Proceedings Paper:
Ouyang, X, Zhou, H, Clement, S orcid.org/0000-0003-2918-5881 et al. (2 more authors) (2018) Mitigate data skew caused stragglers through ImKP partition in MapReduce. In: IEEE International Performance, Computing and Communications Conference, Proceedings. 36th IEEE International Performance Computing and Communications Conference (IPCCC), 10-12 Dec 2017, San Diego, California, USA. Institute of Electrical and Electronics Engineers . ISBN 978-1-5090-6468-7

https://doi.org/10.1109/PCCC.2017.8280475

Mitigate Data Skew Caused Stragglers through ImKP Partition in MapReduce

Xue Ouyang*†, Huan Zhou†, Stephen Clement*, Paul Townend* and Jie Xu*
*School of Computing, University of Leeds, UK
Email: {scxo, P.M.Townend, S.J.Clement, J.Xu}@leeds.ac.uk
†Department of Computer Science and Technology, National University of Defense Technology, China
Email: huanzhou@nudt.edu.cn

***Abstract*—Speculative execution is the mechanism adopted by current MapReduce framework when dealing with the straggler problem, and it functions through creating redundant copies for identified stragglers. The result of the quicker task will be adopted to improve the overall job execution performance. Although proved to be effective for contention caused stragglers, speculative execution can easily meet its bottleneck when mitigating data skew caused stragglers due to its replication nature: the identical unbalanced input data will lead to a slow speculative task. The Map inputs are typically even in size according to the HDFS block configuration, therefore the skew caused stragglers happen mainly in the Reduce phase because of the unknown intermediate key distribution. In this paper, we focus on mitigating data skew caused Reduce stragglers, propose ImKP, an Intermediate Key Pre-processing framework that enables the even distributed partition for Reduce inputs. A group based ranking technique has been developed that dramatically decreases the pre-processing time, and ImKP manages to eliminate this timing overhead through parallelizing the pre-processing with the file uploading procedure (from local file system to HDFS). For jobs that take input directly from HDFS, ImKP minimizes the overhead by storing the $< GroupedKey, Reducer >$ mapping result on every node within the cluster for reuse. Experiments are conducted on different datasets with various workloads. Results show that, compared to the popular hash partition, ImKP can dramatically decrease Reduce skew, achieving a 99.8% reduction in the coefficient of variation of the input sizes in average, and improve up to 29.37% job response performance.**

***Keywords*—Stragglers, MapReduce, Skew-handling, Partition.**

I. Introduction

The MapReduce framework proposed in 2008 [1] has now become the de facto platform to support large-scale parallel processing and data analytics in production systems. Under this framework, the computation of a job is divided into several sub-tasks running on distributed machine nodes. Map phase is prior to the Reduce phase, and within each phase, sub-tasks are executing in a parallelized manner. There are many challenges in efficient MapReduce job execution, and the straggler problem is one of them [2]. In this paper, stragglers refer to the tasks that are suffering from unexpected slow execution and exhibit obvious longer duration compared to other parallelized tasks within the same phase of the MapReduce job.

The dominant method to mitigate stragglers is speculative execution [1], which functions via redundant methodology in a "detection-reaction" manner. Once a straggler is identified, a replication task will be generated to compete with the straggler, and it is expected that the speculative copy will complete prior to the straggler to save the overall job execution time. In addition, in order to avoid the identification delay, some methods push the speculative execution to extreme and functions in a pure cloning based manner [3], in which each sub-tasks will have two running copies co-exist within the system and only the quickest result will be adopted.

These existing redundant based mechanisms achieved some level of success. For example, the speculative execution is officially adopted by Hadoop and is deployed in production systems including Google, Facebook, Yahoo!, etc. [4]. However, the efficiency of such methods is still far from perfect: observations reported in [5] based on Yahoo! data reveal that as many as 90% speculations are actually useless - finish after the original stragglers and end up being killed by the system. Wasted replications make no contributions to improve job execution and can deteriorate system availability. The reason behind such low speculation efficiency varies. For example, some may due to the late identification of stragglers that leaves limited time for the duplication to catch up. Another source of unreasonable speculation is created when dealing with skew caused stragglers. Due to the fact that the replication has to process identical input with the straggler, the speculative copy will still suffer from long execution. And unlike the late detection which can be improved by advanced speculation, the skew caused straggler is the unavoidable bottleneck that requires additional solutions.

Many real world applications exhibit large amounts of data skew [6], and there is much literature that focuses on handling skews in MapReduce framework. For Map, the parallelization process is usually automatic, dependent on the input size and HDFS [7] block size, therefore Map skews can be addressed by further splitting the expensive map tasks or adjusting the HDFS data chunk size. In contrast, skews in the Reduce phase is much more challenging: for an arbitrary application, the distribution of the intermediate data cannot be determined ahead of time. In this paper, we analyze the skew behaviors with various data sets under MapReduce and illustrate their influence toward efficient job execution.

In order to mitigate the Reduce skews, we propose ImKP, the Intermediate Key Pre-processing framework that enables the even distribution of Reduce inputs. By inserting a pre-

processing layer prior to the original Map phase, the mapping decision of $< intermediateKey, Reducer >$ can be directly used by the corresponding ImKP even partitioner. In addition, a group based ranking technique is adopted by ImKP that dramatically cuts back the pre-processing calculation timing overhead, and storing the file of $< GroupedKey, Reducer >$ instead of $< intermediateKey, Reducer >$ saves the space as well as the accessing latency. Another optimization of parallelizing the pre-processing with the file uploading phase is implemented to eliminate the overhead for workloads that take inputs from local file system rather than HDFS. Results show that ImKP is capable of decreasing the skewness of Reducer inputs by 99.8% in average (through the measurement of coefficient of variation), and achieved an improvement of up to 29.37% in job response time.

The rest of the paper is structured as follows: Section 2 presents the related work; Section 3 discusses the problem description; Section 4 illustrates the ImKP design to cope with the reduce skew; Section 5 discusses the experiments setup and results; Section 6 introduces conclusions and future work.

II. RELATED WORK

The MapReduce job performance degradation due to the straggler problem is widely discussed in recent years with speculative execution [1] to be the dominant straggler mitigation method deployed in production systems. It monitors the progress of each parallel task and launches redundant task replicas for identified stragglers with the assumption that the speculation will surpass the original task. There exist numerous techniques which extend speculative execution in terms of specified cases such as LATE [8] for heterogeneous nodes environments. It adopts the metrics of the Longest Approximate Time to End instead of the traditional progress score for MapReduce jobs to enhance the straggler detection precision. Dolly [3] is designed for small jobs with less than 10 parallel tasks, adopting full cloning instead of creating speculations only for identified stragglers.

While these speculation based works are shown to be effective in mitigating stragglers caused by reasons such as resource contention or hardware heterogeneity, they encounter unavoidable bottleneck when dealing with data skew caused stragglers: due to the duplication nature, the replica task processes identical input file with the skew straggler will still suffer from the uneven input distribution. It is shown that a lot of stragglers in MapReduce framework are caused by the curse of skew: the Zipf distribution of the input or intermediate data [6]. In order to alleviate this bottleneck, MCP [9] improves speculation by deliberately avoid creating task copies for skew caused stragglers, however, this avoidance base method did not mitigate the skew at all.

For Reduce skew handling approaches, Co-worker [10] functions in a way that as long as a straggler is identified, the reserved co-worker task will help process the remaining data. Its effectiveness is dependent on the choice of the reserved co-worker number, and will introduce resource overhead when there is no skew. SkewTune [11] is another popular skew mitigation method that works through re-partition. As long as there is a free slot within the system, the task with the greatest remaining time will be re-partitioned. However, the Reduce outputs of both these two methods have to be reconstructed due to the fact that the MapReduce requires all tuples sharing the same key to be dispatched to the same Reducer, and this reconstruction introduces additional complexity.

There are some methods rely on node performance when dealing with Reduce skews. For example, the work detailed in [12] splits the cluster into two groups depending on machine processing capacity. The intermediate data number per Reducer is counted. As long as the number for a certain Reducer surpasses a threshold, this Reducer will be assigned to quick nodes for execution. However, the threshold to decide the skew level differs with different workloads, and the coarse grain node classification is limited for efficient skew mitigation.

The most popular methodology for Reduce skew mitigation focuses on optimized partition approaches to distribute the intermediate keys to Reducers. Hash and range are two of the most common partition methods. Hash partition is relatively straight forward, only requires the Reducer number to generate the $< intermediateKey, Reducer >$ mapping decision through hash calculation, while range partition requires the developer to know the data distribution, therefore sometimes needs sampling. LIBRA [13] is the representative work of this type. It launches selected sample Map tasks first to estimate the intermediate data distribution for partition decision making. However, the efficiency of this method is largely dependent on the estimation accuracy, which varies with different sampling strategies, sample portion selections, etc.

III. BACKGROUND AND PROBLEM DESCRIPTION

Before presenting our algorithm, it is necessary to introduce the background of the MapReduce framework, detailing the types of skews and analyzing why skew handling is important as a complementary part for straggler mitigation schemes. The notions used in the paper are defined as follows: the cluster is consisting of M machine nodes with MapReduce jobs J running on it. Each job J has N subtasks running in parallel, where T_j^{Mi} and T_j^{Ri} represents the i_{th} Map and Reduce task belonging to J respectively. It is assumed that subtasks in the same phase from the same job exhibiting non-straggler behavior have similar response time. The duration of T_j^{Mi} and T_j^{Ri} are defined as the time between scheduling and completion, represented as D_j^{Mi} and D_j^{Ri}, respectively.

A. The MapReduce Framework

MapReduce is the de facto model for applications that process vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner [1], responsible for scheduling, monitoring tasks and re-executing the failed ones automatically. YARN is the second version of Hadoop [14] - the most popular open source version of MapReduce, which consists of a master ResourceManager, a NodeManager per node, and an AppMaster per application [15]. MapReduce jobs J usually split inputs into inde-

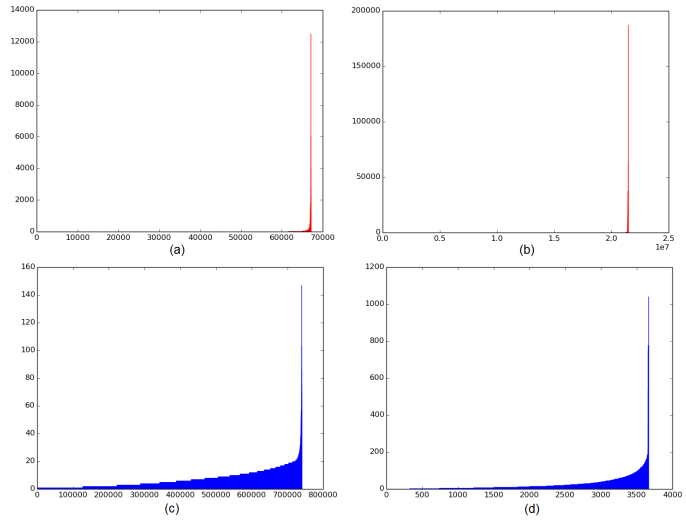

Fig. 1. The word distribution of (a) the Shakespeare collection, (b) the English wiki dataset, and the edge number distribution of (c) the Google web dataset, (d) the Facebook social circles dataset

pendent chunks stored on HDFS, which are then processed by T_j^{Mi}s in a parallel fashion. T_j^{Ri}s take the Map outputs as inputs to generate the final result.

The overall flow of a typical MapReduce job J is as follows: (input) $< K_1, V_1 > \rightarrow$ Map $\rightarrow$ $[< K_1, V_1 >]$ $\rightarrow$ combine $\rightarrow < K_1, [V_1] > \rightarrow$ partition $\rightarrow < K_1, [V_1], R > \rightarrow$ shuffle $\rightarrow < K_2, [V_2] > \rightarrow$ Reduce $\rightarrow$ $[< K_2, V_2 >]$ (output). The T_j^{Mi}s transfer the input file into key value pairs with the customized keys defined by the application developer. For example, in a WordCount job which counts the frequency of each word in a document, the keys are defined as independent words, with the value of "1" indicating one appearance of the key. The *combine* phase is an optional optimization which combines the value of the same keys within each T_j^{Mi} to reduce the network traffic for later shuffle phase. In the previous WordCount example, the *combine* process will generate one record of $< word, 5 >$ out of $5 < word, 1 >$ pairs. The *partition* phase is responsible for marking the keys with Reducers, which determines the Reduce input distribution. The $< K_1, V_1 >$ pairs in the aforementioned flow is marked with suffix 1 to indicate that the corresponding operations are belonging to the Map phase in the framework. The operations in Reduce phase are marked with suffix 2 in the flow. The suffix number is only used to differentiate the MapReduce phases, doesn't mean K_1 and K_2 are different keys.

For the Reduce phase, there are primary three parts: the *shuffle* operation where the T_j^{Ri} copies the output marked with its own ID from each T_j^{Mi} using HTTP across the network; the *sort* operation merges and sorts the intermediate keys for T_j^{Ri} since different T_j^{Mi}s may have output the same key; and the *Reduce* phase where the Reduce function is called for each $< K_2, [V_2] >$ in the sorted inputs. The *shuffle* and *sort* operations often occur simultaneously, i.e. while the Map outputs are being fetched, they are merged and sort.

B. Types of Skews in MapReduce Framework

The skews in MapReduce stem from different reasons. For Map, the most common skew is caused by uneven input file size [16]. For example, if a 150MB size input is processed by the application running on a Hadoop cluster with 128MB HDFS block size, the input will be divided into two sub files with 128MB and 22MB in size. The Map skews can be addressed by splitting the expensive file or adjusting the HDFS block configuration, therefore is relatively straightforward. In contrast, skews in the Reduce phase are more complicated.

There are mainly two types of skews Reduce tasks can encounter: the expensive key group skews and the partition skews. For the former, the MapReduce framework requires that all tuples sharing the same key should be dispatched to the same Reducer. Key groups refer to the sequence of $< K, [V] >$ pairs. Many real world datasets exhibit skews in nature. Fig. 1 shows some examples. Fig. 1 (a) is the word frequency from the Shakespeare collection [17] and Fig. 1 (b) is from the wiki English dataset [18]. Reduce tasks can easily encounter the expensive key group skew if WordCount is run on such data: for Fig. 1 (a), there are altogether 67,056 words with the most frequently used one appeared 23,197 times, while the average word count is 13; for Fig. 1 (b), there are 21,433,355 words with the most frequently used word appeared 46,134,908 times, while the average word count is 43. Another example would be the PageRank application, a link analysis algorithm that assigns weights to each node in a graph by iteratively aggregating the weights of its inbound neighbors. If a graph contains nodes with a large degree of incoming edges such as Fig. 1 (c) and Fig. 1 (d), PageRank will suffer from Reduce skew. Fig. 1 (c) is the Google web dataset and Fig. 1 (d) is the Facebook social circles dataset [19], with X axis referring to the web pages (represented as nodes in the PageRank graph) and Y axis to be the number of hyperlinks in each page (represented as edges in the PageRank algorithm). For Fig. 1 (c), there are 739,454 pages, and the biggest graph node contains 456 linked edges, with 7 to be the average number of edges per node; for Fig. 1 (d), there are 3,363 pages included, the largest graph node contains 1,043 edges while the average edge number per node is 24 in this example.

The other type of skew is exclusive for Reduce tasks. It is called partition skew because it is mainly caused by unreasonable partition decisions. For example in Fig. 2 (a) with two Reducers, if the hash function categorize the intermediate key of "A","B", and "F" to a group and "C", "D", and "E" to another, the Reducer1 will have to process 1.9 times of $< K, V >$ pairs compared with Reducer2, where a partition skew occurs. With different number of Reducers, the severity of the intermediate data skew varies. This is illustrated in Fig. 2 (b) with three Reducers. When processing the same intermediate data with Fig. 2 (a), the hash partitioner may result in a different Reduce skew situation. Sometimes the degree of skew can be alleviated by enlarging the Reducer number, however, such practice can introduce new challenges like overloaded network traffic due to the increased communication, etc. There are some general

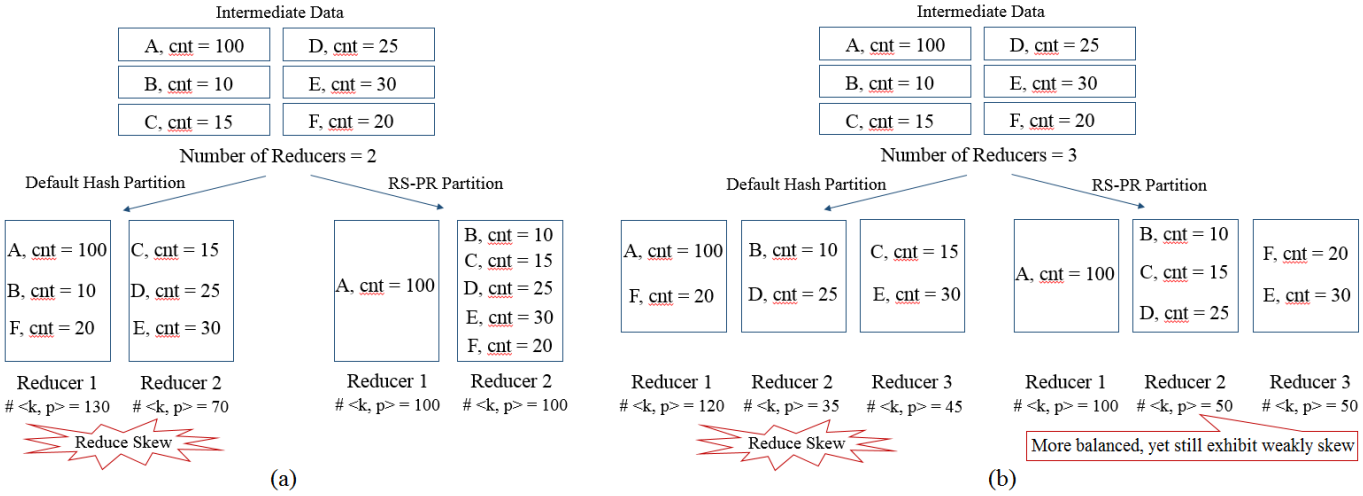

Fig. 2. Reduce skew illustration and possible improvement the ImKP partitioner can achieve

principles of choosing the reasonable Reducer number, which is not the research focus of this paper. We care about the improved partition policy, and any potential achievement will be discussed under the same Reduce number configurations. For an arbitrary application, the distribution of the intermediate data cannot be determined ahead of time. This brings a huge challenge toward partition skew mitigation.

C. When Speculative Execution Breaks Down: Skew-Caused Straggler Mitigation Analysis

Current speculative execution scheme has an unavoidable limitation when dealing with skew caused stragglers. It is shown in [20] that, even with the speculator in function, the OpenCloud cluster still encounter a 5% straggler rate at the task level, and this affects almost half of the parallel jobs to experience extended response time. This statistics is similar to what has been found in other production clusters that do not have speculation deployed [21], revealing a fact that current speculative scheme still has a huge gap in solving the straggler problem. This finding is consistent with other literature such as [5], which claims that in Yahoo!'s system, as many as 90% speculations are actually ended up being killed, with no benefits achieved in execution performance improvement.

One of the reasons behind such speculation breakdown is the skew caused stragglers. One biggest hypothesis assumed by the speculative execution is that, the redundant copy will behave as a quick task like other normal ones that do not fall behind. Therefore even it is launched after the straggler, it still has a chance to catch up. However, when mitigating skew caused stragglers, because the speculative copy needs to process identical input with the original task, itself will again become a straggler caused by the uneven input distribution. Fig. 3 illustrates this scenario with a WordCount example. The job J processes 50 documents with 50 T_j^{Mi}s, the default documents are 10M in size, with 0, 1, 3, 5, and 7 expensive files that are 50M in size to simulate the uneven input distribution for experiments. 0 indicates no skew inputs; 1 to 7 represents lightly skewed data towards more severe skewed inputs. Results show that, the speculation failure rate increases with the number of skewed inputs. In order to prevent such speculation breakdown, it is necessary to develop a skew mitigation scheme for MapReduce framework.

IV. The ImKP Approach for Reduce Skew Mitigation

In this section, we first discuss the design requirements for general skew mitigation systems, followed by the introduction of the proposed ImKP approach. The key preprocessing component is introduced in detail, together with a brief analysis toward how the ImKP system fulfills the above design requirements.

A. Skew Mitigation System Design Requirements

There are some general goals that a good skew mitigation system should accomplish such as minimal developer burden, mitigation transparency and flexibility, as well as minimal overhead. For minimal developer burden, the MapReduce application developer should be able to migrate their code into

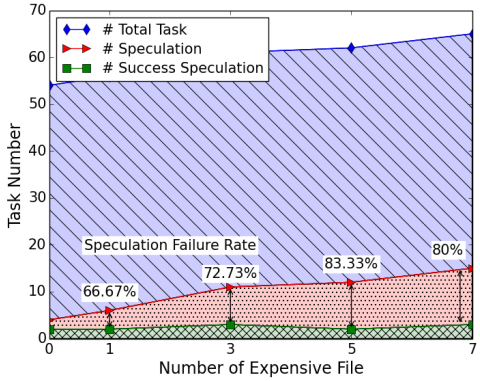

Fig. 3. Speculation failure rate with different input skews

the proposed skew mitigation platform with no requirement of learning new techniques. That is to say, the new system should try to adopt uniform APIs with existing MapReduce platform such as Hadoop YARN to minimize development complexity.

For skew mitigation transparency and flexibility, the former requires the proposed technique to be transparent to the end user. For normal users, when they launch MapReduce applications on the new platform, there should be no need for them to manually conduct additional configurations regarding skew mitigation if they prefer not doing so, and they do not need to get into the algorithm details such as parameter settings for the partition policy, etc. The latter, on the other hand, is for expert users who emphasize certain performance or some level of control. The new framework should provide the possibility for them to insert alternative information to generating flexible partition results.

The requirement of minimal overhead asks for the additional overhead spent on mitigating the skew phenomenon, including extra computation and resources, to be trivial enough that generates no negative impact toward final application level performance indicators such as job execution time.

B. System Model Overview

In order to minimize the skew occurrence for Reducers while fulfilling the aforementioned requirements, we propose ImKP, the Intermediate Key Pre-processing framework that enables the even distribution for Reduce inputs. The overall architecture of the ImKP system is presented in Fig. 4. The shaded parts are added components that belong to ImKP and the rest are compatible with current Hadoop YARN implementation. Texts with green shade represent the workflow that both original YARN application and the ImKP job need to go through. The blue texts are exclusive to original YARN, while the red texts solely belong to the ImKP logic. The procedures with the same sequential number indicate the fact that they are executing in parallel.

Under the ImKP framework, the input file is first sent to the pre-processing component. This additional layer is responsible for generating the key-Reducer mapping result $< K, R >$ file that enables the even partitioner. Details of the pre-processor will be given in the next section. For applications whose input is stored on local file system, ImKP utilizes the multithreading implementation to parallelize the pre-processing with file uploading to mitigate timing overhead. This is reflected in Fig. 4 with two 1.1 steps in red and green texts respectively. For applications whose input is stored on HDFS, we manage to control the pre-processing overhead to a limited level through a group based ranking technique. This guarantees trivial impact toward job execution. The optimization detail is discussed along with the pre-processor in the next section as well.

After pre-processing and file uploading, Map tasks will be generated to handle the input data chunks, which is consistent with the original MapReduce framework. Once the Map function finishes, unlike the default partitioner which does the hash calculation in order to label the intermediate key generated by Map with Reducer ID, the even partitioner in ImKP directly

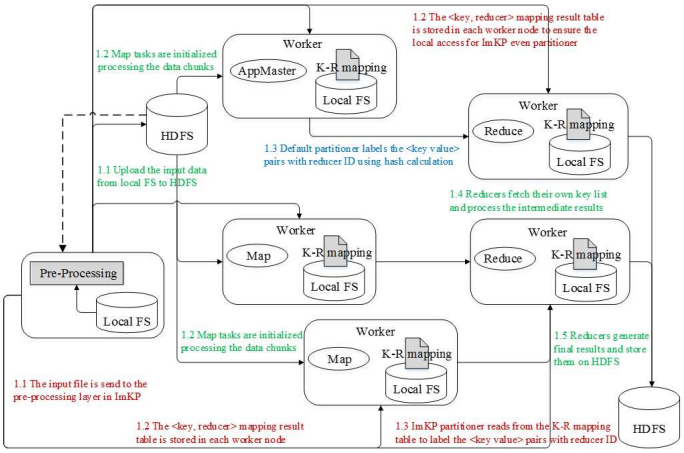

Fig. 4. The system model for the ImKP framework

look up the $< K, R >$ mapping table. The table is stored on every machine node within the YARN cluster to ensure the local access for the even partitioner, regardless of the Mapper positions. And because of the group based ranking optimization, the $< K, R >$ mapping table is extremely small in size, containing only $\#Reducer \times scale$ rows. The notion of $scale$ is a user defined parameter implying the degree of evenness in ImKP pre-processor, with default value of 50. For example, for applications with 10 Reducers, there will only be 500 bi-tuple in the mapping table. This small size guarantees the promptness of the local read operation. We have tested the timing overhead of reading the mapping file from memory and doing the hash calculation, the average time for the former operation is 10,000ns while the latter is 9,000ns, which is only 1,000ns in difference. In other words, the default hash partitioner and the ImKP even partitioner take approximately same time when conducting the partition operation.

C. The Pre-processor in ImKP

The pre-processor is an additional layer before the normal Map phase. Its main purpose is to get the accurate distribution of the intermediate keys and generate a balanced dispatch solution depending on the number of Reducers. The pre-processing procedure forms the most important component of the ImKP system, and it mainly consists of following steps:

- *Define customized keys:* In order to calculate the intermediate key statistics from the input, the definition of the keys must be given. For example, keys are defined as separate terms in a document in WordCount, or as each graph node in PageRank. Because the keys required by this initializer is identical to the keys in the Map phase which is already given by the original MapReduce framework, this step does not need additional developer intervene. The ImKP system can automatically copy the key define function from the user program.
- *Rank the intermediate keys:* In this step, the frequencies of the intermediate key occurrence will be counted and ranked. The biggest challenge encountered here comes

from the fact that the MapReduce framework is designed for big data applications that process large scale intermediate keys. If the frequency for each key is recorded separately for ranking, it will come at huge computational ($\mathcal{O}(n \log n)$ complexity) and storage costs. In order to solve this problem we propose a group based ranking scheme. The assumption supports this optimization is that, we believe the number of keys is way beyond the number of Reducers, therefore one Reducer would have to process multiple keys. Instead of directly rank all the intermediate data, we first map the keys into groups using hash to decrease the number of items that need to be ranked. A parameter of $scale$ is adopted in this procedure to imply the total number of grouped keys one Reducer will later receive. Altogether $\#Reducer \times scale$ number of key groups will be created. The key occurrence frequency will be counted per group for ranking.

- *Even distribute the key groups based on frequency ranking:* This step generates the $< K, R >$ mapping result for the ImKP even partitioner to assign intermediate keys to Reducers. We adopted the best fit policy in our implementation for this bin-packing problem: the key group with the maximum occurrence frequency in the remaining queue will be mapped to the Reducer with the minimum sum of frequencies. The intermediate keys in the same group will be mapped to the same Reducer. For advanced users, we provided an API so that this default best-fit method can be replaced with more dedicated algorithms. For example, if additional information on the performance diversity among machine nodes is introduced, we can always adjust this even distribute scheme. The result mapping file will be stored on every worker node within the cluster so that the local access for the ImKP even partitioner can be guaranteed.

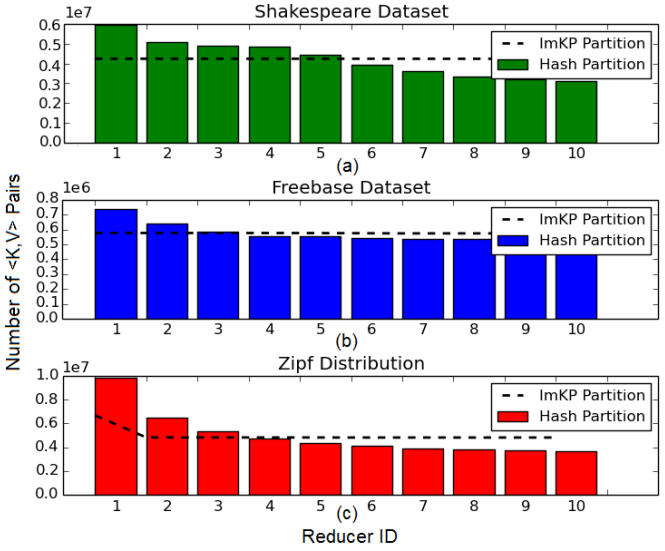

Fig. 5. Number of inputs per Reducer for (a) Inverted Index on Shakespeare data; for (b) PageRank on Freebase data; and for (c) WordCount on Zipf data

V. EVALUATION

The ImKP evaluation focuses on answering following three questions: (1) can it mitigate the Reduce skews by generating a more balanced input size distribution for Reducers; (2) whether the skew mitigation overhead is small enough to be ignored; and (3) whether the overall job response time can be improved. For each question, we either test different workload types or different MapReduce configurations to verify whether the performance improvement remains consistent through different operational situations.

A. Experiment Set Up

In order to evaluate the effectiveness of the ImKP framework, we run various experiments in a 15 virtual machine (VM) cluster build on top of the ExoGENI infrastructure [22]. Each VM contains 1 CPU core, 3G RAM, and 25G disk, running the CentOS6.7 OS. In all experiments, we configured the HDFS to maintain two replicas for each data chunk, and the container for both Map and Reduce tasks are 1G in size.

Popular applications including WordCount, PageRank, and Inverted Index provided in the Bespin toolkit [23] are tested, on both synthetic and real world datasets. We generate 1.6GB synthetic data files following the Zipf distribution with varying σ parameters from 0.4 to 1.4 to control the degree of the skew. The larger σ value represents a heavier skew. Zipf distribution is very common in the data coming from the real world, e.g., the word occurrences in natural language, features of the Internet, etc [6]. For real world data, we run our experiments mainly on the Shakespeare collection [17], the English Wiki dataset [18], and the Freebase data set [24].

B. Skew Mitigation Effectiveness

Fig. 5 illustrates the number of $< K, V >$ pairs processed by each Reducer using the default hash partition and the ImKP even partition algorithm running Inverted Index (a),

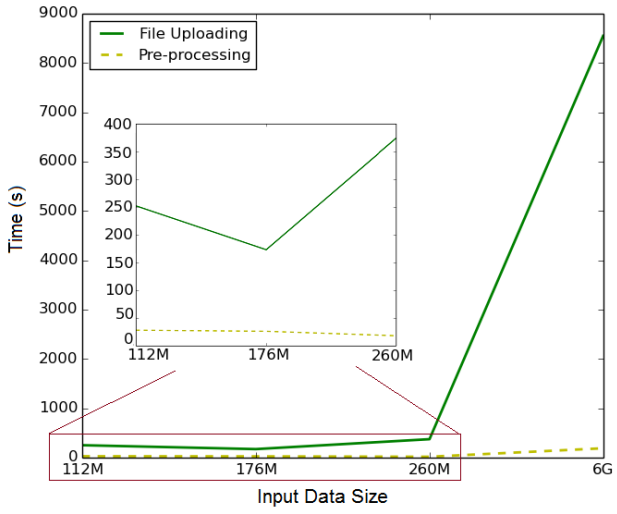

Fig. 6. The pre-processing overhead.

TABLE I
REDUCE INPUT SKEW MITIGATION RESULTS FOR DIFFERENT SKEW DEGREES

Reduce Input Size		σ=0.4	σ=0.5	σ=0.6	σ=0.7	σ=0.8	σ=0.9	σ=1.0	σ=1.1	σ=1.2	σ=1.3	σ=1.4
5 reducer	C_v Improvement	99.76%	94.94%	97.52%	98.92%	99.98%	99.75%	99.87%	99.94%	80.09%	56.28%	40.68%
	C_v Times	416.27	19.75	40.33	92.25	4187.47	406.16	775.18	1665.17	5.02	2.29	1.69
10 reducer	C_v Improvement	90.40%	93.68%	97.35%	98.68%	99.25%	99.72%	69.28%	46.62%	32.06%	21.94%	15.14%
	C_v Times	10.42	15.83	37.71	75.54	133.67	354.84	3.25	1.87	1.47	1.28	1.18

PageRank (b) and WordCount (c). From Fig. 5 (a) and (b), it is observable that ImKP achieved extremely good skew mitigation results: the number of ImKP Reducer inputs are close to the ideal even distribution. We use the coefficient of variation defined in Equation 1 to measure the skewness of the Reduce inputs. Table I details the C_v improvement $((C_v(Hash) - C_v(ImKP))/C_v(ImKP))$ and the C_v times $(C_v(Hash)/C_v(ImKP))$ with varies degree of skews to show the effectiveness of the ImKP in mitigating Reduce skews.

$$C_v = \frac{StdDev(ReducerInputs)}{Avg(ReducerInputs)} \quad (1)$$

Meanwhile, we see from Fig. 5 (c) that the ImKP algorithm has a limitation. The Zipf σ in Fig. 5 (c) is 1.4, which indicates a severe skew and the existence of an expensive key. ImKP is mainly designed for solving the partition skews, for situations of expensive key skews (the number of intermediate data belongs to a certain key surpasses the sum of the others), ImKP can only achieve a more balanced result, yet may still exhibit slightly skew. Fig. 2 (b) illustrates this limitation with a straight forward example. And this explains the C_v improvement changing trend in Table I as well.

C. Skew Mitigation Overhead

The mitigation overhead is one of the biggest concern for the ImKP system design because it inserts an extra pre-processing layer before the normal Map phase. Different with the literature that estimates the complete intermediate key distribution, ImKP pre-processing adopts a group based ranking scheme that dramatically decreased the ranking element numbers based on the fact that one Reducer has to process multiple intermediate keys. Fig. 6 illustrates the exact pre-processing time compared to the file uploading time for the WordCount application run on various input sizes.

From the figure it is observable that, the pre-processing overhead is stable at a low level that remains smaller than the file uploading time. This overhead is small enough even for large inputs such as the 6G input from the English Wiki dataset. And because of the multithreading parallelization implementation, for applications that store their inputs on local file systems, there will be no pre-processing overhead at all. In addition, the pre-processing results are stored on every datanode ready for possible reuse. This will benefit applications that have to go through MapReduce iterations such as PageRank (the PageRank score updates at each iteration before it convergences). Through this way, the influence of the initial timing overhead is further reduced.

D. Job Execution Improvement

According to the aforementioned analysis, the two modifications made by ImKP based on the original Hadoop YARN, the pre-processor and the different partitioner both generate no obvious timing difference for overall job completion. Therefore, the execution time difference listed in Table II is mainly due to the different number of $< K, V >$ pairs processed by each Reducer. Fig. 7 summaries the job execution time improvement for different levels of skew inputs. The results are average values for three running tests, with coefficient of variation C_v to represent the response time variance for each test case. From the result we see that ImKP is capable of improving average response time by up to 29.37%. For the number of Reducers, as discussed in previous sections, different configurations result in different levels of skew severance. The improvements we discuss in this evaluation are conducted under the same Reduce number configurations.

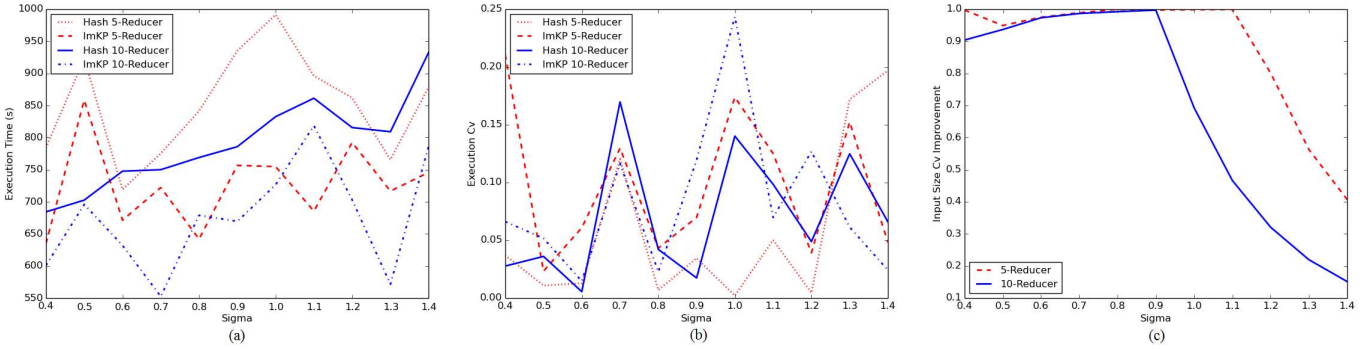

Fig. 7. The (a) execution time; the (b) execution coefficient of variation; and the (c) input size improvement for ImKP and hash partition on Zipf data

TABLE II
RESPONSE TIME IMPROVEMENT FOR WORDCOUNT APPLICATION ON THE ZIPF DATA WHEN σ CHANGES FROM 0.4 TO 1.4.

		σ = 0.4	σ = 0.5	σ = 0.6	σ = 0.7	σ = 0.8	σ = 0.9	σ = 1.0	σ = 1.1	σ = 1.2	σ = 1.3	σ = 1.4
5 reducer	Improvement	19.01%	8.18%	6.63%	6.87%	23.71%	19.10%	23.86%	23.51%	8.08%	6.40%	15.15%
	C_v(Hash)	0.04	0.01	0.01	0.12	0.01	0.03	0.01	0.05	0.01	0.17	0.20
	C_v(ImKP)	0.21	0.02	0.06	0.13	0.04	0.07	0.17	0.13	0.04	0.15	0.05
10 reducer	Improvement	12.19%	0.96%	15.47%	26.34%	11.71%	14.80%	12.75%	4.96%	13.80%	29.37%	15.67%
	C_v(Hash)	0.03	0.04	0.01	0.17	0.04	0.02	0.14	0.10	0.05	0.12	0.07
	C_v(ImKP)	0.07	0.05	0.01	0.12	0.02	0.12	0.24	0.07	0.13	0.06	0.02

VI. CONCLUSION AND FUTURE WORK

In this paper we proposed ImKP, an intermediated key pre-processing framework that enables the even partition for Reduce inputs. ImKP can be used to avoid data skew caused stragglers for Reduce tasks. Main contributions include:

- Analyzed the skew behavior with various datasets and illustrated the type of skews within current MapReduce framework. The influence of data skew stragglers, especially the Reduce skews, toward efficient speculative execution was discussed.
- Proposed ImKP, the Intermediate Key Pre-processing framework that plugged an intermediate key ranking layer before the original Map phase to enable the even partition for Reduce inputs. Results show that the skewness for Reducers can be decreased by 99.8% in average. And overall job response can be improved up to 29.37%.
- Developed a group based ranking technique that dramatically reduced pre-processing overhead for the ImKP system. And through parallelizing the pre-processing with file uploading, we even managed to eliminate the overhead for workloads that take inputs from local file system.

Future work includes the integration of Reducer split solutions into current ImKP to deal with the expensive key skews. The MapReduce requires intermediate data sharing the same key to be processed by the same Reducer, and the related work that splits expensive Reducers often comes with Reducer reconstruct overhead as well as constraints toward workload logic such as ordering preservation. Solving this can be the possible extension of the ImKP framework.

ACKNOWLEDGMENT

This work is supported by the China National Key Research and Development Program (2016YFB1000101, 2016YFB1000103) and the University of Leeds and CSC joint scholarship program.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones." in *NSDI*, vol. 13, 2013, pp. 185–198.

[4] W. Applications powered by Hadoop. (2017). [Online]. Available: https://wiki.apache.org/hadoop/PoweredBy

[5] E. Bortnikov, A. Frank, E. Hillel, and S. Rao, "Predicting execution bottlenecks in map-reduce clusters," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. USENIX Association, 2012, pp. 18–18.

[6] J. Lin *et al.*, "The curse of zipf and limits to parallelization: A look at the stragglers problem in mapreduce," in *7th Workshop on Large-Scale Distributed Systems for Information Retrieval*, vol. 1. ACM Boston, MA, USA, 2009, pp. 57–62.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. IEEE, 2010, pp. 1–10.

[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI*, vol. 8, no. 4, 2008, pp. 7–20.

[9] Q. Chen, C. Liu, and Z. Xiao, "Improving mapreduce performance using smart speculative execution strategy," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 954–967, 2014.

[10] S.-W. Huang, T.-C. Huang, S.-R. Lyu, C.-K. Shieh, and Y.-S. Chou, "Improving speculative execution performance with coworker for cloud computing," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*. IEEE, 2011, pp. 1004–1009.

[11] 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*. ACM, 2012, pp. 25–36.

[12] V. A. Nawale and P. Deshpande, "Minimizing skew in mapreduce applications using node clustering in heterogeneous environment," in *Computational Intelligence and Communication Networks (CICN), 2015 International Conference on*. IEEE, 2015, pp. 136–139.

[13] Q. Chen, J. Yao, and Z. Xiao, "Libra: Lightweight data skew mitigation in mapreduce," *IEEE Transactions on parallel and distributed systems*, vol. 26, no. 9, pp. 2520–2533, 2015.

[14] Hadoop. (2016). [Online]. Available: http://hadoop.apache.org/

[15] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[16] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "A study of skew in mapreduce applications," *Open Cirrus Summit*, vol. 11, 2011.

[17] W. S. The Complete Works of William Shakespeare. (2017). [Online]. Available: http://www.gutenberg.org/ebooks/100

[18] M. Pataki, M. Vajna, and A. Marosi, "Wikipedia as text," *Ercim News - Special theme: Big Data. http://kopiwiki.dsd.sztaki.hu/*, vol. 89, pp. 48–49, 2012.

[19] J. L. Stanford large network dataset collection. (2017). [Online]. Available: http://snap.stanford.edu/data/

[20] X. Ouyang, P. Garraghan, B. Primas, D. McKee, P. Townend, and J. Xu, "Adaptive speculation for efficient internetware application execution in clouds," *ACM Transactions on Internet Technology*, 2017.

[21] X. Ouyang, P. Garraghan, D. McKee, P. Townend, and J. Xu, "Straggler detection in parallel computing systems through dynamic threshold calculation," in *Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on*. IEEE, 2016, pp. 414–421.

[22] ExoGENI. (2017). [Online]. Available: http://www.exogeni.net/

[23] J. Lin, "Bespin: a library that contains implementations of big data algorithms in mapreduce and spark," *https://github.com/lintool/bespin*, 2017.

[24] J. G. Freebase data dumps. (2013). [Online]. Available: https://developers.google.com/freebase/