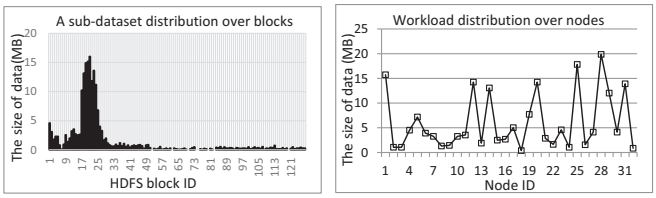
DataNet: A Data Distribution-aware Method for Sub-dataset Analysis On Distributed File Systems

Jun Wang, Jiangling Yin, Jian Zhou, Xuhong Zhang and Ruijun Wang
Department of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL
{jwang, jyin, jzhou, xzhang, ruijun}@eecs.ucf.edu

***Abstract*—In this paper, we study the problem of sub-dataset analysis over distributed file systems, e.g, the Hadoop file system. Our experiments show that the sub-datasets' distribution over HDFS blocks can often cause the corresponding analysis to suffer from a seriously imbalanced parallel execution. This is because the locality of individual sub-datasets is *hidden* by the Hadoop file system and the content clustering of sub-datasets results in some computational nodes carrying out much more workload than others. We conduct a comprehensive analysis on how the imbalanced computing patterns occur and their sensitivity to the size of a cluster. We then propose a novel method to optimize sub-dataset analysis over distributed storage systems referred to as DataNet. DataNet aims to achieve distribution-aware and workload-balanced computing and consists of the following three parts. Firstly, we propose an efficient algorithm with linear complexity to obtain the meta-data of sub-dataset distributions. Secondly, we design an elastic storage structure called ElasticMap based on the HashMap and BloomFilter techniques to store the meta-data. Thirdly, we employ a distribution-aware algorithm for sub-dataset applications to achieve a workload-balance in parallel execution. Our proposed method can benefit different sub-dataset analyses with various computational requirements. Experiments are conducted on PRObEs Marmot 128-node cluster testbed and the results show the performance benefits of DataNet.**

I. Introduction

The advances in sensing, networking and storage technologies have led to the generation and collection of data at extremely high rates and volumes. Large corporations, such as Google, Amazon and Facebook produce and collect terabytes of data with respect to click stream or event logs in only a few hours [28, 22]. In order to ensure system security and gain business intelligence [21] these data usually need to be further grouped or selected for individual analysis. For instance, in recommendation systems and personalized web services, the analysis [15] on the webpage clicks streams needs to perform user sessionization analysis so as to provide better service for each user. Also, in network traffic systems, flow construction [5] based on network traffic traces should differentiate different types of network traffic and conduct analysis accordingly. Also, in business transaction analysis [12], data with specific features are usually selected for fraud detection and risk evaluation. In this paper, the collection of data related to certain events or features is referred to as a **sub-dataset**.



(a) The distribution of a sub-dataset over HDFS blocks. (b) workload distribution over cluster nodes.

Figure 1: The content clustering causes imbalanced data computing in parallel execution.

Currently, the Hadoop file system (HDFS) [26] is the de facto open-source storage system in big data analysis. It can be directly deployed on the disks of cluster nodes for local data access. When storing a dataset, HDFS will divide the dataset into smaller block files and randomly distribute them with several identical copies (for the sake of reliability). In practice, a large dataset may contain millions or billions of sub-datasets such as advertisement clicks or event-based log data [3, 2, 11]. The content of a single sub-dataset can be stored in different HDFS blocks and each block usually contains many sub-datasets.

Unfortunately, most sub-dataset analyses over HDFS have the potential to suffer from a serious imbalance in parallel execution. As shown in previous work [24, 4, 19, 16] sub-datasets pertaining to related topics or features will often be clustered together due to time or spatial locality, e.g. recently uploaded ads/news usually receive clicks at higher rates than older ones. This clustering of related data is referred to as **content clustering** in this paper. Such a phenomenon can result in an imbalanced computing in sub-dataset analysis because analysis tasks, such as MapReduce [26], are usually scheduled based on the HDFS block granularity without considering the distribution of sub-datasets. We will analyze how these imbalanced computing patterns occur and how they are affected by the size of a cluster in Section II-B.

To demonstrate this problem, we launch a job running on a 32-node cluster to perform analysis on a certain movie from a dataset containing millions of movies [11]. The data

1530-2075/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPS.2016.33

is chronologically organized and stored as HDFS blocks, each being 64 MB and containing movies' log data over a period of time. Intuitively, the blocks corresponding to the dates around the time of the release would contain most of our desirable data. The data distribution of the movie within 128 HDFS blocks is shown in Figure 1(a). Clearly, the sub-dataset distribution is far from balanced, e.g, the first 30 blocks contain the most of our desirable data. Since HDFS doesn't maintain the knowledge of sub-dataset distribution, the locality scheduling based on HDFS blocks could result in an imbalanced workload distribution over the cluster or computation nodes during parallel execution as shown in Figure 1(b). This imbalance could seriously degrade the execution performance in many sub-dataset analyses. More examples and detailed configurations are given in Section V.

A great deal of research has been proposed to address the issues associated with logs and sub-dataset processing. Charles [25] proposed two metrics to measure content clustering, topic signatures and collection statistics. Flume [1] is a distributed log collection system that can directly save the collected data into HDFS for MapReduce analysis. Yin [28] et al. proposed a framework that adopts an efficient group-order-merge mechanism to provide faster execution on the event logs. However, these methods do not address the imbalanced distribution of sub-datasets in parallel computing. To address the workload imbalance problem, LIBRA [7] is proposed to adjust the workload among the reducers of MapReduce applications through sampling the intermediate data. SkewTune [17] can mitigate skew in MapReduce applications through observing the job execution and re-balancing workload among the computing resources. However, these methods need to dynamically monitor the task execution and migrate workload for balanced execution during runtime.

In this paper, we aim to give a complete analysis and solve the imbalanced distribution/computing problem of sub-datasets over Hadoop clusters from the source. This will require us to identify the sub-datasets' distribution and foresee the possible imbalanced workload computation among cluster nodes before launching the actual analysis tasks. One challenge to achieve our goal is that obtaining and storing the meta-data pertaining to the distribution of millions or billions of sub-datasets could incur a substantial cost in memory and CPU cycles. Another problem is the overhead involved in the utilization of constructed meta-data. Since sub-datasets with a small amount of data will have a lower probability to cause a workload-imbalance, storing meta-data for all sub-datasets will result in unnecessary computing resource usage.

To address the above challenges, we propose an efficient sub-dataset distribution-aware method called DataNet. DataNet uses an algorithm with a linear time complexity to obtain the sub-datasets' distribution. Instead of precisely maintaining the distribution for all sub-datasets, DataNet adopts an elastic data structure called ElasticMap to capture the approximate distribution of sub-datasets by distinguishing the dominant sub-datasets from the non-dominat sub-datasets within block files. Using ElsaticMap, DataNet enables sub-dataset analysis to achieve balanced computing through a workload-balanced scheduling algorithm. Our contributions in this paper are the following.

- We theoretically analyzed how an imbalanced data distribution caused by the clustering of relevant data within HDFS blocks affects parallel computing.
- We developed an efficient algorithm with linear complexity to obtain the complete distribution of sub-datasets through a single scan.
- We designed a new light-weight data structure called ElasticMap, which can enable applications to quickly obtain the distribution of desired sub-datasets with a low overhead.
- We presented an effective workload-balanced algorithm for parallel sub-dataset analysis. We prototyped and evaluated DataNet against several well-known MapReduce applications. The evaluation results confirm the efficiency and effectiveness of our proposed methods.

The rest of the paper is organized as follows. Section II demonstrates content clustering and formally describes its effect on computational workload-imbalance. Section III describes the design and construction of ElasticMap. Section IV presents a sub-dataset distribution-aware algorithm. Section V shows experimental results. Section VI discusses related work and the final part is the conclusion.

II. Content Clustering and Sub-datasets Imbalanced Computing

A. Sub-Datasets Analysis and Content Clustering

Collecting and analyzing log or event data is important for gaining business intelligence and ensuring system security. For example, the well-known distributed log collection system Flume [1] can directly save log data into a Hadoop File System for distributed analysis. Log or event-based datasets are usually lists of records, each consisting of several fields such as source/user id, log time, destination, etc. To discover knowledge, these data need to be further filtered for individual analysis. Specifically, the sub-dataset $S(e)$ related to a specific event or topic analysis e could be represented as follows,

$$S(e) = \{r | related(r,e), r \in R\} \quad (1)$$

where R is the collection of all log records.

Researches [24, 4, 19, 16] have shown that sub-datasets pertaining to related topics or features will often be clustered together in most large datasets, e.g, the majority of logs for a popular movie would be concentrated around the time of its release. Also, photos or videos recently uploaded to Facebook [24] are often retrieved/commented on at a much higher rate than older ones. In a social network such as LinkedIn or Twitter, users are prompted to group themselves

with others sharing similar skills or interests[4]. Moreover, in graph processing, graph partitioning technologies tend to place highly connected nodes in a single partition and the nodes containing relatively few edges in separate partitions in order to reduce communication between partitions [16].

However, in parallel data analysis applications such as MapReduce, scheduling tasks based on block granularity [26, 30, 29] without the consideration of the sub-datasets' distribution does not result in an optimal scheduling for parallel execution. In the next section, we will present an analysis of the imbalanced workload for sub-dataset processing in parallel execution.

B. Probability Analysis of Imbalanced Workload

Assume a set of parallel processes/executors are launched on an m-node cluster to analyze a specific sub-dataset S, which is distributed among n block files. Due to content clustering, blocks will contain different amounts of data from each sub-dataset. In most situations, the majority of a given sub-dataset is contained in only a few blocks, while other blocks may contain little data related to the sub-dataset. We can model such a distribution using a Gamma distribution, which is widely used to model physical quantities [18, 14] that take positive values such as information or message distribution over time. In our analysis, we let the amount of data contained by each block, X, follow a Gamma distribution $X \sim \Gamma(k,\theta)$, and assume that each X for different blocks is independent. To theoretically discuss the issue of workload imbalance, we suppose that each cluster node performs the analysis on an equal amount of n/m randomly chosen blocks. By taking the summation of the independent random variables X for each block, we obtain the amount of workload processed on a cluster node, Z, which has the distribution $Z \sim \Gamma(\frac{nk}{m},\theta)$ and its density function is

$$f(z;\frac{nk}{m},\theta) = \frac{1}{\Gamma(\frac{nk}{m})\theta^{\frac{nk}{m}}} z^{\frac{nk}{m}-1} e^{-\frac{z}{\theta}} \quad (2)$$

The ideal case is that each cluster node processes the same amount of workload, that is, the expected value $E(Z) = \frac{nk\theta}{m}$. However, with a different number of cluster nodes and data blocks, we could have an imbalanced workload distribution, that is, some cluster nodes process significantly more workload than the average while other cluster nodes process much less than the average. To study this imbalance issue, we compute the cumulative probability of the workload performed by a cluster node as follows.

$$P(Z < w) = \int_{-\infty}^{w} f_Z(t)dt \quad (3)$$

And the probability of a workload greater than w on the node is

$$P(Z > w) = 1 - \int_{-\infty}^{w} f_Z(t)dt \quad (4)$$

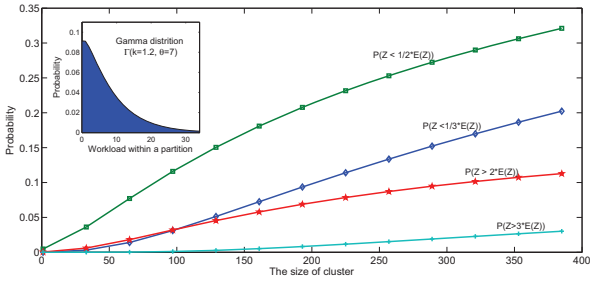


Figure 2: As the size of the cluster increases, more and more cluster nodes tend to have an imbalanced workload.

In general, the probability of the workload size on a cluster node being an extreme value will increase as m increases. For instance, given the value of $k = 1.2, \theta = 7$ and $n = 512$, we can observe from Figure 2 that $P(Z < 1/3 * E(Z))$, $P(Z < 1/2 * E(Z))$, $P(Z > 2 * E(Z))$ and $P(Z > 3 * E(Z))$ will increase with the growth of the cluster size. This implies that a larger number of cluster nodes will result in a higher chance of an imbalanced workload.

The expected number of nodes that will have a workload of at most w is $m * P(Z < w)$ while the expected number of nodes that will have a workload of size greater than w is $m - m * P(z < w)$. Based on the example in Figure 2 and given a cluster size of 128, the expected numbers of nodes that will have a workload of less than $1/2 * E(Z)$ and $1/3 * E(Z)$ are 3.9 and 1.5 respectively; and the expected number of nodes that will have a workload greater than $2 * E(Z)$ is 4.0. This implies that some nodes will have a workload 4 to 6 times greater than others. This confirms the observation shown in Figure 1. On the nodes with larger workloads, a longer execution time is needed to finish the tasks while the nodes with less workload will be idle for a long time before performing the next phase of execution. Experiments in Section V verify the theoretical analysis here that the *imbalanced* distribution could result in an inefficient parallel use of cluster resources and hence a low execution performance.

III. Sub-dataset Distribution Management

The fundamental challenge of DataNet is to create a compact meta-data storage to store the sub-dataset distributions such that blocks with more data from a given sub-dataset will have a higher priority to be considered for workload-balanced computing in comparison with other blocks with less data. In this section, we will present the corresponding solutions for this challenge.

A. ElasticMap: A Meta-data Storage of Sub-datasets Over HDFS Blocks

In order to obtain a sub-dataset distribution as shown in Figure 1(a), DataNet maintains the size of data related to each sub-dataset over block files, that is $|b_i \bigcap s_j|, i = 1,...,n, j = 1,...,m$, where b_i is the set of data records on

the ith block and s_j is one sub-dataset contained by b_i. A simple method of recording this information is to use a table such as a hash map to store the pair $\langle id, quantity \rangle$, which represents the id of a sub-dataset, e.g, source id or event name, and the relative size of data associated with the sub-dataset that resides on a block file. We show an example in Table I, which records the number of reviews corresponding to different movies within a block file.

Table I: The size information of movies within a block file

id	movie_1	moive_2	...	movie_m
# of reviews	3578	3038	...	1

The above method has a memory cost of order $O(n*m)$, where the n is the number of blocks and m is the number of sub-datasets. Since the size information will be stored along with the master node and will be used by the task scheduler to achieve a balanced computational workload as shown in Section IV-B. In the case where the number of sub-datasets is large, the meta-data could incur a high memory cost. Therefore, in our implementation, we develop a data structure called *ElasticMap*, which consists of a hash map together with a bloom filter to store the size information of sub-datasets. In comparison to a hash map, a Bloom Filter uses a bitmap to represent the existence of sub-datasets in a block. Bloom Filters are well-known for space-efficient storage. For example, under a typical configuration, storing a sub-dataset's information over a single block in a HashMap will cost 85 bits while using a bloom filter will cost 10 bits.

Because of content clustering, a small number of sub-datasets could dominate the content of a block file while a large number of sub-datasets could have a small amount of data contained by the block. As the block containing a small amount of a sub-dataset will have a negligible impact on the workload-balance in sub-dataset analysis, a bloom filter is sufficient to provide the information for sub-dataset's tasks assignment. Thus, we design the *ElasticMap* to store the information of dominant sub-datasets in a hash map and store that of non-dominant sub-datasets in a bloom filter. Such a design is very flexible, as we can store all the meta-data into the hash map when the memory is large enough and store most of the information into the bloom filter when the memory is limited.

Let n be the number of block files in a dataset. We maintain an ElasticMap array to record the sub-datasets' distribution information over n blocks. The array has n pointers, each pointing to the meta-data over a block file. Figure 3 shows an example of the data structure, where *id* is the id of a sub-dataset. By querying this structure, we can obtain the distribution of a sub-dataset over all block files.

According to the analysis of a bloom filter [6], if a bloom filter aims to represent sub-datasets with false positive probability ε, the average memory usage per sub-dataset can be expressed as $-\frac{ln(\varepsilon)}{ln^2(2)}$. To evaluate how much memory

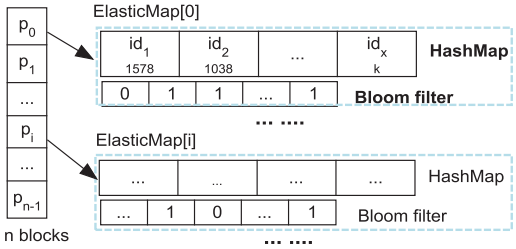


Figure 3: The DataNet meta-data structure over n block files.

space is needed to store an *ElasticMap*, we assume that there are m sub-datasets contained by a block, in which α percent of the sub-datasets will be stored in the hash map and the others are put into the bloom filter. Assume each record in the hash map uses a k-bit representation with the load factor of δ, which is a measure of how full the hash table is allowed to get, the memory cost of the *ElasticMap* on one block is given in Equation 5.

$$Cost(memory) = \frac{m*(1-\alpha)*ln(\varepsilon)}{ln^2(2)} + \frac{m*\alpha*k}{\delta} \quad (5)$$

B. ElasticMap Constructions

The design of the *ElasticMap* needs an efficient method to decide which sub-datasets should be stored into the hash map and which should be stored into the bloom filter. An intuitive method to achieve this is to sort the sub-datasets based on how much of their data is contained by the block file and then store the sub-datasets with larger size values into the hash map and others into the bloom filter. Unfortunately, such a sorting method in the big data era is not efficient, as the time complexity is $O(m \cdot \log m)$, where m is the number of sub-datasets in the block file. In this section, we discuss how to efficiently separate the sub-datasets without sorting.

In order to obtain the size information of sub-datasets over a block b_i, we define a series of size intervals or buckets, and distribute the sub-datasets s_j into the corresponding buckets according to the size $|b_i \cap s_j|$ via a single scan of the block. We maintain a variable S_j for each sub-dataset s_j to compute $|b_i \cap s_j|$. Before scanning, the variable S_j is set to 0. When a data record belonging to sub-dataset s_j is encountered, we increase the variable S_j and adjust the sub-dataset's bucket accordingly. Due to content clustering, the buckets corresponding to larger data sizes will contain a smaller number of sub-datasets. Consequently, we could use non-uniform buckets where larger data sizes have sparser intervals. One instance is the following series of buckets based on fibonacci sequence,

(0,1kb), [1kb,2kb), [2kb, 3kb), [3kb, 5kb), [5kb, 8kb), [8kb, 13kb), [13kb,21kb), [34kb, ∞).

After the scanning is complete, we will have the number of sub-datasets over each bucket. Then, we can decide which sub-datasets should be put into the hash map with the

memory consideration based on Equation 5. Since a block file will be dominated by a small number of sub-datasets and will contain a small amount of data from many other sub-datasets, it is sufficient for us to distinguish dominant sub-datasets using a small number of buckets. For instance, to deal with a block file of $64MB$, one appropriate upper-bound size is $32kb$, since there will at most $64M/32k = 2048$ sub-datasets in the highest bucket, and we may put all of them into the hash map with a small memory cost of around $16kb$. On the other hand, one appropriate lower-bound of the size is $1kb$, since the sub-datasets smaller than $1k$ have little impact on the workload-balance and thus we can put them into a bloom filter. Therefore, tens of buckets could be sufficient to separate the dominant sub-datasets within the block file.

In fact, our algorithm for dominant sub-dataset separation is based on Bucket/Count sorting [10]. However, we are not actually sorting these sub-datasets, we only need to know the statistic value on different buckets to identify the dominant datasets and put them into the hash map. The time complexity of our algorithm is $O(m)$, where m is number of sub-datasets contained by a block. To deal with n blocks, the time complexity is $O(m*n)$, which means only a single scan of the raw data is needed for the meta-data construction.

IV. Sub-dataset Distribution-aware Computing

With the use of ElasticMap, we could identify the imbalanced distribution of sub-datasets before launching the actual analysis tasks. In this section, we will present a distribution-aware method for sub-dataset analysis applications to achieve balanced computing.

A. Sub-dataset Distribution In Hadoop Clusters

Based on the block-locality driven scheduling in Hadoop systems and our analysis in Section II, we can optimize data processing with the knowledge of sub-dataset distribution over HDFS blocks. To achieve this, we build the distribution relationship between cluster nodes and block files, where a block file is mapped to three cluster nodes and different block files contain different amounts of each sub-dataset. The sub-dataset distribution could be retrieved from the ElasticMap during task scheduling.

The distribution relationship with respect to a sub-dataset s is represented as a *Bipartite Graph* $G = (CN, B, E)$, where $CN = \{cn_0, cn_1, ..., cn_n\}$ and $B = \{b_0, b_1, ..., b_m\}$ are the vertices representing the cluster nodes and HDFS block files respectively and $E \subset CN \times B$ is the set of *edges* between CN and B. There exists an edge connecting a computation node $cn_i \in CN$ and a block $b_j \in B$ if and only if b_j is placed on cn_i. There may be several edges connected to a block b_j since a block has several copies stored on different cluster nodes. Each edge is configured with a *weight* equal to $|b_j \cap s|$, the size of the sub-dataset s contained by the block file b_j, which can be obtained through the $ElasticMap$ array.

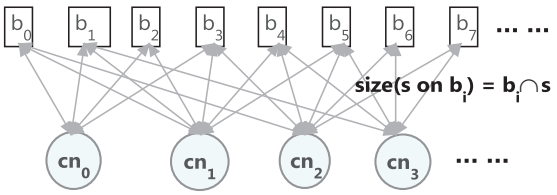


Figure 4: An example of a bipartite graph representing cluster nodes and block files. The edges represent the co-location of block files and cluster nodes, and the weight values of edges adjacent to node b_j is size of the sub-dataset s contained by block file j.

We show a bipartite graph example in Figure 4. The vertices at the bottom represent cluster nodes, while those at the top represent block files. Each edge indicates that a block file b_j is located on a cluster node cn_i with a weight $|b_j \cap s|$. Based on this mapping information, we can optimize sub-dataset assignments according to different computation requirements such as workload balance among cluster nodes or reducing the data transferred for data aggregation.

B. Sub-dataset Distribution-aware Scheduling

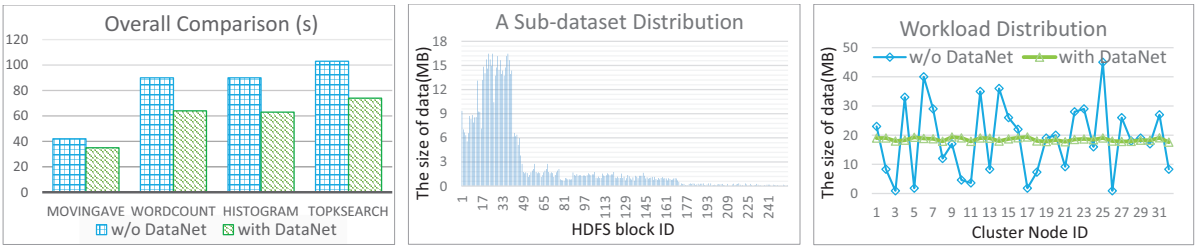
To balance the workload among cluster nodes, we first compute the total size of a sub-dataset as follows,

$$Z = (\sum_{b_j \in \tau_1} |s \cap b_j| + \delta * |\tau_2|) \quad (6)$$

where s is a given sub-dataset, τ_1 is the set of blocks that have the size information of s in the hash map, τ_2 is the set of blocks that have the size information of s in the bloom filter, and δ, the smallest size value of $|s \cap b_j|$, is the approximate data size per block that belongs to sub-datasets stored in the bloom filter. According to the computing capability of computational nodes, we can calculate the amount of sub-datasets to be assigned to each node.

We present a distribution-aware algorithm to balance the workload among computation nodes as shown in Algorithm 1. The algorithm aims to allow each computation node to have an equal amount of workload to be processed. The tasks are assigned with two considerations, the first is to assign local blocks to the requesting computation node i (line 8) while the second is to compare the current workload on node i with the average amount of workload (line 10, 14).

In general, for applications with heavy computational requirements at the map phase, such as similarity computation, Algorithm 1 is useful to balance the parallel execution time. In a homogeneous execution environment, we can actually compute an optimized task assignment through the Ford-Fulkerson method [10]. For applications with aggregation requirements, the output may need to be transferred over the network and finally written into HDFS with several files. For these applications, in which the amount of output could be determined by the size of the input sub-dataset, ElasticMap can also be used to minimize the data transferred



(a) Overall execution comparison. (b) Size of data over HDFS blocks. (c) Workload distribution after selection.

Figure 5: Overall comparisons on a 32-node cluster

Algorithm 1 Distribution-aware Algorithm for Balanced Computing over a Sub-dataset s

Let d_i be the set of blocks adjacent to cluster node cn_i in the bipartite graph G.
Let $T = \{t_0, t_1, ..., t_{n-1}\}$ be the set of tasks corresponding to the n blocks.
Let $|b_j \bigcap s|$ be the size of sub-dataset s in block j.
Let W_i be the current workload on cluster node cn_i.
Steps:
Compute the average workload $\overline{W} = (\sum_{b_j \in \tau_1} |s \cap b_j| + \delta * |\tau_2|)/m$, where m is the total number of cluster nodes
while $|T| \neq 0$ **do**
  if a worker process on cn_i requests a task **then**
    if $|d_i| \neq 0$ **then**
      Find $b_x \in d_i$ such that
      $x = \underset{x}{\text{argmin}} |W_i + b_x \bigcap s - \overline{W}|$
      Assign t_x to the requesting process on node i
    else
      Find $t_x \in T$ such that
      $x = \underset{x}{\text{argmin}} |W_i + b_x \bigcap s - \overline{W}|$
      Assign t_x to the requesting process on node i
    end if
    Remove t_x from T
    for all cn_k adjacent to b_x in G **do**
      Remove the edge (cn_k, b_x) from G
    end for
  end if
end while

with the knowledge of sub-dataset distributions. We leave the optimization of the sub-dataset transfer problem as a future work.

V. Experimental Results and Evaluations

We conduct comprehensive experiments on *Marmot* to show the benefits of DataNet in parallel big data computing with the MapReduce programming model. *Marmot* is a cluster of the PRObE on-site project [13] that is housed at CMU in Pittsburgh. The system has 128 nodes / 256 cores and each node in the cluster has dual 1.6GHz AMD Opteron processors, 16GB of memory, Gigabit Ethernet, and a 2TB Western Digital SATA disk drive. For our experiments, all nodes are connected to the same switch. The Hadoop system is configured as follows: one node is designated to be the NameNode/JobTracker, one node is the secondary NameNode, and other cluster nodes are the DataNodes/TaskTrackers. HDFS is configured with 3-way replication and the size of a chunk file is set to 64 MB.

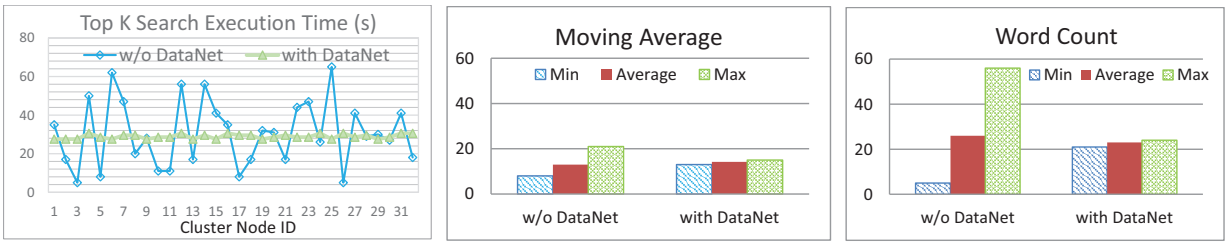
A. DataNet Evaluation

To test DataNet for sub-dataset analysis under Hadoop frameworks, we first launch map tasks to filter out our target sub-dataset and store them locally on the cluster nodes. Then, we run various analysis jobs with different computation patterns to process the filtered sub-dataset and compare their performance. We employ two methods for the map task assignments. The first method (without DataNet) is the default block locality-driven scheduling used by the Hadoop system [26]. The second method (with DataNet) is our proposed distribution-aware method introduced in Section IV-B. For the meta-data stored in ElasticMap, we set the value of α in Equation 5 to 0.3. We will specifically discuss the performance of ElasticMap as α changes in Section V-B.

We implement the following analysis jobs with the MapReduce interface.

- *Moving Average*: analyzing data points by creating a series of averages over intervals of the full dataset. Moving Average is often implemented in the analysis of trend changes and can smooth out short-term fluctuations to highlight longer-term cycles.
- *Top K Search*: finding K sequences with the most similarity to a given sequence. This algorithm needs heavy computation due to the similarity comparison between sequences.
- *Word Count*: reading the sub-dataset and counting how often words occur. Word Count is one of the representative MapReduce benchmark applications.
- *Aggregate Word Histogram*: computing the histogram of the words in the input sub-dataset. This is a fundamental plug-in operation in the MapReduce framework.

In our experiments, we mainly use a dataset consisting of movie ratings and reviews stored in chronological order in



(a) The map execution time distribution (b) The Moving Average map time(s). (c) The Word Count map time(s).

Figure 6: The map execution time(s) comparison on the filtered sub-dataset.

HDFS. The dataset is based on the distribution of the movie names, ratings and categories of "MovieLens" [11]. The text reviews are randomly generated and we also randomly duplicate some ratings for large-scale tests. The total number of block files is 256.

1) Overall Comparison: The overall execution times for the four analysis jobs from 32 cluster nodes are shown in Figure 5(a). As we can see, in all cases, the parallel execution time with the use of DataNet is smaller than that without DataNet. This can be explained by the fact that, without the use of DataNet, certain nodes will have a heavier workload than others, resulting in longer execution times and degrading the overall performance. Besides, we find that DataNet can achieve greater improvements for computationally intensive applications such as TopK Search in comparison to MovingAverage. In all, with DataNet, the improvements of MovingAverage, WordCount, Histogram and TopKSearch are 20%, 39.1%, 40.6% and 42% respectively.

Figure 5(b) shows the sub-dataset distribution over the HDFS blocks, where a small number of blocks contain most of our target data due to content clustering; that is, most reviews about a movie are clustered around the time of the release. Figure 5(c) shows the workload corresponding to the size of the filtered sub-datasets over the cluster nodes. As we can see, without DataNet, the workload of node 25 was significantly higher than the workload of node 17. Such a distribution is far from being balanced for the subsequent analysis and this explains the performance gain in Figure 5(a).

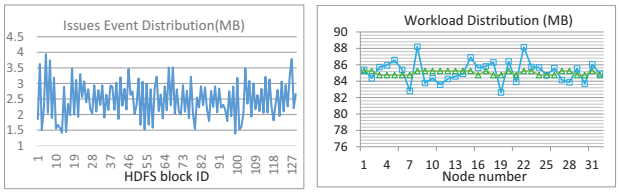
2) Map Execution Time on the Filtered Sub-dataset: To gain further insight into the performance, we monitor the map execution time comparison on the filtered data for the sub-dataset analysis jobs. Figure 6(a) shows the local execution time of Top K Search on all 32 cluster nodes. From the figure, we can find that the slowest execution time is 64 seconds while the fastest execution time is 5 seconds. This could lead to a longer synchronization time to execute the next analysis phase and result in a longer overall execution as shown in Figure 5(a).

With different computational requirements, the imbalanced workload could have different effects on the performance of analysis jobs. To demonstrate this, In Figure 6(b)(c), we show the min, average and max execution times on the the filtered sub-dataset for Moving Average and Word Count on 32 nodes. From the figure, we can find that the gap between the min and max times for Moving Average is much smaller than that of Word Count. This is because Word Count needs to combine words while Moving Average only needs to iterate the data. Therefore, with greater computational requirements, the issue of imbalance becomes more serious.

3) Shuffle Execution Time Comparison: The shuffle phase [26] starts whenever a map task is finished and ends when all map tasks have been executed. We expect that the the shuffle time would be much longer with an imbalanced workload among the cluster nodes. To demonstrate how the imbalance affects the shuffle phase, we collect the min, average and max execution times for shuffle tasks in the Top K Search and Word Count analyses, and show the comparison in Figure 7. From the figure, we can find that the shuffle phase without the use of DataNet takes 4-5X longer than with DataNet. We also find that the speedup of Top K Search is greater than that of Word Count. This is because the Top K Search takes more time for map execution as shown in Figure 6.

Figure 7: The execution time(s) comparison at shuffle phase.

4) More Results and Discussion: We also run experiments on GitHub event log data [2]. The datasets provide more than 20 event types ranging from new commits and fork events to opening new tickets, commenting, and adding members to a project. The size of the raw data is around 34 GB. The experimental setting is the same as with the movie data. We run analysis jobs on "IssueEvent". Figure 8(a) shows the sub-dataset distribution on the first 128 HDFS blocks. As we can see, the sub-dataset distribution doesn't satisfy the property of content clustering. However, since the distribution over HDFS blocks is imbalanced, with the use of



(a) Size of data in HDFS blocks (b) Workload distribution

Figure 8: The imbalanced data distributions and workload of IssueEvent from GitHub data.

ElasticMap, we still can optimize task assignment to meet the balanced computation requirement using Algorithm 1. Specifically, to run the Top K Search job, the longest map execution time is 125 seconds without the use of DataNet and 107 seconds with DataNet. However, we find that the overall improvement is much less than that of the movie dataset. This is because the movie dataset has a more imbalanced sub-dataset distribution due to content clustering, which could cause a more imbalanced workload distribution when scheduling tasks without the distribution knowledge provided by ElasticMap. This can be seen by comparing Figure 6(c) and Figure 8(b).

In order to achieve a workload-balance computation for parallel execution, an alternative method is to dynamically monitor the runtime status [17] and migrate workloads when necessary. Specifically, for sub-dataset processing, we can rebalance the sub-dataset distribution among cluster nodes after the map task's execution on the sub-dataset selection. With the example without DataNet in Figure 5(c), we find that almost every cluster node will transfer or receive sub-datasets and the overall percentage of data migration is more than 30%. Besides the overhead of collecting statistics and adjusting workload during runtime, the data migration could occupy the network resource and prolong the overall execution in comparison with DataNet, which can foresee the imbalanced issue in advance. In comparison, DataNet will scan the raw data once to build all sub-dataset distributions, while the method of dynamic adjugement will migrate the workload for each sub-dataset analyses during runtime. We will specifically discuss the efficiency of DataNet in the next section.

B. Efficiency of DataNet

1) Memory Efficiency and Accuracy of ElasticMap: The goal of ElasticMap is to store the information of sub-datasets' distribution in a compact fashion. With different data distributions, the memory cost on the meta-data storage could vary. For datasets with a high degree of content clustering or a limited number of events, such as GitHub event logs, the size ratio of raw data to meta-data could be very large. We studied the memory efficiency of ElasticMap over the movie dataset and show the results in Table II. The first column of the table represents the percentage of elements stored in the hash map. The last column represents the size ratio of raw data to meta-data. The second column represents the accuracy of ElasticMap calculated as,

$$\chi = 1 - \frac{\sum_{b_j \in \tau_1} (|S \cap b_j| + \delta * |\tau_2|) - Size\_of\_raw\_data}{Size\_of\_raw\_data}$$

where S is the union of all sub-datasets, and b_j, τ_1, τ_2, and δ have the same meaning as in Equation 6

α in Equation 5	Accuracy(χ)	Representation ratio
51%	97%	1857
40%	93%	2270
31%	88%	2751
25%	83%	3196
21%	80%	3497

Table II: The efficiency of ElasticMap

As we can see, for cases in which a small percentage of elements are stored in the hash map, there is a higher representation ratio but a lower overall accuracy. For example, in the case where 21% of the elements are stored in the hash map, 1 MB of meta-data in the ElasticMap can represent around 3497 *MB* of raw data. On the other hand, in the case where 51% of the elements are stored in the hash map, the overall accuracy of DataNet rises to 97% but the representation ratio dropped to 1857. This is due to the fact that the bloom filter can only indicate the existence of a sub-dataset within a block rather than the real size of the sub-dataset.

We also perform accuracy evaluations on individual movies with different sizes. The results are shown in Figure 9. As we can see, for sub-datasets with larger sizes, the difference between the actual size of the sub-dataset and the size calculated through Equation 6 is smaller. This is because the sub-datasets are dominant on most blocks and so they are precisely recorded in the hash map.

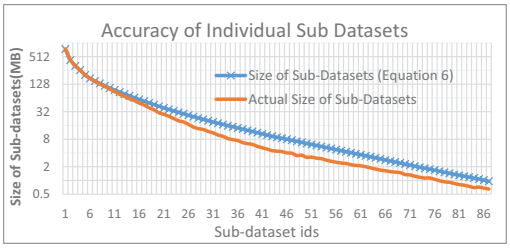


Figure 9: The accuracy of ElasticMap with respect to different sub-datasets.

A greater difference occurs for sub-datasets with a size less than 32 MB. This could be explained by the fact that these sub-datasets are not dominant on most HDFS blocks and they are inaccurately recorded in the bloom filter. Nevertheless, as these sub-datasets have little data, there will be a lower probability for them to cause imbalanced computing. On the other hand, with the knowledge of ElasticMap, we can reduce the I/O cost, since we don't need to process

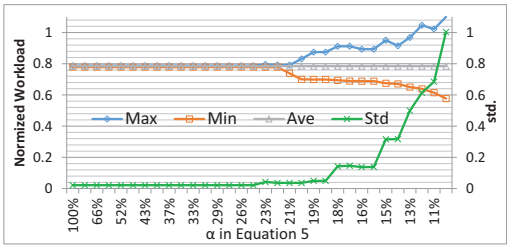


Figure 10: Balancing evaluation; the comparison of maximum, minimum, average workload and the std deviation on 32 compute nodes with different α in Equation 6.

blocks that don't contain our target data (no records in the hash map and bloom filter).

In a real-time or interactive execution environment, recording the meta-data in memory could be efficient. However, as the problem size becomes extremely large, the meta-data may not be able to reside in memory. In such cases, the meta-data can be stored into a database or distributed among multiple machines. We leave this problem as future work and focus on the study and analysis of the imbalanced sub-dataset distribution and computing over Hadoop clusters.

2) The Degree of Balanced Computing: When more sub-datasets are stored in the hash map with a higher memory cost, a higher accuracy could be achieved. This can produce a better workload balance through distribution-aware scheduling. We also run experiments with different α values and study the degree of load balance. The test sub-dataset shares the same distribution in Figure 5. The results are shown as Figure 10. From the Figure, we can find that with only about 15% of the sub-datasets recorded in the hash map, DataNet is able to achieve a satisfactory workload balance, i.e. the max workload is around 0.9 while the min is around 0.7. Changing the percentage from 15 *to* 100 will have little effect on workload balance. The main reason behind these results is that content clustering is the main cause of workload imbalance, and with about 15% of the sub-datasets stored in the hash map, these clustered data could be detected and thus handled using Algorithm 1.

VI. Discussion and Related Works

To provide faster execution on the log files, Yin [28] et al. proposed a framework with a group-order-merge mechanism and Logothetis [23] et al. proposed a in-situ MapReduce architecture to mine the data "on location". To efficiently process ordered datasets in HDFS, Chen [8] et al. proposed a bloom filter-based approach to avoid unnecessary sorting and improve the performance of MapReduce. VSFS [27] is a searchable distributed file system for addressing the needs of data filtering at the file system-level. HBase is an open source implementation of Google's BigTable and the bloom filter used by HBase can greatly reduce the I/O cost during data selection. However, none of these methods address the sub-datasets' imbalanced distribution in parallel computing.

There have also been researches proposed to address data skew problem for MapReduce applications. LIBRA [7] addresses the data skew problem among the reducers of MapReduce applications through sampling the intermediate data. SkewTune [17] can mitigate skew in MapReduce applications through observing the job execution and re-balancing workload among the computing resources. Coppa [9] designs a novel profile-guided progress indicator which can predict data skewness and stragglers so as to avoid excessive costs. CooMR [20] is a cross-task coordination framework that can enable the sorting/merging of Hadoop intermediate data without actually moving the data over the network. DataNet is orthogonal to these techniques and can proactively address the imbalanced computing through its sub-dataset distribution aware algorithm.

VII. Conclusion

In this paper, we investigate the issues of imbalanced sub-dataset analysis over a Hadoop cluster. Due to the missing information of sub-datasets' locality, the content clustering inherent in most sub-datasets prevents applications from efficiently processing them. Through a theoretical analysis, we conclude that an uneven sub-dataset distribution almost always leads to a lower-performance in parallel data analysis. To address this problem, we propose DataNet to support sub-dataset distribution-aware computing. DataNet uses an elastic structure, called ElasticMap, to store the sub-dataset distributions. Also, a dominant sub-dataset separation algorithm is proposed to support the construction of ElasticMap. With the use of DataNet, sub-dataset analyses can easily balance their workload among computational nodes. We conduct comprehensive experiments for different sub-dataset applications with the use of DataNet and the experimental results show the promising performance of DataNet.

Acknowledgments

This work is supported in part by the US National Science Foundation Grant CCF-1527249, CCF- 1337244 and National Science Foundation Early Career Award 0953946.

This material is based upon work supported by the National Science Foundation under the following NSF program: Parallel Reconfigurable Observational Environment for Data Intensive Super-Computing and High Performance Computing (PRObE).

References

[1] Flume: Open source log collection system. https://flume.apache.org/.

[2] Github events. https://www.githubarchive.org/.

[3] World cup 1998 dataset. http://goo.gl/2UqlS.

[4] J.-E. Asbury. Overview of focus group research. *Qualitative health research*, 5(4):414–420, 1995.

[5] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt. A comparative analysis of web and peer-to-peer traffic. In *Proceedings of the 17th International*

Conference on World Wide Web, WWW '08, pages 287–296, New York, NY, USA, 2008. ACM.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[7] Q. Chen, J. Yao, and Z. Xiao. Libra: Lightweight data skew mitigation in mapreduce. *Parallel and Distributed Systems, IEEE Transactions on*, 26(9):2520–2533, Sept 2015.

[8] Z. Chen, D. Wu, W. Xie, J. Zeng, J. He, and D. Wu. A bloom filter-based approach for efficient mapreduce query processing on ordered datasets. In *Advanced Cloud and Big Data (CBD), 2013 International Conference on*, pages 93–98, Dec 2013.

[9] E. Coppa and I. Finocchi. On data skewness, stragglers, and mapreduce progress indicators. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 139–152, New York, NY, USA, 2015. ACM.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[11] S. Dooms, T. De Pessemier, and L. Martens. Movietweetings: a movie rating dataset collected from twitter. In *Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys 2013*, 2013.

[12] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 53–62. ACM, 1999.

[13] G. Gibson, G. Grider, A. Jacobson, and W. Lloyd. Probe: A thousand-node experimental cluster for computer systems research. volume 38, June 2013.

[14] J. Gill. *Bayesian methods: A social and behavioral sciences approach*, volume 20. CRC press, 2014.

[15] J. Hu, H.-J. Zeng, H. Li, C. Niu, and Z. Chen. Demographic prediction based on user's browsing behavior. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 151–160, New York, NY, USA, 2007. ACM.

[16] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. A study of skew in mapreduce applications. *Open Cirrus Summit*, 2011.

[17] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skewtune: Mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 25–36, New York, NY, USA, 2012. ACM.

[18] J. F. Lawless. *Statistical models and methods for lifetime data*, volume 362. John Wiley & Sons, 2011.

[19] Y. Le, J. Liu, F. Ergun, and D. Wang. Online load balancing for mapreduce with skewed data input. In *INFOCOM, 2014 Proceedings IEEE*, pages 2004–2012. IEEE, 2014.

[20] X. Li, Y. Wang, Y. Jiao, C. Xu, and Y. Weikuan. Coomr: Cross-task coordination for efficient data management in mapreduce programs. In *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pages 1–11, Nov 2013.

[21] G. S. Linoff and M. J. Berry. *Data mining techniques: for marketing, sales, and customer relationship management*. John Wiley & Sons, 2011.

[22] D. Logothetis, C. Trezzo, K. C. Webb, and K. Yocum. In-situ mapreduce for log processing. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, pages 9–9, Berkeley, CA, USA, 2011. USENIX Association.

[23] D. Logothetis, C. Trezzo, K. C. Webb, and K. Yocum. In-situ mapreduce for log processing. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, pages 9–9, Berkeley, CA, USA, 2011. USENIX Association.

[24] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, et al. f4: Facebooks warm blob storage system. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, pages 383–398. USENIX Association, 2014.

[25] C. L. Viles and J. C. French. Content locality in distributed digital libraries. *Information Processing & Management*, 35(3):317 – 336, 1999.

[26] T. White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[27] L. Xu, Z. Huang, H. Jiang, L. Tian, and D. Swanson. Vsfs: A searchable distributed file system. In *Proceedings of the 9th Parallel Data Storage Workshop*, PDSW '14, pages 25–30, Piscataway, NJ, USA, 2014. IEEE Press.

[28] J. Yin, Y. Liao, M. Baldi, L. Gao, and A. Nucci. A scalable distributed framework for efficient analytics on ordered datasets. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, UCC '13, pages 131–138, Washington, DC, USA, 2013. IEEE Computer Society.

[29] J. Yin, J. Wang, W.-c. Feng, X. Zhang, and J. Zhang. Slam: Scalable locality-aware middleware for i/o in scientific analysis and visualization. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 257–260, New York, NY, USA, 2014. ACM.

[30] J. Yin, J. Wang, J. Zhou, T. Lukasiewicz, D. Huang, and J. Zhang. Opass: Analysis and optimization of parallel data access on distributed file systems. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 623–632, May 2015.