International Journal of Engineering Research-Online
A Peer Reviewed International Journal
Articles available online http://www.ijoer.in

Vol.3., Issue.5., 2015
(Sept.-Oct.)

RESEARCH ARTICLE

ISSN: 2321-7758

APPLICATION OF THETA JOIN USING MAP REDUCE FRAMEWORK

CH. SRINIVAS[1], K. VENKATA RAMANA[2]

[1,2]Department of CS&SE, Andhra University College of Engineering (A), Visakhapatnam, Andhra Pradesh, India.

CH. SRINIVAS

ABSTRACT

Joins are very important for lot of data evaluation tasks although they are not supported directly from the map reduce paradigm. Map-reduce use commodity computer hardware which is quite simple to acquire. The Proposed join model improves formation of and thinking with respect to joins throughout MapReduce. By utilizing this model, we obtain an extremely easy randomized algorithm, termed 1-Bucket-Theta, for applying theta joins with in a MapReduce framework. This kind of algorithm needs minimal data (input cardinality) and also we offer proof that will for a variety of join issues, it can be near to ideal or the ideal decision. Regarding a number of the complications in which 1-Bucket-Theta isn't really the very best choice, we indicate the best way to much better overall performance by applying additional input statistics. Almost all algorithms could possibly be produced 'memory-aware', and they don't will need virtually any changes for the MapReduce environment.

Keywords: MapReduce, Query Optimization, Theta Join Processing,

1 INTRODUCTION

The rapid growth of data volumes poses a challenge in many scientific disciplines. Scientists must process data sets gathered by means of large-scale experiments along with sensors. The Volume of data that is generated by all of us right from the start of time until finally 2003 ended up being 5 billion gigabytes. When you stack up the particular the shape of disks it may complete an entire football field. The same volume was created in each and every a pair of days within 2011, and in each and every five minutes within rate is growing extremely. Although all of this data created can be meaningful and are needed while highly processed, it is becoming forgotten. 90% the actual world's information was created with in the most recent period.

One among the grand challenges associated with data driven science is always to find interesting patterns in massive high dimensional data sets that may bring out to new hypotheses. This process is currently limited by the large amount of required human effort and the high computational cost.

Our goal is to develop novel scalable exploratory analysis tools and algorithms in order to help scientists search for potentially interesting hypotheses with very large, high-dimensional data sets. We focus on the join operation which is essential for detecting correlations and relationships between patterns in scientific data.

The contributions of this paper are listed in the following.

Given two sets S and T, the join operation returns the set of all pairs (s, t) that satisfy some join condition C(s, t), where $s \in S$, $t \in T$, and C is a Boolean function over the attributes of S and T.

A join operation with such a general join condition C is called a theta-join. The joint operation

is one of the fundamental relational operations and frequently used to combine information from multiple sources. In order to support arbitrary theta-join predicates, we define a join model that will simplifies generation associated with and thought about feasible join algorithms.

We propose a randomized join algorithm called 1-Bucket-Random that achieves near-optimal execution time for output-size dominated join problems. This algorithm is capable of processing any theta-join, Such as the cross-product. This simply desires nominal input statistics (cardinality ratio of input sets) and still efficiently parallelizes the computation. 1-Bucket-Random can be improved by exploiting input statistics for selective joins. We propose algorithms for a popular class of joins, including equality, inequality and also band joins.

2. RELATED WORK

N. Dean et al [2] MapReduce possesses emerged as one of the most common paradigm intended for parallel processing, plus it by now features a excellent effect on data management research. One main reason behind its achievements is the option of to a totally free open-source implementation, Hadoop[1], and also a great energetic developer group which keeps producing upgrades and also introducing capabilities. MapReduce seemed to be offered to make simpler large-scale facts finalizing with spread and also parallel architectures, in particular clusters regarding commodity hardware.

Foto N. Afrati [3] Implementations associated with map reduce are now being helpful to execute several operations on very huge data. We analyze strategies for joining various associations in the map reduce environment. New strategy will begin by simply determining the actual map key the pair of attributes that determine the Reduce process in order to which a Map process should send out a unique tuple. Each attribute from the map-key will get a share. Then, we look at two essential specific cases: chain joins as well as star joins. In each and every case we are able to establish the map-key and establish the actual shares that deliver the least replication.

F. Afrati et al [5] Join processing for example, depends on input replication with in the map phase to be able to calculate multi-way joins. Provides the most effective of a multi-way join in one MapReduce job, which just works for that Equi join case. Since Theta join can't always be answered by simply making the particular join attribute the partition key, thus, the perfect solution proposed in not extended to resolve the situation of multi-way Theta-joins.

3. THETA JOINS

The Theta-Join ∞_θ operator is the most generic and can be any function of the participating attributes from the two relations. The flexibility that this operator gives us, allows for sophisticated queries to be implemented. Several commonly used instances of this operator exist, such as the Equi-Join, where the function is the equality operator (=).

Techniques that will enable effective parallel execution connected with arbitrary theta-joins inside MapReduce. Not any modifications from the MapReduce environment are necessary, and also the end user doesn't even have to write any kind special-purpose code to regulate data flow. Almost everything is achieved simply by specifying the suitable (sequential) Map and also reduce functions.
Particularly, we make the following primary contributions.

1. We propose to hear a reducer primarily based cost model along with join model that will simplifies formation of and reasoning with regards to possible theta-join implementations inside MapReduce.
2. We propose to hear the randomized algorithm known as 1-Bucket- Theta intended for computing any kind of theta-join, such as the cross-product, in single MapReduce job. This kind of algorithm only requires minimal input statistics (cardinality associated input sets) but still effectively parallelizes any kind of theta-join implementation. We show that it must be close to optimal intended for joins having large output size. regarding highly selective joins, we indicate that will although much better implementations inside MapReduce may exist, they generally is not used, leaving behind 1-Bucket-Theta since the very best available option.
3. For any common type of non equi joins, which includes inequality and also band-joins, we propose to here algorithms that will often improve in 1-

Bucket-Theta, provided that sufficiently detailed input statistics are available.

3.1 COST MODEL FOR MAPREDUCE

Since cost intended for transferring data from the DFS through the mapper nodes and also the cost with regard to reading the input tuples locally on each and every mapper is just not impacted by the actual concrete Map along with Reduce functions, we need not take these kinds of costs into consideration for that optimization. Map and Reduce functions impact the costs through providing Map function output till writing a final join result to the DFS. To analyze the particular completion time period of those MapReduce job phases, look at a single reducer.

This reducer receives the particular subset with the mapper output tuples as it's input. It sorts the actual input by key, flows the matches value-list for just a key, computes the join due to this list, then writes their locally produces join tuples towards the DFS.

(S,T)	5	7	7	7	8	9
5						
7						
7						
8						
9						
9						

Table:1 S.A = T.A

(S,T)	5	7	7	7	8	9
5						
7						
7						
8						
9						
9						

Table: 2 abs (S.A = T.A) < 2

(S,T)	5	7	7	7	8	9
5						
7						
7						
8						
9						
9						

Table : 3 S.A >= T.A

Figure 1: Join matrices with equi join, similarity-join, and also inequality-join.

Numbers shows join attribute values through S and T, shaded cells show join results. M(i,j) shows cell numbering.

Our cost model is applicable beyond MapReduce, to any shared-nothing system. However, systems with detailed data fragmentation statistics can exploit data locality and reduce input transfer cost which is not applicable in MapReduce context. For output-size dominated join problems, our cost model is applicable beyond MapReduce.

3.2 JOIN MODEL

a join in between two data sets S along with T that has a join-matrix M as well as use this kind of representation regarding creation of and also reasoning about different join implementations with in MapReduce. Figure 3 indicates example data sets as well as the equivalent matrix intend for a various range of join predicates. For row i as well since column j, matrix access M (i, j) can be defined in order to accurate (shaded inside the picture) when the I^{th} tuple by S and j^{th} tuple by T satisfy the join condition and also false (not filled) usually. Since any kind of theta-join is really a subset from the cross-product, this matrix can easily represent any kind of join condition.

Our objective is usually to have each join output tuple always produced by specifically one reducer, so that expensive post-processing as well as duplicate removing is avoided. Therefore, given r reducers we should map every matrix cell along with value M (i, j) accurate in order to specifically among the r reducers.

We may also suggest that reducer R insures the join matrix cell, just in case this kind of cell will be mapped to R. There are various feasible mappings that handle all true valued matrix cells. Our objective is to discover that mapping from join matrix cells in order to reducers that will minimize job completion time. For this reason we would like to find mappings that will either balance reducer input share or even balance reducer output share or even achieve a compromise between both .

Table 4 :- R1: keys 5,8 Inputs are: S1,S4, T1,T5 and Output: 2 tuples

R2: key 7 Inputs are: S2,S3, T2,T3,T4 and Output: 6 tuples

R1: key 9 Input are : S5,S6,T6 and Output: 2 tuples

max−reducer−input is = 5

max−reducer−output is = 6

(S,T)	5	7	7	7	8	9
5	[5]					
7		[7]				
7						
8					[8]	
9						[9]
9						

Table 4

(S,T)	5	7	7	7	8	9
5	[3]					
7		[2]	[3]	[1]		
7		[3]	[1]	[2]		
8					[1]	
9						[2]
9						[1]

Table 5

(S,T)	5	7	7	7	8	9
5						
7	[1]		[2]			
7						
8						
9					[3]	
9						

Table 6

Figure 2: Matrix-to-reducer mappings along with standard (left) equi-join algorithm, (center) random and also balanced (right) approach

Table 5 : R1: key 1 Inputs are: S3,S2,S6,S4,T4,T3,T6,T5 and Output: 4 tuples

R2: key 2 Inputs are: S2,S5,S3,T2,T6,T4 and Output: 3 tuples

R3: keys 3 Input: S1,S2,S3,T1,T2,T3 and Output is : 3 tuples

max−reducer−input is = 8

max−reducer−output is = 4

Table 6 :- R1: key 1 Inputs are: S1,S2,S3,T1,T2 and also Output: 3 tuples

R2: key 2 Inputs are: S2,S3,T3,T4 and also Output: 4 tuples

R3: keys 3 Inputs are: S4,S5,S6,T5,T6 and also Output: 3 tuples

max−reducer−input is = 5,

max−reducer−output is = 4

the new algorithm denotes the actual practical implementations of this type of simple idea. balance input and also output costs although minimizing replication of reducer input tuples. We may frequently make use the following important lemma.

4. THE 1-BUCKET-THETA ALGORITHM

The actual challenges for implementing joins within MapReduce: data skew and also the difficulty involving implementing non - equi joins along with key-equality based mostly data flow control. We have now expose 1-Bucket-Theta, a algorithm of which addresses these challenges, and gives robust analytical results regarding their properties.

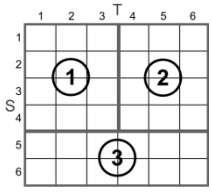

Figure 3: 1-Bucket-Theta Join Matrix Partitioning

Table 1: 1-Bucket-Theta theorems'symbols

Symbol	Explanation
$\|S\|$	sizeofrelationS
$\|T\|$	size of relationT
Q	reducer'sinput
P	number ofreducers
cS	how many optimal squares of the corre-sponding side-length can fit into
cT	how many optimal squares of the corre-sponding side-length can fit into

1-Bucket-Theta inspects almost all tuple pairs, and involves only minimal statistical data, which is the cardinalities from the input, which makes it one of the most tgeneric algorithms.

The actual point on this algorithm could be the fundamental method which it divides the JM, by giving 3 theorems (those are listed below) as well as certain lemmas to guide them, in a manner that almost all cells are covered and simultaneously, the most reducer input metric can be minimized. It can be suited to higher selectivity joins (e.g. >50%). This cross-product includes each and every tuple from S along with every single tuple from T, the particular matching join matrix offers almost all entries set to accurate. We explain the way 1-Bucket-Theta does matrix-to-reducer mapping, indicate that it must be near-optimal for computing the particular cross-product, and also discuss how most of these results extend to processing associated with theta-joins.

These three theorems will be the key points of 1-Bucket-Theta and provide us strong ensures with the near optimality from the algorithm for

implementing this Cross Product case, and can be described as:

•THEOREM 1: If |S| and |T| are multiples of $\sqrt{|S||T|/p}$, the JM can be partitioned into c_s by c_T $\sqrt{|S||T|/p}$ squares of size $\sqrt{|S||T|/p}$ each.

•THEOREM 2: If |S| < |T|/p, then the JM can be partitioned by a single row of r rectangles of size |S| by |T|/p.

•THEOREM 3: If |T/p| ≤ |S| ≤ |T| , then a partitioning can always be found with the properties: not any reducer produces greater than 4|S||T|/p output tuples and not any reducer receives greater than $4\sqrt{|S||T|/p}$ input tuples.

4.1 IMPLEMENTING THETA-JOINS

While we exhibited over that this MapReduce implementation will be close to optimum for cross-product computation, this kind of result will not always carry up to arbitrary joins. This segment provides solid evidence that actually intended for extremely selective join conditions, it is sometimes difficult to help along with a better algorithm compared 1-Bucket-Theta. They will just cannot be seen as correct implementations using the information available at that time when the ideal implement-tation will be selected to get a given join problem, as we show now.

Consider an arbitrary theta-join along with selectivity σ, i.e., it produces σ |S||T| are output tuples. To reduce max-reducer- output, each reducer need to be responsible for σ |S||T|/r are join output tuples. Through 1-Bucket-Theta practically guarantees in order to balance the actual cross-product output throughout reducers.

Algorithm 1: Map (Theta-Join)
Input: input tuple x ∈ S ∪T /* matrix to regionID mapping is loaded into a lookup table during initialization of mapper */ 1: if x ∈S then 2: matrixRow = random(1,|S|) 3: for all regionID in lookup.getRegions(matrixRow) do 4: Output (regionID, (x, "S")) /* key: regionID */ 5: else 6: matrixCol = random(1,|T|) 7: for all regionID in lookup.getRegions(matrixCol)do 8: Output (regionID, (x, |T|))

Algorithm 2: Reduce (Theta-Join)
Input: (ID,[(x_1,$origin_1$), (x_2,$origin_2$),…, (x_k,$origin_k$)])
1: Stuples = ∅; Ttuples = ∅
2: for all (x_i,$origin_i$) in input list do
3: if $origin_i$= "S" then
4: Stuples = StuplesU {(x_i)}
5: else
6: Ttuples = TtuplesU {(x_i)}
7: joinResult = MyFavoriteJoinAlg(Stuples, Ttuples)
8: Output(joinResult)

This particular may not be accurate for other joins. As an example, with some reducer most cross-product tuples may satisfy the join condition, while almost not one do so on a different. Fortunately this is unlikely Due to the randomization which assigns arbitrary samples from S as well as T in order to each reducer. Even though we do not need an analytical proof, the experiments show in which join output is actually generally quite evenly distributed over the reducers. This specific will be estimated so long as join output size will be large enough in order that testing variance will be "averaged out". Significant deviation in output size is most likely whenever join result size is very small, e.g., below a large number of tuples per reducer. However for these cases the entire join output size can be so small which a good significant output difference has only a little effect on the complete runtime.

In short, where ever join output size will be large sufficient in order to significantly impact job completion time,1-Bucket-Theta's randomized approach balances output perfectly across reducers. the idea is extremely A can be quite difficult in order to beat it with output-related costs. For an additional algorithm to attain significantly lower total job completion time, it should have considerably lower input-related costs compared 1-Bucket-Theta.1-Bucket-Theta essentially guarantees that will max-reducer-input value reaches almost all $4\sqrt{|S||T|/r}$ as well as generally it is much closer to $2\sqrt{|S||T|/r}$) hence the actual ratio among max-reducer-input associated with 1-Bucket-Theta versus any kind of competitive theta-join algorithm having a different matrix-to-reducer mapping is at many.

$$\frac{4\sqrt{|S||T|/r}}{2\sqrt{x|S||T|/r}} = \frac{2}{\sqrt{x}}$$

4.2 JOIN CONDITION

When the join condition is often a user refused black box function, after that we have no idea of which join matrix cells include benefit false unless of course we actually assess the join condition intended for these types of cells. However, these particular failures the objective of this algorithm: To discover an effective join algorithm, we may have to compute this join for those cells we are considering since candidates intended certainly not covering them by any kind of reducer. Even though the join condition doesn't consist of user-defined functions, in used it is sometimes hard to identify large regions inside the join matrix intended for which this algorithm can ensure that the entire are contains zero join result tuple.

Before we explore this challenge inside more depth, think about the examples through the above figure. Assume which insides the two cases we compute the exactly same equi-join for that similar inputs. This partitioning inside the left example is much better, since it prevents the large input replication needed for 1-Bucket-Random's cross-product depend dividing in the right example. It is achieved by not really covering huge parts of cells which contain no outcome tuples.

For that block of 3 by 4 cells with in the lower-left corner, histograms with S and also T could mean that predicate (S.A ≥ 8 ^T.A ≤ 7) contains with this matrix region. In order to not assign any kind of cell with this block into a reducer, the actual algorithm has to understand which none of the cells in the region fulfils the join condition. From the example, it's to indicate that ∀ s ∈ S, t ∈ T: (s.A ≥ 8 ^t.A ≤ 7) => ¬ (s.A = t.A). While this can be easy to have an equi-join, it may be difficult and also expensive generally.

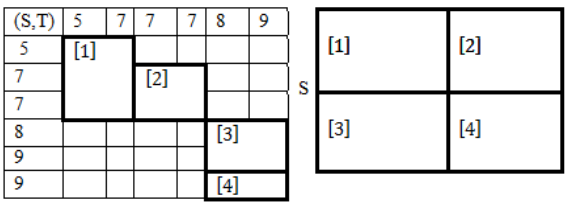

Figure 4 : 8 Matrix-to-reducer mapping examples

For every single candidate region, the particular algorithm needs to obtain the same predicates. After that it has to manage a few satisfiability solver algorithms in order to prove these predicates mean that the join condition is not really satisfiable. When join result tuples are generally "scattered" through-out the join matrix, and then most significant matrix regions will certainly contain a few output tuples. To recognize regions without having join tuples, thus several small regions need to be examined, contributing in higher computational cost.

4.3 M-BUCKET-I AND M-BUCKET-O

For input-size focused joins, we would like to get a matrix-to-reducer mapping that will reduce max-reducer-input. Obtaining this kind of an optimum deal with of all candidate cells generally is usually a hard problem, hence we propose a simple heuristic.

Algorithm 3: M-Bucket-I (Input)
Input: maxInput, k, M
1: row = 0
2: **while** row <M.no Of Rows **do**
3: (row, k) = Cover Sub Matrix(row, maxInput, k, M)
4: if k < 0 then
5: return false
6: End If
7. End While
8. return true

We consider the corresponding algorithm since M-Bucket-I, since it needs more in depth input statistics along with minimizes max-reducer-Input. Recall that will M-Bucket-I has been designed to reduce max-reducer-input. For output-size dominated joins, you should minimize max-reducer-output as a alternative. Due to this issue, we designed a heuristic known as M-Bucket-O. It continues such as M-Bucket-I, but instead of working together with an input-size limit maxInput, this limitations region by area and number of candidate cells along with in a region. observe that M-Bucket-I usually takes far better advantage of input histograms when compared with M-Bucket-O, because it aware of just how many input tuples via each data set are part in order to each single bucket. However, the specific output size of a bucket might be anything actually zero and the product from the

bucket counts. Hence M-Bucket-I may reliably equilibrium input-related costs despite relatively coarse-grained histograms, while M-Bucket-O may show significant output-cost difference actually for quite fine-grained histograms.

5. EXPERIMENTATION RESULTS

The implementation was written in Java with each of the HadoopMapReduce Framework. We evaluated this implementation on a single node running Hadoop 0.21.0.

There is absolutely no individual hardware requirement arranged for installing Hadoop.

TYPE	MINIMUM	RECOMMENDED
Processor	1.8 Ghz	2.3 GHz
Memory	4 GB	8 GB
Hard Disk Space	100 GB	150 GB

We existing results using the following data units:

Cloud: It is a real data set that contains extended cloud reports coming from ships as well as land stations [38]. You will find 382 million records, each single along with 28 attributes, providing a total data size associated with 28.8GB.

Cloud-5-1, Cloud-5-2: These are generally two independent random types of 5 million records each and every single through Cloud. They may be used for experiments along with output-size.

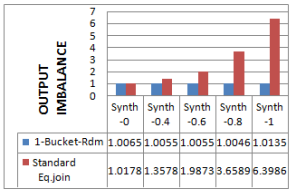

Figure 3: Skew Resistance of 1-Bucket-Random (Output Imbalance) on Synth- α

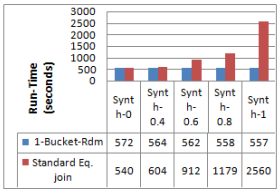

Figure 4: Skew Resistance of 1-Bucket-Random (Run-time) on Synth- α

Synth-α: For a fixed α, it is a set of data set. Both consist of 5 million records, each and every record like a single integer number among 1 and 100.

Table 8: M-Bucket-I cost details (seconds)

Step	Number Of Buckets				
	1	10	100	1000	10000
Quantiles	0	232	233	233	235
Histogram	0	243	245	254	266
Heuristic	0.78	0.58	1.75	1.76	1.78
Join	63600	2153	636	482	465
Total	63600.78	2628.58	1115.75	970.76	967.78

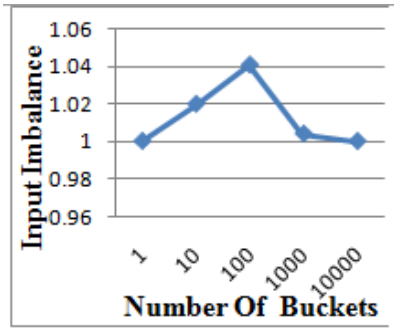

Fig a)Input imbalance for 1-Bucket Random and also M-Bucket-I on Cloud

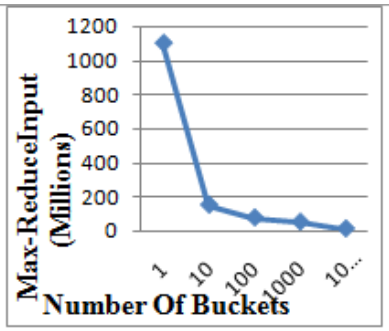

Fig b)Max-Reducer-Input for 1-Bucket-Random and also M-Bucket-I on Cloud

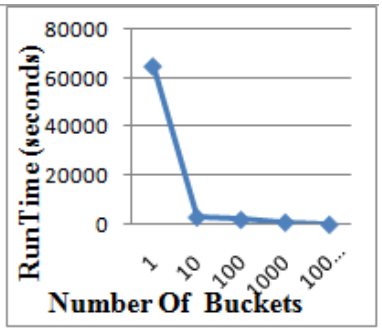

Fig c)Runtime MapReduce for 1-Bucket-Random and also M-Bucket-I on Cloud

Table 9: M-Bucket-O Cost Details (Seconds)

Step	Number Of Buckets				
	1	10	100	1000	10000
Quantiles	0	4.52	4.54	4.8	4.9
Histogram	0	12.6	11.6	11.6	13.8
Heuristic	0.06	0.04	0.05	0.22	0.87
Join	802.8	2332	1076.8	968.6	796.8
Total	802.86	2349.16	1092.99	985.22	816.37

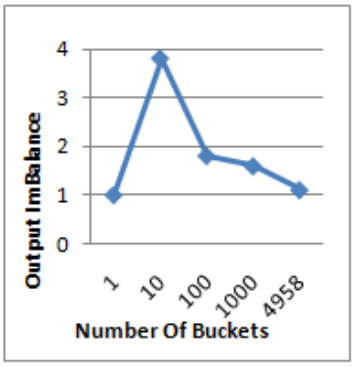

Fig d) Output Imbalance for 1-Bucket-Random and also M-Bucket-O in Cloud-5

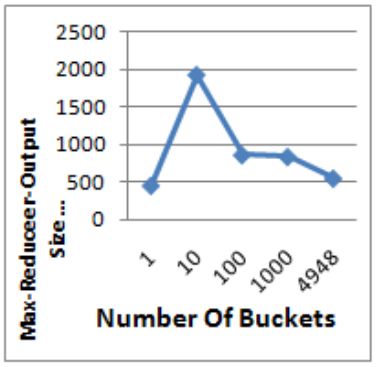

Fig e) Max-Reducer-Output for 1-Bucket-Random and also M-Bucket-O in Cloud-5

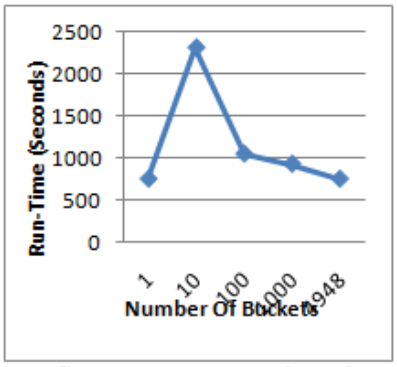

Fig f) Runtime MapReduce for 1-Bucket-Random and also M-Bucket-O in Cloud-5

6. CONCLUSION

Beginning with the purpose of reducing total job completion time, we showed the way to define an excellent various join implementations applying suitable join matrix-to-reducer mappings. To be able to assistance arbitrary joins, we all initial proposed 1-Bucket-Random, the randomized algorithm that may compute any kind of subset from the cross-product, i.e., just about any theta-join. We showed which the matrix-to-reducer mapping achieved through 1-Bucket-Random will be provably near to optimal for any join along with significantly much larger output size as compared to its input size.

For any common class of joins for example equi-joins, inequality joins and also band-joins, we enhanced the runtime achieved through 1-Bucket-Random utilizing our M-Bucket algorithms. These algorithms obtain improved runtime through exploiting input statistics having a equally lightweight test, and so compute selective join conditions effectively.

Our join model allows us to approximate max-reducer-input and also max-reducer-output for each and every single algorithm. An optimizer may apply conventional cost estimation techniques through databases, as the job completion time is depend upon the single reducer will get the greatest input and also the reducer which generates the maximum output. Local reducer computation will be straight responsive to traditional cost evaluation including CPU along with I/O cost.

7. REFERENCES

[1]. Apache hadoop. http://hadoop.apache.org.

[2]. J.Dean and S.Ghemawat. Mapreduce: Simplified data processing on large clusters. InOSDI, 2004.

[3]. F.N.Afrati and J.D.Ullman. Optimizing joins in a map-reduce environment. InEDBT, pages 99-110,2010.

[4]. A.Okcan and et al. Processing theta-joins using mapreduce. InSIGMOD, pages 949–960, 2011.

[5]. F.N.Afrati and J.D.Ullman. Optimizing multiway joins in a map-reduce environment. IEEE Trans. Knowl. Data Eng.,23(9):1282–1298,2011.

[6]. Ioannis K. Koumarelas et al. Binary Theta-Joins using MapReduce: Efficiency Analysis and Improvements.