Design of Algorithms for Big Data Analytics

Raj Bhatnagar(✉)

Department of Electrical Engineering and Computing Systems,
University of Cincinnati, Cincinnati, OH, USA
raj.bhatnagar@uc.edu

Abstract. Processing of high volume and high velocity datasets requires design of algorithms that can exploit the availability of multiple servers configured for asynchronous and simultaneous processing of smaller chunks of large datasets. The Map-Reduce paradigm provides a very effective mechanism for designing efficient algorithms for processing high volume datasets. Sometimes a simple adaptation of a sequential solution of a problem to design Map-Reduce algorithms doesn't draw the full potential of the paradigm. A completely new rethink of the solution from the perspective of the powers of Map-Reduce paradigm can provide very large gains. We present here an example to show that the simple adaptation does not perform as well as a completely new Map-Reduce compatible solution. We do this using the problem of finding all formal concepts from a binary dataset. The problem of handling very high volume data is another important problem and requires newer thinking when designing solutions. We present here an example of the design of a model learning solution from a very high volume monitoring data from a manufacturing environment.

1 Introduction

Dataset sizes in all domains are growing at a pace that is faster than the pace at which the CPU speeds and the hardware performance metrics are growing. Most of the data analytics algorithms have high computational complexity and are rarely bounded within the polynomial time and space limits. Therefore, for analyzing these very large datasets the only viable alternative is to harness the power of multiple processors working in parallel, preferably working on different chunks of the large dataset. Traditional paradigms for parallel processing are too restrictive for the current scales of data. The flexibility in deployment and asynchronous operation of multiple independent servers expected to participate in these computations require an equally flexible and adaptable operating system structure and also a programming paradigm. Map-Reduce is an algorithms design formalism that working within the hadoop kinds of environments is very suitable for processing large datasets. This paradigm can deploy a flexible number of asynchronously working servers without requiring extensive message-passing or requiring shared memory among the processors. Design of data analytics algorithms using the Map-Reduce algorithms requires one to understand the nature, the strengths, and the weaknesses of this programming paradigm.

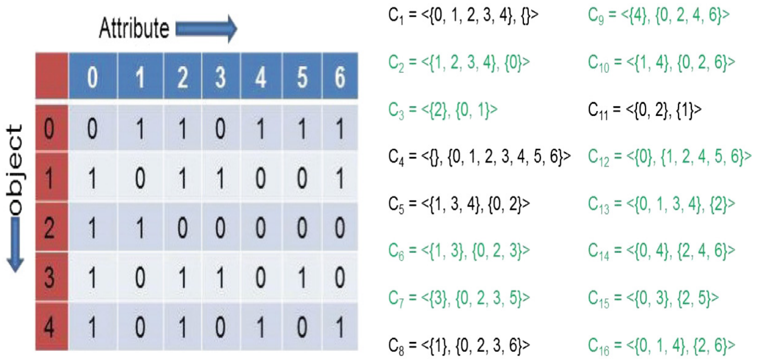
N. Kumar and V. Bhatnagar (Eds.): BDA 2015, LNCS 9498, pp. 101–107, 2015.
DOI: 10.1007/978-3-319-27057-9_7

Most analytics algorithms are designed as iterative algorithms that assume the large original dataset to persist across iterations and use some transient data that stays in dynamic memory. The Map-Reduce paradigm, as typically supported by hadoop type of environments, is not suitable for multi-iteration algorithms.

We illustrate below an example problem, that of finding all formal concepts from a large binary dataset. On single processor systems this problem is typically solved using an iterative formulation that processes the nodes of a Depth-first search tree. We illustrate a completely new formulation that can be performed in a single iteration and is facilitated by the power of Mapper and Reducer Operations. The second example we illustrate relates to a monitored data stream in which very large amounts of data is generated continuously. We need to use this data to predict the health related parameters of the underlying system. A model of the system also needs to be extracted from the same data stream. All of the data does not need to be retained for any long term record but enough information and features must be extracted to enable our model-building and prognostics tasks.

2 Building FCA Lattice

Let us consider the problem of building a lattice of formal concepts from a binary database. This is an important problem from a large number of knowledge extraction perspectives.



Attribute →

object ↓

	0	1	2	3	4	5	6
0	0	1	1	0	1	1	1
1	1	0	1	1	0	0	1
2	1	1	0	0	0	0	0
3	1	0	1	1	0	1	0
4	1	0	1	0	1	0	1

(a) Binary Data Table

C_1 = <{0, 1, 2, 3, 4}, {}>	C_9 = <{4}, {0, 2, 4, 6}>
C_2 = <{1, 2, 3, 4}, {0}>	C_{10} = <{1, 4}, {0, 2, 6}>
C_3 = <{2}, {0, 1}>	C_{11} = <{0, 2}, {1}>
C_4 = <{}, {0, 1, 2, 3, 4, 5, 6}>	C_{12} = <{0}, {1, 2, 4, 5, 6}>
C_5 = <{1, 3, 4}, {0, 2}>	C_{13} = <{0, 1, 3, 4}, {2}>
C_6 = <{1, 3}, {0, 2, 3}>	C_{14} = <{0, 4}, {2, 4, 6}>
C_7 = <{3}, {0, 2, 3, 5}>	C_{15} = <{0, 3}, {2, 5}>
C_8 = <{1}, {0, 2, 3, 6}>	C_{16} = <{0, 1, 4}, {2, 6}>

(b) Concepts in Data Table

Fig. 1. Example binary dataset and its concepts

The binary dataset in Fig. 1a shows a dataset that is typically encountered in many application domains. The objects along the rows may be genes and the columns may correspond to diseases. The objects may also correspond to documents and the columns may correspond to words that occur in these documents. An entry of '1' in the table means that the column entry is related to row object and a '0' entry means that the column entry is not related to the row objects. In many applications binary datasets typically have about 50,000 rows and 1,000 columns. A "concept" existing in such a binary dataset is defined to be a rectangle of only '1' entries that can be formed by arbitrary shuffling of rows and columns of the binary dataset, such that the rectangle is the largest possible in the following sense. A rectangle of '1's corresponding to a "concept" should not be extendable by adding any additional row or column while still meeting the condition of containing only the '1's in the rectangle. These concepts found a database are very meaningful and concise representation of co-occurrence or associational knowledge embedded in the dataset. Figure 1b shows all the concepts embedded in the binary dataset of Fig. 1a.

All the concepts of a binary dataset can also be organized in the form of a lattice as shown in Fig. 2. The parent-child relationships of this lattice are defined using the subset/superset relation between their object sets and between their itemsets. The theory of formal concept analysis [2].

2.1 Inefficient Map-Reduce Based Design

Most of the existing algorithms for finding the concepts are not scalable to very large datasets. A dataset of 5000 rows and 60 columns takes more than twelve hours of CPU time on standard desktop computers. We need scalable algorithms that can process much larger datasets in reasonable time. Power of parallel computing using Map-Reduce paradigm can be employed for designing algorithms that are much more scalable than the existing ones. There are various ways in which the concept-discovery problem can be formulated using the Map-Reduce paradigm and one such formulation has been presented in [3]. This parallel formulation takes the traditional Depth First Search (DFS) based algorithm for finding the concepts, uses the Map-Reduce formulation, and seeks to parallelize the processing of various branches at each node of the corresponding Breadth First Search tree. This formulation is not very efficient because it needs to run as many iterations of Map-Reduce as there are nodes in the DFS/BFS search tree. Multiple iterations of Map-Reduce make the algorithm computationally very expensive. Inherently, the search for concepts remains structured as DFS or BFS-based and therefore, at core, a serial processing of different nodes in the search tree takes place.

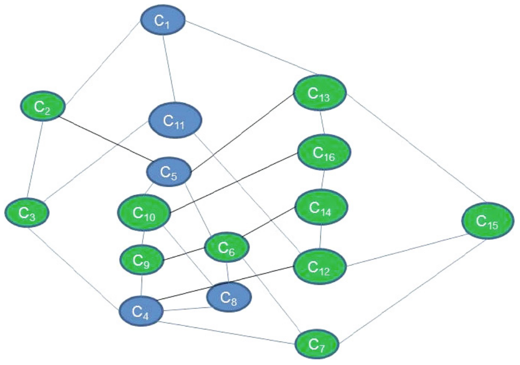


Fig. 2. Example binary dataset and its concepts

Map-Reduce paradigm has been employed to develop faster concept-discovery algorithms. The first such algorithm was presented in [3] and is a close Map-Reduce adaptation of the ClosebyOne algorithm [4]. The authors of [5] present two more algorithms called *MRGanter* and *MRGanter+* that are Map-Reduce adaptations of the basic DFS based algorithm presented by Ganter in [2]. The latter of these implementations is shown to perform the best among all possible algorithms, but the algorithms are run on Twister system that facilitates iterations of Map-Reduce operations. This is facilitated by splitting the data into static and dynamic parts and the static parts consisting the large volumes of the data, are not removed from the servers between the iterations. On the Mushrooms dataset having 125 attributes and 8124 objects the best of these algorithms requires 14 Map-Reduce iterations. On a commonly available hadoop implementations these iterations will become extremely expensive. Our algorithms has abandoned the DFS skeleton for finding the concepts and is using a different approach better suited to the Map-reduce paradigm. This proposed algorithms performs the tasks in less than tenth of the time required by the above mentioned algorithms.

2.2 Efficient Map-Reduce Based Design

We have developed and presented a completely new formulation for the concept discovery problem, using theoretical basis from the FCA theory and adapting the Map-Reduce paradigm for its implementation [1]. We show in [1] theoretically and empirically, that our algorithm is highly efficient when compared to the BFS/DFS based algorithms, and also much faster compared to Map-Reduce formulations presented in [3,5].

One major difference of our approach is that we do not seek to enumerate the entire lattice of concepts for a dataset. Such exhaustive enumeration is always too large to store in memory and takes too long to compute. Our algorithm generates a sufficient set of concepts and stores it. This sufficient set of concepts can then be used to enumerate any other concept in the lattice. The lattice shown in 2 shows in green color the nodes generated by our algorithm as the sufficient set of concepts. The remaining nodes of a lattice can then be generated by simple set union and intersection operations on the concepts in the sufficient set. This example shows a large number of concepts in the sufficient set, but for very large datasets typically the sufficient set of concepts is a very small fraction of the number of all the concepts in the lattice. The process for generating the sufficient set of is driven in part by a Map-Reduce formulation. This formulation looks at the problem of finding the rows or columns corresponding to a set of columns or rows in one single operation. That is, given a set of items, we should find all those rows for which all the item-columns contain '1's. And then, given a set of row ids, we should find all those columns such that the given rows have all '1's in them. For very large datasets these two mapping operations cannot be easily done as single step macro operations. But map-Reduce enables us to achieve this and that helps us design a very efficient algorithm. The process of merging the concept's components that appear in different regions of rows or columns can only be partially merged within the map-reduce based formulation. The final merging of concept components is much less complex problem because the problem size has been reduced from that of the whole big original data to only the size of the sufficient set of concepts. This last merging of concept components is done on a single processor machine by our algorithm Table 1.

Table 1. Time (Seconds) enumerating sufficient set and of complete lattice

Dataset # of concepts	Mushroom 219010	Anon-Web 129009	CensusIncome 86532
NextClosure *Sequential*	618	14671	18230
CloseByOne *Sequential*	2543	656	7465
MRGanter map-red	20269 (5 nodes)	20110 (3 nodes)	9654 (11 nodes)
MRCbO map-red	241 (11 nodes)	693 (11 nodes)	803 (11 nodes)
MRGanter+ map-red	198 (9 nodes)	496 (9 nodes)	358 (9 nodes)
Our algorithm suffic. set only map-red	42 (10 nodes)	26 (10 nodes)	69 (10 nodes)
Our algorithm enumeration on single processor	OutOfMemory	361	OutOfMemory
Number of concepts in sufficient set	117	365	147

Exhaustive enumeration from the sufficient set may then be performed by a non-parallelized algorithm on a single processor. It is efficient to create and store only the sufficient set of concepts. This set implicitly contains information about all the other concepts in the lattice but we don't need to keep them in an enumerated form.

One primary idea underlying a Map-Reduce formulation of an algorithm is to split the table horizontally into a number of partitions. Each partition is then processed on an independent server by a *Mapper* function. The outputs from the mapper functions are gathered at a central location, sorted, and then split again among a number of independent servers that run the *Reducer* function. The outputs of the reducer function from different servers are then collected at a central location and form the output of a Map-Reduce iteration. One last step of the algorithm of merging of concepts' components is then performed on a single processor system.

A comparison of two different formulations as reported in [1] can be seen in the table here.

3 High Velocity Datasets

There are many situations in which a large amount of data is being generated as a result of monitoring some physical or social system. Storing all the data is neither useful nor feasible. However, we want to learn from this high volume stream of data all the essential details about the monitored system and discard the data. There is some buffer available for storing all monitored data for some limited window. We consider the case of a spindle from a manufacturing domain as shown in Fig. 3.

The problem to be addressed is that the bearings of these spindles wear out over time and crash during a manufacturing operation. Such disruptive crashes cause a lot of loss by first damaging the piece being manufactured and second by holding up the pipeline of operations on the shop floor. It is therefore very valuable to predict the remaining life of the bearings installed within a spindle. We installed an angular acceleration sensor on the spindle and this continuously monitors the acceleration and stores the data. In order to detect the vibrations that indicate deteriorating health we need to sample the acceleration at a rate of 25,000 samples per second. In addition to predicting the remaining life of the bearings we need to find out the characteristics of the operating mode in which the spindle is running at any time. We would also like to detect any slow of rapid drifts in the operating modes of the bearings.

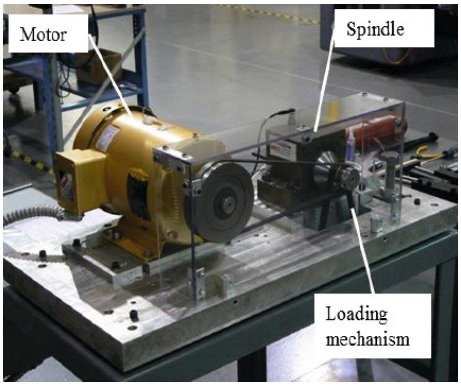


Fig. 3. Spindle testbed

The above can be achieved by learning a model of the current mode of operation and then using this model to perform the above two tasks. The model of current operating mode can be compared to some pre-existing models in a library to characterize its current mode. However, it turns out that there is so much uncertainty and continuous evolution in the models of operation that no one model come very close to any one other model learned in the past. Predictions based on the closest models from the past did not work very well. Our models were constructed using the frequency domain features derived from the sensor data.

We used these features to build regression models to predict the remaining life of a spindle's bearings. The plots in Fig. 4 show the results of predicting the health of the bearings based on the features derived from a 72 h window immediately preceding the time at which the prediction is made.

In this plot the light blue points show the actual time to failure (TTF) for the spindle at any point of time. The three other curves show the prediction results of three different types of regression models constructed by using different types of features. All these models seem to have an acceptable performance in predicting

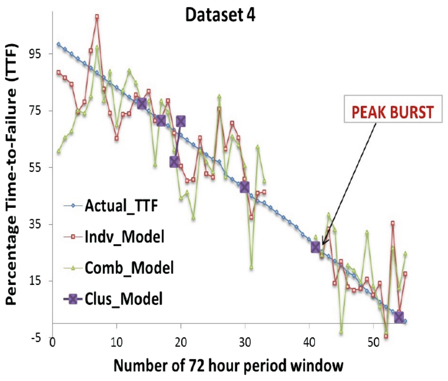


Fig. 4. Prediction results based on 72 h window

the remaining life of the bearings. However, the biggest drawback is that we need to store data for the preceding 72 h and use it to generate features to make these predictions. This is very long time window given that we need to collect 25,000 samples per second, and also generate Fourier transform signatures from 0.5 s time slices as the streaming data comes in. Any window size smaller than 72 h does not perform well enough to predict the remaining life of a spindle. From practical shop-floor perspective it is impractical to leave a machine unused for 72 h just so that it can be monitored for remaining life prediction. The challenge of using and exploiting the very high volume data still remains. Solutions that we are considering including monitoring the spindle for few minutes every couple of hours. Such data may not be able to generate the rich and informative features that help predict the remaining life of the bearings.

4 Conclusions

In the discussion above we have presented some challenges that are faced while designing systems for analytics of Big Data. We have presented the example of finding formal concepts from a large binary dataset and have shown that designing Map-reduce compatible algorithms must be attempted and this can result in significantly enhanced performance. We have also presented an example of a high volume data stream situation and described the challenges that need to be faced in exploiting the data stream effectively.

References

1. Bhatnagar, R., Kumar, L.: An efficient map-reduce algorithm for computing formal concepts from binary data. In: 2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara (to appear, 2015)
2. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Heidelberg (2012)
3. Krajca, P., Vychodil, V.: Distributed algorithm for computing formal concepts using map-reduce framework. In: Adams, N.M., Robardet, C., Siebes, A., Boulicaut, J.-F. (eds.) IDA 2009. LNCS, vol. 5772, pp. 333–344. Springer, Heidelberg (2009)
4. Kuznetsov, S.O.: A fast algorithm for computing all intersections of objects in a finite semi-lattice. Autom. Documentation Math. Linguist. **27**(5), 11–21 (1993)
5. Xu, B., de Fréin, R., Robson, E., Ó Foghlú, M.: Distributed formal concept analysis algorithms based on an iterative mapreduce framework. In: Domenach, F., Ignatov, D.I., Poelmans, J. (eds.) ICFCA 2012. LNCS, vol. 7278, pp. 292–308. Springer, Heidelberg (2012)