Distributed Diversification of Large Datasets

Mahbub Hasan [1], Abdullah Mueen [2], Vassilis Tsotras [1]

[1] *University of California, Riverside*
{hasanm,tsotras}@cs.ucr.edu

[2] *University of New Mexico*
mueen@cs.unm.edu

***Abstract*—Diversification has been recently proposed as an approach to allow a user to better grasp a large result set without having to look through all relevant results. In this paper, we expand the use of diversification as an *analytical* tool to explore large datasets dispersed over many nodes. The diversification problem is in general NP-complete and existing uniprocessor algorithms are unfortunately not suitable for the distributed setting of our environment. Using the MapReduce framework we consider two distinct approaches to solve the distributed diversification problem, one that focuses on optimizing disk I/O and one that optimizes for network I/O. Our approaches are iterative in nature, allowing the user to continue refining the diversification process if more time is available. Moreover, we prove that (i) this iteration process converges and (ii) it produces a 2-approximate diversified result set when compared to the optimal solution. We also develop a cost model to predict the run-time for both approaches based on the network and disk characteristics. We implemented our approaches on a cluster of 40 cores and showed that they are scalable and produce the same quality results as the state-of-the-art uniprocessor algorithms.**

***Keywords*-Diversity, MapReduce, Parallel Processing**

I. Introduction

Many search related applications have recently incorporated *result diversification* as a means of increasing user understanding of the result space. The idea is to return to the user results that are as *relevant* as possible to the query and at the same time, as *diverse* as possible from each other. Diversification has been applied on several domains including recommendation and web search [22][16][19][25][3], structured databases [13][17], personalized systems [18], topic summarization [8], etc. However, all these previous works have advocated diversification only for the online applications and proposed algorithms that are suitable to diversify a typically smaller result set (e.g. thousands of results). In this paper, we use diversification as an analytical tool to explore large datasets (e.g. millions of elements). Consider the scenario where the user specifies some criteria (expressed as SQL text, keywords, images etc.); for simplicity we call these criteria as the *query*. These criteria order elements from the large dataspace according to their relevance to the user query. (Despite the use of the term query, the reader should note that our environment is not on-line querying but off-line.) Returning the top-k elements based on relevance-only may return many elements from the same category, therefore is not suitable for exploring the dataspace. Instead returning the top-k most diverse elements provides the user with a better understanding of the dataspace.

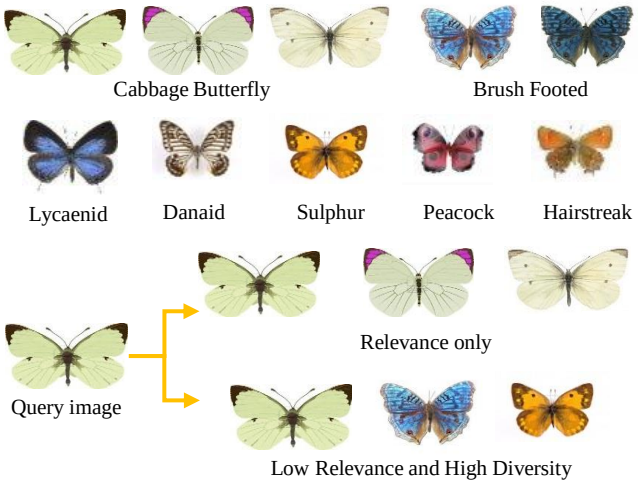

Fig. 1. A set of different types of butterflies match with a sample query image and 3 images selected satisfying different criteria: relevance only and relevance with diversity

As an example consider a dataset (dataspace) with images crawled from the web [1]. A user is interested in butterflies so s/he provides a query image of a butterfly as shown in Figure 1. Assuming that relevance is based on shape similarity, millions of images match this query (for simplicity, the dataspace shown in the figure contains only ten images) and the top-3 based on relevance-only are depicted. However, we could have better explored the dataspace by using other image features such as texture, color, etc. to capture diversity among the returned results. When both diversity and relevance are considered, results are ranked according to a score function that considers a weighted sum of similarity and diversity based on a trade-off parameter [16]. The second top-3 results depict a lower-relevance higher-diversity set of butterflies.

The diversification problem is in general NP-complete [23] and many uniprocessor approximation algorithms have been proposed in the literature [22][23][10][24][7]. For large datasets, even the approximate algorithms are lethargic and therefore, parallelization becomes the method of choice for faster response time. MapReduce[12] has become extremely popular over the last few years for processing large volume of data in a distributed setting. Several database problems (e.g. clustering[9], join[5]) have been solved successfully using MapReduce. In this paper we propose two distinct approaches for result diversification using the mapreduce framework.

While there exist domain specific diversification algorithms [6][3][11] that assume availability of prior information such as query log, taxonomy, etc., our approaches provide a more general distributed framework for result diversification. The choices for the relevance and diversity measures are orthogonal with our approaches.

Extending diversification on a distributed environment is a challenging task because none of the previous uniprocessor diversification algorithms can be directly extended to a MapReduce setting. For example, parallelizing the computation of the method in [7] requires one read of the entire file (dataset) to produce one diversified result in the output. Therefore, computing the top-k diverse results require at least k file reads which becomes infeasible (in disk I/O [20]) for large datasets. Another challenge is to minimize the communication cost between the distributed nodes (network I/O [9]). This motivates the need to develop distributed algorithms that can diversify large datasets (e.g. millions of elements) fast while preserving the quality of the diversification, as well as achieving linear speedup for increasing number of nodes.

We propose two distributed diversification approaches addressing these challenges. In Section IV, we show that different approaches perform better in different scenarios (e.g. disk rate, network speed, number of cluster nodes, data size). We present a cost model that can dynamically choose the most suitable approach for a given computation environment.

In particular, we make the following contributions:

- We describe the first distributed solution for diversifying large datasets using the MapReduce framework.
- We propose two approaches; one optimized for the disk access cost while the second optimized towards the network transfer cost. We present a cost model that can dynamically choose the suitable approach considering the environment parameters (disk rate, network speed, number of cluster nodes) and data size.
- We propose an approach to improve the quality of the diversification by iteratively refining the output. The final output is a 2-approximation over the optimal solution.

The rest of the paper is organized as follows: we define the top-k diversification problem in Section II. Section III explains the core components of diversification (i.e. diversification approaches, cost model, iterative refinement). Section IV provides an experimental evaluation of the components using two real life datasets. Section V describes the related work and finally we conclude in Section VII.

II. Problem Definition

Consider a set of n elements $\mathcal{D} = \{e_1, e_2, \ldots, e_n\}$, a query Q and an integer k $(\leq n)$. Each element $e_i \in \mathcal{D}$ has a *relevance* score, $rel : \mathcal{D} \rightarrow \mathbb{R}^+$, to the query Q, where higher relevance score implies the element e_i is more *relevant* to the query Q. The *dissimilarity* between two elements e_i, e_j is defined by the function, $dis : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}^+$, where a higher score implies that the elements e_i and e_j are highly *dissimilar* to each other. Our goal is to find a set of k elements, $S_k \subseteq \mathcal{D}$, such that the elements in S_k are highly *relevant* to the query Q and highly *dissimilar* to each other.

Most previous works define top-k diversification as a bi-criteria optimization problem: to each k-element subset $S_k \subseteq D$ a score $\mathcal{F}(S_k)$ is assigned, using both the *relevance* and *dissimilarity* of the elements in S_k. In particular, let $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}^+$ be the distance metric defined as,

$$d(e_i, e_j) = (1 - \lambda) \frac{rel(e_i) + rel(e_j)}{2} + \lambda \, dis(e_i, e_j) \quad (1)$$

where, the trade-off parameter, $\lambda \in [0, 1]$, balances the relative weights between *relevance* and *dissimilarity* [23]. Then $\mathcal{F}(S_k)$ is the sum of all pairwise distances between the elements in S_k, namely:

$$\mathcal{F}(S_k) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} d(e_i, e_j) \quad (2)$$

Problem 1 (Top-k Diverse Elements): Given $\mathcal{D}$, Q and k identify the subset S_k for which $\mathcal{F}(S_k)$ is maximum.

III. Diversification Framework

There are three main components in our diversification framework (Figure 2): 1) the *Diversification Stage*, 2) the *Cost Analysis*, and, 3) the *Iterative Refinement*. The *Diversification Stage* component reads data from HDFS and generates the set S_k using a diversification algorithm. In Section III-A we propose two distributed diversification approaches for this component. In Section III-B we present a cost model to estimate the execution time of each diversification approach given the environment parameters and data characteristics. Using this model, the *Cost Analysis* component chooses the best approach dynamically at runtime. Finally, the *Iterative Refinement* component iteratively refines the set S_k returned from the diversification stage until it either converges (no further score improvement) or a user time threshold is reached (Section III-C).

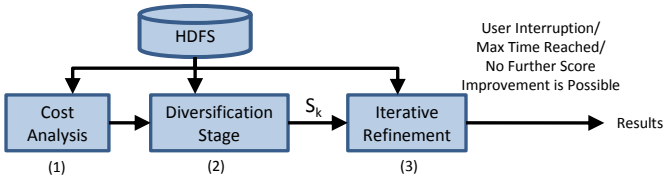

Fig. 2. The Diversification Framework Architecture

A. Diversification Approaches

In the rest of the paper, we assume a distributed MapReduce framework with m mappers and r reducers. Previous works [9][15] on designing algorithms in a MapReduce framework generally consider the following approach: First the data is divided (mapped) into partitions and each partition is assigned to a single node. Each node solves (reduces) the problem on the assigned partition and generates an output. Then, all outputs are merged together in single node which produces the final output. Our first approach, the **D**ivide and **M**erge based

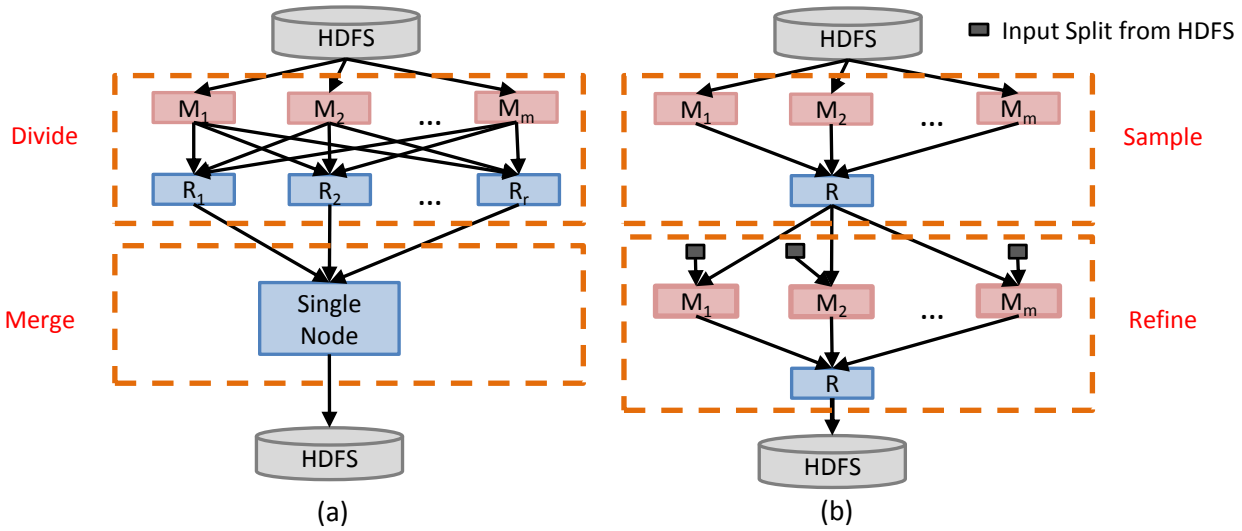

Fig. 3. Overview of (a)DM and (b)SR Diversification Approaches

diversification (DM), follows a similar strategy. This approach is disk I/O efficient since it reads the whole data only once from the disk. However, it incurs high network cost to send all the elements through the network for partitioning.

To reduce the network cost, recently another approach, namely "sample and ignore", has been proposed for the problem of clustering large data [9]. It reduces the network cost by first finding the major clusters from a sample of the input data and then ignoring the elements that are contained in the major clusters from passing through the network. The sampling idea is useful for our purposes as it maintains the charasteristics of the dataset; however, we replaced the "ignore" phase with a novel "refine" phase which reduces the network cost significantly compared to the "ignore" phase by sending only k results from each node. This resulted to the **S**ample and **R**efine based diversification (SR) approach.

Note that both of our approaches use a uniprocessor diversification algorithm as a plug-in when each single node performs diversification on its local element set. The choice of this diversification algorithm is independent with our framework. Therefore, any of the algorithms proposed in [22][23][7] can serve the purpose.

1) Divide and Merge (DM): As the name implies, there are two phases, *divide* and *merge* (Figure 3(a)). In the *divide* phase, data is partitioned randomly to different nodes maintaining a balanced load. Each node executes the uniprocessor diversification algorithm on the assigned partition and generates a top-k diversified subset from its own partition. One key assumption in this phase is that each node has enough memory to store its assigned partition (the case that this does not hold is considered in section III-C). In the *merge* phase, all k-diverse results generated by different nodes are merged in a single node to compute the overall top-k diverse results.

The *divide* phase is implemented by a single MapReduce job. In the map phase, each map task reads a split (block) of $\mathcal{D}$ from HDFS. For each element in the file split, it outputs the pair $< key, element >$. The key denotes the ID of the reducer (between 1 to r). In the shuffle phase, each reducer gets the pairs with the same key, and pairs with distinct key values are forwarded to distinct reducers to be processed separately. In the reduce phase, each reducer executes the uniprocessor diversification algorithm on the assigned elements and generates k-diverse elements. Note that, the value of k is relatively small (in tens) while the value of r is assumed in the hundreds. Therefore, the total number of merged elements (rk) is small enough to fit in the memory of a single node at the *merge* phase. This node executes the uniprocessor diversification algorithm on the rk elements and generates the overall k diverse elements S_k.

2) Sample and Refine (SR): Although the DM approach reads the input data only once in its divide phase (therefore, is disk I/O efficient), it sends all the elements through the network for partitioning. For a slow network, this might cause a bottleneck. Instead, the SR approach reduces the network load significantly by sending a small subset of the elements through the network.

SR (Figure 3(b)) also works in two phases, namely, *sample* and *refine*. In the *sample* phase, each mapper reads a split of $\mathcal{D}$ from HDFS and selects a small random sample from the split. Let α be the sampling ratio. A single reducer collects all the samples, executes the uniprocessor algorithm on the sampled elements and computes the k diverse elements S'_k. Note that, only the selected samples need to be shuffled through the network. Thus the SR algorithm reduces the network cost significantly compared to the DM algorithm.

The key challenge of the *sample* phase is to select a good representative sample from each mapper's file, such that the single node that diversifies the samples, can still produce a good k-diverse result. Since the quality of the diversification depends on the score $\mathcal{F}(S'_k)$ (higher score means more diversified result), we investigate the effect of α on the diversification score. Figure 4 shows the $\mathcal{F}(S'_{10})$ score of top-10 diverse tweets computed from 10 million tweets[2] by varying the sampling ratio α. The MMR uniprocessor algorithm [7] was used for diversification. For small k, a sample of about 1% is good enough. However, for higher k, a larger sample is needed

(around 30% for $k = 25$).

Note that the SR approach assumes that all samples taken from the mappers will fit in the memory of the single reducer node. If this is not the case (assuming that a limited amount of memory is available for the diversification task), the sampling rate needs to be adjusted, thus reducing the quality of the sample. This motivates us to further refine the k diverse element set (S'_k) generated in the *sample* phase by a novel *refine* phase using the elements in $\mathcal{D}$.

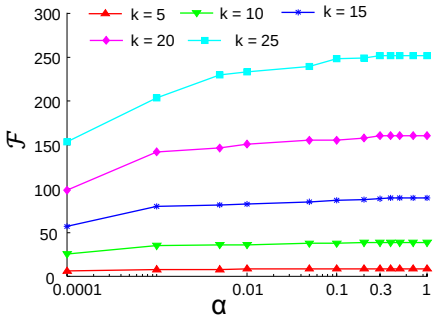

Fig. 4. Effect of α on $\mathcal{F}$

In the *refine* phase, S'_k is broadcasted to all mappers. Each mapper reads a split of data from the disk and tries to refine S'_k using the elements from the split. Since in MapReduce framework the mapper works on a single element at a time, we use the swap strategy [24] for refinement. Each element e in the partition is checked against all the elements e_i in S'_k to see if there exists a replacement operation, e for e_i, that can improve the quality of S'_k. If there exists multiple such operations, e replaces the element e_i in S'_k that improves the quality the most. When all elements in the split are checked, the refined k-diverse element set is forwarded to a single reducer. This reducer combines all refined element sets returned from different mappers, executes the uniprocessor algorithm on the combined element set and computes the final k-diverse results. Note that, the total number of elements shuffled in the *refine* phase is mk. The value of k is in tens (as discussed before) and the value of m is in hundreds depending on the split size and data size. Therefore, the network cost of the *refine* phase is negligible compared to the DM algorithm.

Algorithm 1 shows the pseudocode of the SR algorithm in high level. The sample phase corresponds to the lines (2-3). The refine phase (line 5) is further elaborated by Algorithm 2. In the map phase of Algorithm 2 (lines 2-5), each mapper works on a split of $\mathcal{D}$ and call the subroutine $get\_refined\_set$ for each element in the file split. When all the elements are processed, the refined element set is forwarded to a single reducer. Finally, one reducer merges the refined element sets and computes the final k-diverse results (line 7).

In Section IV, we show that the quality of the diversified result set produced by each of our approaches, DM and SR, matches the quality of the diversified result set that a uniprocessor diversification algorithm would produce (if it was fed the full data set). Furthermore, in section III-C we theoretically prove that combined with some *refine* phases, both of our algorithms produce a diversified result set whose score is 2-approximate to the score of the optimal diversified result set.

Algorithm 1 $SR(\mathcal{D},k,\alpha)$

Require: Element Set $\mathcal{D}$, value of k and Sampling Ratio α
Ensure: Return k diverse results
1: // sample
2: In parallel, each mapper reads a split of $\mathcal{D}$ from HDFS, selects some elements with probability α and sends the elements to a single reducer
3: one reducer gets the elements, executes the uniprocessor algorithm and produces a k-diverse results, S'_k
4: // refine
5: $S_k \leftarrow Refine(\mathcal{D},S'_k)$
6: return S_k

Algorithm 2 $Refine(\mathcal{D},S_k)$

Require: Element Set $\mathcal{D}$ and k-element set S_k
Ensure: Return refined k diverse results
1: // map
2: In parallel, each mapper reads a split of $\mathcal{D}$ from HDFS and do the following
3: **for** each element e in the file split **do**
4: $S_k \leftarrow get\_refined\_set(e, S_k)$
5: sends S_k to a single reducer
6: // Reduce
7: one reducer gets the refined result sets, executes the uniprocessor algorithm, and produces k-diverse results S_k
8: return S_k

B. Cost Model

The performance of the two approaches depends on various parameters of the distributed environment (disk speed, network speed, number of nodes etc.) For example, Figures 8(a), 8(c) show the wall clock time needed to compute top-10 diverse results (for twitter[2] and image[1] datasets respectively), by varying the number of reducers. As seen from these experiments, the SR approach performs better for smaller number of reducers while the DM dominates as the number of reducers increases. Ideally, depending on the environment parameters and data characteristics, we would like to choose the best diversification approach. We thus proceed with a cost model

Algorithm 3 $get\_refined\_set(e, S_k)$

Require: an element e and k-element set S_k
Ensure: Return k diverse results
1: $S'_k \leftarrow S_k$
2: **for** each element e_i in S_k **do**
3: **if** $\mathcal{F}(\{S_k - e_i\} \cup e) > \mathcal{F}(S'_k)$ **then**
4: $S'_k \leftarrow \{S_k - e_i\} \cup e$
5: **if** $\mathcal{F}(S'_k) > \mathcal{F}(S_k)$ **then**
6: $S_k \leftarrow S'_k$
7: return S_k

Symbols	Definitions
$F_{\mathcal{D}}$	File Size of $\mathcal{D}$ in bytes
D_r	Disk Rate in bytes/sec (average read/write)
N_r	Network Rate in bytes/sec
α	Sampling Ratio
β	Dispersion Ratio
m	Number of Mappers
r	Number of Reducer

TABLE I
COST MODEL PARAMETERS

that captures the disk I/O, network I/O and CPU cost for DM and SR.

Our cost model is developed on a standard deployment of Hadoop 1.0.4 for massive data. Therefore, we do not assume data piping from memory to memory, instead, pessimistically assume disk is being used in between the mappers and reducers. Table I summarizes the parameters used. Using an approach similar to [9] we first model the cost of a single MapReduce job which is the sum of costs for the Map and Reduce phases.

Map Cost: Let the cost to start m mappers be $delay(m)$, where $delay(1)$ denotes the time required to start a single task (map/reduce). To read $F_{\mathcal{D}}$ bytes from the disk by m mappers incurs $\frac{F_{\mathcal{D}}}{m.D_r}$ cost. On average, each mapper gets $\frac{F_{\mathcal{D}}}{m}$ bytes. Let $costP(\frac{F_{\mathcal{D}}}{m})$ be the cost to process the $\frac{F_{\mathcal{D}}}{m}$ bytes by each mapper. A factor of α bytes are picked during the processing. To spill the sampled bytes to disk each mapper takes $\frac{\alpha.F_{\mathcal{D}}}{m.D_r}$ cost. Note that, we ignore the additional bytes required to store the key part in the output pairs since this is negligible compared to input bytes $\alpha.F_{\mathcal{D}}$. Therefore the cost to execute the map phase, $costM(F_{\mathcal{D}}, m, \alpha)$, is:

$$costM(F_{\mathcal{D}}, m, \alpha) = delay(m) + \frac{F_{\mathcal{D}}}{m.D_r} + costP(\frac{F_{\mathcal{D}}}{m}) + \frac{\alpha.F_{\mathcal{D}}}{m.D_r}$$

Reduce Cost: In the reduce phase, the data stored in the mapper local disks are copied in the reducer memory. To read $\alpha.F_{\mathcal{D}}$ bytes from m mappers' local disks takes $\frac{\alpha.F_{\mathcal{D}}}{m.D_r}$ cost. Note that, a fraction of these $\alpha.F_{\mathcal{D}}$ bytes are shuffled through the network to reach the other cluster nodes running reduce tasks. Let β is the dispersion ratio, denotes the fraction of mapper output are shuffled though the network. To shuffle $\beta.\alpha.F_{\mathcal{D}}$ bytes to r reducers requires $\frac{\beta.\alpha.F_{\mathcal{D}}}{r.N_r}$ cost.

Once the data is copied in reducer memory, the uniprocessor diversification algorithm is executed on the data. On average each reducer gets $\frac{\alpha.F_{\mathcal{D}}}{r}$ bytes. Let $costD(\frac{\alpha.F_{\mathcal{D}}}{r})$ be the cost to execute the uniprocessor algorithm on $\frac{\alpha.F_{\mathcal{D}}}{r}$ bytes. Finally the output records in written in HDFS. Since each reducer needs to write only k elements, which is small in size, thus the cost to write the output is ignored.

Therefore, the cost to execute the refine phase, $costR(F_{\mathcal{D}}, m, r, \alpha)$, is defined as,

$$costR(F_{\mathcal{D}}, m, r, \alpha) = delay(r) + \frac{\alpha.F_{\mathcal{D}}}{m.D_r} + \frac{\beta.\alpha.F_{\mathcal{D}}}{r.N_r} + costD(\frac{\alpha.F_{\mathcal{D}}}{r})$$

Note that, in both the DM and SR approaches, each reducer gets the elements with the same key. Therefore, the sorting time is negligible in the reduce phase.

DM Cost: In the *divide* phase, m mappers read the data and do the partitioning with cost $costM(F_{\mathcal{D}}, m, 1)$. The reducers execute the uniprocessor algorithm with cost $costR(F_{\mathcal{D}}, m, r, 1)$. Note that, in the *merge* phase, one single machine runs the uniprocessor algorithm on $r.k$ elements. Since the value of $r.k$ is relatively small (discussed in Section III-A), the cost associated with the merging phase is ignored from calculation. Therefore, the cost to execute the DM algorithm is:

$$costDM = costM(F_{\mathcal{D}}, m, 1) + costR(F_{\mathcal{D}}, m, r, 1) \quad (3)$$

SR Cost: The cost to execute SR algorithm consists of the cost associated with the two phases, *sample* and *refine*. In the *sample* phase, m mappers do the sampling with cost $costM(F_{\mathcal{D}}, m, \alpha)$. One reducer processes the remaining data with cost $costR(F_{\mathcal{D}}, m, 1, \alpha)$. In the *refine* phase, m mappers do the refinement with cost $costM(F_{\mathcal{D}}, m, 1)$. Note that, in the map phase, each mapper outputs k elements. Thus the single reducer gets a total $m.k$ elements by m mappers; since this is small, the cost associated in diversifying these mk elements is ignored. Hence, the cost to execute the SR algorithm is:

$$costSR = costM(F_{\mathcal{D}}, m, \alpha) + costR(F_{\mathcal{D}}, m, 1, \alpha) + costM(F_{\mathcal{D}}, m, 1) \quad (4)$$

Using Equations 3 and 4, the cost analysis component in Figure 2 estimates the execution times of two diversification approaches DM, SR respectively, and picks the best one at runtime.

C. Iterative Refinement

Our DM approach assumes that in the *divide* phase each reducer has enough memory to store the assigned partition. However, in case of limited memory, each reducer gets a sample of the partition, thus reduces the quality of diversification (Figures 10(a)) by the DM approach. Therefore, we propose an iterative refinement component that further refines the output of the DM approach and guarantees a 2-approximate solution compared to the optimal. Note that this component is also applicable after the SR approach to guarantee the 2-approximate solution.

The iterative refinement component works as a plug-in after the diversification stage and iteratively improves the quality of the k-diverse elements returned by the diversification approaches. Each iteration corresponds to a single mapreduce job and does exactly the same task like the *refine* phase in SR algorithm. This process continues until no further score improvement is possible or a user threshold time is reached. Also note that, the user can stop the execution at any point in the iterative refinement to get the best result set produced at the elapsed period of time. This ensures a possible adaptation of our approach as an *anytime* algorithm.

Algorithm 4 ($DivF$) describes the overall workflow of our diversification framework. At first, the cost analysis component computes the cost of our two diversification approaches DM and SR (line 1). Based on the costs, the best approach is executed in diversification stage (lines 2-5). Finally the iterative component refines the k diverse set returned from diversification stage (lines 6-11).

Algorithm 4 $DivF(\mathcal{D},k,\alpha)$

Require: Element set $\mathcal{D}$ and size of k, Sampling Ratio α
Ensure: Return set $S_k \subseteq \mathcal{D}$ of size k
1: compute $costDM$ and $costSR$ using the cost model proposed in Section III-B
2: **if** $costDM < costSR$ **then**
3: $S_k \leftarrow DM(\mathcal{D},k)$
4: **else**
5: $S_k \leftarrow SR(\mathcal{D},k,\alpha)$
6: **repeat**
7: $S' \leftarrow Refine(\mathcal{D},S_k)$
8: $diff = score(Q,S') - score(Q,S_k)$
9: **if** $diff > 0$ **then**
10: $S_k \leftarrow S'$
11: **until** $diff \leq 0$ or max time elapsed or user interrupts
12: return S_k

Next, we prove that $DivF$ halts after finite number of iterations, and when it halts (no refinement is possible in the mappers of the *refine* phase), it produces a 2-approximate solution compared to the optimal solution.

Lemma 1: $DivF$ halts after finite number of iterations.

Proof: Since the while loop in Algorithm 4 is the only iterative component, it suffices to show that the while loop halts after finite number of iterations. Let S_{opt} denotes the optimal k-diverse element set of $\mathcal{D}$. Each iteration of the while loop changes the current k element set S_k to a new refined set and improves the current score by a positive value $(0,\mathcal{F}(S_{opt})-\mathcal{F}(S_k)]$. Note that, the total number of k elements subsets of $\mathcal{D}$ is $\binom{|\mathcal{D}|}{k}$, and each subset has a fixed score $\mathcal{F}$. Therefore, $DivF$ halts after at most $\binom{|\mathcal{D}|}{k}$ iterations. ■

Although the proof section of Lemma 1 considers the worst case scenario ($\binom{|\mathcal{D}|}{k}$ iterations), in Section IV, we show empirically (Figures 10(b)) that there is a high probability that $DivF$ halts after $2 \sim 3$ iterations.

To prove that our iterative algorithm $DivF$ produces 2-approximate solution we assume that the distance metric d follows the triangular inequality. We can rewrite the score $\mathcal{F}(S_k)$ as, $\mathcal{F}(S_k) = \frac{1}{2}\sum_{e_i \in S_k}\sum_{e_j \in S_k, e_j \neq e_i} d(e_i,e_j) = \frac{1}{2}\sum_{e_i \in S_k} C_{e_i}^{S_k}$. Here $C_{e_i}^{S_k}$ denotes the contribution of an element $e_i \in S_k$ to $\mathcal{F}(S_k)$ which is the sum of all $d(e_i,e_j)$ between e_i and the other elements e_j in S_k. Let S is the final k element set returned by $DivF$ when it halts.

Theorem 1: $\mathcal{F}(S_{opt}) \leq 2\mathcal{F}(\mathcal{S})$.

Proof: Let us consider the worst case scenario where S_{opt} and S have no elements in common, that means $S_{opt} \cap S = \emptyset$. We will establish a one to one mapping between the elements in S_{opt} and S. Let e_i and e'_i are two arbitrary elements from S and S_{opt} respectively. Since, the total score of an element set is half to the sum of all individual element contributions to the score, to prove Theorem 1, it suffices to show that $C_{e'_i}^{S_{opt}} \leq 2C_{e_i}^{S}$.

Let e'_j is an element in S_{opt} and $e'_j \neq e'_i$. Also let C_1 (or C_2) denotes the sum of all $d(.,.)$s between the elements in S/e_i and e'_i (or e'_j). Note that C_1 (or C_2) is less than $C_{e_i}^S$, otherwise e_i would be replaced by e'_i (or e'_j) in the refine phase. By the triangular inequality, we can say that the sum of C_1 and C_2 is at least as $(k-1)d(e'_i,e'_j)$ (as seen in Figure 5). Thus, $(k-1)d(e'_i,e'_j) \leq C_1 + C_2 \leq 2C_{e_i}^S$. Therefore, we can write, $C_{e'_i}^{S_{opt}} = \sum_{e'_j \in S_{opt}/e'_i} d(e'_i,e'_j) \leq \sum_{e'_j \in S_{opt}/e'_i} \frac{2}{(k-1)} C_{e_i}^S = 2C_{e_i}^S$. ■

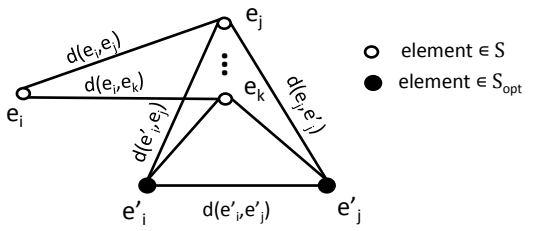

Fig. 5. 2-approximation of $DivF$

IV. Experimental Evaluation

We proceed with an experimental evaluation of the three main components (*diversification approaches*, *cost model*, *iterative refinement*) of our diversification framework. Section IV-A describes the setup along with the datasets, methodology, cluster parameters used for the experiments. In Section IV-B, we evaluate the quality and performance of our two *diversification approaches*, *DM* and *SR*, by changing the environmental and algorithmic parameters, and the dataset size. In Section IV-C, we analyze the accuracy of our *cost model*. Finally, Section IV-D provides an empirical evaluation of the *iterative refinement* component using real life datasets.

A. Setup

All of our experiments are performed on a five node cluster running Hadoop 1.0.4. Each node has 8 cores (3.30GHz Intel Xeon CPU) with 16GB RAM and 1TB of raw disk storage. We configure each node to run 8 tasks (map/reduce) at a time. Thus, at any point of time, we can run at most 40 tasks concurrently on our cluster.

Datasets: We use two datasets for the experiments in this section,

- **TwitterCrawl:** This dataset contains 82,774,328 tweets crawled from Twitter[2]. Each tweet has 8 terms on average. The total size of the dataset on disk is 7.55GB.
- **Image:** This dataset has 79,302,017 feature vectors extracted from collection of images[1]. Each vector has 16 features. The total size on disk is 5.12GB.

Methodology: We randomly select 100 elements from each dataset and use them as the queries. The results shown in this

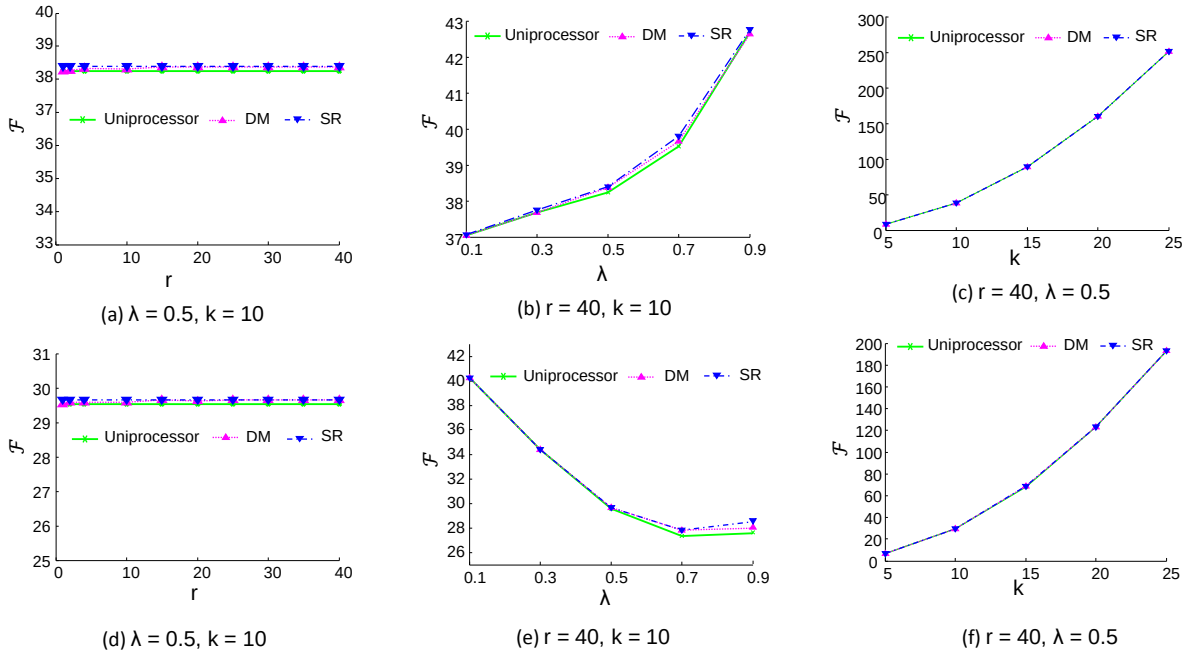

Fig. 6. For *TwitterCrawl* dataset, (a) Avg. $\mathcal{F}$ vs. r, (b) Avg. $\mathcal{F}$ vs. λ, (c) Avg. $\mathcal{F}$ vs. k. (d),(e),(f) show the same figures for *Image* dataset

section are averaged over these 100 queries. We use the same distance metric for rel and dis calculation; *euclidean* distance for *Image* dataset, and *cosine* similarity for *TwitterCrawl* dataset. Note that, the relevance features are always taken from the user in the form of a query and the user always expects relevant answers. However, since the user has given a subset of features, diversification is necessary before presenting the potential large set of relevant results. Therefore, in our experiments, a subset of the feature set is used for rel calculation and the whole feature set is used for dis calculation [23]. For the *Image* dataset, first half of the features are used for rel calculation (i.e. $rel = (1 - L_2(1..8))$), and the whole feature set is used for dis calculation (i.e. $dis = L_2(1..16)$). For the *TwitterCrawl* dataset, three random terms are used for rel calculation (i.e. $rel = cosine(terms)$), and the whole tweet is used for dis calculation (i.e. $dis = 1 - cosine(tweets)$). However, we have also tested other feature combinations for relevance computation which produced similar results, thus are skipped in this paper (due to limited space).

Uniprocessor Diversification: Several algorithms have been proposed in literature for result diversification in a uniprocessor system [22][23][10][24][7]. For our experiments, we use the MMR [7] diversification because of its efficiency within a single node [23]. MMR picks diverse results using the greedy strategy. The first element is always picked as the most relevant one. Successive diverse results are picked (one at a time) that maximize the $\mathcal{F}$ score with respect to the current selected diverse results. This process continues until k diverse results are picked. Note that, recently two other algorithms (GMC and GNE) have been proposed for diversification[23]. These algorithms may provide better diversification quality but require quadratic running time on the size of the data set $\mathcal{D}$. Instead MMR's running time is linear to the size of $\mathcal{D}$.

Cluster Parameters: We perform several experiments on our cluster to estimate the values of the parameters in Table I. Based on our experiments, the values are estimated as, disk rate $D_r = 20MB/sec$, network rate $N_r = 10MB/sec$, $delay(1) = 0.1sec$, dispersion ratio $\beta = 0.8$. The uniprocessor MMR cost is estimated as $costD(bk) = 3.37E^{-6}bk\ sec$, which is the cost to generate k diverse results from b bytes. The mapper cost during the *refine* phase of *SR* approach is estimated as, $costP(bk) = 2.9E^{-7}bk\ sec$. Note that, the mapper processing cost in the first phase of the diversification approaches (*DM* and *SR*) are ignored from our calculation since the values are estimated as close to zero.

In all of our experiments, the number of mappers m is set by the Hadoop framework and we vary the number of reducers r from 1 to 40 (default value 40). We set the sampling ratio such that the number of elements processed in the single reducer during the *sample* phase of *SR* algorithm is ~ 1 million, i.e. $\alpha = \frac{1million}{|\mathcal{D}|}$. The results shown in this section are averaged over 10 distinct runs. Note that, unless specified explicitly, the default value used for k is 10 and for λ is 0.5

B. Evaluation of Diversification Approaches

We proceed with an evaluation of our two diversification approaches, *DM* and *SR*, in terms of quality and performance.

1) Qualitative Analysis: The motivation of the qualitative analysis is to answer the question, *Does the distributed implementation of the diversification algorithm degrades the quality when compared to the uniprocessor algorithm?* Therefore, we compare the quality achieved by the top-k result set generated by our two distributed approaches, *DM* and *SR*, with the quality of the uniprocessor *MMR* algorithm (i.e., a naive implementation where all elements are forwarded to a single reducer which computes the top-k diverse results

using *MMR*). The comparison with a uniprocessor optimal algorithm has been skipped due to the optimal algorithm's time complexity (given that the dataset we consider is in the millions of records). However, [24] contains a comparison between the uniprocessor MMR and the optimal algorithm for a small dataset ($|\mathcal{D}| = 200$), which shows that MMR produced good quality results with respect to the optimal algorithm. We use score ($\mathcal{F}$) as the measurement of quality[23] for our experiments (i.e. higher $\mathcal{F}$ denotes better quality results).

Due to memory constraints and problem complexity, we could run the uniprocessor *MMR* only on a small set of elements. Figure 6 shows the quality comparisons of three algorithms, *DM*, *SR* and uniprocessor *MMR*, using a dataset with 10 million elements. We vary the number of reducers r from 1 to 40 (Figures 6(a),6(d)), the λ values from 0.1 to 0.9 (Figures 6(b),6(e)), the k values from 5 to 25 (Figures 6(c),6(f)) for two datasets *TwitterCrawl*, *Image* (respectively). Note that, since both the *SR* and uniprocessor *MMR* algorithms use only one reducer, the ($\mathcal{F}$) scores are fixed, independent from the changing r values (Figures 6(a),6(d)).

In all the experiments of Figure 6, the DM and SR approaches produce equal or better quality results when compared to the uniprocessor MMR algorithm. This is due to the nature of the DM and SR approaches. Note that, the quality of the final k diverse results produced by the uniprocessor MMR algorithm depends heavily on the first chosen element. If the first element is not chosen properly then the final k diverse results may be of low quality. In comparison, during the *divide* phase of our DM approach, each reducer uses a separate initial element to compute the k-diverse results from the assigned sample. Thus r different initial elements (one in each reducer) are used to compute a total of rk diverse elements. These rk elements form a better candidate set to be considered for diversification since these elements are selected by a uniprocessor MMR (run on different reducers). Therefore, the final k diverse results computed from these rk elements (during the *merge* phase) would be of better quality when compared to the results produced by the uniprocessor MMR on the whole dataset (as seen in most of the experiments of Figure 6).

In the case of the SR approach, a k diverse set is computed using the uniprocessor MMR from a sample of the dataset (in the *sample* phase) which is further refined by an additional scan of the whole dataset (in the *refine* phase). Therefore, the second scan of the dataset improves the quality of the diverse results produced by the SR approach when compared to the results produced by the uniprocessor MMR on the whole dataset.

Figure 7 shows the wall clock time needed by the three algorithms (uniprocessor *MMR*, *DM* and *SR*) for the quality experiments. As seen from the figure, our algorithms, *DM* and *SR*, compute diverse results much faster compare to the uniprocessor *MMR*. In fact, *DM* (*SR*) decreases the running time of uniprocessor *MMR* to a factor of 0.037 (0.119) and a factor of 0.039 (0.107) for the *TwitterCrawl* and *Image* datasets respectively.

Dataset	Wall Clock Time (sec)		
	Uniprocessor	DM (r = 40)	SR
TwitterCrawl	3053.94	113.19	363.49
Image	2193.72	85.1	235.63

Fig. 7. Avg. wall clock time needed by Uniprocessor *MMR*, *DM* and *SR* algorithms for $k = 10$ and $|\mathcal{D}|= 10$ millions

2) Performance Analysis: Figures 8(a),8(c) show the wall clock time needed to compute top-10 diverse results by varying the number of reducers r, 1 to 40, using 10 million elements from the *TwitterCrawl* and *image* datasets respectively. As described earlier, the *SR* approach uses only one reducer, thus the total time taken by this approach is independent from the increasing number of reducers. However, the total time for the *DM* algorithm is decreasing nonlinearly with the increasing r. Note that, the *SR* algorithm performs better for smaller number of reducers $1 \sim 10$. If we increase the number of reducers, then *DM* algorithm starts to dominate.

Figures 8(b),8(d) show the wall clock time needed to compute top-10 diverse results by varying the number of records $|\mathcal{D}|$, 10 to $\sim$$80$ millions, from the *TwitterCrawl* and *Image* datasets respectively. The number of reducers is fixed to 40. As seen from the figures, both of the approaches, *DM* and *SR*, show linear scale-up with the increasing $|\mathcal{D}|$ value. Unlike to the Figures 8(a) and 8(c), *DM* algorithm performs better for small number of elements (up to $\sim$ 25 millions), while *SR* works better for larger dataset.

Therefore, Figures 8(a)-(d) conclude that none of two diversification approaches, *DM* and *SR*, is a universal winner in all scenarios. *DM* performs better for small dataset and large number of reducers, while *SR* works better for larger dataset and small number of reducers. This provides a strong motivation of our *cost model* which has been analyzed in the next subsection.

C. Evaluation of Cost Model

In this subsection, we analyze the accuracy of the cost model. Figures 9(a), 9(d) show the same figures as in Figures 8(a), 8(c) with one more red curve (denoted as *best*) that, using cost model in Section III-B, can consistently pick the best approach at runtime (as shown from figures 9(a), 9(d)). Similar characteristics are found for figures 8(b),8(d), thus skipped in this paper. Figures 9(b), 9(e) (Figures 9(c), 9(f)) show the actual and predicted running times for *DM* (*SR*) algorithm using two datasets *TwitterCrawl* and *Image* respectively. As seen from the figures, the predicted and actual times are close to each other. This confirms the fact that the cluster parameters estimated in Section IV-A are realistic.

D. Evaluation of Iterative Refinement

Our last experiment is to evaluate our iterative refinement component. For the space limitation, only the results of the *TwitterCrawl* dataset are shown in this paper. As discussed in Section III-C, our iterative component is useful when we

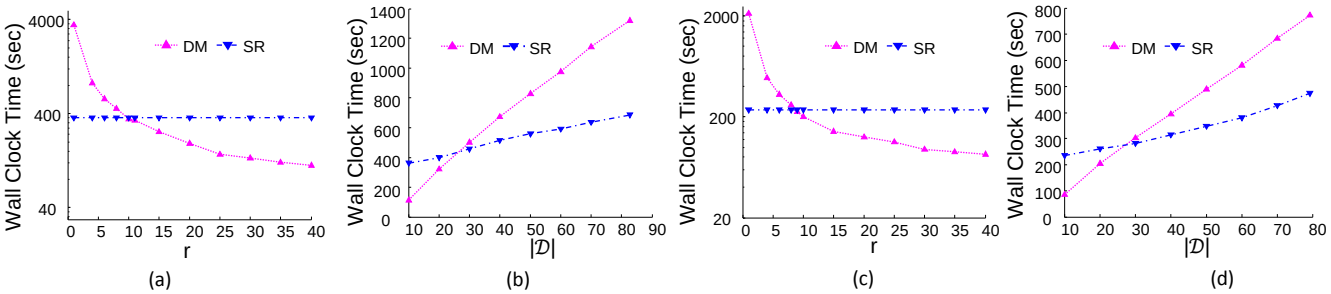

Fig. 8. For *TwitterCrawl* dataset, (a) Avg. wall clock time vs. r, (b) Avg. wall clock time vs. $|\mathcal{D}|$. (c),(d) show the same figures for *Image* dataset

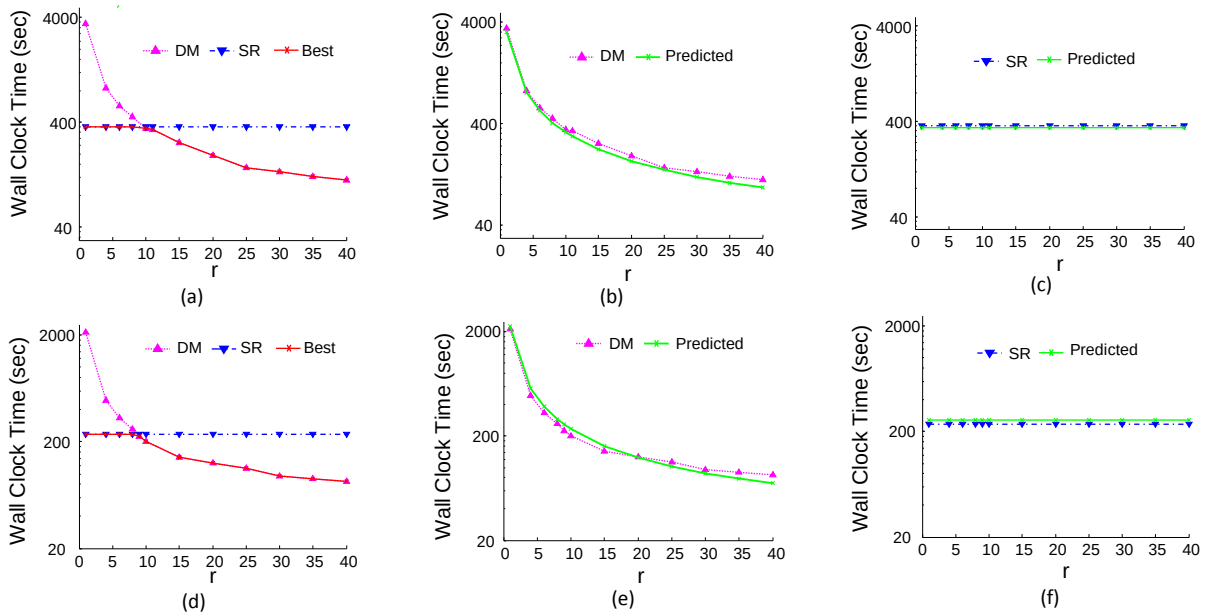

Fig. 9. For *TwitterCrawl* dataset, (a) the ability of the cost model to select the best strategy (red curve) from *DM* and *SR* (b) actual and predicted times for *DM*, (c) actual and predicted times for *SR*. (d),(e),(f) show the same figures for *Image* dataset

have limited amount of resources (e.g. number of reducers, memory). Therefore, in the *divide* phase of DM approach, we set r to 8 and assign a random sample of 0.5 millions to each reducer. Similarly, the single reducer in the *sample* phase of SR approach gets a sample of size 0.5 millions.

Figure 10(a) shows the average values of $\mathcal{F}$ over iterations (0 to 4) for the top-10 diverse results calculated from the whole *TwitterCrawl* dataset. Note that, the scores at iteration 0 denote the $\mathcal{F}(S_{10})$ scores of the 10 diverse results returned from the diversification stage by the two diversification approaches DM and SR. As seen in Figure 10(a), the SR approach produces better quality result compare to the DM approach for limited resources. This is due to the *refine* phase in SR approach which makes a whole pass on $\mathcal{D}$. The DM approach reaches the quality of SR approach at iteration 1. After that the quality remains similar for both of the approaches.

We then estimate the number of iterations needed to converge the *while* loop in Algorithm 4. Figure 10(b) shows our estimated probability to converge vs. number of iterations using 100 random queries selected from the *TwitterCrawl* dataset. As seen from the figure, both of the diversification approaches, *DM* and *SR*, have high probability of convergence within $1 \sim 3$ iterations. The maximum number of iterations needed in our experiment was 7. Therefore, empirically we conclude that, with high probability, our iterative refinement strategy guarantees 2-approximate solution after 3 iterations.

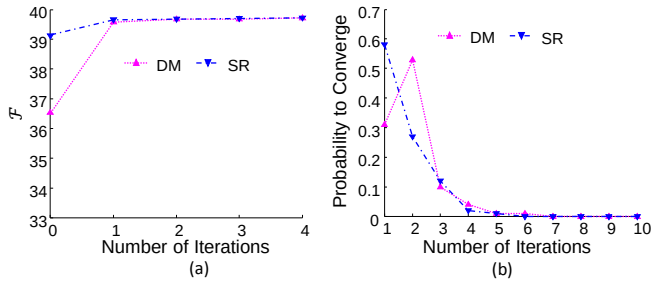

Fig. 10. For *TwitterCrawl* dataset, (a) Avg. $\mathcal{F}$ vs. Number of Iterations, (b) Probability to Converge vs. Number of Iterations

V. Related Work

Diversification has recently been researched heavily for a uniprocessor environment. Several approaches have been proposed in literature for result diversification in many domains. [22][13][17][14] propose diversification in structured databases. [22] introduces diversity ordering between the result attributes of structured data and proposes diversification based on this ordering. [13] proposes diversification based on the

possible interpretation of keyword queries and novelty of the search results. [17] tries to identify a set of differentiating features that help the users to analyze and compare the search results. [14] computes a diverse result set where the results are dissimilar to each other as well as cover the whole dataspace. Diversification has also been applied in recommendation systems [25][24]. [25] proposes topic diversification in the recommendation lists to cover the user's different range of interests. [24] introduces explanation based diversification where the explanation of an item is defined by a set of similar items rated by the user in the past. However, all these works consider online applications of diversification and propose algorithms assuming the size of the dataset is small enough (e.g. thousands of elements) to be stored in a single node's memory, thus fail to diversify large datasets (e.g. millions of elements). In this paper, we consider diversification on massive datasets.

There are also diversification frameworks [6][11] proposed in literature that generate the diverse result set in polynomial time. [6] uses the query log information to measure the ambiguity of query terms (e.g. java, apple, jaguar) and proposes diversification which retrieves k documents that cover different specializations. [11] proposes diversification by proportionality; the number of documents in the final top-k result set covering a particular query aspect is proportional to the popularity of the aspect. However, all these approaches require some prior knowledge (e.g. query log, specializations, query aspects) which may not be available in all applications (when workloads and query information are not known in advance [23]). In this paper, we adapt the general diversification framework described in [23] and propose distributed solution to the problem.

One may argue using an existing clustering approach [9][15], that clusters large datasets using the MapReduce framework, to generate k clusters and then pick one representative from each of the clusters, to provide a diversified result. In fact [21] uses clustering technique to generate diverse result set. However, as pointed out in [23] this approach does not guarantee good quality of diversification. This is because it is not trivial to identify which representative point to select from each cluster for the most diversified result. [23] showed experimentally that obvious choices of picking cluster representatives for the diversified result (k-medoids, etc.) provides low diversification score.

VI. Acknowledgments

This work was partially supported by NSF grants: IIS-0910859 and IIS-1161997.

VII. Conclusion

In this paper, we propose two distributed approaches, *DM* and *SR*, for result diversification on massive datasets. The actual winner of these two approaches depends on the environment parameters and data characteristics. Therefore, we propose a cost model that can choose the best approach at runtime. We also propose an iterative refinement component that iteratively refines the diverse result set returned by the diversification approach and guarantees a 2-approximate solution when converses. Our iterative refinement component is useful when we have limited resources (e.g. limited memory, small number of nodes).

Our discussion considered diversification on a static dataset; it is an interesting open problem whether we can iteratively identify a good quality diversified answer if the dataset changes over time (i.e., a new element is added to $\mathcal{D}$ or an element is deleted). For further future work, we are planning to add diversification on the Asterix platform (both as an extension of AQL and on top of the Hyracks engine) [4].

References

[1] Tiny image dataset, http://horatio.cs.nyu.edu/mit/tiny/data/index.html.
[2] Twitter, https://twitter.com/.
[3] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proc. ACM WSDM*, 2009.
[4] A. Behm, V. R. Borkar, M. J. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, and V. J. Tsotras. Asterix: towards a scalable, semistructured data platform for evolving-world models. *Distributed and Parallel Databases*, 29(3), 2011.
[5] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In *SIGMOD*, 2010.
[6] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri. Efficient diversification of web search results. *VLDB*, 4(7), 2011.
[7] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.
[8] C. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR*, 2008.
[9] R. L. F. Cordeiro, C. T. Junior, A. J. M. Traina, J. López, U. Kang, and C. Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *KDD*, 2011.
[10] M. Coyle and B. Smyth. On the importance of being diverse. In *IIP*. 2005.
[11] V. Dang and W. B. Croft. Diversity by proportionality: an election-based approach to search result diversification. In *SIGIR*, 2012.
[12] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*. 2004.
[13] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. Divq: Diversification for keyword search over structured databases. In *SIGIR*, 2010.
[14] M. Drosou and E. Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *VLDB*, 6(1), 2012.
[15] A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In *SIGKDD*, 2011.
[16] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, 2009.
[17] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. In *PVLDB*, 2009.
[18] F. Radlinski and S. Dumais. Improving personalized web search using result diversification. In *SIGIR*, 2006.
[19] D. Rafiei, K. Bharat, and A. Shukla. Diversifying web search results. In *WWW*, 2010.
[20] A. Rasmussen, M. Conley, R. Kapoor, V. T. Lam, G. Porter, and A. Vahdat. *ThemisMR: An I/O-Efficient MapReduce*. Technical Report CS2012-0983, UCSD, 2012.
[21] R. H. van Leuken, L. Garcia, X. Olivares, and R. van Zwol. Visual diversification of image search results. In *WWW*, 2009.
[22] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, 2008.
[23] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. Traina, and V. J. Tsotras. On query result diversification. In *ICDE*, 2011.
[24] C. Yu, L. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, 2009.
[25] C. N. Ziegler, S. M. Mcnee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, 2005.