THESIS CONTRIBUTION

This thesis studies the design and development of a co-clustering algorithm, a process of discovering concepts of a data matrix emerging from the correlations of row and column subsets, targeted for big data analysis. Our implementation is based on the Map-Reduce model and its open-source implementation, Hadoop.

Earlier work view co-clustering as a "hard" partitioning process, where all input rows and columns belong to only one co-cluster. In contrast to this, we approach Map-Reduce co-clustering through distributed optimization techniques exploiting the idea of Sparse Matrix Regression (SMR) which can define "soft" co-clusters capturing important correlations and masking noise in the input data.

This is extremely valuable in the case of big data analysis, because usually as data availability grows, so does level of possible noise in the data.

We give novel formulations of the different SMR subproblems in a parallelizable way over the Map-Reduce model of computation, and present an experimental study on real-life data that establishes the accuracy and scalability of our approach, as well as its ability to consistently discover new, robust concepts in large data collections.

1.2 THESIS ORGANIZATION

Chapter 2 initiates the discussion upon the big data initiative, which provides the necessary motivation to focus on a different paradigm of algorithm design, oriented towards data-intensive applications. Furthermore, a brief view of the scientific field we are examining is provided, as well as a description of programming models and frameworks utilized in this thesis. In Chapter 3, we describe the main idea behind co-clustering, by pointing out its importance to many applications and its difference from its one-sided counterpart. We also refer to the previous work published as of this method, regarding both sequential and parallel (large-scale) co-clustering approaches. We describe, in Chapter 4, the main concept of Sparse Matrix Regression and provide the background of the Alternating Direction of Multipliers Method, which enables us to tackle co-clustering through Map-Reduce model. In Chapter 5, we present the way of expressing the problem in Map-Reduce by pointing out the reasons why this approach is scalable to large-scale datasets and explaining various implementation details and choices. We provide, in Chapter 6, our experimental results on real-life data and discuss them thoroughly, in terms of accuracy on both matrix dimensions (pointing out the significance of co-clustering) and scaling. Finally, we

conclude this thesis, by mentioning some general inferences about this work, while pointing out at the same time future research directions.

2

BACKGROUND

In this section, we adduce the challenges posed by our era's technological advancements, which reflect the significance of a different paradigm of algorithm design. Furthermore, a brief view of the scientific field we are examining is provided, as well as a description of programming models and frameworks utilized in this thesis.

2.1 BIG DATA INITIATIVE

Imagine every thousandth blood cell in human body has a tiny radio transmitter in it. Imagine that 10 times a second that transmitter sends each cell's location to a computer storing the data. Along with position, it also sends the concentration of a list of 10 chemicals encountered at receptors distributed at 10 sites over the surface of each cell. Now imagine following all those blood cells for an hour. That makes a billion blood cells being sampled 10 times a second for 3,600 seconds. Now imagine a task of sorting through all those numbers and extracting something meaningful about the human body. That problem,(which is not far from our era [1]), would be an example of "Big Data".

In information technology (branch of engineering dealing with the use of computers and telecommunications equipment to store, retrieve, transmit and manipulate data), Big Data is a term applied to voluminous unstructured and structured data sets, which, because of their size, cannot be reasonably stored in typical databases for easy processing and managing. Data sources are everywhere, from Web 2.0 and user-generated content to large scientific experiments, from social networks to wireless sensor networks - Figure 1 contains an interesting infographic on how much data are being generated within 60 seconds by many famous web services. Typical examples of massive data sources are the following:

- The Large Hadron Collider (LHC) near Geneva is the world's largest particle accelerator, designed to probe the mysteries of the universe, including the fundamental nature of matter, by recreating conditions shortly following the Big Bang. Experiments at the LHC produce 30 Petabytes - 30 million Gigabytes - of data every year, which has to be stored, backed up, and made available to more than 8,500 scientists around the globe.[2]

- Astronomers have long recognized the importance of a "digital observatory" that would support the data needs of researchers across the globe - the Sloan Digital Sky Survey is perhaps the most well known of these projects. Looking into the future, the Large Synoptic Survey Telescope (LSST) is a wide-field instrument that is capable of observing the entire sky every few days. When the

Figure 1: Data being generated from various web sources within 1 minute [5]

telescope comes online around 2015 in Chile, its 3.2 Gigapixel primary camera will produce approximately half a Petabyte of archive images every month. [3]

- The advent of next-generation DNA sequencing technology has created a deluge of sequence data that needs to be stored, organized, and delivered to scientists for further study. The European Bioinformatics Institute (EBI), which hosts a central repository of sequence data called EMBL-bank, has increased storage capacity from 2.5 Petabytes in 2008 to 5 Petabytes in 2009. Scientists are predicting that, in the not-so-distant future, sequencing an individual's genome will be no more complex than getting a blood test today - ushering a new era of personalized medicine, where interventions can be specifically targeted for an individual.

One of the greatest challenges for 21st-century science is how we respond to this new era of data-intensive science. This is recognized as a new paradigm beyond experimental and theoretical research and computer simulations of natural phenomena - one that requires new tools, techniques, and ways of working, which has been hailed as the emerging "fourth paradigm" of science [6]. To this direction, institutions and organizations of all size are turning to people who are capable of translating this trove of data, into valuable knowledge, establishing the data scientist profession as one of the most sought-after professions of our times.

2.2 COMPUTING IN THE CLOUDS

Lately cloud computing has received a substantial amount of attention from industry, academia and press. As a result, the term "Cloud Computing" has become a buzzword, overloaded with meanings. There

is lack of consensus on what is and what is not cloud. Even simple client-server applications are sometimes included in the category [7]. The boundaries between similar technologies are fuzzy, so there is no clear distinction among grid, utility, cloud, and other kinds of computing technologies. In spite of the many attempts to describe cloud computing [8], there is no widely accepted definition.

However, within cloud computing, there is a more cohesive subset of technologies which is geared towards data analysis. We refer to this subset as Data Intensive Scalable Computing (DISC) systems. These systems are aimed mainly at I/O intensive tasks, are optimized for dealing with large amounts of data and use a data-parallel approach. An interesting feature is that they are "dispersed": computing and storage facilities are distributed, abstracted and intermixed - while implementing a term known as "utility computing", which, as the name implies, supposes treatment of computing resources as a metered service, like electricity or natural gas. This idea harkens back to the days of time-sharing machines, and in truth is not very different from this antiquated form of computing. Under this model, a "cloud user" can dynamically provision any amount of computing resources from a "cloud provider" on demand and only pay for what is consumed. In practical terms, the user is paying for access to virtual machine instances that run a standard operating system such as Linux. These systems attempt to move computation as close to data as possible because moving large quantities of data is expensive. Finally, the burden of dealing with the issues caused by parallelism is removed from the programmer. This provides the programmer with a scale-agnostic programming model. More details on the programming model used in this thesis (implemented above a cloud computing system with the aforementioned characteristics) are given in Section 2.5.

2.3 DATA MINING

The aforementioned data revolution provides, through its manifestations, the data from which a data scientist initiates his path, possibly using a cloud computing framework as described above. The destination of this path consists of extracting *knowledge* about the initial data, through a procedure called Data Mining. The most commonly accepted definition of data mining is the discovery of "models" for data [9]. A "model," however, can be one of several things and below we refer to the most important directions.

2.3.1 *Statistical Modeling*

Statisticians were the first to use the term "data mining". Originally, "data mining" or "data dredging" was a derogatory term referring to attempts to extract information that was not supported by the data. Today, "data mining" has taken on a positive meaning. Now, statisticians view data mining as the construction of a statistical model, that is, an underlying distribution from which the visible data is drawn.

2.3.2 *Machine Learning*

There are some who regard data mining as synonymous with machine learning. There is no question that some data mining appropriately uses algorithms from machine learning. Machine-learning practitioners use the data as a training set, to train an algorithm of one of the many types used by machine-learning practitioners (such as Bayes nets, support-vector machines, decision trees, hidden Markov models, and many others).

There are situations where using data in this way makes sense. The typical case where machine learning is a good approach is when we have little idea of what we are looking for in the data. For example, it is rather unclear what it is about movies that makes certain moviegoers like or dislike it. Thus, in answering the "Netflix challenge" [10] to devise an algorithm that predicts the ratings of movies by users, based on a sample of their responses, machine-learning algorithms have proved quite successful.

2.3.3 *Computational Approaches to Data Modeling*

More recently, computer scientists have looked at data mining as an algorithmic problem and there are many different approaches to this direction. Having already mentioned the possibility of constructing a statistical process whereby the data could have been generated, most other approaches to data modeling can be described as either:

1. Extracting the most prominent features of the data and ignoring the rest, or
2. Summarizing the data succinctly and approximately.

2.3.3.1 *Feature Extraction*

The typical feature-based model looks for the most extreme examples of a phenomenon and represents the data by these examples. Some of the important kinds of feature extraction, of particular interest because of the abundance of data they are usually accompanied by, are:

1. Frequent Itemsets. This model makes sense for data that consists of "baskets" of small sets of items, as in the market-basket problem. We look for small sets of items that appear together in many baskets, and these frequent itemsets are the characterization of the data that we seek. The original application of this sort of mining was true market baskets: the sets of items, that people tend to buy together when checking out at the cash register of a store or super market.
2. Similar Items. Often, data looks like a collection of sets, and the objective is to find pairs of sets that have a relatively large fraction of their elements in common. An example is treating customers at an on-line store like Amazon as the set of items they have bought.

> In order for Amazon to recommend something else they might like, Amazon can look for "similar" customers and recommend something many of these customers have bought. This process is called "collaborative filtering."

2.3.3.2 *Summarization*

One of the most important types of data mining for massive datasets is summarization and one of the most interesting forms of it, is the PageRank idea, which made Google successful. In this form of Web mining, the entire complex structure of the Web is summarized by a single number for each page and as an oversimplification, this number, the "PageRank" of the page, is the probability that a random walker on the graph of all worldwide websites, would be at that page at any given time. The remarkable property this ranking has is that it reflects very well the "importance" of the page - the degree to which typical searchers would like that page returned as an answer to their search query.

Another important form of summarizing data constitutes clustering (or cluster analysis), which will introduce us to the topic of this thesis. Cluster analysis (clustering) is the organization of a collection of patterns (usually represented as vectors of measurements, or as points in a multidimensional space) into clusters based on similarity. Intuitively, patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster. An example of clustering is depicted in Figure 3, where the input patterns are shown in (a) and the desired clusters are shown in (b). Here, points belonging to the same cluster are given the same label. Data modeling puts clustering in a historical perspective rooted in mathematics, statistics, and numerical analysis. From a machine learning perspective clusters correspond to hidden patterns, the search for clusters is unsupervised learning, and the resulting system represents a data concept. From a practical perspective clustering plays an outstanding role in data mining applications such as scientific data exploration, information retrieval and text mining, spatial database applications, Web analysis, CRM, marketing, medical diagnostics, computational biology, and many others [11].

Before approaching clustering through a more algorithmic point of view, it would be interesting to refer to a famous instance of it, with historical significance. Long ago in London, the physician John Snow, dealing with a Cholera outbreak plotted the cases on a map of the city. A small illustration suggesting the process is shown in Figure 2.

The cases clustered around some of the intersections of roads. These intersections were the locations of wells that had become contaminated and as a result people who lived nearest these wells got sick, while people who lived nearer to wells that had not been contaminated did not get sick. Without the ability to cluster the data, the cause of Cholera would not have been discovered. This discovery came to influence public health and the construction of improved sanitation facilities beginning in the 19th century [12].

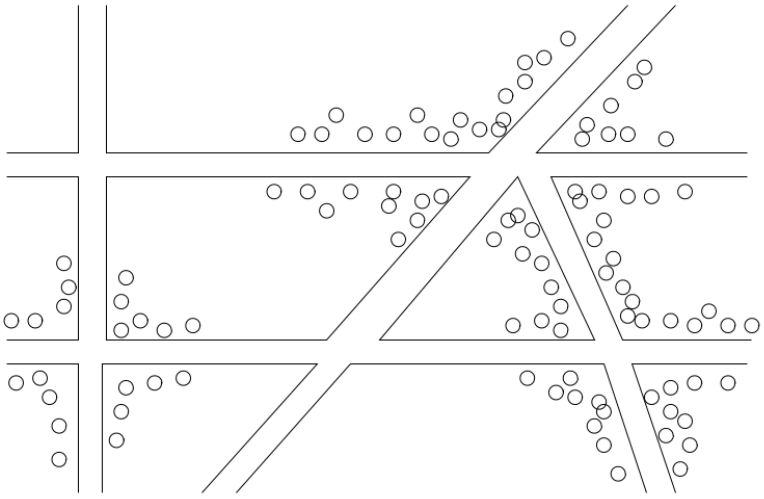

Figure 2: Plotting cholera cases on a map of London

2.4 CLUSTERING

The variety of techniques for representing data, measuring proximity between data elements, and grouping data elements has produced a rich and often confusing assortment of clustering methods, which may be distinguished in two categories:

1. *Partitional methods* Given a database of n objects, a partitional clustering algorithm constructs k partitions of the data, so that an objective function is optimized. One of the issues with such algorithms is their high complexity, as some of them exhaustively enumerate all possible groupings and try to find the global optimum. Even for a small number of objects, the number of partitions is huge. That is why common solutions start with an initial, usually random, partition and proceed with its refinement. A better practice is to run the partitional algorithm for several different sets of k initial points (considered as representatives), and keep the result with the best quality.

   Partitional clustering algorithms try to locally improve a certain criterion. The majority of them could be considered as greedy algorithms, i.e., algorithms that at each step choose the best solution and may not lead to optimal results in the end. The best solution at each step is the placement of a certain object in the cluster for which the representative point is nearest to the object. The most famous algorithm of this family is *k-means* [15] and an instance of its application is depicted below.

2. *Hierarchical methods* They create a hierarchical decomposition of the objects, being either agglomerative (bottom-up) or divisive (top-down):

   a) Agglomerative algorithms start with each object being a separate cluster itself, and successively merge groups according to a distance measure. The clustering may stop when all

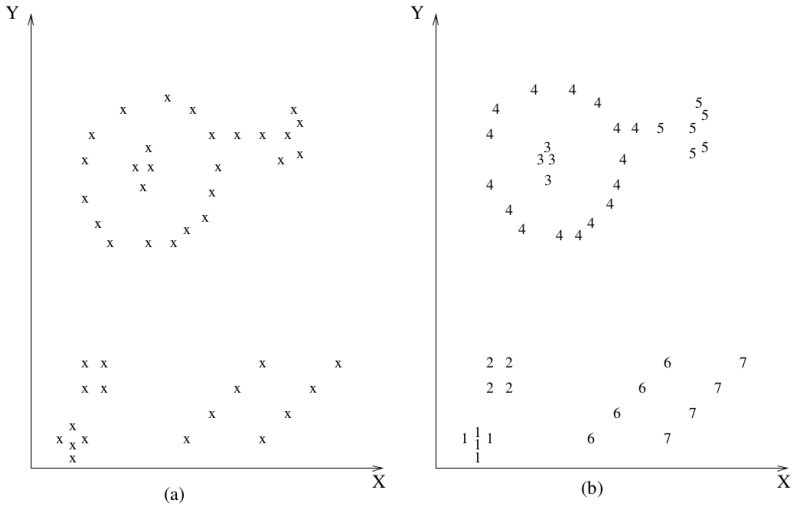

Figure 3: Clustering example [14]

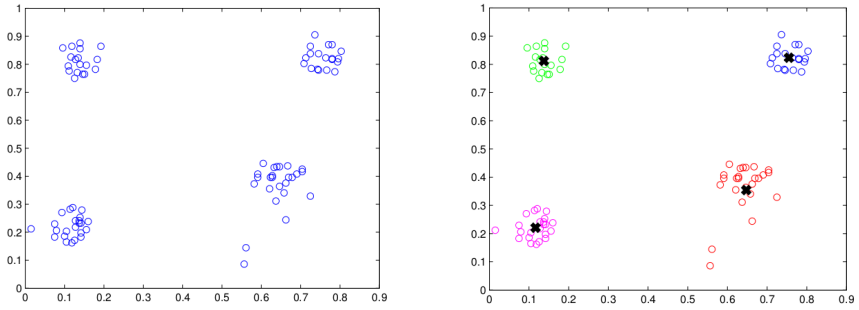

Figure 4: K-means goal: partition points into k clusters, by minimizing squared Euclidean distance from points to their cluster centroids

objects are in a single group or at any other point the user chooses. These methods generally follow a greedy bottom-up merging.

b) Divisive algorithms follow the opposite strategy. They start with one group of all objects and successively split groups into smaller ones, until each object falls in one cluster, or until a desired number of clusters is reached. This is similar to the approach followed by divide-and-conquer algorithms, i.e., algorithms that, given an instance of the problem to be solved, split this instance into several, smaller, sub-instances (of the same problem), independently solve each of the sub-instances and then combine the sub-instance solutions so as to yield a solution for the original instance.

Partitional and hierarchical methods can be integrated. For example, a result given by a hierarchical method can be improved via a partitional step, which refines the result via iterative relocation of points. A representation of the above description is given in Figure 5.

Returning to the classical K-means clustering, it is shown [17] to be computationally difficult, despite that its objective seems straightfor-

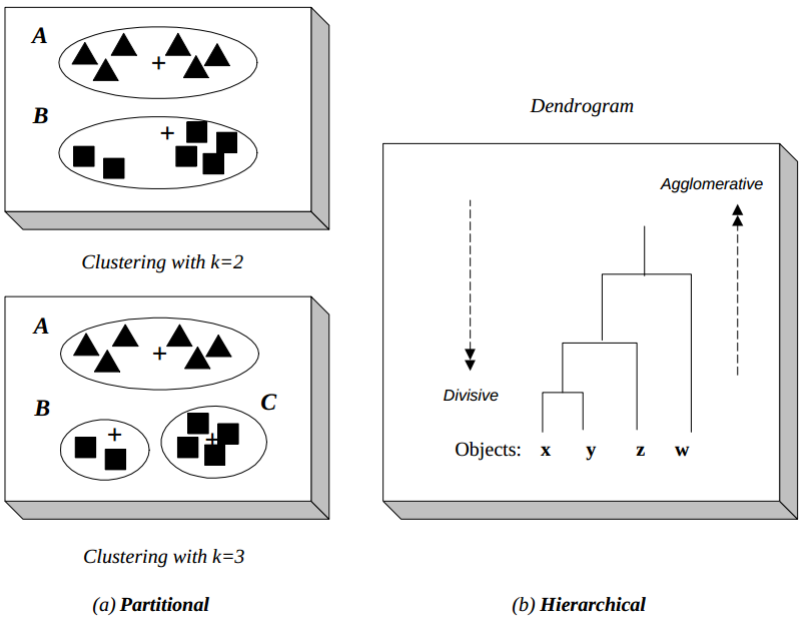

Figure 5: Partitional clustering for k=2,3. Hierarchical clustering dendrogram

ward. In particular, its hardness is described by the class of NP-Hard problems, which means that an *optimal* solution is unattainable within a reasonable amount of time [1]. However, it is wise to sacrifice optimality and settle for a good feasible solution that can be computed efficiently.

2.5 MAP-REDUCE FRAMEWORK

An introduction to Map-Reduce programming model is provided below, along with descriptions of open-source implementations utilized in this thesis that are inextricably related to this model.

Map-Reduce is a programming model introduced by Google [33] in 2004, to support distributed computing on large datasets. Today Google's Map-Reduce framework is used inside Google to process data on the order of petabytes on a network of few thousand computers. The framework is inspired by *map* and *reduce* functions, commonly used in functional programming.

The basic characteristic of this model is that the whole processing is divided in two steps: *map* and *reduce*, while all input/output/intermediate data are being encoded as pairs of a key and a value attached to this key.

Input data have to be organized in such pairs and the framework is responsible for the *map* function's execution and the output of intermediate $< key, value >$ pairs. Map-Reduce framework processes this output before transferring it to the *reduce* phase. This processing sorts and groups the $< key, value >$ pairs *by key*. Finally, the *reduce* function

1 A more strict definition of NP-hard class problems is as follows: a problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.

processes the aforementioned tuples and usually emits a "reduced" set of them.

To better understand the Map/Reduce, lets consider an example. Given below are the map and reduce function for categorizing a set of numbers as even or odd.

Algorithm 1 Even or Odd MR example

```
1: procedure MAP(String key, Integer values)
2: //key: File Name
3: //values: list of numbers
4:     for each v in values do
5:         if v%2 == 0 then
6:             emit < "even", v >
7:         else
8:             emit < "odd", v >
9:         end if
10:    end for
11: end procedure
12: procedure REDUCE(String key, Iterator values)
13: //key: Even or Odd
14: //values: Iterator over list of numbers
15: //(categorized as odd or even)
16:    String val = values.next()
17:    while values.hasNext() do
18:        val = val +", "+ values.next()
19:    end while
20:    emit < key, val >
21: end procedure
```

This is a very simple example where both the map and reduce function do not extract anything much interesting. But a programmer has the freedom to write something a lot more complex through these functions, as will be seen in the coming chapters.

Map-Reduce is Turing Complete. This definition describes a system in which a program can be written that will find an answer - although with no guarantees regarding runtime or memory ([13]), so all problems can be expressed in this model. However, it does not provide advantages for all problems. The ones that match its philosophy are those that:

- process parts of data independently from each other,
- require only batch computations (on static data sets),
- work with input data easily expressed as $< key, value >$ pairs,
- handle huge load, even TB's of data,
- can be expressed as a sequence of Map and Reduce functions.

For problem cases handling moderate size of input data, usage of this model is unworthy, since it leads to delays concering data partitioning and task sharing, that are comparable with execution's runtime.

2.5.1 *Map-Reduce model key ideas*

In this subsection, we discuss a number of "big ideas" within Map-Reduce that established it as the most popular parallel programming model handling massive data, over the last few years. [3]

- **Scale *out*, not *up*:** For data-intensive workloads, a large number of commodity low-end servers (i.e., the scaling out approach) is preferred over a small number of high-end servers (i.e., the scaling up approach). The latter approach of purchasing symmetric multi-processing (SMP) machines with a large number of processor sockets (dozens, even hundreds) and a large amount of shared memory (hundreds or even thousands of gigabytes) is not cost effective, since the costs of such machines do not scale linearly (i.e., a machine with twice as many processors is often significantly more than twice as expensive). On the other hand, the low-end server market overlaps with the high-volume desktop computing market, which has the effect of keeping prices low due to competition, interchangeable components, and economies of scale.

- **Assume failures are common - providing fault tolerance:** At warehouse scale, failures are not only inevitable, but commonplace. A simple calculation suffices to demonstrate: let us suppose that a cluster is built from reliable machines with a mean-time between failures (MTBF) of 1000 days (about three years). Even with these reliable servers, a 10,000-server cluster would still experience roughly 10 failures a day. For the sake of argument, let us suppose that a MTBF of 10,000 days (about thirty years) were achievable at realistic costs (which is unlikely). Even then, a 10,000-server cluster would still experience one failure daily. This means that any large-scale service that is distributed across a large cluster (either a user-facing application or a computing platform like MapReduce) must cope with hardware failures as an intrinsic aspect of its operation. That is, a server may fail at any time, without notice. For example, in large clusters disk failures are common and RAM experiences more errors than one might expect. Datacenters suffer from both planned outages (e.g., system maintenance and hardware upgrades) and unexpected outages (e.g., power failure, connectivity loss, etc.).

  Mature implementations of the Map-Reduce programming model are able to robustly cope with failures through a number of mechanisms such as automatic task restarts on different cluster nodes.

- **Move processing to the data:** In traditional high-performance computing (HPC) applications (e.g., for climate or nuclear simulations), it is commonplace for a supercomputer to have processing nodes and storage nodes linked together by a high-capacity interconnect. Many data-intensive workloads are not very processor-demanding, which means that the separation of compute and storage creates a bottleneck in the network. As an alternative to moving data around, it is more efficient to move the processing

around. That is, MapReduce assumes an architecture where processors and storage (disk) are co-located. In such a setup, we can take advantage of data locality by running code on the processor directly attached to the block of data we need. The distributed file system is responsible for managing the data over which MapReduce operates.

- **Process data sequentially and avoid random access:** Data - intensive processing by definition means that the relevant datasets are too large to fit in memory and must be held on disk. Seek times for random disk access are fundamentally limited by the mechanical nature of the devices: read heads can only move so fast and platters can only spin so rapidly. As a result, it is desirable to avoid random data access, and instead organize computations so that data is processed sequentially. A simple scenario poignantly illustrates the large performance gap between sequential operations and random seeks: assume a 1 terabyte database containing 10^{10} 100-byte records. Given reasonable assumptions about disk latency and throughput, a back-of-the-envelop calculation will show that updating 1% of the records (by accessing and then mutating each record) will take about a month on a single machine. On the other hand, if one simply reads the entire database and rewrites all the records (mutating those that need updating), the process would finish in under a work day on a single machine. Sequential data access is, literally, orders of magnitude faster than random data access.

  The development of solid-state drives is unlikely to change this balance for at least two reasons. First, the cost differential between traditional magnetic disks and solid-state ones remains substantial: large-data will for the most part remain on mechanical drives, at least in the near future. Second, although solid- state disks have substantially faster seek times, order-of-magnitude differences in performance between sequential and random access still remain. Map-Reduce is primarily designed for batch processing over large datasets. To the extent possible, all computations are organized into long streaming operations that take advantage of the aggregate bandwidth of many disks in a cluster. Many aspects of Map-Reduce's design explicitly trade latency for throughput.

- **Hide system-level details from the application developer:** The challenges in writing distributed software are greatly compounded - the programmer must manage details across several threads, processes, or machines. Of course, the biggest headache in distributed programming is that code runs concurrently in unpredictable orders, accessing data in unpredictable patterns. This gives rise to race conditions, deadlocks, and other well-known problems. Programmers are taught to use low-level devices such as mutexes and to apply high-level "design patterns" such as producer-consumer queues to tackle these challenges, but the

truth remains: concurrent programs are notoriously difficult to reason about and even harder to debug.

Map-Reduce addresses the challenges of distributed programming by providing an abstraction that isolates the developer from system-level details (e.g., locking of data structures, data starvation issues in the processing pipeline, etc.). The programming model specifies simple and well-defined interfaces between a small number of components, and therefore is easy for the programmer to reason about. Map-Reduce maintains a separation of what computations are to be performed and how those computations are actually carried out on a cluster of machines. The first is under the control of the programmer, while the second is exclusively the responsibility of the execution framework or runtime.

2.5.2 *Map-Reduce or libraries (MPI/OpenMP) for parallel execution?*

Before the development of Map-Reduce programming model, program parallelization could be achieved through appropriate modification so that they make use of certain libraries, such as MPI, or OpenMP. However, these methods have certain drawbacks:

1. These libraries do not facilitate users who are not experts at parallel/distributed programming. Even though, people who posess these skills face difficulties, as they are responsible both for the resource allocation at cluster's machines, as well as for the processing part. As mentioned above, for the Map-Reduce case, the whole resource allocation process remains transparent to the user.

2. Map-Reduce communicates between nodes by disk I/O (on HDFS/GFS, which will be mentioned below, which is faster than NTFS/EXT3), while MPI performs communication by message passing.

3. Map-Reduce provides a fault-tolerant mechanism, that is, when one node fails, map-reduce restarts the same task on another node. All MPI processes will exit if one of them fails.

Generally, libraries, such as MPI, give more freedom to the programmer, which leads to more difficulties during program development. These MPI drawbacks are effectively eliminated through the simple Map-Reduce model.

2.5.3 *Hadoop & HDFS*

Today, Hadoop [34] is the most well-known open-source implementation of the Map-Reduce programming model. It has a worldwide impact and some of the companies using it (apart from Yahoo!), are Last.fm, Facebook, New York Times etc. It is being implemented in Java and supports multiple classes facilitating code development. Map-Reduce

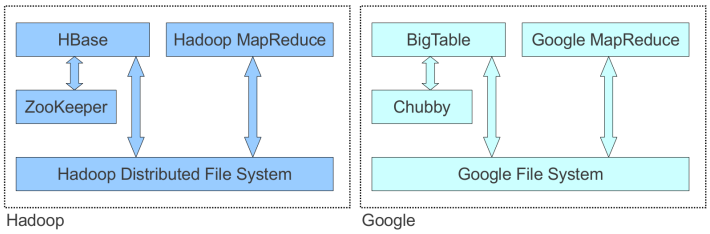

Figure 6: Google's and the respective open-source terminology

programs executed in Hadoop may be developed in other languages as well (apart from Java), such as Python, Ruby, C++.

HDFS is the distributed file system used by Hadoop, sharing many common characteristics with Google's File System(GFS) [38]. Because those characteristics were analyzed in Map-Reduce section, we refer to them, from a viewpoint focused at HDFS:

- High fault tolerance: In large-scale distribyted systems, hardware failures are a commonplace and HDFS cares to locate those nodes and protects the user from losing data and unpredictable crashes.
- Streaming data access: HDFS was created to process large scale data in batches (batch processing) and for high throughput achievement.
- Large input data: HDFS supports very large file storage by splitting them into blocks.
- Calculation transfer is cheaper than data transfer: it is obvious that calculations are more efficient when they are executed close to the data in use. Performance difference is perceptible for large scale input data. HDFS prefers to transfer calculations to other nodes, than transferring respective data, so it posesses mechanisms permitting applications to be moved closer to data - achieving better *data locality*.

2.5.4 *HBase*

HBase ([35]) is an Apache open-source project, the goal of which is to provide BigTable-like [36] storage(designed to scale to very large databases) for the Hadoop DFS. As Hadoop and HDFS constitute open-source implementations of Map-Reduce and GFS, HBase is an open-source platform emulating Google's BigTable (this nomenclature is shown below in Figure 6).

HBase is a type of "NoSQL" database. "NoSQL" is a general term meaning that the database is not an RDBMS (Relational DataBase Management System) which supports SQL as its primary access language. Technically speaking, HBase is really more a "Data Store" than "Data Base" because it lacks many of the features we find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages.

In general words, it constitutes a *sparse*, *consistent*, *distributed*, *multidimensional*, *sorted map* [37, 39]. Below we refer to each of these terms separately to focus on HBase's capabilities:

- **Map:** HBase maintaints maps of *keys* to *values* (key→value). Each of these mappings is called a *KeyValue* or a *Cell*, and each value can be retrieved by each respective key.
- **Sorted:** These cells are sorted by the key. This is a very important property as it allows efficiency in searching (ability to retrieve for example all values between keys X and Y), rather than just retrieving a value for a known key.
- **Multidimensional:** The key itself has structure. Each key consists of the following parts: row-key, column family, column, and timestamp. So the mapping is actually: (rowkey, column family, column, timestamp) → value. Rowkey and value are bytes, so anything that can be serialized into a byte array can be stored into a cell.
- **Sparse:** This follows from the fact that HBase stores key → value mappings and that a "row" is nothing more than a grouping of these mappings (identified by the rowkey mentioned above). Unlike NULL in most relational databases, no storage is needed for absent information. In this case, there will be just no cell for a column that does not have any value. It also means that every value carries all its coordinates with it.
- **Distributed:** One key feature of HBase is that the data can be spread over 100s or 1000s of machines and reach billions of cells. HBase manages the load balancing automatically and if the application-specific context requires it, user is able to take control of distribution management in certain ways.
- **Consistent:** HBase guarantees that all changes referring to the same rowkey are *atomic*. A reader will always read the last written (and committed) values.

As concerns **inner structure**, HBase partitions the key space in *Tables* and each one of them declares one or more *column families*, which define the storage properties for an arbitrary set of *columns*.

As depicted in Figure 7, the structure of records in HBase's tables consist of mappings from (rowkey, column family, column, timestamp) to a value.

- **Rowkey** is application-specific and allows the user to define the desired sort order. Defining the right sort order is extremely important as scanning is the only way of retrieving any value for which the key is not known a-priori.

  The rowkey also provides a logical grouping of cells and HBase ensures that all cells with the same rowkey are co-located on the same server (called a RegionServer in HBase), which allows for

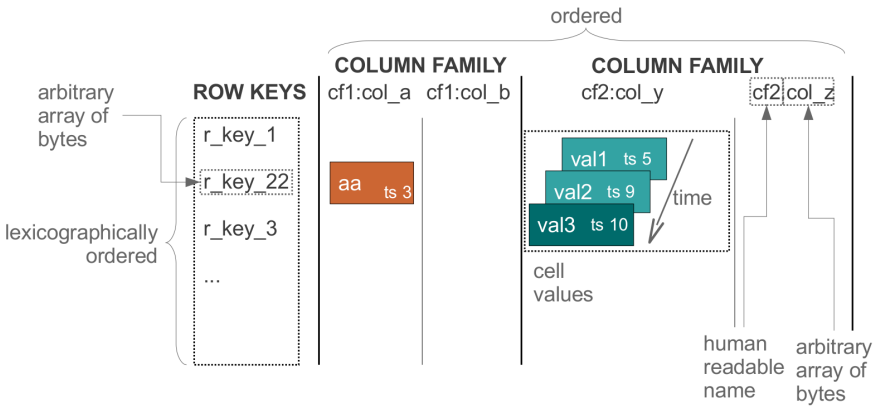

Figure 7: HBase Datamodel view

ACID (atomicity, consistency, isolation, durability) guarantees for updates with the same rowkey without complicated and slow two-phase-commits.

- **Column families** are declared when a table is created. They define storage attributes such as compression, number of versions to maintain etc.
- **Columns** are arbitrary names (or labels) assigned by the application.
- **Timestamp** is a Long identifying (by default) the creation time of the cell. Each cell is versioned, and every "update" creates a new version of the affected set of cells.

Furthermore, referring to **consistency**, HBase ensures that all new versions created by single put operations for a particular rowkey are either all seen by other clients or seen by none, as well as that a Get or Scan will only return a combination of versions of cells for a row that existed together at some point. This ensures that no client will ever see a partially completed update or delete.

This is being achieved by a variation of Multi Version Concurrency Control (MVCC) - which is defined as the implementation of updates not by deleting an old piece of data and overwriting it with a new one, but instead by marking the old data as obsolete and adding the newer version. This variation used by HBase, is called Multi Version *Consistency* Control, and the main idea revolves around the existence of internal timestamps (apart from the ones discussed above), called memstoreTS's, which handle the whole update process and ensure atomicity. However, further discussion upon this fact is beyond the scope of this introduction.

Finally, as for the **physical storage**, puts and deletes are collected into an in-memory structure called the MemStore. When it reaches a certain size, MemStore is flushed to disk into StoreFiles.

Periodically StoreFiles are compacted into fewer StoreFiles. For reading and writing HBase employs Log Structured Mergetrees, which in

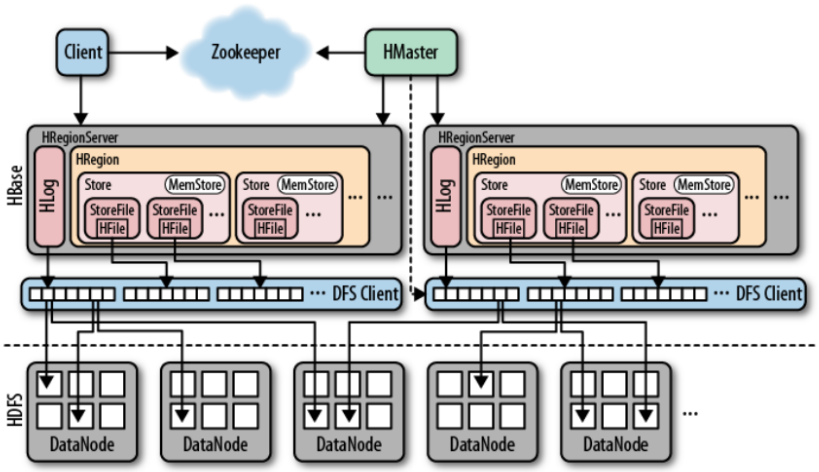

Figure 8: HBase's physical storage general view

```
Table       (HBase table)
    Region       (Regions for the table)
        Store          (Store per ColumnFamily for each Region for the table)
            MemStore           (MemStore for each Store for each Region for the table)
            StoreFile          (StoreFiles for each Store for each Region for the table)
                Block              (Blocks within a StoreFile within a Store for each Region for the table)
```

Figure 9: HBase's object hierarchy

other words means that, reading and compacting in HBase is performing a merge sort (a scan looks at the heads of all StoreFiles and the Memstore picks the smallest element first, which in case of a Scan is returned to the client and in case of a compaction is written to the new StoreFile).

The aforementioned compactions constitute the *minor* ones, while *major compactions* eventually compact the entire set of files into a single one, after which the flushes start adding smaller files again.

Since all store files are immutable, there is no way to delete a particular value out of them, nor does it make sense to keep rewriting large store files to remove the deleted cells one by one. Instead, a tombstone marker is written, which masks out the "deleted" information-which can be a single cell, a range of cells, or entire rows.

The older (referring to timestamp) versions of each column are kept into HBase's files, but deletion, in fact, is only possible when a major compaction has been performed, after which the older versions are removed forever, using the predicate delete based on the configured maximum versions to retain. This is the reason why we have mentioned above that the "delete" term is not accurate enough - true removals happen only after a major compaction.

A bird's eye view of the physical storage is provided in Figure 8. The Figure shows that HBase handles basically two kinds of file types: one is used for the write-ahead log (used to secure data from crashes) and the other for actual data storage. The files are primarily handled by the HRegionServers. HRegionServer opens a region and creates a corresponding HRegion object. When an HRegion is opened, it sets up a Store instance for each HColumnFamily for every table as defined by the user beforehand. Each Store instance can, in turn, have one or more StoreFile instances, which are lightweight wrappers around the

actual storage file called HFile. A Store also has a MemStore, and the HRegionServer a shared HLog instance. A complete object hierarchy is shown in 9.

3

PROBLEM STATEMENT-RELATED WORK

In this section, we attempt to provide the concept of co-clustering, by pointing out its importance to many applications and its difference from one-sided clustering. Furthermore, we discuss related work on both sequential and large-scale co-clustering approaches.

3.1 KEY INTUITION

In several applications, the data itself has a lot of structure, which may be hard to capture using a traditional clustering objective. Consider the example of a boolean/binary matrix, whose rows correspond to objects and columns correspond to their features, and an entry is one if and only if an object is related to a feature. The goal is to cluster both objects and features of the matrix. One way to accomplish this would be to independently cluster rows and columns using the standard notion of clustering (i.e. cluster similar advertisers and cluster similar keywords).

To be more explanatory, given a data matrix, if we would like to learn more about its structure and possible partitions, as a first approach we could consider each row as a vector in multi-dimensional space (where the number of dimensions would be equal to the number of columns), and perform clustering (e.g., by k-means) of these vectors. The same process could then be followed for column vectors.

Even though for some criteria this might be a reasonable solution, such an endeavor might fail to elicit subtle structures that might exist in the data. The *key intuition and main power* of co-clustering against one-sided clustering can be seen through the following examples:

- Consider that rows correspond to keywords and columns correspond to advertisers, and an entry is one if and only if the advertiser has placed a bid on the keyword. The goal, again, is to cluster both the advertisers and the keywords. Perhaps, there are two disjoint sets of advertisers A_1, A_2 and keywords K_1, K_2 such that each advertiser in A_i bids on each keyword in K_j if and only if i = j.

- Another example would be a marketing application, where each customer is represented by a vector, across a list of products and vice-versa, where we would not be interested in grouping customers (or products), but rather in spotting *subsets* of customers that tend to buy the same *subset* of products - *even though their overall buying patterns could otherwise be very different.* These subsets of interest are not known a-priori but had we known them, the problem would be "reduced" to clustering across a subset of dimensions.

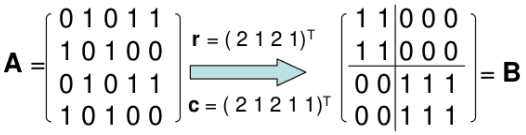

Figure 10: A co-clustering example: Given A, find group assignments r and c such that the resulting sub-matrices in matrix B are highly correlated. B is permuted according to co-clustering assignments.

To be able to discover the aforementioned structures, the clustering objective has to simultaneously intertwine information about both the advertisers and keywords that are present in the matrix. So, if we would prefer extracting "blocks" (or "co-clusters") of inter-related rows and columns rather than similar rows or columns as a whole, co-clustering (also mentioned as biclustering, block/direct/multi-way/simultaneous/two-mode/two-sided/two-way/subspace clustering) is precisely the technique we should aim at, for solving our problem. A typical example of co-clustering is depicted in Figure 10.

3.2 APPLICATIONS

A number of empirical studies have demonstrated the usefulness of co-clustering algorithms. A brief summary of them follows:

- Simultaneous clustering of documents and words in text mining ([29]): The key task is to identify document and word clusters from a bag-of-words model represented in a vector space in the form of word-by-document matrix.
- Microarray (i.e., genes and experimental conditions) in bioinformatics ([19]): The main idea of discovering latent local patterns is compatible with the current understanding of cellular processes that a subset of genes are coregulated under certain experimental conditions, but to behave almost independently under other conditions. The main task is to identify groups of similar genes and similar conditions based on their expression levels. For an extended survey referring to co-clustering approaches of this type of data, see [20].
- Metabolic Screening: Many governments have employed screening of newborns in order to detect metabolic diseases in the earliest possible moment. For this purpose, a blood sample is taken from each newborn and the concentrations of specific metabolites (i.e., metabolic products) in these blood samples are measured. In the resulting data matrix, rows represent the newborns and columns represent the metabolites. Biologists usually want to identify homogeneous groups of newborns suffering from a common metabolic disease. Usually, each metabolic disease causes a correlation between the concentration of a specific set of metabolites. Thus, any clustering algorithm should take into account that newborns should be grouped together only if they exhibit a

common correlation among a set of metabolites. In addition, the set of participating metabolites and the type of correlation can be different for different diseases (i.e., clusters).

- Customer Recommendation Systems: Generalizing the aforementioned example about marketing applications, in customer recommendation systems, customers of a company can vote for the company's products. Depending on the portfolio of the company, there may be a very large set of products. It is now interesting, for example, for target marketing purposes, to cluster the customers into groups of homogeneous voting schemata. Customers that have similar preferences should be grouped together. For each group, special marketing strategies can be applied taking each group's preferences into account. The problem for a cluster analysis process is that different customers may be grouped together according to different sets of products. In other words, customer A may share a preference for a given set S of products with customer B but not with C, whereas A may share another preference for a different set T of products with C but not with B. To make the problem even more challenging, the relationships between the preferences of the customers of one cluster may be arbitrary complex, like "The lower the products $p1$ and $p2$ are rated, the higher the products $p3$ and $p4$ are rated." Symmetrically, clustering products based on similar customer preferences (where only a small subset of customers may be sufficient to establish similarity among products) is also a common problem.

- Tokens and contexts in natural language processing: For most natural language processing applications, the number of tokens and contexts is extremely large, making it infeasible to directly employ computationally intensive learning algorithms. Co - clustering alleviates this problem by constructing new features in a more compact but highly informative representation from co-cluster centroids.

- Quantized image patches are represented as image features in co-occurrence matrices of images and low level features [22]. Then, using the spectral bipartite graph partitioning algorithm [21], the authors demonstrate that the co-clustering has better retrieval performance as well as a computational advantage over the traditional k-means clustering algorithm, especially for high dimensional feature vectors.

- Users and movies in recommender systems [23]: An efficient real-time collaborative filtering (CF) framework for the movie rating matrix consisting of users and items (i.e., movies) is proposed. The key idea is to simultaneously obtain user and item neighborhoods via the co-clustering and generate predictions based on the average ratings of the co-clusters, a hybrid of incremental and batch versions of the algorithm is proposed to reflect new users' ratings. Furthermore, parallel CF based on parallel co-clustering is discussed.

- Missing value prediction in recommender systems & co - clustering categorical data matrices: Banerjee et al. ([24]) propose co - clustering based missing value estimation for collaborative filtering-based recommender systems. The authors assume a low parameter structure by using the Bregman co-clustering algorithm with a suitably weighted loss function, where weight is one for known ratings and zero otherwise (i.e., missing ratings). On the other hand, the authors consider data matrices consisting of categorical values from a finite set such as market-basket data matrices with users by products with the entries corresponding to preferred brands.

3.3 INFLUENTIAL WORKS

In this subsection, we give a short overview of a few influential works in the field of co-clustering. For a deeper review of co-clustering approaches, we refer to the excellent surveys of the field, including [26], [20], [27].

As a preamble, we would like to point that, as proved in [25], partition-based (hard) [1] co-clustering belongs to the class of NP-Hard problems, by reduction from the one-sided clustering problem. This finding explains mathematically the reason why all co-clustering works seek approximations of the optimal solution (similar to one-sided clustering solutions in the literature).

- Apparently the earliest (1972) biclustering algorithm that may be found in the literature is so-called direct clustering by Hartigan [28], also known as block clustering. This approach relies on statistical analysis of submatrices to form the biclusters. Namely, the quality of a bicluster (k,l) is assessed by the variance

$$Var(k,l) = \sum_{i \in k} \sum_{j \in l} (a_{ij} - \mu_k)^2$$

where μ_k is the average value in the bicluster, i.e.:

$$\mu_k = \frac{\sum_{i \in k} \sum_{j \in l} a_{ij}}{|k||l|}$$

A bicluster is considered perfect if it has zero variance, so biclusters with lower variance are considered to be better than biclusters with higher variance. This, however, leads to an undesirable effect: single-row, single-column submatrices become ideal biclusters as their variance is zero. The issue is resolved by fixing the number of biclusters and minimizing the objective:

1 meaning that all rows/columns must be present in one and only row/column cluster forming a checkerboard partitioning of input matrix

$$Var(k,l) = \sum_{k=1}^{r} \sum_{i \in k} \sum_{j \in l} (a_{ij} - \mu_k)^2$$

Hartigan mentioned that other objective functions may be used to find biclusters with other desirable properties, e.g., minimizing variance in rows, variance in columns, or biclusters following certain patterns.

- A more sophisticated criterion for constructing patterned biclusters was introduced by Cheng and Church [19]. It is based on minimization of the so-called mean squared residue. To formulate it, let us introduce the following notation. Let

$$\mu_{ik} = \frac{1}{|k|} \sum_{j \in k} a_{ij}$$

be the mean of the ith row in the row cluster k,

$$\mu_{jk} = \frac{1}{|l|} \sum_{i \in l} a_{ij}$$

be the mean of the jth column in the column cluster l and μ_k the same as in the above case (expressing the mean value of the bicluster (k,l)). The *residue* of an element a_{ij} is defined as

$$r_{ij} = a_{ij} - \mu_{ik} - \mu_{jk} + \mu_k$$

and the mean squared residue (H_k) of the bicluster (k,l) is defined as the sum of squares of all r_{ij} of the resulting matrix. This value is equal to zero if all columns of the bicluster are equal to each other - that would imply that all rows are equal too. A bicluster (k,l) is called a δ-bicluster if $H_k \leq \delta$. Cheng and Church proved that finding the largest square δ-bicluster is NP-hard([19]). So, the general idea was that using a greedy procedure starting from the entire data matrix and successively removing columns or rows contributing most to the mean squared residue score, one could end up having a well-defined partitioning of the initial matrix.

- Dhillon ([21]) approaches co-clustering as a problem of partitioning a bipartite graph via the algorithm Singular Value Decomposition (known as SVD), and Dhillon et al. ([29]) provide a fundamental analysis for co-clustering via information theory. The main intuition of the latter is as follows:

  Consider a joint probability distribution p(X,Y). The relative entropy, or the Kullback-Leibler (KL) divergence between two probability distributions $p_1(x)$ and $p_2(x)$ is defined as

$$p(X,Y) = \begin{bmatrix} .05 & .05 & .05 & 0 & 0 & 0 \\ .05 & .05 & .05 & 0 & 0 & 0 \\ 0 & 0 & 0 & .05 & .05 & .05 \\ 0 & 0 & 0 & .05 & .05 & .05 \\ .04 & .04 & 0 & .04 & .04 & .04 \\ .04 & .04 & .04 & 0 & .04 & .04 \end{bmatrix} \qquad p(\hat{X},\hat{Y}) = \begin{bmatrix} .3 & 0 \\ 0 & .3 \\ .2 & .2 \end{bmatrix}$$

Figure 11: ITCC: On the left, we see the joint probability distribution of X & Y and on the right, the same amount concerning the "clustered" random variables.

$$D(p_1||p_2) = \sum_x p_1(x) log \frac{p_1(x)}{p_2(x)}$$

Kullback-Leibler divergence can be considered as a distance of a true distribution p_1 to an approximation p_2.

The mutual information I(X;Y) of two random variables X and Y is the amount of information shared between these two variables. In other words, I(X;Y) = I(Y;X) measures how much X tells about Y and, vice versa. It is defined as:

$$I(X;Y) = \sum_y \sum_x p(x,y) log \frac{p(x,y)}{p(x)p(y)} = D(p(x,y)||p(x)p(y))$$

The authors define the quality of a co-clustering by the resulting *loss in Mutual Information*:

$$\min_{\hat{X},\hat{Y}} I(X;Y) - I(\hat{X};\hat{Y})$$

where $\hat{X}, \hat{Y}$ represent the "clustered" random variables, which are extracted through the following formula:

$$p(\hat{x},\hat{y}) = \sum_{x \in \hat{x}} \sum_{y \in \hat{y}} p(x,y)$$

An example of this calculation is depicted in Figure 11, where rows of the joint probability distribution $p(x,y)$ are grouped into three clusters: $\hat{x_1} = \{x_1, x_2\}$, $\hat{x_2} = \{x_3, x_4\}$ and $\hat{x_3} = \{x_5, x_6\}$. Similarly, the natural column clustering is : $\hat{y_1} = \{y_1, y_2, y_3\}$ and $\hat{y_2} = \{y_4, y_5, y_6\}$.

A main observation is that if

$$q(x,y) = p(\hat{x},\hat{y})p(x|\hat{x})p(y|\hat{y})$$

where

$$p(x|\hat{x}) = p(x)/p(\hat{x})$$

if row x is assigned to cluster $\hat{x}$ and zero otherwise, then the above loss in mutual information can be expressed as:

$$I(X;Y) - I(\hat{X};\hat{Y}) = D(p(X,Y)||q(X,Y))$$

In other words, finding an optimal co-clustering is equivalent to finding a distribution q as defined above, which is close to p in KL divergence.

Dhillon et al. [29] succeed in expressing the loss in MI, with respect to only row or column distributions. This leads to an iterative algorithm that alternatively updates the row and column clustering, This enables them to alternatively updates the row and column clustering, by successively minimizing the respective costs.

- In [24], Banerjee et al. also view co-clustering as a partitional problem that is driven by the search for a good approximation of the original matrix, where the quality of each co-clustering solution is determined by the approximation error. In this context, they formulate a unified view of co-clustering algorithms, termed as Bregman co-clustering (BCC), which allows the aforementioned error to be measured using a large class of loss functions called *Bregman divergences*. These divergences constitute (by an oversimplistic view) a large class of well-behaved loss functions with a number of desirable properties [24].

  Two sub-cases of this class are:

  – I-Divergence: Given $z \in R_+$, let $\phi(z) = zlog(z)$
  For $z_1, z_2 \in R_+ : d_\phi(z_1, z_2) = z_1 log(z_1/z_2) - (z_1 - z_2)$
  – Squared Euclidean Distance: Given $z \in R$, let $\phi(z) = z^2$
  For $z_1, z_2 \in R : d_\phi(z_1, z_2) = (z_1 - z_2)^2$

  These are used by [29] and [30], respectively, as the aforementioned Information - Theoretic approach, and Minimum Sum-Square Residue Co-clustering, are special cases of this generalized framework.

  Another significant quantity for this framework is Bregman information which is defined as the expected Bregman divergence to the expectation, i.e.:

$$I_\phi(Z) = E[d_\phi(Z, E[Z])]$$

  Extending the same sub-cases as above, we have:

  – I-Divergence: Given a real non-negative random variable Z, the Bregman information is $I_\phi(Z) = E[Zlog(Z/E[Z])]$
  – Squared Euclidean Distance: Given any real random variable Z, the Bregman information is $I_\phi(Z) = E[(Z - E[Z])^2]$

In addition, Banerjee et al. [24], permit multiple structurally different co-clustering schemes that preserve various linear statistics of the original data matrix. They focus on summary statistics that correspond to conditional expectations over partitions that result from the rows, columns and co-clusterings, while establishing that there are exactly six non-trivial co-clustering schemes. Each of these schemes corresponds to a unique co-clustering basis, that is, a combination of conditional expectations over various partitions. Existing partitional co-clustering algorithms presented in [29] and [30], employ one of the six co-clustering bases.

To sum up, this co-clustering process is guided by the search for the matrix approximation that has the minimum Bregman information while preserving the specified co-clustering statistics, and a pseudocode of its general view is provided below:

Algorithm 2 Bregman CC algorithm

Inputs: Matrix Z, Bregman divergence d_ϕ, #row clusters l, #column clusters k,representation scheme C

Outputs: Locally optimal co-clustering $(\rho^\star, \gamma^\star)$

1: Randomly initialize ρ^0 and γ^0
2: **repeat**
3: $t \leftarrow t + 1$
4: Obtain minimum Bregman information solution for $\hat{Z}(\rho^t, \gamma^t, C)$ (matrix approx.)
5: $\rho^{t+1} \leftarrow \arg\min_\rho E[d_\phi(Z, \hat{Z}(\rho, \gamma^t, C))]$ (row clustering)
6: $\gamma^{t+1} \leftarrow \arg\min_\gamma E[d_\phi(Z, \hat{Z}(\rho^{t+1}, \gamma, C))]$(column clustering)
7: **until** convergence

- In [31], a method for extracting co-clusters (referred to as cross - associations) for binary data is proposed. The main idea is to divide each iteration into two alternating steps: the search for a good co-clustering solution for a given number of row/column clusters and the search for this number of row/column groups. The former step is achieved through the "classic" way (as in [29] and the more general [24]) of placing each row in the best (in terms of cost minimization) row group, given a column clustering - the symmetric step for columns w.r.t. to row clusters follows. The latter step is tackled through splitting the group (row/column respectively) posessing the maximum entropy per row/column.

  The cost function used, is based on the Minimum Description Language principle [32] and the general idea is to minimize the bits required, in order for a transmitter to broadcast *accurately* (i.e., by lossless compression), each given co-clustering solution to a hypothetical receiver.

  More specifically, let A denote an a x b binary matrix. Define:

$$n_1(A) := \text{number of nonzero entries in A}$$
$$n_0(A) := \text{number of zero entries in A}$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Figure 12: Example matrix A, from [31]

$$n(A) := n_1(A) + n_0(A) = \text{a x b}$$
$$P_A(i) := n_i(A)/n(A), \text{ i} = 0, 1.$$

Given the knowledge of the matrix dimensions (a,b) and the distribution P_A, matrix A can be encoded as follows:

If we attempt to find the number of necessary bits to encode a *uniform* distribution of e.g. 8 possible outcomes, we would need at least 3 bits to encode each event, and this number results by taking $log(8) = 3$ [2], because each event can occur with probability $1/8$. If this distribution is not uniform though, the probability of each event would not be equal among the possible events, so the number of bits necessary to encode each one of them, would be equal to $log(8/N_i)$, where N_i is the number of occurences of each event i. So, by generalizing, we have that whenever an event $i(i = 0/1)$ is encountered, it can be encoded using $log(n(A)/n_i(A))$ bits, on average.

As a result, if we multiply each event's occurences by the above amount of bits it needs to be encoded, and sum for all possible events, we end up to the total number of necessary bits to describe the matrix as a whole. Mathematically:

$$C(A) = \sum_{0}^{1} n_i(A) log(\frac{n(A)}{n_i(A)}) = n(A)H(P_A(0)) \tag{1}$$

where H is the binary Shannon entropy function.

For instance, using the example matrix of Figure 12, we can compute the following: $n_1(A) = 4, n_0(A) = 12, n(A) = 16, P_A(1) = 1/4, P_A(0) = 3/4$. So, according to 1, the total code length for matrix A is: $4log(4) + 12log(4/3) = 16H(1/4)$.

By summing the above code length objective, used to quantify the number of bits required to describe *the contents of each co-cluster*, with the number of necessary bits to describe *each co-clustering assignement* (i.e.: the bits required to send the number of row/column groups, the number of rows/columns in each of the groups as well as the number of ones for all the co-clusters), we have the MDL objective function being minimized in [31].

2 all logarithms in this section are base 2

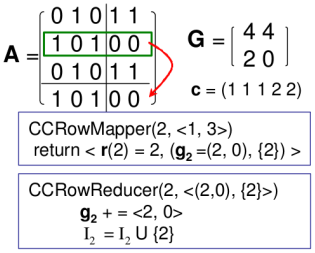

Figure 13: Example of Map and Reduce function of [40]
Best choice for 2nd row is the 2nd row group

3.4 LARGE-SCALE CO-CLUSTERING

Below we provide a brief description of works which specifically tackled the co-clustering problem through the Map-Reduce model [33], targeting big data analysis.

- Papadimitriou and Sun [40] propose a *hard* co-clustering solution for *binary* data expressed in Map-Reduce. Their approach is based on the same principle followed by [24], where summary statistics of a (permuted according to a co-clustering assignement) matrix are maintained (as depicted in Figure 11 for the information theoretic sub-case), and row with column cluster assignements are alternatively updated separately until the algorithm converges. The cost function for those summary statistics is the MDL-based cost proposed in [31] (Equation 1).

  Initially, the transpose of original matrix is pre-computed, to be used uring the column cluster assignement process; furthermore, (row/column) labeling vectors (r, c) that depict the cluster to which each row/column has been assigned so far are initialized (as is G, representing summary statistics and maintaining the number of non-zeros for each row-column group intersection). Then, in the Map phase, each row is placed in every row cluster sequentially, and the assignement that results in the lowest MDL-cost is chosen. The row group in which each row ended up in, constitutes the key, while the partial statistics of this row with respect to the fixed column groups, as well as the row's id, form together the value of each $< key, value >$ pair being emitted from Map phase.

  As a fundamental property of the Map-Reduce framework, at the Reduce phase, $< key, value >$ pairs sharing the same key, are being processed by the same Reducer's instance. Consequently, partial statistics for each row cluster are summed and a set union adds each row to the respective row group. An example of this Map-Reduce phase is being shown in Figure 13.

  An additional *global sync* step follows each iteration in order to collect new results for global statistics' matrix G and row group assignements. The same process as a whole is followed for columns, and row/column cluster improvement steps are executed until convergence.

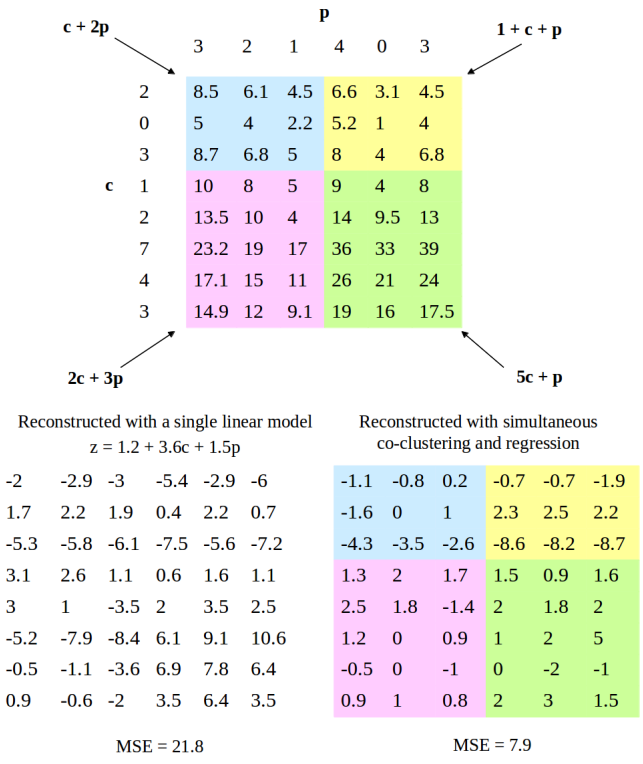

Figure 14: Difference between attempting to build separately a linear model for each co-cluster and the matrix a as whole [41]

- Deodhar and Ghosh [41], employ an approach of simultaneous co-clustering and classification, in order to exploit the neighborhood information provided by each co-clustering assignment for a better classification accuracy, assuming we want to classify the contents of a real-valued data matrix. They attempt to model each matrix value by an approximate linear combination of both objects and features, in order to finally, end up to a separate linear model for each co-cluster (rather than having a single model for the whole matrix). In Figure 14, the difference between classification error of both cases is obvious. Referring solely to the co-clustering method, this approach uses a sub-case of the aforementioned generalized Bregman co-clustering [24], by minimizing the total squared error between the original matrix and its approximation (but this approximation here is built upon each co-cluster's linear model). In [42], this method has been expressed in Map-Reduce. The co-clustering approach does not essentially differ from the aforementioned DisCo work [40] and a third step is added to the row and column cluster assignments to express the model building step - these three steps represent each algorithm's iteration.

4

SMR CO-CLUSTERING AND ADMM

In this section, we introduce both the problem of Sparse Matrix Regression (SMR) and the Alternating Direction of Multipliers optimization method (ADMM). The solution we give in Chapter 5 of SMR using ADMM, is what allows us to tackle the Sparse Matrix Regression problem through the Map-Reduce model.

4.1 INTRODUCTION TO SPARSE MATRIX REGRESSION

A brief review of fundamental definitions and background of Sparse Matrix Regression are provided below, to facilitate understanding of the approach.

4.1.1 *Lasso Regression*

Linear Regression aims to model variables using linear combinations of certain observations or measurements. The observations are usually called predictors and are symbolized by $\mathbf{x}$ and the outputs are called responses and are symbolized by y. Linear Regression provides an estimate $\hat{y}$ of the actual output. Consider the input vector:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$$

The linear regression model has the form:

$$\hat{y} = f(x) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j$$

We assume that the relationships between $\mathbf{x}$ and y are linear or approximately linear. The parameters β_i are initially unknown and are computed via a training process. Let us denote as $\mathbf{X}$ the $N \times p$ input matrix, with each row containing an input vector. Let us denote as $\mathbf{y}$ the vector with the corresponding outputs to each row of $\mathbf{X}$. The most popular estimator for β_i is the least squares estimator which minimizes the *residual sum of squares* (RSS):

$$RSS(\beta) = \sum_{i=1}^{N} (\mathbf{y}_i - f(\mathbf{x}_i))^2 = \sum_{i=1}^{N} (\mathbf{y}_i - \sum_{j=1}^{p} \mathbf{X}_{ij} \beta_j)^2$$

Instead of just minimizing the Residual Sum of Squares function, the group of shrinkage least squares methods also introduces an additional penalty to the cost function which eventually leads to more interpretable models by shrinking or discarding some of the predictor values.

The Lasso is such a method for linear regression and was first introduced in [47]. The lasso minimizes the RSS, penalizing the optimization

process with the absolute value of the coefficients. By posing this constraint, some coefficients tend to become zero, thus providing a more conceptually interpretable model.

The setting for lasso is the same as the one described in the linear regression model. We assume $N \times p$ matrix $\mathbf{X}$ containing the predictor values and a vector $\mathbf{y}$ keeping the responses. We also denote:

$$\hat{\beta} = \begin{bmatrix} \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{bmatrix}$$

Then, the lasso estimate $(\hat{a}, \hat{b})$ is obtained through the following optimization formula:

$$\begin{aligned} \min_{\beta} \quad & \sum_{i=1}^{N} (\mathbf{y}_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i,j})^2 \\ \text{subject to} \quad & \sum_{j=1}^{p} |\beta_j| \leq t \end{aligned}$$

The upper bound t is a tuning parameter. For relatively small values of t, the solutions are shrunken versions of the least squares estimates, justifying the term "shrinkage method". Often, some of the coefficients β_j are zero, a property that is quite often desirable. An alternative way of expressing the lasso equation is the following, with $\lambda \geq 0$ playing an equivalent role to t:

$$\min_{\beta} \quad \sum_{i=1}^{N} (\mathbf{y}_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i,j})^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

4.1.2 *Non-Negative Matrix Factorization*

Non-Negative Matrix Factorization (NNMF) can be expressed as a non-linear, constrained optimization problem, as follows:

Consider a (generally) non-negative matrix $\mathbf{M} \in \mathbb{R}^{I \times J}$. We want to decompose $\mathbf{M}$ into two factors $\mathbf{A} \in \mathbb{R}^{I \times \hat{k}}$, $\mathbf{B} \in \mathbb{R}^{J \times \hat{k}}$, where $\hat{k} \leq rank(M)$ [1] minimizing the following objective function:

$$\begin{aligned} \min_{\mathbf{A},\mathbf{B}} \quad & ||\mathbf{M} - \mathbf{A}\mathbf{B}^T||_F^2 \\ \text{subject to} \quad & a_{i,j} \geq 0 \text{ and } b_{i,j} \geq 0 \end{aligned}$$

In the factorization principle, each row of the initial data matrix $\mathbf{M}$ is expressed as a linear combination of the columns of $\mathbf{B}$, weighted by the elements of the corresponding row of $\mathbf{A}$. The fact that this linear

1 The rank of a matrix is the maximum number of linearly independent rows (or columns).

combination contains strictly additive relations (imposed by the non-negativity constraints) between its elements, provides an interpretable model.

In regards to the computation of NNMF, besides the most popular multiplicative method, a more simplistic approach for solving it is Alternating Least Squares (ALS). It works by decomposing the non-linear optimization problem into two linear ones. In general, the method works as follows:

1. Fix matrix $\mathbf{A}$ to initially random values
2. Solve the Non-negative Least Squares problem (NNLS):

$$\begin{aligned} \min_{\mathbf{B}} \quad & ||\mathbf{M} - \mathbf{A}\mathbf{B}^T||_F^2 \\ \text{subject to} \quad & b_{i,j} \geq 0 \end{aligned}$$

3. Fix $\mathbf{B}$ to the value obtained at the previous step.
4. Solve the Non-negative Least Squares problem (NNLS):

$$\begin{aligned} \min_{\mathbf{A}} \quad & ||\mathbf{M} - \mathbf{A}\mathbf{B}^T||_F^2 \\ \text{subject to} \quad & a_{i,j} \geq 0 \end{aligned}$$

5. Repeat from step 2 until convergence.

4.1.3 *Co-clustering using Sparse Matrix Regression*

Papalexakis et al. [43, 44], introduced a novel approach to co-clustering, which benefits from modeling the problem of matrix decomposition into a combination of the aforementioned lasso regression and NNMF.

Mathematically, the simplest form of SMR stated as Lasso regression problem, is as follows:

$$\min_{\mathbf{B}} \quad ||\mathbf{M} - \mathbf{A}\mathbf{B}^T||_F^2 + \lambda||\mathbf{B}||_1$$

The above equation is similar to the one introduced by Lasso regression, if we consider each column of $\mathbf{M}$ as an output vector, matrix $\mathbf{A}$ as the input matrix and each column of $\mathbf{B}$ as the β_j coefficients we need to estimate.

By reforming this regression formula and using matrix transposition, in order to express it with respect to matrix $\mathbf{A}$, we have:

$$\min_{\mathbf{A}} \quad ||\mathbf{M}^T - \mathbf{B}\mathbf{A}^T||_F^2 + \lambda||\mathbf{A}||_1$$

Both of the above equations assume a fixed value for $\mathbf{A}$ and $\mathbf{B}$ respectively. By combining them into one formula, and concurrently imposing sparsity, as well as non-negativity for both factors, we have:

$$\begin{aligned} \min_{\mathbf{A},\mathbf{B}} \quad & ||\mathbf{M} - \mathbf{A}\mathbf{B}^T||_F^2 + \lambda||\mathbf{A}||_1 + \lambda||\mathbf{B}||_1 \\ \text{subject to} \quad & a_{i,j} \geq 0 \; \beta_{i,j} \geq 0 \end{aligned}$$

Due to the non-linearity of the above problem, it cannot be solved in this direct form. One way to solve this is by solving two linear Lasso problems in an alternating fashion(in a similar way to the aforementioned ALS). The complexity per iteration is $\mathcal{O}(IJ\hat{k}^2)$ for the Alternating Sparse Regression algorithm, the pseudocode of which follows:

Algorithm 3 Alternating SMR with NN constraints

Inputs: $\mathbf{M}$(of size I,J), k, λ **Outputs:** $\mathbf{A}, \mathbf{B}$

1: $\mathbf{A} = rand(I, k)$
2: $\mathbf{B} = rand(J, k)$
3: **repeat**
4: $\quad \mathbf{B} = \min_{\mathbf{B}\geq 0} ||\mathbf{M} - \mathbf{A}\mathbf{B}^T||_F^2 + \lambda||\mathbf{B}||_1$
5: $\quad \mathbf{A} = \min_{\mathbf{A}\geq 0} ||\mathbf{M}^T - \mathbf{B}\mathbf{A}^T||_F^2 + \lambda||\mathbf{A}||_1$
6: **until** convergence

4.1.3.1 *Method properties - interpretation*

Having established the above optimization target, besides the attractive characteristics of NNMF concerning model interpretability of the factors to which the initial matrix is decomposed, *sparsity*[2] of these resulting factors is being exploited, as it further improves interpretability, by "pushing" small values to exactly zero. Thus, the most powerful correlations between objects and features are unveiled - as sparsity "selects" these correlations, uniqueness of solution is improved and noise reduction is performed.

The above concept is examined in [45, 44, 46] and proven experimentally to co-cluster the input matrix in a "soft" and "lossy" way. This means that we do not seek for a "checkerboard" partitioning of input, where all objects and features will belong to a co-cluster at the result. We prefer rather a model which ignores indifferent, noisy data and may create overlapping co-clusters - where some objects/features may belong to more than one output concepts, which is desirable for several applications.

The main intuition behind the resulting factors $\mathbf{A}$ and $\mathbf{B}$ is that each of them corresponds to each of the contracting dimensions forming the input matrix and the non-zero values existing in the kth column of the resulting matrices, indicate which rows/columns of $\mathbf{A}/\mathbf{B}$ respectively, belong to each of the k co-clusters. Apart from that, each of these non-zero values, indicate a measure of "belongingness" to the co-cluster they are members of. Thus, for each of the resulting factors, a division by the maximum value of each column is a useful post-processing step, as to further highlight the most important elements belonging to each co-cluster.

This method's properties are depicted in an example from the Chemometrics domain [3] in Figure 15, where species sharing the same char-

2 A matrix is called sparse if the zero elements largely outnumber the non-zero ones.
3 Chemometrics is the science of extracting information from chemical systems by data-driven means.

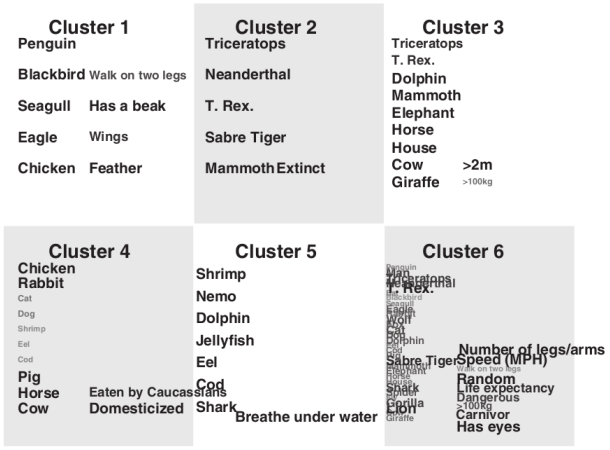

Figure 15: Application of SMR co-clustering in the Chemometrics context

acteristics are being grouped together and the font-size indicates the aforementioned "belongingness" to each co-cluster [45].

4.1.3.2 *Applicability of this method for large-scale data*

In this point, it would be useful to enumerate the reasons why this method is particularly desirable for large-scale data analysis, by summing up its advantages from the perspective of "big data", in order to justify the selection of this approach against others for parallelization:

1. The "soft" co-clustering this method forces to input data is a fundamental reason for its choice. A reasonable argument to this direction would be that as data availability grows, so does level of possible noise in the data. As a result, not all parts of input data will be important for concept discovery - these are the parts that we expect this method to extract.

2. The imposition of latent sparsity followed by this approach aids in the incremental extraction of co-clusters, as the number of requested co-clusters is increased - establishing this method as appropriate for large datasets.

3. The extension of this model to multiple dimensions (tensor co-clustering [44]) does not make any assumptions about the structure of contracting spaces (e.g.: star structure of domains). Thus, the tensor extension of this approach through Map-Reduce would be a promising research direction.

4.2 ALTERNATING DIRECTION OF MULTIPLIERS METHOD

In this section, we briefly analyze the background behind general optimization methods, as well as ADMM.

4.2.1 *Optimization preliminaries*

Many methods for global optimization require a cheaply computable lower bound on the optimal value of the nonconvex problem, instead of directly solving the original one. This is often possible by converting the original form of the optimization problem (i.e. the primal problem) to a dual form.

To illustrate this procedure, consider an optimization problem in the standard form [50]:

$$\begin{aligned} \min \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0 \; i = 1, \ldots, m. \\ & h_i(x) = 0 \; i = 1, \ldots, p. \end{aligned} \tag{2}$$

with variable $x \in \mathbb{R}^n$ and where the functions from $\mathbb{R}^n \to \mathbb{R}$, $f_0, f_1, \cdots, f_m$ and $h_1, \cdots, h_m$ are the objective, inequality constraints and equality constraints respectively. For now, we do not assume the problem 2 is convex [4].

The basic idea in *Lagrangian duality* is to augment the objective function of 2, with a weighted sum of the constraint functions. Thus, the *Lagrangian* L, associated with the problem 2, is defined as:

$$L(x, \lambda, v) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} v_i h_i(x) \tag{3}$$

We refer to λ_i as the Lagrange multiplier associated with the ith inequality constraint; similarly we refer to v_i as the Lagrange multiplier associated with the ith equality constraint $h_i(x) = 0$. The vectors λ and v are called *dual variables* or Lagrange multiplier vectors associated with problem 2. They can be thought of "costs" associated with violating different constraints.

We define the *Lagrange dual function* (or just dual function) as the *minimum* value of the Lagrangian over x:

$$g(\lambda, v) = \inf_{x \in \mathbb{D}} \quad (f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} v_i h_i(x)) \tag{4}$$

It is shown (proof omitted for brevity - can be found in Boyd et al. optimization book [51]) that for any $\lambda \geq 0$ and any v, the *dual function yields lower bounds on the optimal value* $p^\star$ of problem 2:

$$g(\lambda, v) \leq p^\star$$

Thus, we have a lower bound that depends on some parameters λ, v - a natural question is: What is the *best* lower bound that can be obtained

4 A function is convex iff the region above its graph is a convex set,i.e.: if for every pair of points within the set, every point on the straight line segment that joins them is also within the set.

from the Lagrange dual function? This leads to the following optimization problem, which is called the respective *Lagrange dual problem*:

$$\begin{aligned} \max \quad & g(\lambda, v) \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{5}$$

In this context, the original problem 2, is sometimes called the *primal problem*. Besides, each pair (λ, v) with $\lambda \geq 0$ and $g(\lambda, v) \geq -\infty$, is dual feasible (i.e.: to imply feasibility of this solution for the dual problem). Finally, $(\lambda^\star, v^\star)$ is dual optimal or optimal Lagrange multipliers if they are optimal for problem 5.

The Lagrange dual problem 5 is a convex optimization problem, since the objective to be maximized is concave and the constraint is convex. This is the case whether or not the primal problem 2 is convex. The above is a useful property, as generally non-convex optimization problems are harder to solve than convex ones.

4.2.2 *ADMM precursors and main concept*

The Alternating direction method of multipliers, is a simple but powerful algorithm that is well suited to distributed convex optimization and in particular to problems arising in applied statistics and machine learning. It takes the form of a *decomposition-coordination* procedure, in which the solutions to small local subproblems are coordinated to find a solution to a large global problem. ADMM can be viewed as an attempt to blend the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization, two approaches that we briefly review below.

As Boyd et al. [49] remark:

> It is worth emphasizing that the algorithm itself is not new. It was first introduced in the mid-1970s by Gabay, Mercier, Glowinski, and Marrocco, though similar ideas emerged as early as the mid-1950s. The fact that ADMM was developed so far in advance of the ready availability of large-scale distributed computing systems and massive optimization problems *may account for why it is not as widely known today as we believe it should be.*

4.2.2.1 *Dual Decomposition*

Consider the following equality-constrained convex optimization problem [52]:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & Ax = b \end{aligned} \tag{6}$$

with variable $x \in \mathbb{R}^n$, where $A \in \mathbb{R}^{m \times n}$ and $f : \mathbb{R}^n \to \mathbb{R}$ is convex. The Lagrangian for problem 6 is

$$L(x, y) = f(x) + y^T(Ax - b)$$

where $y \in \mathbb{R}^m$ is the dual variable. The dual function is

$$g(y) = \inf_x L(x, y)$$

The dual problem is expressed as the maximization of $g(y)$, as mentioned above. Furthermore, assuming strong duality holds, [5] the optimal values for the primal and dual problems are the same, so we can recover a primal optimal point $x^\star$ from a dual optimal point $y^\star$ as

$$x^\star := \operatorname*{argmin}_x L(x, y^\star)$$

The following *dual ascent* algorithm solves the dual problem using gradient ascent. Assuming that g is differentiable, the gradient $\nabla g(y)$ can be evaluated as follows. We first find $x^+ = \operatorname*{argmin}_x L(x, y)$; then we have $\nabla g(y) = Ax^+ - b$, which is the residual for the equality constraint. The dual ascent method consists of iterating the updates:

$$x^{k+1} := \operatorname*{argmin}_x L(x, y^k) \tag{7}$$

$$y^{k+1} := y^k + a^k(Ax^{k+1} - b) \tag{8}$$

where $a^k \geq 0$ is a step size, and the superscript is the iteration counter. This iteration continues until we reach a local optimum - i.e. when the residual $Ax - b$ equals zero. This algorithm is called dual ascent, since, with appropriate choice of a^k, the dual function increases in each step, i.e.: $g(y^{k+1}) > g(y^k)$.

If a^k is chosen appropriately and several other assumptions hold, then x^k converges to an optimal point and y^k converges to an optimal dual point. However, these assumptions do not hold in many applications, so *dual ascent often cannot be used.*

Nevertheless, it has a major benefit that we could take advantage of. *It can lead to a decentralized algorithm in some cases.* Suppose, the objective f is separable, meaning that:

$$f(x) = \textstyle\sum_{i=1}^N f_i(x_i)$$

Partitioning the matrix A conformably as $A = [A_1 \cdots A_N]$, so $Ax = \sum_{i=1}^N A_i x_i$, the Lagrangian can be written as

$$L(x, y) = \textstyle\sum_{i=1}^N L_i(x_i, y) = \sum_{i=1}^N (f_i(x_i) + y^T A_i x_i - (\frac{1}{N} y^T b),$$

which is also separable in x. This means that the minimization step in 7, splits into N separate problems that can be solved in parallel. In other words:

5 the best bound that can be obtained by the Lagrange dual function is tight, so that the optimal value obtained by the dual problem is the optimal value for the initial one

$$\begin{aligned} x_i^{k+1} &:= \underset{x_i}{\operatorname{argmin}}\, L_i(x_i, y^k) \\ y^{k+1} &:= y^k + a^k(\sum_{i=1}^{N} A_i x_i^{k+1} - b) \end{aligned}$$

If N processors were to iterate through the above formulas, a "scatter-gather" process would take place. We gather the x_i solutions to sum and evaluate the residual, while scattering the vector y to all i processors in order to update each x_i in parallel.

The aforementioned algorithm is an alternative of dual ascent, called *dual decomposition.*

4.2.2.2 *Augmented Lagrangian methods*

In the previous section, we accentuated the fact that dual ascent (as well as its decomposition), needs several assumptions (e.g., strict convexity or finiteness of f) to reach convergence, but these assumptions do not hold in many applications. This is the reason why Augmented Lagrangian (AL) methods were developed: in order to *robustify* dual ascent.

This process of *augmenting* the Lagrangian typically consists of adding a quadratic term to the Lagrangian multiplied by a positive parameter. Mathematically,

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + (\rho/2)||Ax - b||_2^2$$

is the AL for Problem 6, where $\rho > 0$ is called the penalty parameter. L_ρ is the Lagrangian of the problem:

$$\begin{aligned} \text{minimize} \quad & f(x) + (\rho/2)||Ax - b||_2^2 \\ \text{subject to} \quad & Ax = b \end{aligned}$$

which is clearly equivalent to the original problem 6, since for any feasible x the term added to the objective zero. By applying the dual-ascent method to the above problem we end up with the following algorithm:

$$\begin{aligned} x^{k+1} &:= \underset{x}{\operatorname{argmin}}\, L_\rho(x, y^k) \\ y^{k+1} &:= y^k + \rho(Ax^{k+1} - b) \end{aligned}$$

which is called the method of multipliers. The only difference from the above dual ascent process, is (besides the use of Augmented Lagrangian), the selection of step size ρ for the gradient ascent step. This is justified as follows:

The optimality conditions for primal and dual feasibility, are:

$$Ax^\star - b = 0 \text{ and } \nabla f(x^\star) + A^T y = 0$$

i.e.: the achievement of the initial constraint, as well as the minimization of x over L_ρ which happens when the derivative of L_ρ with respect to x is zero.

What it means for x^{k+1} to minimize $L_\rho(x, y)$ is that the gradient of L_ρ with respect to x should be zero, so:

$$\begin{aligned} 0 &= \nabla_x L_\rho(x^{k+1}, y^k) \\ &= \nabla_x f(x^{k+1}) + A^T(y^k + \rho(Ax^{k+1} - b)) \\ &= \nabla_x f(x^{k+1}) + A^T y^{k+1} \end{aligned}$$

What we see is that by using ρ as the step size in the dual update, we get *dual feasibility*. This is the reason why ρ is chosen as step size in this algorithm.

This greatly improves convergence properties of the method of multipliers over dual ascent - it converges under far more relaxed conditions (almost always) - comes at a cost. When f is separable, the Augmented Lagrangian L_ρ is not separable, so the x-minimization step cannot be carried out separately in parallel for each x_i. This means that the basic method of multipliers cannot be used for decomposition.

4.2.2.3 *Alternating Direction Method of Multipliers*

ADMM is a powerful algorithm for solving structured convex optimization problems and is intented to *blend* the *decomposability of dual ascent* with the *superior convergence properties of the method of multipliers*. This is a very general justification of why it is particularly applicable to large-scale decision problems. It solves problems of the form:

$$\begin{aligned} \text{minimize} \quad & f(x) + g(z) \\ \text{subject to} \quad & Ax + Bz = c \end{aligned} \tag{9}$$

with variables $x \in \mathbb{R}^m$ and $z \in \mathbb{R}^m$, where $A \in \mathbb{R}^{p \times n}, B \in \mathbb{R}^{p \times m}$ and $c \in \mathbb{R}^p$.

We will assume that f and g are convex. The only difference from the general problem 6 is that the variable, called x there, has been split into two parts, called x and z here, with the objective function separable across this splitting. The optimal value of problem 9 will be denoted by $p^\star = \inf\{f(x) + g(z) | Ax + Bz = c\}$.

As in the method of multipliers, we form the augmented Lagrangian

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz + c) + \rho/2||Ax + Bz - c||_2^2$$

and solve by minimizing separately over x and z and perform the dual update step for our dual variable y:

$$\begin{aligned} x^{k+1} &= \underset{x}{\operatorname{argmin}}\ L_\rho(x, z^k, y^k) \\ z^{k+1} &= \underset{z}{\operatorname{argmin}}\ L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \end{aligned}$$

where $\rho > 0$.

The separate minimizations with respect to x and z, is exactly the feature permitting the exploitation of parallelism, when any of f,g is separable (thus combining the method of multipliers with dual ascent).

The optimality conditions (for differentiable case) are again, primal (which is expressed as $Ax + Bz - c = 0$), as well as dual feasibility (expressed as $\nabla f(x) + A^T y = 0$ and $\nabla g(z) + B^T y = 0$).

Since z^{k+1} minimizes $L_\rho(x^{k+1}, z, y^k)$ we have:

$$\begin{aligned} 0 &= \nabla g(z^{k+1}) + B^T y^k + \rho B^T (Ax^{k+1} + Bz^{k+1} - c) \\ &= \nabla g(z^{k+1}) + B^T y^{k+1} \end{aligned}$$

As a result, with ADMM's dual variable update, $(x^{k+1}, z^{k+1}, y^{k+1})$ satisfies second dual feasibility condition. Primal and first dual feasibility are achieved as $k \to \infty$ and the algorithm's stopping criteria are applied upon them.

In general, the conditions which are always true about this method are:

- iterates approach feasibility: $Ax^k + Bz^k - c \to 0$
- objective approaches optimal value: $f(x^k) + g(z^k) \to p^\star$

An important fact about the statements above, is that all other conditions not asserted by this method (e.g. that x^k converges), are irrelevant for the method's success and our stopping criteria will be based on the above two statements.

For more on convergence/optimality theory behind ADMM, we refer to [49].

As a final point on non-distributed ADMM form, it is useful to provide the standard lasso regression formulation in ADMM. As we discussed in the respective section, lasso regression's objective is typically:

$$min \tfrac{1}{2}||Ax - b||_2^2 + \lambda ||x||_1$$

which is expressed in ADMM ([49]) as follows:

$$\begin{aligned} x^{k+1} &= (\mathbf{A}^T\mathbf{A} + \rho\mathbf{I})^{-1}(\mathbf{A}^T b + \rho(z^k - u^k)) \\ z^{k+1} &= \mathbf{S}_{\lambda/\rho}(x^{k+1} + u^k) \\ y^{k+1} &= y^k + x^{k+1} - z^{k+1} \end{aligned} \tag{10}$$

where $\mathbf{S}$ denotes the Soft-Thresholding operator:

$$\mathbf{S}_k(a) = \begin{cases} a - k & a > k \\ 0 & |a| \le k \\ a + k & a < -k \end{cases} \tag{11}$$

and $u^k = (1/\rho)y^k$.

This problem is depicted in Figure 16.

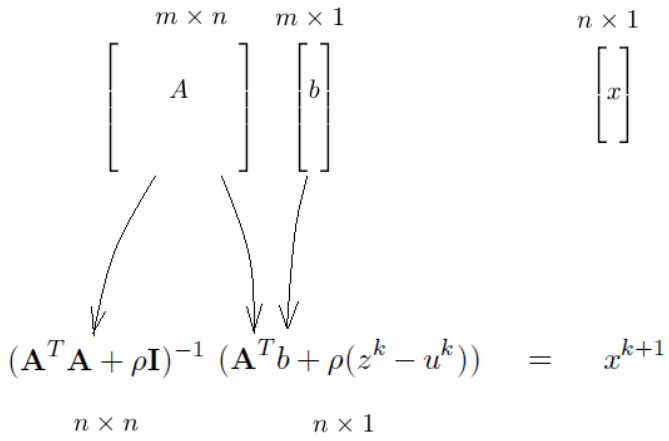

Figure 16: Solving x-minimization step of standard Lasso problem via ADMM

4.2.2.4 *Consensus*

Consider a problem of the form:

$$\text{minimize} \quad \sum_{i=1}^{N} f_i(x) \tag{12}$$

where f_i could be the loss function for the ith block of training data.

The great advantage of ADMM we are exploiting, is its expression for solving distributed optimization problems. The respective formulation is called as "consensus" [49] and the reasons for that are described below. In ADMM through consensus, Problem 12 takes the following form:

$$\begin{aligned} &\text{minimize} \ \sum_{i=1}^{N} f_i(x_i) \\ &\text{subject to} \ x_i - z = 0 \end{aligned}$$

Instead of a global variable x, here we allow each block of data to have its "local opinion" x_i, but all of these have to agree finally to a global variable z.

This holds when the cost function f is decomposable. It is called global consensus, because all the "local" x_i for i=1,..., N, have to be equal to each other - i.e.: to "consent" to the slack variable z.

The resulting general ADMM iterations are the following [49]:

$$\begin{aligned} x_i^{k+1} &= \min_{x_i}(f_i(x_i) + y_i^{kT}(x_i - z^k) + (\rho/2)||x_i - z^k||_2^2) \\ z^{k+1} &= \tfrac{1}{N}\sum_{i=1}^{N}(x_i^{k+1} + (1/\rho)y_i^k) \\ y_i^{k+1} &= y_i^k + \rho(x_i^{k+1} - z^{k+1}) \end{aligned}$$

The first and last steps are carried independently for each i=1,...,N. In the literature, the processing element that handles the global variable z is sometimes called the central collector or the fusion center. This global variable, is obviously just the average of the x_i and y_i/ρ variables.

Getting closer to our problem, the global variable consensus *with regularization* is being expressed as:

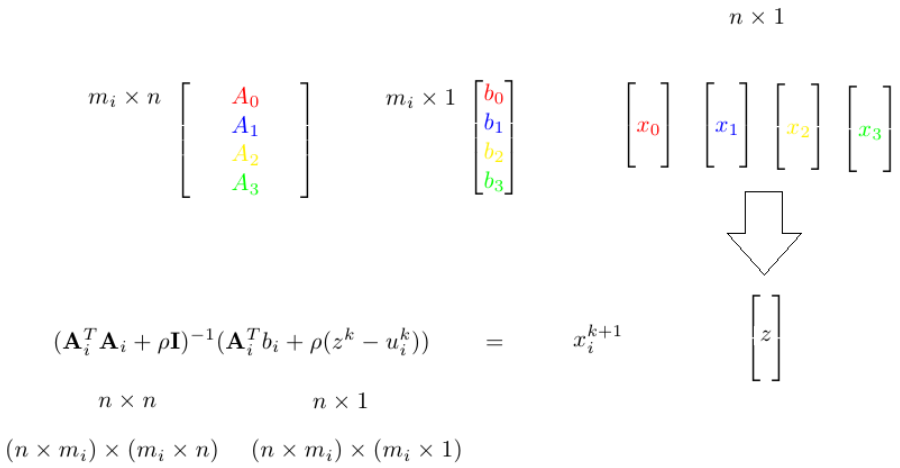

Figure 17: Solving x_i-minimization step of consensus Lasso problem via ADMM. Partitions sharing the same color participate in the solution of the respective x_i vector. We consider 4 partitions as an example.

$$\begin{aligned} &\min_{x_1,\ldots,x_n,z} \textstyle\sum_{i=1}^{N} f_i(x_i) + g(z) \\ &\text{subject to: } x_i - z = 0, \quad i = 1, \ldots, N \end{aligned}$$

Standard lasso regression represents a sub-case of this expression and the resulting consensus ADMM algorithm for the lasso, extending the aforementioned case of non-distributed lasso in 10, is ([49], at section 8.2.1):

$$\begin{aligned} x_i^{k+1} &= (\mathbf{A}_i^T\mathbf{A}_i + \rho\mathbf{I})^{-1}(\mathbf{A}_i^T b_i + \rho(z^k - u_i^k)) \\ z^{k+1} &= \mathbf{S}_{\lambda/\rho N}(\bar{x}^{k+1} + \bar{u}^k) \\ y_i^{k+1} &= y_i^k + x_i^{k+1} - z^{k+1} \end{aligned} \tag{13}$$

This formulation is depicted in Figure 17, where we attempt to emphasize the basic intuition of the consensus method. This is that each subsystem calculates a solution of the *whole* problem, based on the knowledge of a *portion* of the whole data. Consequently, these local solutions x_i participate in the extraction of global solution z. In this figure, $\mathbf{A}_i$, b_i have $(m_i \times n)$ and m_i as respective sizes.

5 IMPLEMENTATION ANALYSIS

Our initial goal is to tackle each of the Lasso regression problems of Sparse Matrix Regression through the Alternating Direction of Multipliers Method (ADMM), in order to be able to handle large-scale datasets through Map-Reduce, by exploiting all the aforementioned useful properties of the SMR algorithm.

The reason for this choice lies in the fact that ADMM is *distributed* in its *nature* and can intuitively be *expressed* as a *Map-Reduce procedure*. The problem form we end up with, can be characterized as a "scatter-gather" process, as we discuss below. Note that our choice of ADMM is in contrast to the already proposed Coordinate Descent method in [43], which is neither intuitively expressed in Map-Reduce, nor targeted specifically for distributed optimization.

5.1 EXPRESSION OF SMR CO-CLUSTERING THROUGH ADMM

At this point we provide the equations which form the basis of our Map-Reduce implementation.

5.1.1 *General Form*

As mentioned above, our target is to tackle each of the Lasso regression problems of SMR co-clustering via ADMM. We chose to approach the problem's formulation from first principles. Another choice would be to vectorize it and employ the standard lasso solution provided above, but the following formulation through matrices, ensures both the avoidance of the respective transformations, involving computation of Kronecker products, and a more intuitive representation at storage. In general, the problem has the following form:

$$\min_{\mathbf{B}\geq 0} ||\mathbf{X} - \mathbf{A}\mathbf{B}^T||_F^2 + \lambda||\mathbf{B}||_1$$

We will analyze this case which applies also for the symmetric case of transposing $\mathbf{X}$ and minimizing over $\mathbf{A}$.

At first, we transform the problem to:

$$min_{\mathbf{B}} ||\mathbf{X} - \mathbf{A}\mathbf{B}||_F^2 + \lambda||\mathbf{B}||_\infty$$

Importing $\mathbf{C}$ as slack variable:

$$\begin{aligned} \text{minimize} \quad & ||\mathbf{X} - \mathbf{A}\mathbf{B}||_F^2 + \lambda||\mathbf{C}||_\infty \\ \text{subject to} \quad & \mathbf{B} = \mathbf{C} \end{aligned}$$

Forming the Lagrangian of the above problem:

$$L(\mathbf{B},\mathbf{C},\mathbf{\Lambda}) = ||\mathbf{X}-\mathbf{AB}||_F^2 + \lambda||\mathbf{C}||_\infty + vec(\mathbf{\Lambda}^T) * vec(\mathbf{B}-\mathbf{C})$$

Forming the respective Augmented Lagrangian $L_\rho(\mathbf{B},\mathbf{C},\mathbf{\Lambda})$:

$$||\mathbf{X}-\mathbf{AB}||_F^2 + \lambda||\mathbf{C}||_\infty + vec(\mathbf{\Lambda}^T)vec(\mathbf{B}-\mathbf{C}) + \tfrac{\rho}{2}||\mathbf{B}-\mathbf{C}||_F^2$$

1. Zeroing the derivative of $L_\rho(\mathbf{B},\mathbf{C},\mathbf{\Lambda})$ with respect to $\mathbf{B}$:

$$\begin{aligned} -2\mathbf{A}^T(\mathbf{X}-\mathbf{AB}) + \mathbf{\Lambda} + \rho(\mathbf{B}-\mathbf{C}) = 0 \Rightarrow \\ \mathbf{B} = (2\mathbf{A}^T\mathbf{A} + \rho\mathbf{I})^{-1}(2\mathbf{A}^T\mathbf{X} + \rho\mathbf{C} - \mathbf{\Lambda}) \end{aligned}$$

2. Zeroing the derivative of $L_\rho(\mathbf{B},\mathbf{C},\mathbf{\Lambda})$ with respect to $\mathbf{C}$, through element-wise cases (following the paradigm of [57]):

   - If $C > 0$:

$$\begin{aligned} \lambda\mathbf{I} - \mathbf{\Lambda} + \rho(\mathbf{C}-\mathbf{B}) = 0 \Rightarrow \\ \mathbf{C} = \mathbf{B} + \frac{1}{\rho}(\mathbf{\Lambda} - \lambda\mathbf{I}) \\ \text{This happens when: } \mathbf{B} + \frac{\mathbf{\Lambda}}{\rho} > \frac{\lambda\mathbf{I}}{\rho} \end{aligned} \tag{14}$$

   - If $C < 0$:

$$\begin{aligned} -\lambda\mathbf{I} - \mathbf{\Lambda} + \rho(\mathbf{C}-\mathbf{B}) = 0 \Rightarrow \\ \mathbf{C} = \mathbf{B} + \frac{1}{\rho}(\mathbf{\Lambda} + \lambda\mathbf{I}) \Rightarrow \\ \text{This happens when: } \mathbf{B} + \frac{\mathbf{\Lambda}}{\rho} < -\frac{\lambda\mathbf{I}}{\rho} \end{aligned} \tag{15}$$

   - If $C = 0$, from 14, 15 covering the other 2 cases, we get that:

$$\begin{aligned} -\frac{\lambda\mathbf{I}}{\rho} \leq \mathbf{B} + \frac{\mathbf{\Lambda}}{\rho} \leq \frac{\lambda\mathbf{I}}{\rho} \Rightarrow \\ |\mathbf{B} + \frac{\mathbf{\Lambda}}{\rho}| \leq \frac{\lambda\mathbf{I}}{\rho} \end{aligned} \tag{16}$$

   From 14, 15 and 16, we get that:

$$\mathbf{C} = \mathbf{S}_{\frac{\lambda}{\rho}}(\mathbf{B} + \tfrac{\mathbf{\Lambda}}{\rho}), \text{ element-wise}$$

3. For primal feasibility to hold we have: $B - C = 0$, whereas for dual feasibility to hold we have(as in [49], at section 2.3):

$$\frac{d||\mathbf{X}-\mathbf{AB}||_F^2}{d\mathbf{B}} + \mathbf{\Lambda}^n + \rho(\mathbf{B}-\mathbf{C}) = 0 \tag{17}$$

$$\frac{d||\mathbf{C}||_\infty}{d\mathbf{C}} - \mathbf{\Lambda}^n - \rho(\mathbf{B}-\mathbf{C}) = 0 \tag{18}$$

By definition (which was used before to find solutions of $\mathbf{B}$ & $\mathbf{C}$ in closed form), we have:

$$\begin{aligned} 0 &= \nabla_{\mathbf{B}}\mathbf{L}(\mathbf{B},\mathbf{C},\mathbf{\Lambda}) \\ &= \rho(\mathbf{B}-\mathbf{C}) + \mathbf{\Lambda}^n + \frac{d||\mathbf{X}-\mathbf{AB}||_F^2}{d\mathbf{B}} \end{aligned} \tag{19}$$

Furthermore:

$$\begin{aligned} 0 &= \nabla_{\mathbf{C}}\mathbf{L}(\mathbf{B},\mathbf{C},\mathbf{\Lambda}) \\ &= -\rho(\mathbf{B}-\mathbf{C}) - \mathbf{\Lambda}^n + \frac{d||\mathbf{C}||_\infty}{d\mathbf{C}} \end{aligned} \tag{20}$$

Thus, we notice that in order for the dual feasibility expressed in (17), (18) to hold, it is valid to choose ρ as our step size at (19), (20) - to perform our dual update. As a result, it holds that:

$$\mathbf{\Lambda}^{n+1} = \mathbf{\Lambda}^n + \rho(\mathbf{B}-\mathbf{C})$$

It is tested against CVX optimization tool that in order to impose non-negative constraints on the above problem, our slack variable needs to be transformed to: $\mathbf{C} = \max(0, \mathbf{S}_{\frac{\lambda}{\rho}}(\mathbf{B} + \frac{\mathbf{\Lambda}}{\rho}))$.

To sum up, our ADMM iterations consist of the following:

$$\begin{aligned} \mathbf{B} &= (2\mathbf{A}^T\mathbf{A} + \rho\mathbf{I})^{-1}(2\mathbf{A}^T\mathbf{X} + \rho\mathbf{C} - \mathbf{\Lambda}) \\ \mathbf{C} &= \max(0, \mathbf{S}_{\frac{\lambda}{\rho}}(\mathbf{B} + \frac{\mathbf{\Lambda}}{\rho})) \\ \mathbf{\Lambda} &= \mathbf{\Lambda} + \rho(\mathbf{B}-\mathbf{C}) \end{aligned} \tag{21}$$

5.1.2 *First consensus approach*

We have discussed above the way the standard lasso ADMM form in 10, turns into its consensus counterpart in 13. If we follow this paradigm to transform our lasso ADMM form (consisting of matrices) in 21, to a consensus form, we would have:

$$\begin{aligned} \mathbf{B}_i^{k+1} &= (2\mathbf{A}_i^T\mathbf{A}_i + \rho\mathbf{I})^{-1}(2\mathbf{A}_i^T\mathbf{X}_i + \rho\mathbf{C}^k - \mathbf{\Lambda}_i^k) \\ \mathbf{C}^{k+1} &= \max(0, \mathbf{S}_{\frac{\lambda}{\rho N}}(\bar{\mathbf{B}} + \frac{\bar{\mathbf{\Lambda}}}{\rho})) \\ \mathbf{\Lambda}_i^{k+1} &= \mathbf{\Lambda}_i^k + \rho(\mathbf{B}_i^{k+1} - \mathbf{C}^{k+1}) \end{aligned} \tag{22}$$

In Figure 18, this process is depicted with 4 partitions.

As it happens with the simplest form of consensus we discussed about, each subsystem calculates a local opinion of the *whole* problem, based on the knowledge of the *portion* of data it processes and these partial solutions "agree" to a global solution through successive iterations.

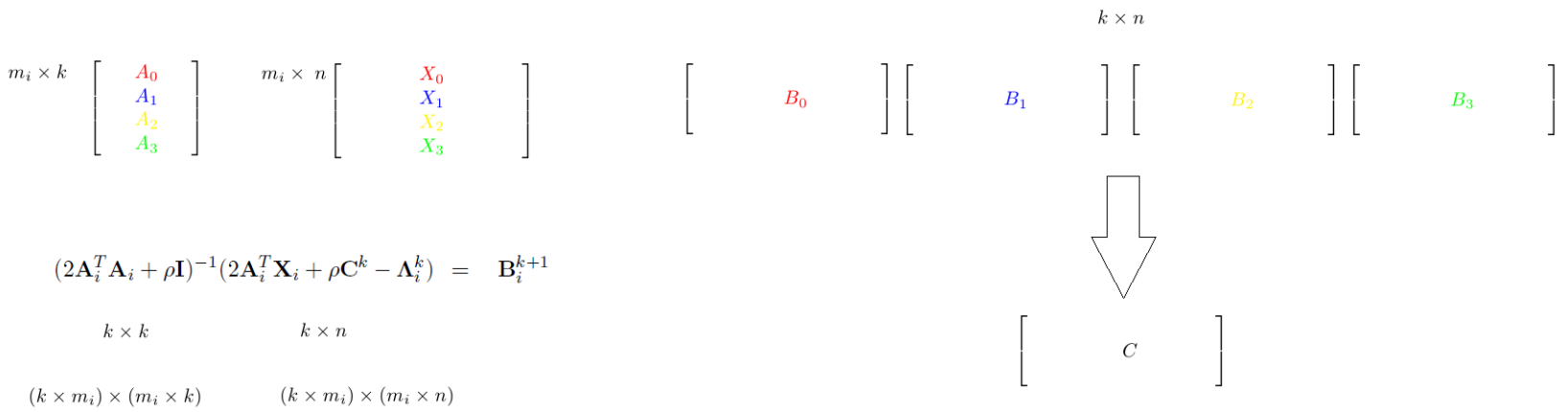

Figure 18: Example of first approach of extending the Lasso consensus ADMM to our matrix representation

In this case, each processor is responsible to work with its own inputs $\mathbf{X}_i$ and $\mathbf{A}_i$, while keeping its own solutions ($\mathbf{B}_i$ and $\mathbf{\Lambda}_i$), and sharing the global solution for $\mathbf{C}$. Assuming sizes of $\mathbf{X}, \mathbf{A}$, are $(m \times n)$, $(m \times k)$ respectively (where k is the number of desired co-clusters), $\mathbf{B}_i, \mathbf{\Lambda}_i, \mathbf{C}$ are all of size $(k \times n)$. As a result, with this decomposition, we require that each node calculates a local solution B_i with the upper bound on total size being $(k \times n)$.

5.1.3 *Improved consensus approach*

We observe that the computation of the amount of $2\mathbf{A}_i^T\mathbf{X}_i$ permits further decomposability, as we can perform this matrix multiplication in blocks. This enables us to split our input dataset with respect to columns as well.

With this change, the algorithm in 22 for the case of minimizing over $\mathbf{B}$, transforms to:

$$\begin{aligned} \mathbf{B}_i^{k+1} &= (2\mathbf{A}_j^T\mathbf{A}_j + \rho\mathbf{I})^{-1}(2\mathbf{A}_j^T\mathbf{X}_i + \rho\mathbf{C}_q^k - \mathbf{\Lambda}_i^k) \qquad (23) \\ \mathbf{C}_q^{k+1} &= \max(0, \mathbf{S}_{\frac{\lambda}{\rho r}}(\bar{\mathbf{B}} + \frac{\bar{\mathbf{\Lambda}}}{\rho})) \\ \mathbf{\Lambda}_i^{k+1} &= \mathbf{\Lambda}_i^k + \rho(\mathbf{B}_i^{k+1} - \mathbf{C}_q^{k+1}) \end{aligned}$$

In the above solution, matrices $\mathbf{X}$ and $\mathbf{A}$, are partitioned as follows:

$$\begin{bmatrix} X_0 & \cdots X_{c-1} \\ \vdots & \ddots & \vdots \\ X_{rc-c} \cdots & X_{rc-1} \end{bmatrix}, \begin{bmatrix} A_0 \\ \vdots \\ A_{r-1} \end{bmatrix}$$

where r is the number of row groups and c the number of column groups to which our input matrix is partitioned, $i = 0 \ldots rc-1$, $j = \left\lfloor \frac{i}{c} \right\rfloor$ and $q = i \mod c$. This is depicted by an example with $r = 4, c = 2$ in Figure 19.

By employing this approach, we require that each node calculates a local solution B_i with an upper bound on total size of $(k \times n_i)$, where n_i is the number of elements each column split of our input matrix $\mathbf{X}$ contains.

As a final step to the above form, we transpose the equations in order to be feasible for the problem to be expressed in Map-Reduce environment with minimal assumptions - i.e. just by assuming that *vectors of size k* (the desired number of co-clusters) fit in main memory. This is the **"basic unit"** of our iterations, which is fundamental for this method to scale to large datasets, without having enormous memory requirements, as we demonstrate in the next section.

Please note the notation for $\mathbf{B}_i, \mathbf{C}_q, \mathbf{L}_i$ (they are considered to have size equal to $n_i \times k$). For aesthetic reasons, from now on, though transposing them, notation stays the same.

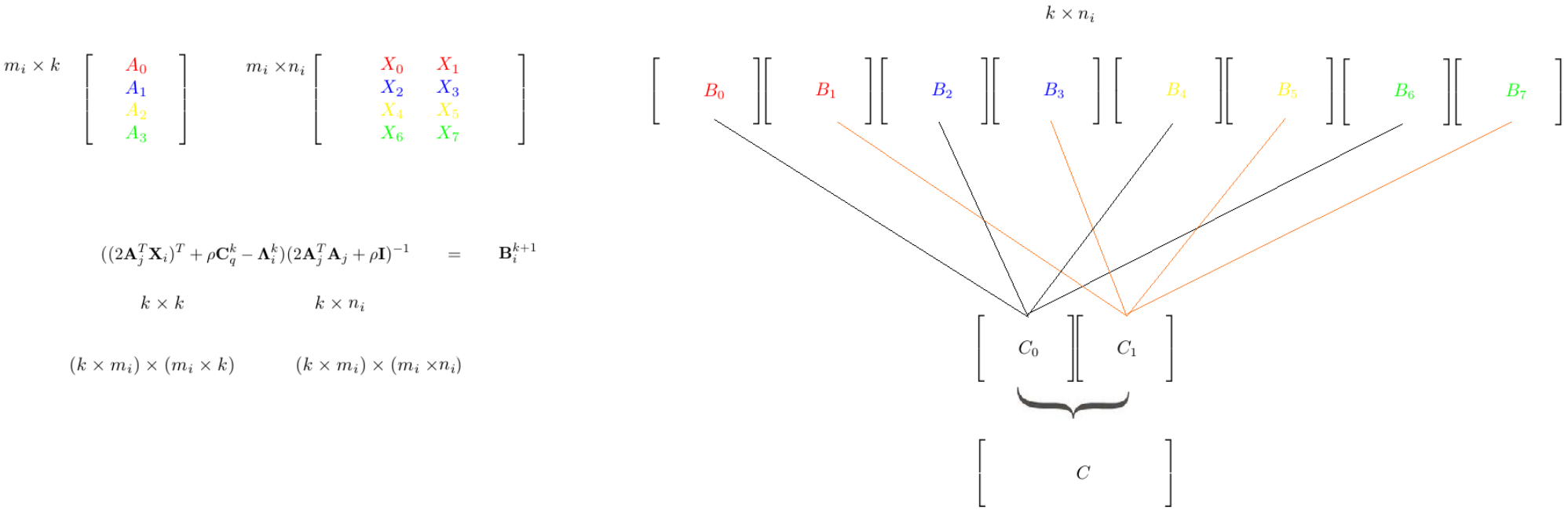

Figure 19: Example of improved approach of Lasso consensus ADMM through our matrix representation, with $r = 4, c = 2$

$$\begin{bmatrix} X_0 & \cdots X_{c-1} \\ \vdots & \ddots & \vdots \\ X_{rc-c} \cdots & X_{rc-1} \end{bmatrix}$$

$$\begin{bmatrix} A_0 \\ \vdots \\ A_{r-1} \end{bmatrix} \cdots \begin{bmatrix} A_{rc-r} \\ \vdots \\ A_{rc-1} \end{bmatrix} \qquad \begin{bmatrix} B_0 \\ \vdots \\ B_{c-1} \end{bmatrix} \cdots \begin{bmatrix} B_{rc-c} \\ \vdots \\ B_{rc-1} \end{bmatrix}$$

Figure 20: Structure of contracting matrices of our problem

$$\begin{aligned} \mathbf{B}_i^{k+1} &= ((2\mathbf{A}_j^T\mathbf{X}_i)^T + \rho\mathbf{C}_q^k - \mathbf{\Lambda}_i^k)(2\mathbf{A}_j^T\mathbf{A}_j + \rho\mathbf{I})^{-1} \qquad (24) \\ \mathbf{C}_q^{k+1} &= \max(0, \mathbf{S}_{\frac{\lambda}{\rho r}}(\bar{\mathbf{B}} + \frac{\bar{\mathbf{\Lambda}}}{\rho})) \\ \mathbf{\Lambda}_i^{k+1} &= \mathbf{\Lambda}_i^k + \rho(\mathbf{B}_i^{k+1} - \mathbf{C}_q^{k+1}) \end{aligned}$$

where again, r is the number of row groups and c the number of column groups to which our input matrix is partitioned, $i = 0 \dots rc-1$, $j = \left\lfloor \frac{i}{c} \right\rfloor$ and $q = i \mod c$. It is obvious that both the *amounts* of $(2\mathbf{A}_j^T\mathbf{X}_i)^T$, as well as $(2\mathbf{A}_j^T\mathbf{A}_j + \rho\mathbf{I})^{-1}$ (if parameter ρ remains stable through iterations) *could be precomputed and cached*, in order to be used at subsequent iterations.

5.2 MAP-REDUCE IMPLEMENTATION

In this section, we analyze the way of expressing in Map-Reduce (with HBase's support), the form we derive in Equation 24, by pointing out the reasons why this approach is scalable to large-scale datasets and explaining various implementation details and choices.

5.2.1 *Structure*

Let r be the number of groups into which we initially partition the rows of our input matrix X and our target matrix A and c the number of groups into which the columns of X as well as matrix B are partitioned. Using these, in Figure 20, we depict the final subsystems' structure of our matrices. Each of A and B subsystems contain their local solution as well their local Lagrange multiplier $\mathbf{\Lambda}_i$ and share the same global solution with all subsystems of the same level. In other words, $\mathbf{B}_0, \mathbf{B}_0 + c, \dots, \mathbf{B}_{rc-c}$ share the same global solution, as happens with subsystems $\mathbf{A}_0, \mathbf{A}_0 + r, \dots, \mathbf{A}_{rc-r}$. Besides, each "slice" of $\mathbf{A}$ and $\mathbf{B}$ matrices, as shown in Figure 20, contains a view of the whole matrix if assembled (e.g.: subsystems from $\mathbf{A}_0$ to $\mathbf{A}_{r-1}$).

The above discussion about partitioning and sharding the partial solutions of our problem does not intentionally contain Map-Reduce as

a part of it, in order to designate a problem at this point. The way Hadoop handles I/O, is opposed to what is needed in this case. This happens because its manner of reading and splitting data to subsets, in order to fuel map tasks, is being optimized to achieve high data locality and is usually "transparent" to the user and it is not created for cases in which we need to preserve state in the mappers across iterations.

What is necessary for us to tackle the problem, is a mechanism which accomplishes the following (the discussion addresses the minimization of $\mathbf{B}$ but the minimization of $\mathbf{A}$ is symmetric):

- Stores data referring to the same row in different times, that need to be processed together (e.g. store the calculation of $(2\mathbf{A}_j^T\mathbf{X}_i)^T$ and retrieve the pth row of this result together with the pth row of $\mathbf{B}_i$, $\mathbf{L}_i$ and $\mathbf{C}_q$ in order to calculate the new pth row of $\mathbf{B}_i$): in other words, we need *structure* to our tasks input and output
- Is timestamp-oriented in order to have the same row stored for the last two subsequent iterations and utilize this to check for convergence

Pleasingly enough, both the above properties are of the fundamental ones of HBase's storage system (details at the respective section of 2).

5.2.1.1 *Design choices*

More specifically, each of our matrices can be represented as an HBase's *table*. By pre-splitting it, we explicitly define *region* boundaries of our table. An important property of the frameworks in use is that if a Map-Reduce job reads data from an HBase's table, it reads data from exactly one region. As a result, the *region* is perfectly equivalent to the meaning we would like a *subsystem* to have - accessing always the same portion of data which is updated through successive iterations.[1]

The *rowkey* of choice for all tables, is a combination of subsystem's ID (depicted as subscripts in Figure 20) and the row ID - more accurately, a concatenation of them. This enables us to store information for the same subsystem together and efficiently access it, as HBase maintains data in lexicographic order by row key.

We chose to take advantage of Apache Mahout libraries ([53]) in order to represent, process and store data in a vectorized form. As mentioned above, the fundamental unit of our calculations is a vector of size equal to the number of desired co-clusters, so we store vectors of this size into our table's columns - each vector is serialized into one HBase's cell. All the data in HBase are stored in byte arrays, so we implemented the respective byte serializers for the purposes of our vectors' storage.

Besides, a functionality of HBase worth mentioning, that plays a crucial role in the scalability potential of the whole process, is bulk importing. Typically, bulk importing bypasses the HBase API and writes contents, which are properly formatted HBase data files - HFiles, directly to the file system. The main advantage of this method is that

1 From now on, we consider these meanings equivalent and use them in alternating fashion.

by directly mapping the output of M/R jobs to HFiles, we minimize the potential overhead of multiple Remote Procedure Calls from several tasks, as we scale to bigger datasets and utilize many nodes. In that way, we both gain the structural advantages HBase has to offer, without the overhead that concurrent RPC's from many nodes could provoke to the running time. The only limitation about this method, is that each HFile contents have to span across one and only region.

5.2.2 *Map-Reduce job analysis*

At this point, we provide an overview of each of the procedures needed to tackle each lasso regression problem:

$$\min_{\mathbf{B}\geq 0}||\mathbf{X}-\mathbf{A}\mathbf{B}^T||_F^2+\lambda||\mathbf{B}||_1$$

For the sake of brevity, we will focus on the case of minimizing the above expression over $\mathbf{B}$. Despite this, the extention to the case of minimizing over $\mathbf{A}$ is not that straightforward, as we pursued a solution which only stores $\mathbf{X}$ and not its transpose and the retrieval of the right subsystem ID for each of the contracting matrices in some jobs required further configurations.

Algorithm 4 Lasso Regression process via ADMM

r: #row partitions c:#column partitions of input matrix
$i=0,\ldots,rc-1$ and $j=\left\lfloor\frac{i}{c}\right\rfloor$.

1: Compute $(2\mathbf{A}_j^T\mathbf{X}_i)^T$ and store to the ith subsystem
2: Compute $(2\mathbf{A}_j^T\mathbf{A}_j+\rho\mathbf{I})^{-1}$ and store to the respective subsystems
3: **repeat**
4: Iterate through the equations in 24
5: **until** convergence

The above process is followed for the minimization w.r.t. $\mathbf{B}$ and a symmetric one is followed for the minimization w.r.t. $\mathbf{A}$ until convergence, as Sparse Matrix Regression algorithm (Algorithm 3) dictates.

We describe in detail each of the above procedures, as follows:

5.2.2.1 *First caching process*

This first inner procedure requires computing a large matrix-matrix multiplication in blocks. More specifically, we are dealing with parts of a narrow matrix $\mathbf{A}$, which have to be multiplied with parts of matrix $\mathbf{X}$. We employ a standard technique of matrix-matrix multiplication, illustrated in [9, 48]. A difference from these approaches is that we have to break multiplication with respect to each subsystem's ID, in order to store the correct results at respective positions, which is achieved by including the subsystem's ID into the key being sent from the first map task. As noticed above, the ith part of $\mathbf{X}$, will participate in the multiplication corresponding to the ith subsystem of our target matrix $\mathbf{B}$.

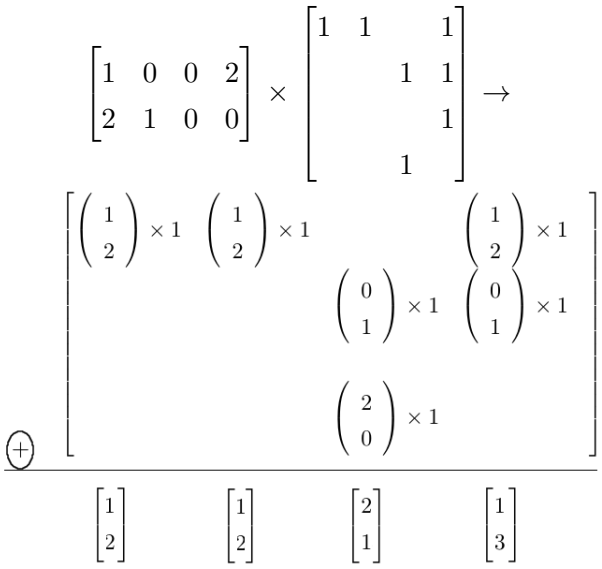

Figure 21: A simple example illustrating the multiplication of our first process: the partial products corresponding to each column are being accumulated to calculate each column of the result

In detail, at our first task of this process, we set the input of mappers to consist of the regions of matrix $\mathbf{X}$. As mentioned above, each map task will receive exactly one region's data, by sequentially reading vectors of size n_i, which depends on the partition of our input data w.t.r. to columns.

Let w_j denote the jth column (which is considered as row at the end of the calculation, in order to achieve the transposition imposed) of the result of each part of $\mathbf{A}^T\mathbf{X}$. A basic formula of the whole calculation is the following [48]:

$$w_p = \sum_{i=1}^{m} \mathbf{X}_{i,j} a_i^T = \sum_{i \in \mathbf{O}^j} \mathbf{X}_{i,j} a_i^T$$

This indicates that w_p is a linear combination of $\{a_i^T\}$ over the nonzero cells on the jth column of $\mathbf{X}$. Thus, the first step we have to accomplish is to multiply each a_i vector with all the non-zero cells of ith row of $\mathbf{X}$ and as a second step, we have to sum the partial vectors corresponding to the same column of matrix $\mathbf{X}$. An example of this method's processing is given in 21. We have to accentuate that because of the fact the third column (i.e. the third row we would read in such an example at the first M/R job) of the first matrix being multiplicated is a zero vector, it is not transported to reducers at all, with the goal of minimizing the network load.

As we are handling a soft co-clustering algorithm and as a result not all rows of a matrix have to participate in a co-cluster, we expect that a significant amount of these rows are being successively minimized to zero. Thus, this choice reduces significantly the network load, as algorithm iterations proceed.

The Map-Reduce jobs participating in this process are the following:

- `Map-I`: Map $<$ `SubId,i` $: j, X_{i,j} >$ from $\mathbf{X}$. At the job's cleanup, open table containing matrix $\mathbf{A}$ and map $<$ `SubIdA,i` $: a_i >$

for respective subsystems as dictated above by process 4. Figure 21 indicates that the ith column vector of first matrix is being multiplicated with the elements of the ith row of the second one. However, we are reading the first matrix transposed (i.e. we are reading row vectors of $\mathbf{A}$) at map phase. As a result, $<$ `key` : $value >$ pairs sharing the same subsystem id and the same row will end up to the same reducer.

- `Reduce-I`: Emit $<$ `SubId,j` : $\mathbf{X}_{i,j}a_i^T >$ for each value list sharing the same subsystem id and the same row.
- `Map-II`: Map $<$ `SubId,j` : $\mathbf{X}_{i,j}a_i^T >$ such that vectors corresponding to the same column of the result end up to the same reducer.
- `Reduce-II`: Emit $<$ `SubId,j` : $\sum_{i \in \mathbf{O}^j} \mathbf{X}_{i,j}a_i^T >$ as the result of each result's column for each subsystem separately.
- Finally, a last M/R job is being utilized in order to bulk import data to the table's respective regions.

An example of data flow for this process is depicted in 22.

5.2.2.2 *Second caching process*

In the next job, we have to cache the amount of $(2\mathbf{A}_j^T\mathbf{A}_j + \rho\mathbf{I})^{-1}$ where $i = 0, \ldots, rc-1$ and $j = \left\lfloor \frac{i}{c} \right\rfloor$ and r, c represent row/column partitions respectively. As mentioned above concerning region structure of our matrices, as shown in Figure 20, beyond the local solutions for each part of data, the global solution of matrix $\mathbf{A}$ is being (redundantly) stored as well in each respective part of data, for efficient access from mappers(from each "slice" we could retrieve the whole matrix), as a result, we have to scan subsystems zero up to $r-1$ to retain the complete $\mathbf{A}$ matrix in mappers and calculate the desired multiplication.

The setting of how many regions/subsystems of a table our mappers will read, is being facilitated by our rowkey design and our tables' structure. As mentioned above, the rowkey consists of a concatenation of subsystem's id and the respective row id (keys are stored by default in lexicographic order in HBase's table). Thus, we simply set in the wrapper program the configuration of *stopRow* of our scan, to be equal to the number of row partitions to which our $\mathbf{A}$ table has been divided minus one $(r-1)$: the first "slice" of regions as concerns Figure 20.

What we intend to highlight at this point, is that this task, though it seems simple enough, would not be straightforward at all, without the aid of this particular structure we followed considering contracting matrices/tables of our problem.

As for the main task, we exploit the fact of multiplying a matrix by itself, by completing this step in one M/R job. In particular:

- `Map-I`: We place each row we read into two iterators. For each non-zero value of the first one, we iterate over all non-zero values of the second one. This represents typically a double for-loop. Its outer iteration sets x equal to each index of non-zero value (let

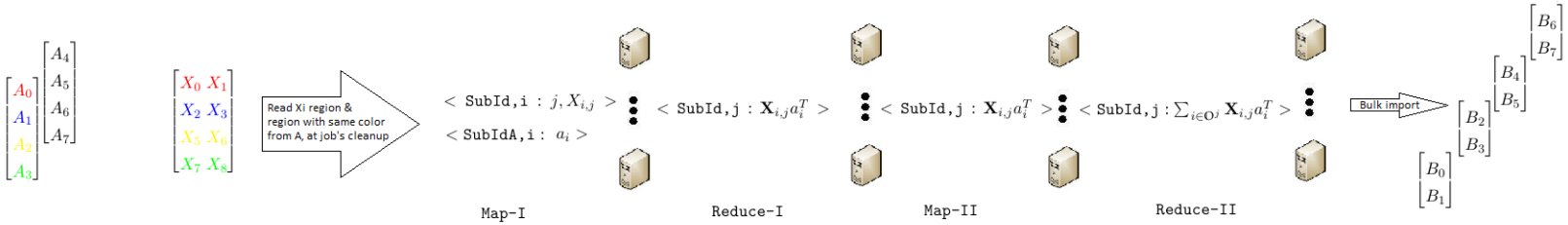

Figure 22: Data flow for our first caching process. Example with $r = 4, c = 2$

the value be val_1) and the inner iteration sets y as well to each non-zero value's (let it be val_2) index. The $<$ `key` : $value$ $>$ pairs we emit are $<$ `SubId, x ,y` : $val_1 \times val_2$ $>$ so that values sharing the same subsystem, as well as matrix position (x, y) end up to the same reducer.

This process is depicted below by a simple example:

Consider the following matrix:

$$\begin{bmatrix} 1 & 0 & 2 \\ 4 & 3 & 1 \end{bmatrix}$$

The $<$ `key` : $value$ $>$ pairs emitted from the first row are:

$<$ `0,0` : 1 $>, <$ `0,2` : 2 $>, <$ `2,0` : 2 $>, <$ `2,2` : 4 $>$

From the second row, we emit:

$<$ `0,0` : 16 $>, <$ `0,1` : 12 $>, <$ `0,2` : 4 $>, <$ `1,0` : 12 $>, <$ `1,1` : 9 $>, <$ `1,2` : 3 $>, <$ `2,0` : 4 $>, <$ `2,1` : 3 $>, <$ `2,2` : 1 $>$

By accumulating the values corresponding to the same matrix positions, we lead to the result of calculating the transpose of a matrix with itself:

$$\begin{bmatrix} 17 & 12 & 6 \\ 12 & 9 & 3 \\ 6 & 3 & 5 \end{bmatrix}$$

- `Reduce-I`: For each value list, we accumulate over it, to retrieve the value for each x, y position of $\mathbf{A}_j^T \mathbf{A}_j$ matrix and after that, we multiply by 2 each extracted value, as our formula dictates. The number of reducers is being defined as r (the number of row partitions), and each of them keeps in memory a matrix of size $k \times k$, which is updated as results referring to different matrix positions are being calculated. At the job's cleanup, we add ρ parameter to the values lying in the matrix's diagonal and compute its inverse. Then, we write this matrix to sequence file, in order to provide access from respective map tasks of ADMM's main iteration.

The data flow of this job is graphically presented in 23. Notice the number of reduce tasks, they are equal to the number of row partitions.

5.2.2.3 *Main Iteration*

This job represents the core functionality of an ADMM's iteration.

Each map task initially retrieves the inversed matrix of size $k \times k$ from the second step and stores it. It reads sequentially vectors of size k from local "opinion" $\mathbf{B}_i$, Lagrange multiplier $\mathbf{L}_i$, global solution $\mathbf{C}_q$ and $(2\mathbf{A}_j^T \mathbf{X}_i)^T$.

It computes each vector of $\mathbf{L}_i$, which is trivial as is depicted by the respective equation. For each vector from $\mathbf{B}_i$, we multiply finally with

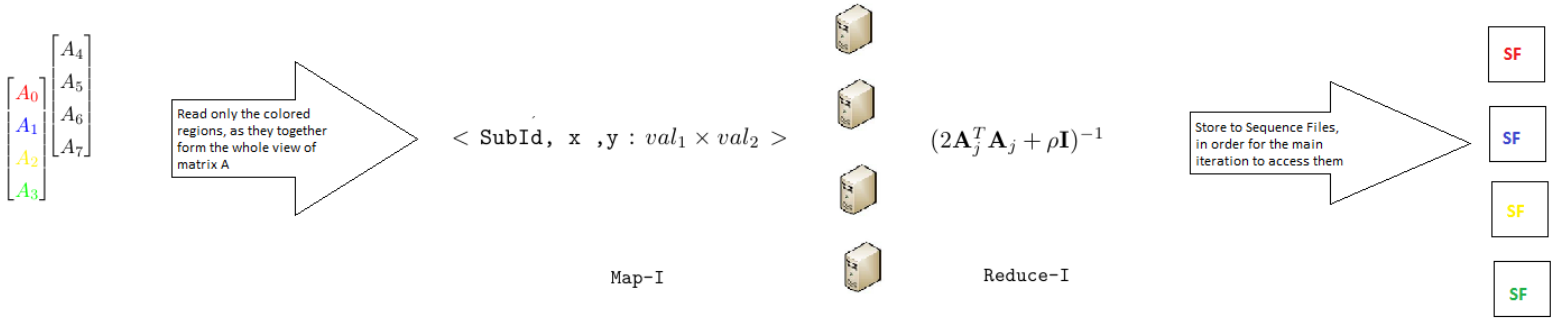

Figure 23: Data flow for our second caching process. Example with $r = 4, c = 2$

the stored inverse matrix of size $k \times k$ from step 2, which is also a simple operation.

It is worth noticing that in order to compute the amounts necessary for convergence in a distributed form, we read the last two versions of each row and accumulate partial results to variables. More details are provided in the respective section.

- `Map-I`: Each map task computes

$$\mathbf{B}_i^{k+1} = ((2\mathbf{A}_j^T\mathbf{X}_i)^T + \rho\mathbf{C}_q^k - \mathbf{\Lambda}_i^k)(2\mathbf{A}_j^T\mathbf{A}_j + \rho\mathbf{I})^{-1}, \text{ and}$$
$$\mathbf{\Lambda}_i^{k+1} = \mathbf{\Lambda}_i^k + \rho(\mathbf{B}_i^{k+1} - \mathbf{C}_q^{k+1})$$

  where, r is the number of row groups and c the number of column groups to which our input matrix is partitioned, $i = 0 \dots rc - 1$, $j = \left\lfloor \frac{i}{c} \right\rfloor$ and $q = i \mod c$.

  The computation of $\mathbf{C}_q$, which takes place in the reduce step, does not need to access the local solutions of $\mathbf{B}_i, \mathbf{L}_i$, but their sum (with $\mathbf{L}$ divided by ρ). In addition, an important remark here is that we are not obligated to transfer these partial solutions to reducers in order to store them, because we could append them directly to HFiles and bulk load them at each map task's cleanup operation.

  This is the approach we follow for this operation and the tuples being emitted to reducers are of the following form:
  $<$ `SubId % c, rowId` : $vectorSum, partialNorms >$, where $vectorSum$ is the sum of each row's $\mathbf{B}_i + \mathbf{L}_i/\rho$ and partialNorms correspond to the partial results that need to be accumulated at each reduce task, in order to distributedly compute the amounts necessary to check if stopping criteria have been reached. Besides, we are emitting the region's identifier remainder of its division by the number of row partitions r, so that regions computing local "opinions" about the same portion of data, are reduced by the same task. These regions were mentioned above as being at the same "level", as concerns Figure 20.

  A combine step is also provided in order to accumulate values sharing the same key.

- `Reduce-I`: As a result, partial sums referring to the same row of the global solution $\mathbf{C}_q$ are being received from the same reducer which accumulates them and performs the Soft-Thresholding operator as below:

$$\mathbf{C}_q^{k+1} = \max(0, \mathbf{S}_{\frac{\lambda}{\rho r}}(\bar{\mathbf{B}} + \frac{\bar{\mathbf{\Lambda}}}{\rho}))$$

  It is useful at this point to accentuate the division of the subscript of Soft-Thresholding operator (formula in 11) by r. This operator typically constitutes a comparator, and since the sum calculated into the parenthesis is averaged by the number of subsystems that

"consent" to each value (i.e. r in this case) [2], we also have to divide the subscript of this operator with r in order to maintain the values depicted in the parenthesis and the subscript comparable.

Because of the aforementioned issue of classic HBase API, concerning the delays associated with a possible plethora of RPC's received from many sources, we first write the global solution for each row of $\mathbf{C}_q$ to context, and then with a second M/R job, we shard these solutions to respective subsystems using bulk loading, in order to make it efficient to access them. It is not possible to bulk load data at our first job of this process, because (as mentioned above), each HFile to which we append to, has to span across one and only region. However, in this case, we desire to store each vector of global solution $\mathbf{C}_q$ to many regions, in order to make it efficient for them to access it in the subsequent map tasks.

A graphical representation of this process is provided in Figure 24.

5.2.2.4 *SMR algorithm convergence check*

Beyond the convergence criteria of each ADMM round (alternating between minimizing the cost for each one of the contracting matrices), we have to control if the outer criterion we are aiming to minimize has reached an optimum. This criterion has the following form:

$$\min_{\mathbf{A}\geq 0,\mathbf{B}\geq 0} ||\mathbf{X} - \mathbf{A}\mathbf{B}^T||_F^2 + \lambda||\mathbf{A}||_1 + \lambda||\mathbf{B}||_1$$

Thus, we have to compute the above amount after each pair of ADMM rounds (one for each of $\mathbf{A}, \mathbf{B}$). We have to focus on two main points:

- how the multiplication between global solutions of matrices $\mathbf{A}, \mathbf{B}^T$ will be achieved in an efficient way and
- in what manner should we retrieve the initial matrix $\mathbf{X}$ from its respective HBase's table by ensuring that we are performing the minimun possible number of RPC's (remote proceduce calls to HBase). It is worth mentioning that we are not able to base upon assumptions that an entire region ($\mathbf{X}_i$ as of Figure 20) of contents of matrix $\mathbf{X}$, (or respectively of $\mathbf{A}\mathbf{B}^T$), could fit into memory, as we are aiming the whole procedure to be applied to large-scale data.

The matrix multiplication is being based upon the standard (sparse matrix-oriented) method of multiplying two matrices in Map-Reduce. The difference (as in the aforementioned first caching process) is again the multiplication in blocks, so that each reduce task of the multiplication's second job, has to process around the same amount of data and has to retrieve a specific subsystem's (region) contents of $\mathbf{X}$ in order to make the respective norm calculation.

2 This is the general aforementioned meaning of the "consensus" approach, i.e.: compute a mean value over the sum of local "opinions".

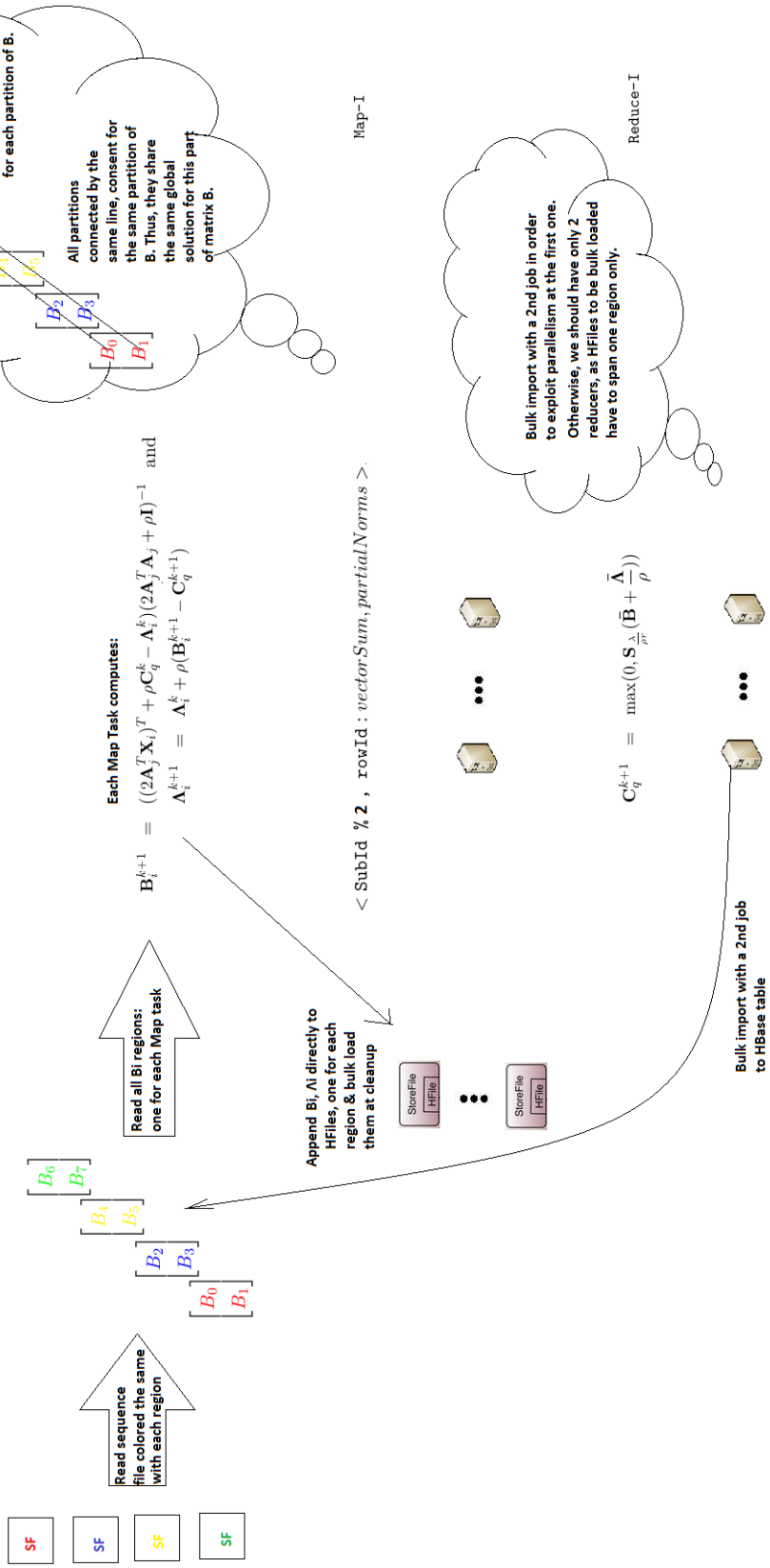

Figure 24: Data flow for our main iteration process. Example with $r = 4, c = 2$

The aim is that elements of the resulting $\mathbf{AB}^T$ multiplication, will arrive in each reduce task, and (limited by the memory requirements) will be row-by-row subtracted by each respective row we retrieve from matrix $\mathbf{X}$. With this solution though, we fetch *each and every row* of $\mathbf{X}$, even when rows of the matrix multiplication are *zero vectors* - and we expect many of them to be completely zero as the algorithm progresses, because of its sparse nature. In this case though (of a resulting zero vector), we just have compute the sum of squares of the matching row from $\mathbf{X}$.

Motivated by this, we expand the minimization formula, as follows:

$$||\mathbf{X} - \mathbf{AB}^T||_F^2 + \lambda||\mathbf{A}||_1 + \lambda||\mathbf{B}||_1 =$$
$$||\mathbf{X}||_F^2 - 2\sum(\mathbf{X} \circ (\mathbf{AB}^T)) + ||\mathbf{AB}^T||_F^2 + \lambda||\mathbf{A}||_1 + \lambda||\mathbf{B}||_1$$

where $\circ$ denotes the Hadamard (element-wise) product between the contracting matrices.

Employing the above formulation, we remark that if rows of the resulting matrix multiplication of $\mathbf{AB}^T$ are zero vectors, then the term $(\mathbf{X} \circ (\mathbf{AB}^T))$ for the respective row would be zeroed out, as well. In that case, we would have to simply (even separately from this matrix multiplication) compute the sum of squares of the matching row from $\mathbf{X}$ matrix and accumulate it to the result.

As a result, at the algorithm's beginning, we compute the term $||\mathbf{X}||_F^2$ and store it to sequence files. Thus, each time we have to compute the total cost of our solution, we trace back to those files to retrieve the unchanged $\mathbf{X}$ norms and accumulate them to the total cost.

The Map-Reduce jobs participating in this process are the following:

- `Map-I`: We read matrix $\mathbf{B}$ at map tasks. As mentioned above, the ith region of $\mathbf{B}$ will be processed by the ith mapper, for $i = 0, \ldots, rc - 1$. At each map job's cleanup function, we retrieve the jth region of $\mathbf{A}$, where $j = \left\lfloor \frac{i}{c} \right\rfloor$ and c is the number of column partitions of input matrix.

  At this stage, we accumulate the respective amounts for the calculation of l_1 norms of $\mathbf{A}$ and $\mathbf{B}$ and store them to sequence files, in order to be included to the final result.

  From map tasks, we map each $<$ `SubId,j` $: i, value >$ for each non-zero value of each of matrix. We ensure by a partitioning function that tuples sharing the same region Id, will be processed by the same reducer.

- `Reduce-I`: For each key j, examine its list of associated values and for each value that comes from $\mathbf{A}$ (say $\mathbf{A}, i, a_{ij}$) and each one that comes from $\mathbf{B}$ (say $\mathbf{B}, k, b_{jk}$), produce the tuple $<$ `SubId,j` $: i, k, a_{ij}b_{jk} >$

- `Map-II`: A grouping and aggregation follows, by mapping tuples at a form in which subsystem and matrix coordinates constitute the key (i.e. $<$ `SubId,i,j` $: value >$). We again reassure by the partitioner that tuples sharing the same subsystem Id, will lead to the same reduce task.

- `Reduce-II`: In this reduce task, we take advantage of the sorting by key performed by the Map-Reduce framework to all tuples leading to the same reducer. We set the number of reduce tasks to be equal to the number of regions ($r * c$), to which we have partitioned our input matrix and as a result we receive results in need to be matched with the ith region of $\mathbf{X}$ to the ith reducer. Matrix positions though, come sorted (as we have configured their object's "compare" function), by row number. In this way, we do not receive results for row $p+1$ until the reception of all elements of row p completes.

  Thus, we store a vector of size equal to the number of columns of each $\mathbf{X}$ region and while accumulating the value lists of respective matrix positions, we wait until the row key changes. If this happens, we fetch the respective row key from $\mathbf{X}$ and compute the term $-2\sum(\mathbf{X} \circ (\mathbf{A}\mathbf{B}^T)) + ||\mathbf{A}\mathbf{B}^T||_F^2$ for this vector.

  In this way, we ensure that rows of $\mathbf{X}$ will be retrieved from HBase, if it is necessary to compute their element-wise product with the respective row from the result of matrix multiplication.

  Finally, partial results from reducers are written to files, and the wrapper program calculates the total cost of SMR algorithm.

5.3 ADMM RELATED CHOICES

5.3.1 *ADMM convergence criteria*

The convergence criteria for the consensus form of Alternating Direction Method of Multipliers, are essentially primal and dual residuals. The former of these amounts typically constitute the measure of accumulated deviations of each local "opinion" from the global problem's solution, while the latter represents the amount of difference of global solution from each previous value (i.e.: the typical convergence criteria of most known algorithms). As proved in [49] for the non-distributed form and extended for the consensus one, these residuals are formulated respectively as:

$$||r^k||_2^2 = \sum_{i=1}^{N} ||\mathbf{B}_i^k - \mathbf{C}^k||_F^2 \text{ and } ||s^k||_2^2 = N\rho^2||\mathbf{C}^k - \mathbf{C}^{k-1}||_F^2$$

with $||\mathbf{Z}||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|z_{ij}|^2}$ being the Frobenius norm of matrix $\mathbf{Z}$.

The convergence criteria as a whole consist of checking the following conditions:

$$||r^k||_2^2 \leq eps_{feas} \text{ and } ||s^k||_2^2 \leq eps_{conv}$$

By applying the formulas for eps_{feas} and eps_{conv} as defined in [49], we end up to :

$$eps_{feas} = e_{abs}\sqrt{k * r} + e_{rel} * max(\sqrt{\sum_{i=1}^{N}||\mathbf{B}_i||_F^2}, \sqrt{r}||\mathbf{C}||_F^2) \text{ and}$$
$$eps_{conv} = e_{abs}\sqrt{k * r} + e_{rel}\sqrt{\sum_{i=1}^{N}||\mathbf{L}_i||_F^2}$$

where k is the number of desired co-clusters, r is the number of row partitions, and e_{abs} and e_{rel} are an absolute and a relative tolerance respectively, and are adjusted according to the approximation of convergence criteria the user desires to reach.

As mentioned above, the calculation of all necessary amounts for convergence checking is being performed distributedly as to avoid delays - while the above amounts that need accumulation over subsystems are passed from map to reduce tasks at ADMM's main iteration.

It is useful to remark at this point, that after each iteration's completion, we employ the so-called *warm-start* technique, by initializing our local $\mathbf{B}_i, \mathbf{L}_i$ with the previous iteration's values for these matrices. This choice accelerates algorithm's convergence.

5.3.2 *Automatic adjustment of parameter ρ*

The speed of this method's convergence depends on the selection of parameter ρ. In [56], analytical solutions for finding the appropriate ρ are given for some sub-cases of the general form of ADMM. Our case of interest (l_1 - reguralized loss minimizations), is left by authors as a future research problem. Their empirical results as concerns this case, show a connection between the first and last (if sorted) eigenvalues of $\mathbf{A}^T\mathbf{A}$ and the value of lambda we choose for the lasso regression (e.g. if $\lambda \leq e_1(\mathbf{A}^T\mathbf{A}), \rho = e_1(\mathbf{A}^T\mathbf{A})$, where e_1 is the smallest eigenvalue in magnitude etc.)

These choices did not seem always stable in our case, thus we employed a simpler and more practical solution.

Initially, a crucial point to understand is that large values of ρ place a large penalty on violations of primal feasibility and so tend to produce small primal residuals. Conversely, the definition of $||s^k||_2^2$ suggests that small values of ρ tend to reduce the dual residual, but at the expense of reducing the penalty on primal feasibility, which may result in a larger primal residual. This penalty on primal feasibility refers to the equation of updating the Lagrange multiplier:$\mathbf{\Lambda}_i^{k+1} = \mathbf{\Lambda}_i^k + \rho(\mathbf{B}_i^{k+1} - \mathbf{C}_q^{k+1})$

What we pursue as an optimal scheme of minimizing these residuals, is their reduction happening with the same pace. However, empirically we noticed that these residuals may be orders of magnitude away from each other, so we employed the following solution after each complete iteration (in order to use the updated ρ at next ADMM round) for each side separately (i.e.: the cases of minimizing $\mathbf{A}, \mathbf{B}$ have its own ρ parameter): Let $div_1 = \frac{||r^k||_2^2}{eps_{feas}}$, $div_2 = \frac{||s^k||_2^2}{eps_{conv}}$ and $diff = |div_1 - div_2|$.

$$\rho = \begin{cases} \begin{cases} \rho * t_{incr} & div_1 > div_2 \\ \frac{\rho}{t_{decr}} & \text{else} \end{cases} & \text{if both/none crit. converged and diff>T} \\ \begin{cases} \rho * t_{incr} & \text{if} ||s^k||_2^2 \leq eps_{conv} \\ \frac{\rho}{t_{decr}} & \text{if} ||r^k||_2^2 \leq eps_{feas} \end{cases} & \text{if reached max. iterations} \end{cases}$$

Common values for parameters are: $T = 0.75, t_{incr} = 1.25, t_{decr} = 2$
The above conditions depict the following: If the ADMM round did not

lead to convergence and only one of the two criteria did not reached a desirable value, increase/decrease ρ accordingly. On the other hand, if both or none criteria converged, we check the "distance" from their goals, by dividing with eps_{feas} and eps_{conv} respectively. If the relative difference between these two distances is bigger than a threshold, adjust ρ as necessary.

A similar, but simpler technique of adjusting ρ is provided in [49], that is employed into the same round of ADMM iterations. This would imply for our case that we should compute the amount of $(2\mathbf{A}_j^T\mathbf{A}_j + \rho\mathbf{I})^{-1}$ after each change in ρ, and would also contrast to the algorithm's theory of convergence.

This is the reason why we choose to adjust ρ at the end of each ADMM round of iterations, as to be updated in the subsequent one.

6

EXPERIMENTAL RESULTS

In this section, we adduce experimental results of our work on real data and discuss upon the purity and the associated attributes with each co-cluster in a number of cases. Results on horizontal scaling of our Map-Reduce jobs are provided as well.

6.1 DATASET DESCRIPTION AND MOTIVATION

Our inspiration for the current experimentation has been the work in [46], where co-clustering is being applied to the problem of network intrusion detection.

In general, the problem in context comes from the fact that security constitutes an increasingly large problem in today's Internet. However, its initial architecture did not consider security to be a high priority, leaving the problem of managing security concerns to end hosts. This is therefore a growing problem for system administrators having to continually avoid a variety of attacks and intrusion attempts by both individuals and large botnets.

This security issue creates a challenge for system administrators of how to distinguish normal from malicious connections. A key challenge for an administrator is how to block the latter, by leaving the former connections, coming from legitimate users, unaffected. We can never know for sure to which of these categories any given connection falls into, but we can attempt to isolate a set of connections that stand out from normal user behavior, so as to help system administrators detect attacks. This problem is being called as network intrusion detection.

In [59, 60], the authors show that certain types of attacks are strongly correlated with only *subsets* of a connection's parameters. In other words, it is possible to determine the type of an attack just by looking at the certain subset of parameters that best characterizes this attack. This is the reason why co-clustering is a useful method at this context, as it employs an unsupervised technique aiding to find these particular parameters that best identify the type of an abnormal connection.

Our experiments are based on KDD 1999 Cup data set [58], which constitutes a standard set of data that can be used to evaluate proposed approaches in the field on intrusion detection. It contains $4,898,431$ connections each of which is characterized by 41 attributes, and a label defining the identity of each connection and used for evaluation purposes. The attributes, as well as the distribution of types of connections belonging to this dataset are in Figures 25 and 26 respectively.

The reasons for choosing this dataset to run our experiments on are bilateral: First, the labels it contains are a useful tool of *quantitatively* evaluating our method's efficiency. On the other hand, it originates from real measurements and its size (close to 1GB after the normalization we imposed to it) is a challenge for the standard Sparse Matrix Regression

Measurement Name	Measurement Type
duration	continuous
protocol_type	symbolic
service	symbolic
flag	symbolic
src_bytes	continuous
dst_bytes	continuous
land	symbolic
wrong_fragment	continuous
urgent	continuous
hot	continuous
num_failed_logins	continuous
logged_in	symbolic
num_compromised	continuous
root_shell	continuous
su_attempted	continuous
num_root	continuous
num_file_creations	continuous
num_shells	continuous
num_access_files	continuous
num_outbound_cmds	continuous
is_host_login	symbolic
is_guest_login	symbolic
count	continuous
srv_count	continuous
serror_rate	continuous
srv_serror_rate	continuous
rerror_rate	continuous
srv_rerror_rate	continuous
same_srv_rate	continuous
diff_srv_rate	continuous
srv_diff_host_rate	continuous
dst_host_count	continuous
dst_host_srv_count	continuous
dst_host_same_srv_rate	continuous
dst_host_diff_srv_rate	continuous
dst_host_same_src_port_rate	continuous
dst_host_srv_diff_host_rate	continuous
dst_host_serror_rate	continuous
dst_host_srv_serror_rate	continuous
dst_host_rerror_rate	continuous
dst_host_srv_rerror_rate	continuous

Figure 25: List of connection measurements in the data set

label	total_conn
back	2203
buffer_overflow	30
ftp_write	8
guess_passwd	53
imap	12
ipsweep	12481
land	21
loadmodule	9
multihop	7
neptune	1072017
nmap	2316
normal	972780
perl	3
phf	4
pod	264
portsweep	10413
rootkit	10
satan	15892
smurf	2807886
spy	2
teardrop	979
warezclient	1020
warezmaster	20

Figure 26: Distribution of types of connections belonging to the data set

algorithm, as reported in [46], where the authors employ this algorithm to samples of $\frac{1}{50}$ of the whole dataset.

6.2 ACCURACY AND INTERPRETATION

We run experiments for cases when the number of desired co-clusters was $k = 2, 3, 4, 5$. For the case when 2 co-clusters were set, the algorithm created consistently co-clusters dominated by attacks. In particular, one of them contained mainly "neptune" while the other contained mainly "smurf" attacks. The fact that as it is shown in 26, these two types of connections consist of the 80% of the whole dataset, could explain this behavior. As our context is anomaly detection, we prefer to focus on cases when (pure) clusters with normal connections participate in the analysis. For three co-clusters, the number of connections and purity results (normal/abnormal) of each co-cluster are depicted in table 1. Besides, the individual participation of neptune and smurf attacks as of the whole cluster contents, are being pointed out.

Pleasingly enough, the algorithm extracts the 3 main data "concepts" - (neptune,smurf attacks and normal connections) with high purity at each co-cluster. These three groups of connections constitute the 99% of input data, as shown in Figure 26.

In order to visualize and facilitate interpretation of the associated measurements, we graphically present them, by setting the font size of each attribute to be equivalent with its belongingness to the co-cluster. The pleasing property of SMR algorithm not only to define the participations to each cluster, but also to posess a numerical value at the respective matrix entry for each row/column, aids to this direction. The indicated font size is typically the magnitude of each numerical value divided by the maximum value of the respective co-cluster for this matrix. The associated parameters for each group of connections, for $k = 3$, are shown in Figure 27.

In [46], it is being reported that a certain subset of parameters was consistently associated with attacks, in general. A finding at this point, concerning our work, is that this exact set of parameters are associated with our "smurf" dominated co-cluster (and not with both clusters dominated by attacks) and only two of these parameters are present in both the "neptune" and the "smurf" co-clusters. These two parameters shared by both co-clusters are: dst_host_count and count, as shown in Figure 27.

For $k = 4$, co-cluster purity is again very high as of normal/abnormal distinction, as shown in table 2. As mentioned above, the basic data "concepts" are three, so the fact that overlapping co-clusters start to form and smurf connections are being mixtured with neptune ones, is considered normal. Besides, both types of connections are distinguished more generally as Denial of Service (DoS) attacks, so this algorithm's "choice" seems rational.

As for the associated parameters for $k = 4$ shown in Figure 28, the first co-cluster posesses exactly the same attributes as the smurf co-cluster for $k = 3$. This comes from the fact that the percentage of

Co-Cluster	Number of Connections	Percent Normal	Neptune | Smurfs Percent Attacks
1	879,404	0.147%	98.637% | 0% 99.853%
2	837,457	98.698%	0.005% | 0.018% 1.301%
3	3,006,618	5.91%	0.416% | 93.4% 94.1%

Table 1: Results for $k = 3$

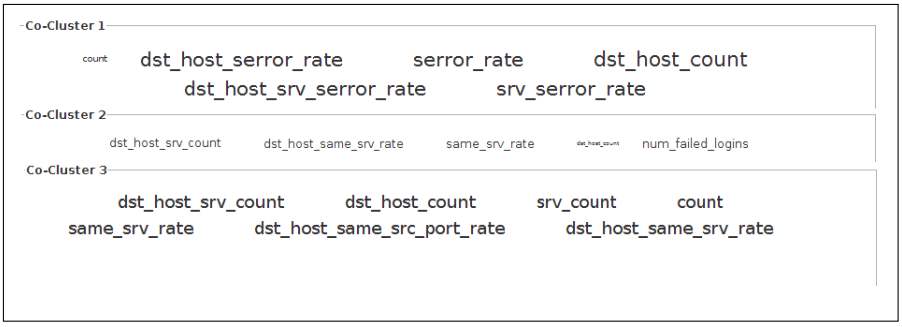

Figure 27: Associated attributes with each group of connections for $k = 3$

smurfs is very high, as related to the respective percentage of neptune connections. In the case when neptune participation increases in the second co-cluster full of attacks, we remark that dst_host_count is the most correlated attribute with this co-cluster. It is worth mentioning that this is the one of two features, which are shared by both totally homogeneous co-clusters of smurf and neptune attacks, in the case of $k = 3$.

As concerns the two pure normal co-clusters formed in this case, we notice that attributes dst_host_same_srv_rate, same_srv_rate and dst_host_srv_count represent the most important correlations with normal connections, which agrees with the respective results for the normal co-cluster, for $k = 3$. In addition to these, land feature is also considered a strong one at the fourth co-cluster.

For the case of $k = 5$, the results are equally satisfactory with both previous cases, as we notice again a very high level of homogeneity of the resulting co-clusters, in distinguishing normal from anomalous connections. As concerns the first co-cluster, which is full of normal connections, we notice that all the features that appeared in both previous cases ($k = 3, 4$) for the normal co-clusters,i.e.: dst_host_same_srv_rate, same_srv_rate and dst_host_srv_count along with land attribute which appeared in the case of $k = 4$, have the strongest correlation with this normal cluster.

Another favorable property appears in the case of the second co-cluster, where we notice that the two parameters being shared by both pure neptune and smurf co-cluster (as for $k = 3, 4$) are those appearing in this case (i.e.: dst_host_count and count). This establishes our

Co-Cluster	Number of Connections	Percent Normal	Neptune | Smurfs Percent Attacks
1	3,304,994	0.28%	14.89% | 84.64% 99.72%
2	2,580,602	2.84%	23.5% | 72.86% 97.17%
3	23,670	96.34 %	0.097% | 0.097% 3.658%
4	575,804	98.185 %	0.196% | 0.33% 1.814%

Table 2: Results for $k = 4$

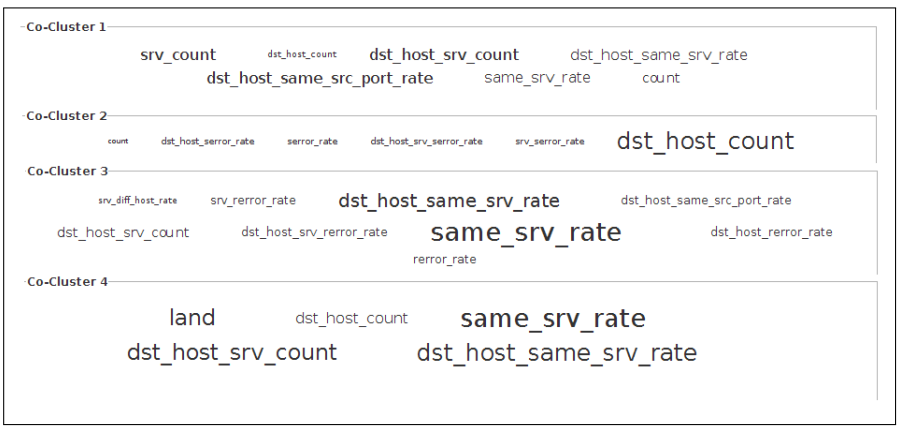

Figure 28: Associated attributes with each group of connections for $k = 4$

Co-Cluster	Number of Connections	Percent Normal	Neptune | Smurfs Percent Attacks
1	729,174	97.68%	0.195% | 0.327% 2.32%
2	3,683,424	0.49%	23.39% | 76.01% 99.51%
3	40,726	5.89 %	63.25% | 0% 94.1%
4	871,202	0.075 %	99.57% | 0% 99.92%
5	418,981	3.165 %	0.02% | 96.392% 96.835%

Table 3: Results for $k = 5$

hypothesis that when we face an homogeneous attack co-cluster (e.g. 99.51% purity in particular), having a percentage of either smurf or neptune connections below 80% (i.e. it is not highly pure on either one of them), then the features associated, tend to be the *intersection* of the contracting parties.

Results on features of third co-cluster indicate the mixture of neptune attacks with other malicious connections (e.g.: portsweep, ipsweep etc.), as the attributes appeared in this cluster, are not matched with neptune's correlated attributes. This is justified by the relatively low percentage of neptune connections as of the whole cluster contents (63.25%), which makes allowance for other types of connections to define the co-cluster attributes. It is worth mentioning though, that the percentage of attacks (94.1%) again indicates a highly homogeneous cluster, as of normal/abnormal distinction. As for the last two co-clusters, the one containing mostly neptune attacks is being associated with most of the connections we considered to be correlated with this kind of attack, in cases for $k = 3, 4$, while the last one is again correlated with all seven attributes defining a highly coherent "smurf" co-cluster.

As an inference, we notice that our approach continuously distinguishes normal connections from malicious ones, by building highly homogeneous respective clusters and is in general on par with the winning entries of the KDD Cup 1999 competition. Besides, we discover a consistent correlation with attributes on most types of clusters, which proves the method's efficiency for the context in use - i.e.: for detecting which subset of features is "responsible" for each type of connections.

6.3 SCALING

At this point, we provide diagrams depicting the horizontal scaling (i.e.: how run-time is affected as we increase node capacity) of our Map-Reduce jobs - grouped as a process. It is worth mentioning that

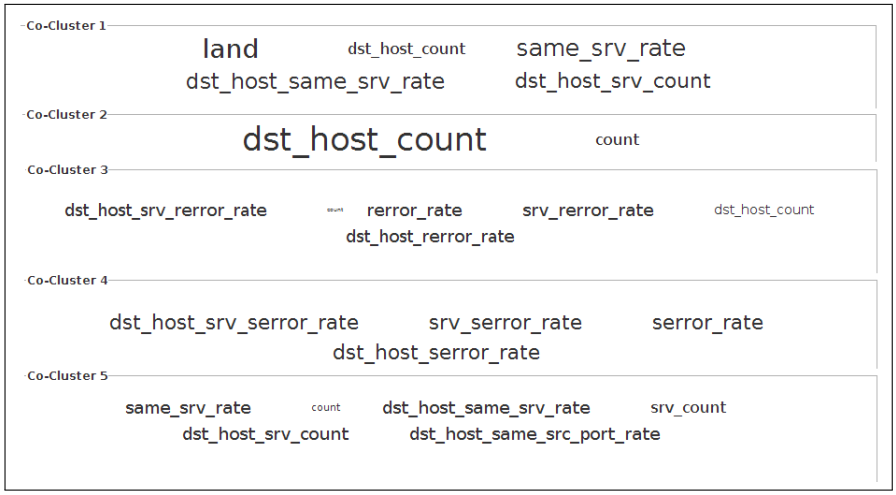

Figure 29: Associated attributes with each group of connections for $k = 5$

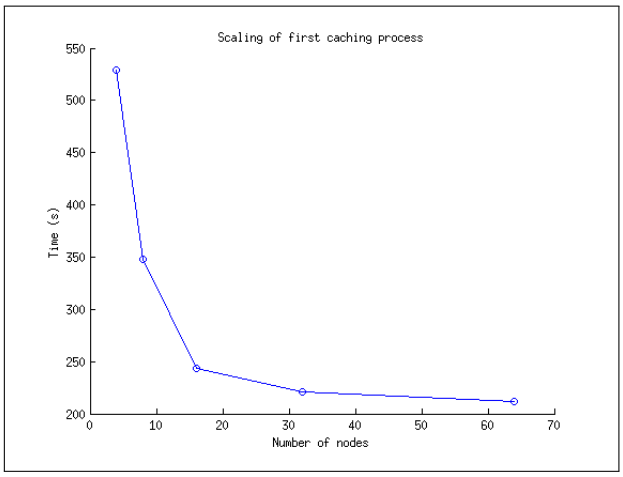

Figure 30: Scaling of first caching process (concerning all 3 jobs)

all 3 processes (except the check of convergence) share similar linear behavior, as concerns horizontal scaling.

As for the last group of jobs (checking if whole SMR converged), we performed these experiments just after random initialization of target matrices and as a result we had to handle with dense matrices - which (as mentioned above) is not the common case the algorithm has to handle. In fact, it is an extreme worst-case scenario. This justifies both the long running time of this job, as well as our choice to start our experiments with 16 nodes.

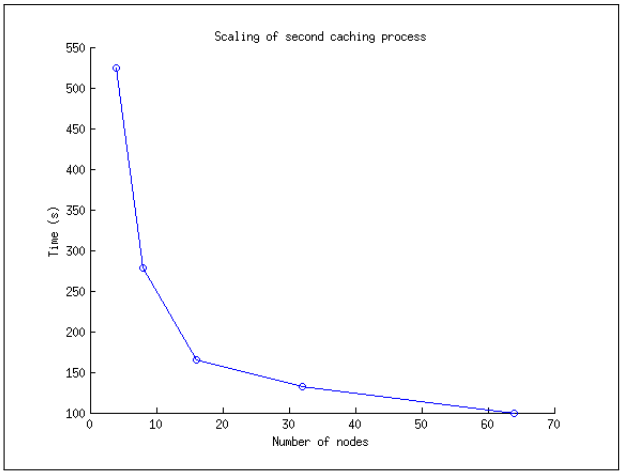

Figure 31: Scaling of second caching process

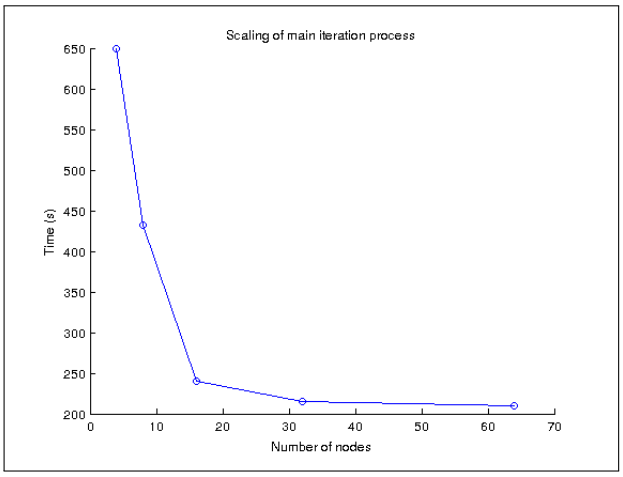

Figure 32: Scaling of main iteration process

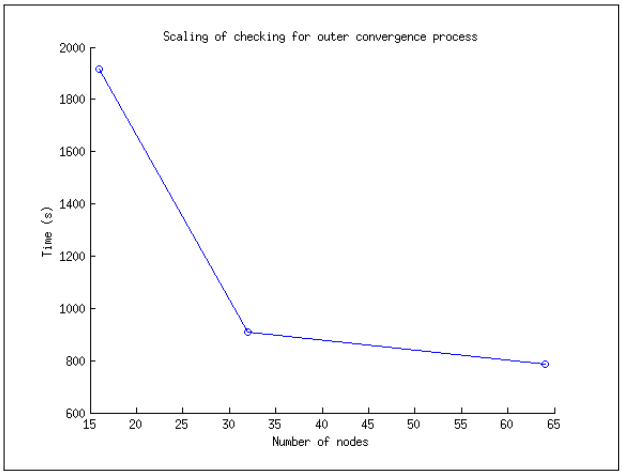

Figure 33: Scaling of checking for outer convergence process (worst-case scenario)

7 CONCLUSION

In this thesis, we tackled the problem of co-clustering through Sparse Matrix Regression algorithm, by employing an approach of each SMR subproblem, through Alternating Direction Method of Multipliers. This formulation enables us to express the problem in Map-Reduce, and implement on its most popular open-source platform, Hadoop, with the aid of structural properties HBase provides.

We notice that our approach builds highly homogeneous co-clusters between the two matrix dimensions that are meaningful. Furthermore, we discover a consistent correlation with attributes on most types of clusters, which proves the method's efficiency for the context in use - i.e. for detecting which subset of features is "responsible" for each type of connections.

7.1 FUTURE WORK

- Adapt the current solution to other problems involving large-scale Lasso computations.
- Tensor extension based on this approach could prove to be an interesting research direction, by focusing on how to express formulations such as SPARAFAC [44], through ADMM and tackle the problem of large-scale tensor (multi-way array) co-clustering efficiently.
- We aim to experiment with iterative platforms, such as Apache Hama [61], with the aim of reducing Hadoop's overhead for setting up each Map-Reduce task.

BIBLIOGRAPHY

[1] http://bits.blogs.nytimes.com/2012/09/07/big-data-in-your-blood

[2] http://www.uslhc.us/files/factsheets/us_computing.pdf

[3] Data-Intensive Text Processing with MapReduce **Jimmy Lin and Chris Dyer**

[4] Designing a multi-petabyte database for LSST. **Jacek Becla, Andrew Hanushevsky, Sergei Nikolaev, Ghaleb Abdulla, Alex Szalay, Maria Nieto-Santisteban, Ani Thakar, and Jim Gray** SLAC Publications SLAC-PUB-12292, Stanford Linear Accelerator Center, May 2006.

[5] http://www.go-globe.com

[6] The Fourth Paradigm: Data-Intensive Scientific Discovery. **Tony Hey, Stewart Tansley, and Kristin Tolle** Microsoft Research, Redmond, Washington, 2009.

[7] Cloud Computing: An Overview. **Mache Creeger** ACM Queue, 7(5) 2009. 13

[8] Definition of Cloud Computing, **Peter Mell and Tim Grance** October 2009. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[9] Mining of Massive Datasets **Anand Rajaraman, Jeff Ullman, Jure Leskovec** http://i.stanford.edu/~ullman/mmds.html

[10] http://en.wikipedia.org/wiki/Netflix_Prize

[11] Survey of Clustering Data Mining Techniques **Pavel Berkhin** Accrue Software, Inc.

[12] http://en.wikipedia.org/wiki/1854_Broad_Street_cholera_outbreak

[13] http://en.wikipedia.org/wiki/Turing_completeness

[14] Data Clustering: A Review **A.K. Jain, M.N. Murty, P.J. Flynn**

[15] Algorithms for Clustering Data **Anil K. Jain and Richard C. Dubes** Prentice-Hall, 1988.

[16] Scalable Clustering of Categorical Data and Applications **Periklis Andritsos** Phd Thesis, University of Toronto, 2004

[17] The hardness of k-means clustering **Sanjoy Dasgupta**

[18] Information-theoretic Co-clustering **I. S. Dhillon, S. Mallela, and D. S. Modha** KDD, 2003

[19] Biclustering of expression data. **ChengY, Church GM** In: Proceedings of the eighth international conference on intelligent systems for molecular biology. 2000. p. 93-103.

[20] Biclustering algorithms for bilogical data analysis: a survey. **Madeira SC, Oliveira AL.**IEEE Transactions on Computational Biology and Bioinformatics 2004;1:24-45.

[21] Co-clustering documents and words using bipartite spectral graph partitioning.**Dhillon IS.** In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining(KDD), August 26-29, 2001, San Francisco, CA, USA

[22] Spectral images and features coclustering with application to content-based image retrieval. **J. Guan, G. Qie, and X. Y. Xue.** IEEE International Workshop on Multimedia Signal Processing (MMSP'05), 2005.

[23] A scalable collaborative filtering framework based on co-clustering. **T. George and S. Merugu** In Proceedings of the 5th IEEE Conference on Data Mining (ICDM'05), pages 625-628, 2005.

[24] A Generalized Maximum Entropy Approach to Bregman Coclustering and Matrix Approximation **Arindam Banerjee, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, Dharmendra S. Modha** Journal of Machine Learning Research 8 (2007)

[25] Approximation Algorithms for Co-Clustering **Aris Anagnostopoulos,Anirban Dasgupta,Ravi Kumar** PODS'08, June 9-12, 2008, Vancouver, BC, Canada.

[26] Biclustering in data mining **Stanislav Busygin , Oleg Prokopyev, , Panos M. Pardalos** Computers & Operations Research 35 (2008) 2964 - 2987

[27] Clustering High-Dimensional Data: A Survey on Subspace Clustering, Pattern-Based Clustering, and Correlation Clustering **Hans-Peter Kriegel, Peer Kroger and Arthur Zimek**

[28] Direct clustering of a data matrix **J.A. Hartigan** Journal of the American Statistical Association, Vol.67, No. 337. (Mar.,1972) pp. 123-129

[29] Information-Theoretic Co-clustering **Dhillon IS, Mallela S, Modha DS** In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining(KDD). August 2003. p. 89-98.

[30] Coclustering of human cancer microarrays using Minimum Sum-Squared Residue coclustering. **Cho H, Dhillon IS.** IEEE/ACM Trans Comput. Biol. Bioinform.

[31] Fully-Automatic Cross-Associations **Chakrabarti, Modha, Papadimitriou, Faloutsos** KDD'04, August 22-25, 2004, Seattle, Washington, USA

[32] Modeling by shortest data description **J. Rissanen** Automatica, vol. 14, pp. 465-471, 1978

[33] MapReduce: Simplified Data Processing on Large Clusters **Jeffrey Dean and Sanjay Ghemawat** Google, Inc.

[34] http://hadoop.apache.org/

[35] http://hbase.apache.org/

[36] Bigtable: A Distributed Storage System for Structured Data **Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber** Google, Inc.

[37] http://hadoop-hbase.blogspot.gr/2011/12/introduction-to-hbase.html

[38] Hadoop: The definitive guide, Third edition **Tom White**

[39] HBase: The definitive guide, First edition **Lars George**

[40] DisCo: Distributed Co-clustering with Map-Reduce, A Case Study Towards Petabyte-Scale End-to-End Mining **Spiros Papadimitriou, Jimeng Sun** ICDM '08 Proceedings of the 2008 Eighth IEEE International Conference on Data Mining

[41] A Framework for Simultaneous Co-clustering and Learning from Complex Data **Meghana Deodhar, Joydeep Ghosh** KDD 07

[42] Parallel Simultaneous Co-clustering and Learning with Map-Reduce **Meghana Deodhar, Clinton Jones and Joydeep Ghosh** GrC 10

[43] Reviewer Profiling Using Sparse Matrix Regression **Evangelos E. Papalexakis, Nicholaos D. Sidiropoulos, Minos Garofalakis** IEEE OEDM 2010 Workshop, held in conjuction with ICDM 2010, Sydney, Australia

[44] Co-clustering as multilinear decomposition with sparse latent factors **Evangelos E. Papalexakis, Nicholaos D. Sidiropoulos** IEEE ICASSP 2011, Prague, Czech Republic

[45] Coclustering - a useful tool for Chemometrics **Rasmus Bro, Evangelos E. Papalexakis, Evrim Acar, Nicholaos D. Sidiropoulos** Journal of Chemometrics, January 2012

[46] Network Anomaly Detection using Co-clustering **Evangelos E. Papalexakis, Alex Beutel, Peter Steenkiste** IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining ASONAM 2012, Istanbul, Turkey

[47] Regression shrinkage and selection via the lasso. **Tibshirani, R.** J. Royal. Statist. Soc B., Vol. 58, No. 1, pages 267-288). (1996).

[48] Distributed Nonnegative Matrix Factorization for Web-Scale Dyadic Data Analysis on MapReduce, **Chao Liu et al.** WWW 2010, ACM

[49] Distributed optimization and statistical learning via the alternating direction method of multipliers **S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein** Foundations and Trends in Machine Learning, Michael Jordan, Editor in Chief, 3(1):1-122, 2011

[50] A Tutorial on Convex Optimization II: Duality and Interior Point Methods **Haitham Hindi**

[51] Convex Optimization **S.P. Boyd and L. Vandenberghe**Cambridge University Press, 2004

[52] Advanced Concepts in Telecommunication Systems, **Athanasios Liavas** Lecture Notes, Spring 2012, ECE, TUC

[53] http://mahout.apache.org/

[54] Local Linear Convergence of ADMM on Quadratic or Linear Programs **Daniel Boley** University of Minnesota

[55] Efficient Distributed Linear Classification Algorithms via the Alternating Direction Method of Multipliers **Caoxie Zhang, Honglak Lee, Kang G. Shin**

[56] On the Optimal Step-size Selection for the Alternating Direction Method of Multipliers **Euhanna Ghadimi, Andre Teixeira, Iman Shames and Mikael Johansson**

[57] http://www.simonlucey.com/lasso-using-admm/

[58] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[59] Category-based selection of effective parameters for intrusion detection. **P. Kabiri and G.R. Zargar.** International Journal of Computer Science and Network Security (IJCSNS), 9(9):181-188, 2009.

[60] Identifying significant Features for Network Forensic Analysis Using Artificial Intelligent Techniques **S. Mukkamala, A. H. Sung** International Journal of Digital Evidence, Winter 2003, Vol. 1, Issue 4.

[61] HAMA: An Efficient Matrix Computation with the MapReduce Framework **Sangwon Seo, Yoon, E.J, Jaehong Kim, Seongwook Jin, Jin-Soo Kim, Seungryoul Maeng**