VOL. 11, NO. 5, MARCH 2016 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

ALGORITHMS USING MAP REDUCE-A SURVEY

M. Karpagam, N. Saranya and M. Sujatha
School of Computing, SASTRA University, Thanjavur, Tamilnadu, India
E-Mail:karps@it.sastra.edu

ABSTRACT

Despite increasing data volumes much faster than compute power. This growth demands new strategies for processing and analyzing information. Organizations are determining that significant forecasting can be through sorting and analyze Big Data. Ever since a large amount of data is "amorphous", it should be structured in a manner which is appropriate for mining and succeeding analysis. Hadoop helps in structuring Big Data, and solves with solution for analytics purposes. Hadoop exploits a technique called MapReduce to carry out this extensive analysis rapidly. In this paper, we provide an extensive survey of MapReduce the popular open-source implementation Hadoop which can be obtained, while highlighting the various algorithms and computing selection by Map Reduce. We conclude the paper with a critical analysis of challenges that have not yet been fully met.

Keywords: bigdata, MapReduce, data storage, global pulse, data analysis algorithm.

INTRODUCTION

Recent data-mining applications, often called "big-data" analysis, involve us to manage massive amounts of data quickly. In many of these applications, the data is extremely regular, and there is abundant opportunity to exploit parallelism. Vital examples are:

- The ranking of Web pages by importance, which involves an iterated matrix-vector multiplication where the dimension is many billions.
- Searches in "friends" networks at social-networking sites, which involve graphs with hundreds of millions of nodes and many billions of edges.

Central to the new software stack is a programming system called MapReduce. Implementation of MapReduce enables universal computation on large-scale data to be performed on computing clusters efficiently and to tolerate failures during the computation [2].MapReduce systems are evolving and extending rapidly. Today, it is common for MapReduce programs to be created from still higher-level programming.

Our last topic for this chapter is the design of good MapReduce algorithms, a subject that often differs significantly from the matter of designing good parallel algorithms to be run on a supercomputer. When designing MapReduce algorithms, we often find that the greatest cost is in the communication. We thus investigate communication cost and what it tells us about the most efficient MapReduce algorithms. For several common applications of MapReduce we are able to give families of algorithms that optimally trade the communication cost against the degree of parallelism.

Organization of paper

The rest of this paper is organized as follows: In Section 2, we present the motivation for Big Data Analytics, and discuss potential applications and scenarios. In Section 3, we elaborate MapReduceExecution with example. Next, in Section 4 we discuss various algorithms used in MapReduce.We provide a discussion on Challenges in MapReduce in Section 5. Finally, conclusions and directions for future research are identified in Section 6.

MOTIVATION FOR BIG DATA ANALYTICS, RESEARCH TRENDS AND CHALLENGES IN HANDLING BIG DATA

Motivation

Big Data market is constantly increasing each year. In March 2012, The White House announced a national "Big Data Initiative" that consisted of six Federal departments and agencies committing more than $200 million to big data research projects [3].

Global Pulse - an innovative lab that uses big data to improve the life in developing countries. In recent competitive & complex business world the various aspects of business are blend together [12]. Changing one facet has direct or indirect effect on the other facet. Inside an organization, this complication makes it tricky for industry leaders to rely exclusively on intuition to make conclusion. They are in need to rely on data (structured, unstructured or semi-structured) to support their decisions. The tools which are present today don't allow themselves for sophisticated data analysis at the extent that the user involves. Tools like SAS, R, and Matlab sustain the influential analysis but they are not designed for the immense datasets and neither DBMS nor Map Reduce can handle the data that are arrived at high rates. To fill this gap the "Big Data" arrives. Big Data provides the organization a novel path to analyze and visualize their data in an effective manner. For example:

Industry: Customer Feedback, trends etc.

Health care: Health care organizations are influencing big data technology to avail all the information

concerning patient to get more inclusive view into care synchronization, health management & outcome. Usage of big data helps to construct a sustainable healthcare system and enlarge the access to healthcare.

Energy and utility: Big data is the key to deploy condition based maintenance program and progress forecasting and scheduling of resources.

Challenges in handling Big Data

- Heterogeneous Data Source
- Unstructured Nature of Data Sources
- High Scalability
- Timeliness
- Privacy
- Data Integration

Role of MapReduce in Big Data

In the Big Data community, MapReduce is one of the key facilitating approach for the convention of incessant rising needs on computing resources that is enforced by enormous data sets. Due to this parallel and distributed execution over a large number of computing nodes is achieved [4].

MapReduce overview

MapReduce is a style of computing that has been implemented in several systems, including Google's internal implementation (simply called MapReduce) and the popular open-source implementation Hadoop which can be obtained, along with the HDFS file system from the Apache Foundation. You can use an implementation of MapReduce to manage many large-scale computations in a way that is tolerant of hardware faults. To support this you need to write the two main tasks namely map and reduce which manages the parallel execution of large task [5].

MapReduce computation executes as follows [6]:

1. Each Map tasks each is given one or more chunks from a distributed file system. These Map tasks turn the chunk into a sequence of key-value pairs. The way key-value pairs are produced from the input data is determined by the code written by the user for the Map function.

2. The key-value pairs from each Map task are collected by a master controller and sorted by key. The keys are divided among all the Reduce tasks, so all key-value pairs with the same key wind up at the same Reduce task.

3. The Reduce tasks work on one key at a time, and combine all the values associated with that key in some way. The manner of combination of values is determined by the code written by the user for the Reduce function.

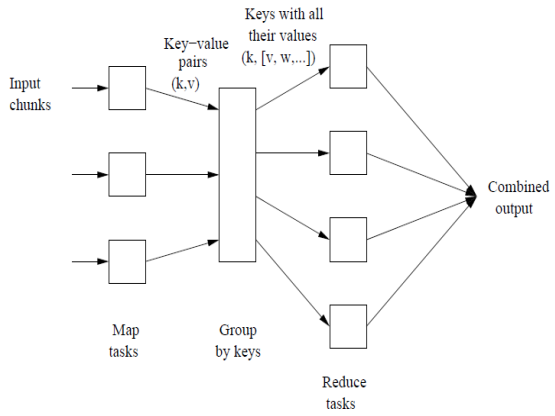

Figure-1. Schematic of a MapReduce computation.

Map tasks

We view input files for a Map task as consisting of elements, which can be any type: a tuple or a document, for example. A chunk is a collection of elements, and no element is stored across two chunks. Inputs to Map tasks and outcomes of Reduce tasks are of the key-value-pair form [7], but normally the input elements keys are not pertinent and we shall tend to ignore them. Insisting on this form for inputs and outputs is motivated by the desire to allow composition of several MapReduce processes. The Map function takes an input element as its argument and generates either zero or more key-value pairs. The types of keys and values are each arbitrary. Further, keys are not "keys" in the usual sense; they do not have to be unique. Rather a Map task can produce several key-value pairs of same key, even from the same element.

Example 1: We shall illustrate a MapReduce computation with what has become the standard example application: To count the number of occurrences for each word in a collection of documents. In this example, the input file is a repository of documents, and each document is an element. The Map function for this example uses keys that are of type String (the words) and values that are integers. The Map task reads a document and breaks it into its sequence of words $w_1, w_2, \ldots, w_n$. It then break a sequence of key-value pairs whose value is always 1. That is, the output of the Map task for this document is the sequence of key-value pairs:

$(w_1, 1), (w_2, 1), \ldots, (w_n, 1)$

Note that a single Map task will typically process many documents - all the documents in one or more chunks. Thus, its output will be more than the sequence for the one documented suggested above. Note also that if a word w appears m times among all the documents assigned to that process, then there will be m key-value pairs (w, 1) among its output. An option, which we discuss in Section 2.2.4, is to combine these m pairs into a single

pair (w, m), but we can only do that because, as we shall see, the Reduce tasks apply an associative and commutative operation, addition, to the values.

Grouping by key

As soon as the Map tasks have all completed successfully, the key-value pairs are grouped by key, and the values associated with each key are formed into a list of values. The grouping performed by the system, regardless of what the Map and Reduce tasks do. The master controller process knows how many Reduce tasks there will be, say r such tasks. The user typically tells the Map Reduce system what r should be. Then the master controller picks a hash function that applies to keys and produces a bucket number from 0 to r - 1. Each key that is output by a Map task is hashed and its key-value pair is put in one of r local files. Each file is destined for one of the Reduce tasks.

To perform the grouping by key and distribution to the Reduce tasks, the master controller merges the files from each Map task that are destined for a particular Reduce task and feeds the merged file to that process as a sequence of key-list-of-value pairs. That is, for each key k, the input to the Reduce task that handles key k is a pair of the form (k, [v_1, v_2, v_n]), where (k, v_1), (k, v_2), . . . , (k, v_n) are all the key-value pairs with key k coming from all the Map tasks.

Reduce tasks

The Reduce function's argument consisting of key-value pair. The outcome of the Reduce function is a series of zero or more key-value pairs. These key-value pairs can be of a type different from those sent from Map to Reduce tasks, but often they are the same type. A Reduce task receives one or more keys and their associated value lists. That is, a Reduce task executes one or more reducers. The outputs from all the Reduce tasks are merged into a single file. Reducers may be partitioned among a smaller number of Reduce tasks is by hashing the keys and associating each Reduce task with one of the buckets of the hash function.

Example 2: Let us continue with the word-count example of Example 1.

The Reduce function simply adds up all the values. The output of a reducer consists of the word and the sum. Thus, the result of all the Reduce tasks is a series of (w,m) pairs, where w is a word that appears at least once among all the input documents and m is the total number of occurrences of w among all those documents.

Combiners

Sometimes, a Reduce function is associative and commutative. That is, the values to be combined can be combined in any order, with the same result. The addition performed in Example 3.2 is an example of an associative and commutative operation. It doesn't matter how we group a list of numbers v_1, v_2, . . . ,v_n; the sum will be the same. When the Reduce function is associative and commutative, we can push some of what the reducers do to the Map tasks. For example, instead of the Map tasks in Example 1 producing many pairs (w, 1), (w, 1), . . ., we could apply the Reduce function within the Map task, before the output of the Map tasks is subject to grouping and aggregation. These key-value pairs would thus be replaced by one pair with key w and value equal to the sum of all the 1's in all those pairs. That is, the pairs with key w generated by a single Map task would be replaced by a pair (w,m), where m is the number of times that w appears among the documents handled by this Map task. Note that it is still necessary to do grouping and aggregation and to pass the result to the Reduce tasks, since there will typically be one key-value pair with key w coming from each of the Map tasks.

Details of map reduce execution

The Figure-1 offers an outline of how processes, tasks, and files interact. Taking advantage of a library provided by a Map Reduce system such as Hadoop, the user program splits a Master controller process and some number of Worker processes at different compute nodes. Normally, a Worker handles either Map tasks (a Map worker) or Reduce tasks (a Reduce worker), but not both [8].

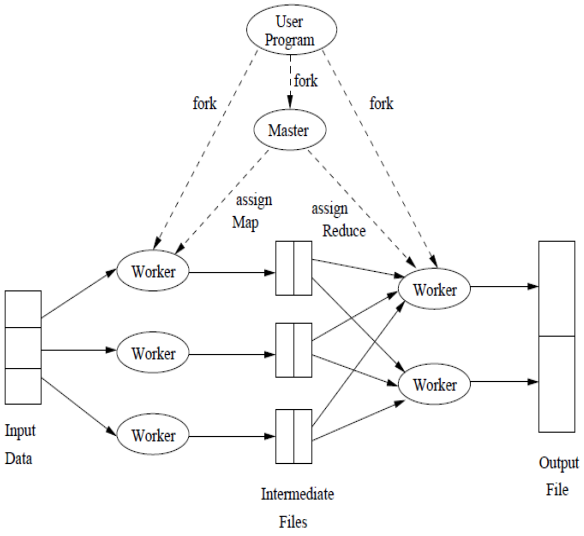

Figure-2. Overview of the execution of a MapReduce program.

The Master has many responsibilities. One is to create some number of Map tasks and Reduce tasks, these numbers being selected by the user program. These tasks will be allocate to Worker processes by the Master. It is reasonable to create one Map task for every chunk of the input file(s), but we may wish to create fewer Reduce tasks. The reason for limiting the number of Reduce tasks is that it is necessary for each Map task to create an

intermediate file for each Reduce task, and if there are too many Reduce tasks the number of intermediate files explodes.

The Master keeps analyzing the status of each Map and Reduce task frequently. A Worker process informs the Master after completing a task, and a new task is scheduled by the Master for that Worker process.

Each Map task is assigned one or more chunks of the input file(s) and executes on it. The Map task creates a file for each Reduce task on the local disk of the Worker that executes the Map task. The Master is informed of the location and sizes of each of these files, and the Reduce task for which each is destined. When a Reduce task is assigned by the Master to a Worker process, that task is given all the files that form its input.

The Reduce task executes code written by the user and writes its output to a file that is part of the surrounding distributed file system.

Algorithms using Map Reduce

The original purpose for which the Google implementation of MapReduce was created was to execute very large matrix-vector multiplications as are needed in the calculation of Page Rank. We shall see that matrix-vector and matrix-matrix calculations fit nicely into the MapReduce style of computing.

Matrix -vector multiplication by Map Reduce

Suppose we have an n×n matrix M, whose element in rows i and column j will be denoted m_{ij} . Suppose we also have a vector v of length n, whose j^{th}element is v_j. Then the matrix-vector product is the vector x of length n, whose i^{th}element x_i is given by

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

If n = 100, we do not want to use a DFS or MapReduce for this calculation. But this sort of calculation is at the heart of the ranking of Web pages that goes on at search engines, and there, n is in the tens of billions. Let us first assume that n is large, but not so large that vector v cannot fit in main memory and thus be available to every Map task.

The matrix M and the vector v each will be stored in a file of the DFS. We assume that the row-column coordinates of each matrix element will be discoverable, either from its position in the file, or because it is stored with explicit coordinates, as a triple (i, j,m_{ij}). We also assume the position of element vj in the vector v will be discoverable in the analogous way.

The map function: The Map function is written to apply to one element of M. However, if v is not already read into main memory at the compute node executing a Map task, then v is first read, in its entirety, and subsequently will be available to all applications of the Map function performed at this Map task.

Each Map task will operate on a chunk of the matrix M. From each matrix element mij it produces the key-value pair ($i,m_{ij}v_j$). Thus, all terms of the sum that make up the component xi of the matrix-vector product will get the same key, i.

The reduce function: The Reduce function simply sums all the values associated with a given key i. The result will be a pair (i, xi).

Vector v cannot fit in main memory

However, it is possible for the vector v which is so large to make it fit in its main memory. It is not required that v fit in RAM at a compute node, but if it does not then there will be a very large number of disk accesses as we move pieces of the vector into main memory to multiply components by elements of the matrix. Thus, as an alternative, split the matrix into vertical stripes of equivalent width and divide the vector into an equal number of horizontal stripes, of the same height. Our goal is to use enough stripes so that the portion of the vector in one stripe fits conveniently into RAM at a compute node. Figure-2.4 suggests what the partition looks like if the matrix and vector are each divided into five stripes.

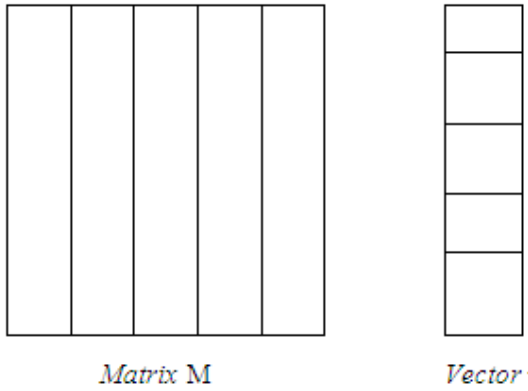

Figure-3. Division of a matrix and vector into five stripes.

The i^{th}stripe of the matrix multiplies only components from the i^{th} stripe of the vector. Thus, we can divide the matrix into one file for each stripe, and do the same for the vector. Every Map task is allocated a chunk from one of the stripes of the matrix and gets the entire corresponding stripe of the vector. The Map and Reduce tasks can then act exactly as was described above for the case where Map tasks get the entire vector. We shall take up matrix-vector multiplication using MapReduce. There, because of the particular application (Page Rank calculation), we have an additional constraint that the result vector should be partitioned in the same way as the input vector, so the output may become the input for another iteration of the matrix-vector multiplication. We

shall see there that the best strategy involves partitioning the matrix M into square blocks, rather than stripes.

Relational- algebra operations

There are a number of operations on large-scale data that are used in database queries. Many traditional database applications involve retrieval of small amounts of data, even though the database itself may be large. For example, a query may ask for the bank balance of one particular account. Such queries are not useful applications of MapReduce. However, there are many operations on data that can be described easily in terms of the common database-query primitives, even if the queries themselves are not executed within a database management system. Thus, a good starting point for exploring applications of MapReduce is by considering the standard operations on relations. We assume you are familiar with database systems, the query language SQL, and the relational model, but to review, a relation is a table with column headers called attributes. Rows of the relation are called tuples. The set of attributes of a relation is called its schema. We often write an expression like R (A_1,A_2, . . . ,A_n) to say that the relation name is R and its attributes are A_1, A_2, . . . ,A_n.

From	To
url1	url2
url1	url3
url2	url3
....	....

Figure-4. Relational links.

In Figure-4 we see part of the relation Links that describes the structure of the Web. There are two attributes, from and to. A row, or tuple, of the relation is a pair of URL's, such that there is at least one link from the first URL to the second. For instance, the first row of Figure-2.5 is the pair (url1, url2) that says the Web page url1 has a link to page url2. While we have shown only four tuples, the real relation of the Web, or the portion of it that would be stored by a typical search engine, has billions of tuples.A relation, however large, can be stored as a file in a distributed file system. The elements of this file are the tuples of the relation.

Imagine that a social-networking site has a relation

Friends (User, Friend)

This relation has tuples that are pairs (a, b) such that b is a friend of a. The site might want to develop statistics about the number of friends members have. Their first step would be to compute a count of the number of friends of each user. This operation can be done by grouping and aggregation, specifically

$\gamma_{User,\ COUNT(Friend)}$(Friends)

This operation groups all the tuples by the value in their first component, so there is one group for each user. Then, for each group the count of the number of friends of that user is made. The result will be one tuple for each group, and a typical tuple would look like (Sally, 300), if user "Sally" has 300 friends.

Computing selections by Map Reduce

Selections really do not need the full power of MapReduce. They can be done most conveniently in the map portion alone, although they could also be done in the reduce portion alone. Here is a Map Reduce implementation of selection σC(R).

The map function: For each tuple t in R, test if it satisfies C. If so, produce the key-value pair (t, t). That is, both key- value are t.

The reduce function: The Reduce function is the identity. It simply passes each key-value pair to the output.

The output is not exactly a relation, because it has key-value pairs. However, a relation can be obtained by using only the value components (or only the key components) of the output.

Computing projections by MapReduce

Projection is performed similarly to selection, because projection may cause the same tuple to appear several times, the Reduce function must eliminate duplicates. We may compute π_S(R) as follows.

The map function: For each tuple t in R, construct a tuple t′ by eliminating from t those components whose attributes are not in S. Output the key-value pair (t′, t′).

The reduce function: For each key t′ produced by any of the Map tasks, there will be one or more key-value pairs (t′, t′). The Reduce function turns (t′, [t′, t′, . . . , t′]) into (t′, t′), so it produces exactly one pair (t′, t′) for this key t′.

The Reduce operation is duplicate elimination. This operation is associative and commutative, so a combiner associated with each Map task can eliminate whatever duplicates are produced locally. However, the Reduce tasks are still needed to eliminate two identical tuples coming from different Map tasks.

Union, intersection and difference by Map Reduce

First, consider the union of two relations. Suppose relations R and S have the same schema. Map tasks will be assigned chunks from either R or S; it doesn't matter which. The Map tasks don't really do anything except pass their input tuples as key-value pairs to the Reduce tasks. The latter need only eliminate duplicates as for projection

The map function: Turn each input tuple t into a key-value pair (t, t).

The reduce function: Associated with each key t there will be either one or two values. Produce output (t, t) in either case.

To compute **the intersection**, we can use the same Map function. However, the Reduce function must produce a tuple only if both relations have the tuple. If the key t has a list of two values [t, t] associated with it, then the Reduce task for t should produce (t, t). However, if the value-list associated with key t is just [t], then one of R and S is missing t, so we don't want to produce a tuple for the intersection.

The map function: Turn each tuple t into a key-value pair (t, t).

The reduce function: If key t has value list [t, t], then produce (t, t). Otherwise, produce nothing.

The difference R - S requires a bit more thought. The only way a tuple t can appear in the output is if it is in R but not in S. The Map function can pass tuples from R and S through, but must inform the Reduce function whether the tuple came from R or S. We shall thus use the relation as the value associated with the key t. Here is a specification for the two functions.

The map function: For a tuple t in R, produce key-value pair (t,R), and for a tuple t in S, produce key-value pair (t, S). Note that the intent is that the value is the name of R or S (or better, a single bit indicating whether the relation is R or S), not the entire relation.

The reduce function: For each key t, if the associated value list is [R], then produce (t, t). Otherwise, produce nothing.

Computing natural join by Map Reduce

The idea behind implementing natural join via MapReduce can be seen if we look at the specific case of joining R(A,B) with S(B,C). We must find tuples that agree on their B components, that is the second component from tuples of R and the first component of tuples of S. We shall use the B-value of tuples from either relation as the key. The value will be the other component and the name of the relation, so the Reduce function can know where each tuple came from.

The map function: For each tuple (a, b) of R, produce the key-value pair b, (R, a). For each tuple (b, c) of S, produce the key-value pair b, (S, c).

The reduce function: Each key value b will be associated with a list of pairs that are either of the form (R, a) or (S, c). Construct all pairs consisting of one with first component R and the other with first component S, say (R, a) and (S, c). The output from this key and value list is a series of key-value pairs. The key is irrelevant. Each value is one of the triples (a, b, c) such that (R, a) and (S, c) are on the input list of values.

The same algorithm works if the relations have more than two attributes. You can think of A as representing all those attributes in the schema of R but not S. B represents the attributes in both schemas, and C represents attributes only in the schema of S. The key for a tuple of R or S is the list of values in all the attributes that are in the schemas of both R and S. The value for a tuple of R is the name R together with the values of all the attributes belonging to R but not to S, and the value for a tuple of S is the name S together with the values of the attributes belonging to S but not R.

The Reduce function looks at all the key-value pairs with a given key and combines those values from R with those values of S in all possible ways. From each pairing, the tuple produced has the values from R, the key values, and the values from S.

Grouping andaggregation by Map Reduce

The minimal example of grouping and aggregation, where there is one grouping attribute and one aggregation. Let R(A,B,C) be a relation to which we apply the operator $\gamma_{A,\theta(B)}(R)$. Map will perform the grouping, while Reduce does the aggregation.

The map function: For each tuple (a, b, c) produce the key-value pair (a, b).

The reduce function: Each key a represents a group. Apply the aggregation operator θ to the list [b1, b2, . . . ,bn] of B-values associated with key a. The output is the pair (a, x), where x is the result of applying θ to the list. For example, if θ is SUM, then $x = b_1 + b_2 + \cdots + b_n$, and if θ is MAX, then x is the largest of b1, b2, . . . , bn.

If there are several grouping attributes, then the key is the list of the values of a tuple for all these attributes. If there is more than one aggregation [9], then the Reduce function applies each of them to the list of values associated with a given key and produces a tuple consisting of the key, including components for all grouping attributes if there is more than one, followed by the results of each of the aggregations.

Matrix multiplication

If M is a matrix with element m_{ij} in row i and column j, and N is a matrix with element n_{jk} in row j and column k, then the product P = MN is the matrix P with element p_{ik} in row i and column k, where [10]

$p_{ik}=\sum m_{ij}n_{jk}$

It is required that the number of columns of M equals the number of rows of N, so the sum over j makes sense.

We can think of a matrix as a relation with three attributes: the row number, the column number, and the value in that row and column. Thus, we could view matrix M as a relation M(I, J, V), with tuples (i,j,m_{ij}), and we could view matrix N as a relation N(J,K,W), with tuples (j, k, n_{jk}). As large matrices are often sparse (mostly 0's), and since we can omit the tuples for matrix elements that are 0, this relational representation is often a very good one for a large matrix. However, it is possible that i, j, and k are implicit in the position of a matrix element in the file that

represents it, rather than written explicitly with the element itself. In that case, the Map function will have to be designed to construct the I, J, and K components of tuples from the position of the data.

The product MN is almost a natural join followed by grouping and aggregation. That is, the natural join of M(I, J, V) and N(J,K,W), having only attribute J in common, would produce tuples (i, j, k, v,w) from each tuple (i, j, v) in M and tuple (j, k,w) in N. This five-component tuple represents the pair of matrix elements (m_{ij}, n_{jk}). What we want instead is the product of these elements, that is, the four-component tuple (i, j, k, v × w), because that represents the product $m_{ij}n_{jk}$. Once we have this relation as the result of one Map Reduce operation, we can perform grouping and aggregation, with I and K as the grouping attributes and the sum of V ×W as the aggregation. That is, we can implement matrix multiplication as the cascade of two Map Reduce operations, as follows. First:

The map function: For each matrix element m_{ij}, produce the key value pair j, (M, i,m_{ij}). Likewise, for each matrix element n_{jk}, produce the key value pair j, (N, k, n_{jk}) [11]. Note that M and N in the values are not the matrices themselves. Rather they are names of the matrices or (as we mentioned for the similar Map function used for natural join) better, a bit indicating whether the element comes from M or N.

The reduce function: For each key j, examine its list of associated values. For each value that comes from M, say (M, i, m_{ij}), and each value that comes from N, say (N, k, n_{jk}), produce a key-value pair with key equal to (i, k) and value equal to the product of these elements, $m_{ij}n_{jk}$.Now, we perform a grouping and aggregation by another MapReduce operation.

The map function: This function is just the identity. That is, for every input element with key (i, k) and value v, produce exactly this key-value pair.

The reduce function: For each key (i, k), produce the sum of values allied with this key. The result is a pair (i, k), v, where v is the value of the element in row i and column k of the matrix P = MN.

Matrix multiplication with one Map Reduce step

There often is more than one way to use Map Reduce to solve a problem. You may wish to use only a single Map Reduce pass to perform matrix multiplication P = MN. It is possible to do so if we put more work into the two functions. Start by using the Map function to create the sets of matrix elements that are needed to compute each element of the answer P. Notice that an element of M or N contributes to many elements of the result, so one input element will be turned into many key-value pairs. The keys will be pairs (i, k), where i is a row of M and k is a column of N. Here is a synopsis of the Map and Reduce functions.

The map function: For each element mij of M, produce all the key-value pairs (i, k), (M, j,m_{ij}) for k = 1, 2, . . ., up to the number of columns of N. Similarly, for each element n_{jk} of N, produce all the key-value pairs (i, k), (N, j, n_{jk}) for i = 1, 2, . . ., up to the number of rows of M. As before and N are really bits to tell which of the two relations a value comes from.

The reduce function: Each key (i, k) will have an associated list with all the values (M, j,m_{ij}) and (N, j, n_{jk}), for all possible values of j [11]. The Reduce function needs to connect the two values on the list that have the same value of j, for each j. An easy way to do this step is to sort by j the values that begin with M and sort by j the values that begin with N, in separate lists. The jth values on each list must have their third components, mij and n_{jk} extracted and multiplied. Then, these products are summed and the result is paired with (i, k) in the output of the Reduce function.If a row of the matrix M or a column of the matrix N is so large that it will not fit in main memory, then the Reduce tasks will be forced to use an external sort to order the values associated with a given key(i, k). However, in that case, the matrices themselves are so large, perhaps 1020 elements, that it is unlikely we would attempt this calculation if the matrices were dense. If they are sparse, then we would expect many fewer values to be associated with any one key, and it would be feasible to do the sum of products in main memory.

Map Reduce challenges

The recognized Map Reduce challenges are grouped into four main categories corresponding to Big Data tasks types: data storage, analytics, online processing, security and privacy is presented in Table-1 [1].

Table-1. Overview of the Map Reduce challenges.

	Major challenges	Solution approaches
Data storage	Schema-free, index-free	In-database Map Reduce
		NoSQL stores - Map Reduce with various indexing approaches
	Lack of standardized SQL-like language	Apache Hive - SQL on top of Hadoop
		NoSQL stores: proprietary SQL-like languages (Cassandra, MongoDB) or Hive (HBase)
Analytics	Scaling complex linear algebra	Use computationally less expensive, though less accurate, algebra
	Interactive analysis	Map interactive query processing techniques for handling small data, to MapReduce
	Iterative algorithms	Extensions of MapReduce implementation such as Twister and HaLoop
	Statistical challenges for learning	Data pre-processing using MapReduce
Online processing	Performance /Latency issues	Direct communication between phases and jobs
	Programming model	Alternative models, such as MapUpdate and Twitter's Storm
Privacy and security	Auditing	Trusted third party monitoring, security analytics
	Access control	Optimized access control approach with semantic understanding
	Privacy	Privacy policy enforcement with security to prevent information leakage

Current business analytics

According to the Business Strategy and BI Research group, data volumes are doubling every year [13]:

- 42.6 percent of respondents are keeping more than three years of data for analytical purposes.
- Novel sources are promising at massive volumes, in different industries, such as utilities.
- 80 % of data is unstructured and not effectively used in the organization.

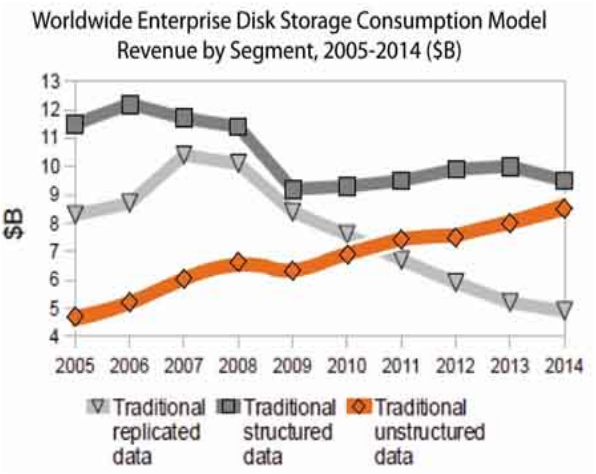

Figure-5. Recent business analysis.

To keep pace with the need to ingest, store and process vast amounts of data, both computational and storage solutions have had to evolve leading to:

- The Emergence of new programming frameworks to enable distributed computing on large data sets (for example, MapReduce).

- New data storage techniques (for example, file systems on commodity hardware, like the Hadoop file system, or HDFS) for structured and unstructured data.

Most of the storage models already exist to support enterprise- class needs. Reliable, flexible distributed data grids have been proven to scale to very large data sizes (petabytes) and are now being offered at an affordable cost.

CONCLUSIONS

Map Reduce is a programming paradigm that allows developing applications which process enormous amounts of data in parallel across a distributed cluster of processors or stand-alone computers. This paper contains various basic algorithms that are fed into Map Reduce for performing sorting, searching and analyzing vast amount of data, along with workflow and challenges of Map Reduce.

REFERENCES

[1] K. Grolinger, M. Hayes, W. Higashino, A. L'Heureux, D. S. Allison, M. A. M. Capretz. Challenges for MapReduce in Big Data.To appear in the Proc. of the. IEEE 10th World Congress on Services (SERVICES 2014), June 27-July 2, 1014, Alaska, USA.

[2] HabibWafaa M. A., Hoda M. O. Mokhtar and Mohamed El-Sharkawi. 2014. Processing Universal uantification Queries using MapReduce. 2014 International Conference on Big Data and Smart Computing (BIGCOMP).

[3] Liu, Simon. 2013. Exploring the Future of Computing. IT Professional.

[4] Aboul Ella Hassanien,Janusz Kacprzyk, Jemal H. Abawajy. Big Data in Complex Systems Challenges and Opportunities.http://link.springer.com/book/10.1007%2F978-3-319-11056-1.

[5] Taoxiao. 2011. PSON: A parallelized Son Algorithm with MapReduce for mining frequent sets. 2011 fourth international symposium on parallel architectures algorithms and programming, 12/2011.

[6] Advances in Visual Informatics 4th International Visual Informatics Conference, IVIC 2015, Bangi, Malaysia, November 17-19, 2015, Proceedings http://link.springer.com/book/10.1007%2F978-3-319-25939-0.

[7] Maitrey, Seema, and C.K. Jha. 2015. Handling Big Data efficiently by using map reduce technique. 2015 IEEE International Conference on Computational Intelligence and Communication Technology.

[8] KatalAvita, Mohammad Wazid and R. H. Goudar. 2013. Big data: Issues, challenges, tools and Good practices. 2013 Sixth International Conference on Contemporary Computing (IC3).

[9] Bellettini Carlo, MatteoCamilli, Lorenzo Capra, and Mattia Monga.Distributed CTL model checking using mapreduce: theory and practice: Distributed Ctl Model.

[10] Zhang, Yanfeng, QixinGao, LixinGao, And Cuirong Wang. 2014. Imapreduce: Extending Mapreducefor Iterative Processing. Large Scale and Big Data.

[11] YanfengZhang. 2012. Imapreduce: A Distributed Computing Framework For Iterative Computation.Journal of Grid Computing, March.

[12] NeelamSingh, NehaGarg, Varsha Mittal. 2012. Big Data - Insights, Motivation and Challenges, Journal of Grid Computing, March.

[13] RahmanMdWasi-ur-, Xiaoyi Lu, NusratSharmin Islam and Dhabaleswar K. (DK) Panda. 2014.HOMR: a hybrid approach to exploit maximum overlapping in MapReduce over high performance interconnects. Proceedings of the 28thACMInternational Conference on Supercomputing - ICS 14.