Front.Comput.Sci.
DOI

REVIEW ARTICLE

Reducing Partition Skew on MapReduce: An Incremental Allocation Approach

Zhuo WANG , Qun CHEN (✉), Bo SUO, Wei PAN, ZhanHuai LI

School of Computer Science and Engineering, North Western Polytechnical University, Xi‘an, PR China,

Abstract MapReduce, a parallel computational model, has been widely used in processing big data in a distributed cluster. Consisting of alternate Map and Reduce phases, MapReduce has to shuffle the intermediate data generated by mappers to reducers. The key challenge of ensuring balanced workload on MapReduce is to reduce partition skew among reducers without detailed distribution information on mapped data.

In this paper, we propose an incremental data allocation approach to reduce partition skew among reducers on MapReduce. The proposed approach divides mapped data into many micro-partitions and gradually gathers the statistics on their sizes in the process of mapping. The micro-partitions are then incrementally allocated to reducers in multiple rounds. We propose to execute incremental allocation in two steps, micro-partition scheduling and micro-partition allocation. We propose a Markov Decision Process (MDP) model to optimize the problem of multiple-round micro-partition scheduling for allocation commitment. We present an optimal solution with the time complexity of $O(K \cdot N^2)$, in which K represents the number of allocation rounds and N represents the number of micro-partitions. Alternatively, we also present a greedy but more efficient algorithm with the time complexity of $O(K \cdot NlnN)$. Then, we propose a Min-Max programming model to handle the allocation mapping between micro-partitions and reducers, and present an effective heuristic solution due to its NP-completeness. Finally, we have implemented the proposed approach on Hadoop, an open-source MapReduce platform, and empirically evaluated its performance. Our extensive experiments show that compared with the state-of-the-art approaches, the proposed approach achieves considerably better data load balance among reducers as well as overall better parallel performance.

Keywords incremental partitioning, data balance, MapReduce

Received month dd, yyyy; accepted month dd, yyyy

E-mail: *wzhuo*918@*mail.nwpu.edu.cn*

1 Introduction

MapReduce [1] is an enormously popular simplified parallel data programming model for big data analytics. There are currently many distributed systems built on MapReduce [2], including Hadoop [3], a popular open-source implementation. A MapReduce program consists of alternate Map and Reduce phases. In the Map phase, mappers transform input tuples into key-value pairs, which are then divided into partitions and shuffled to reducers. In the Reduce phase, each reducer executes specified operations on the partitions allocated to it.

It has been pointed out [4] [5] that workload unbalance is a common phenomenon on MapReduce. We observe that the key challenge of ensuring workload balancing among reducers is partition skew. The basic strategy, which is the default solution implemented on Hadoop, is to specify a hash or range partition function before mapping. The function can usually result in each reducer receiving roughly the same number of key blocks. However, due to size skew of key blocks, it can not ensure data load balance among reducers. Actually, without detailed distribution information on mapped data, it is nearly impossible to predefine a partition function that can balance data load in the Reduce phase.

Currently, there exist two approaches to address the shortcoming of the basic strategy. Some work [6] proposed to first sketch data distribution by pre-sampling input tuples, and then use the obtained information to design a balanced partition function. This approach requires an additional run of mapping on sampled data. Moreover, its effectiveness depends on accurate sampling, which by itself remains a challenging task. The other approach [7] [8] [9] [10] [11] trades slow shuffle start for distribution statistics garnering. It first defines a tentative partition function to produce arbitrary-sized partitions and gradually gathers the statistics on partition sizes in the mapping process. Once the mapping task is completed up to a preset progress point, it then adjusts the partitioning plan based on the garnered distribution information. In this approach, mapped data are actually allocated to reducers at adjustment point. Note that the basic strategy as implemented on Hadoop can start to shuffle mapped data once the first mapper finishes its job. In contrast, this approach can only begin to shuffle mapped data after adjustment point. Since mapped data allocation occurs in a single round, the challenge of implementing this approach is to determine the optimal adjustment point. If the adjustment point is set to be at an early stage of mapping, the gathered distribution information may not be accurate. On the other hand, if it is set to be later, data shuffling has to be delayed accordingly.

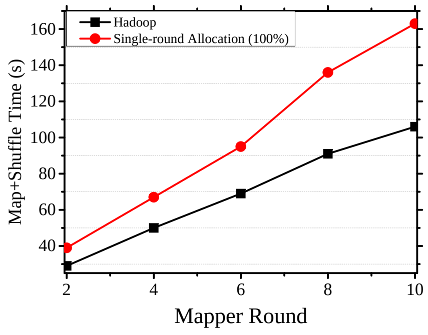


Fig. 1: Illustration of Slow Reduce Job Start

We illustrate the shortcoming of the single-round allocation approach by a comparative experiment, as shown in Figure 1. The naive Hadoop approach starts to shuffle data immediately after the first mapper finishes its job. In contrast, the single-round approach starts to shuffle data only after all the mappers finish their jobs. The X-axis represents the number of mapper rounds required by a mapping task. The Y-axis represents the total time consumed by the Map and Shuffle phases. In the experiment, each round has totally 10 mappers running simultaneously, each of which processes 64M data. It can be observed that the single-round approach can consume considerably more time than the naive approach, and its performance disadvantage tends to deteriorate with the increasing number of required mapper rounds.

To better manage the trade-off between sampling accuracy and early reduce job start, this paper proposes a novel incremental allocation approach. Our major contributions can be summarized as follows:

- We propose an incremental data allocation approach to reduce partition skew on MapReduce. It first divides mapped data into many micro-partitions while gradually gathering the statistics on their sizes in the process of mapping, and then allocates them to reducers in multiple rounds. Compared with existing techniques, it provides with a more effective mechanism to manage the trade-off between sampling accuracy and early shuffle start. On one hand, it enables immediate shuffling after the first mapper finishes its job. On the other hand, partition skew resulting from inaccurate sampling can be corrected in later rounds of micro-partition allocation. (**Section 4**)
- We propose to execute incremental allocation in two steps, micro-partition scheduling and micro-partition allocation. Provided with multiple allocation decision points, the problem of micro-partition scheduling chooses the micro-partitions for allocation commitment at each decision point, and the problem of micro-partition allocation maps committed micro-partitions to reducers. For micro-partition scheduling, we propose a Markov Decision Process (MDP) model and present an optimal algorithm with the time complexity of $O(K \cdot N^2)$, in which K represents the number of allocation rounds and N represents the number of micro-partitions. We also present a greedy but more efficient algorithm with the time complexity of $O(K \cdot N \ln N)$. For micro-partition allocation, we propose a MinMax programming model and present an effective heuristic solution due to its NP-completeness. (**Section 5&6**)
- We implement the proposed incremental approach on Hadoop (**Section 7**) and compare its performance with that of the state-of-the-art solutions. Our extensive experiments show that it achieves considerably better data load balance among reducers as well as overall better parallel performance. (**Section 8**)

The rest of this paper is structured as follows: Section 2 reviews related work. Section 3 presents the preliminaries and gives an overview on the existing solutions. Section 4 introduces the overview of incremental approach. Section 5 presents the solution for micro-partition scheduling. Section 6 presents the solution for micro-partition allocation. Section

7 presents the implementation on the native Hadoop. Section 8 presents the experimental results. Finally, we conclude our work with Section 9.

2 Related work

Since the MapReduce programming model was first proposed [1], the problem of workload unbalance has been widely recognized [12] [13] [14] [5] [6] [15] [16] [17]. The existing workload balancing techniques proposed for traditional parallel systems [18] [19] optimized job and task scheduling to minimize the parallel execution time. The Map-Shuffle-Reduce phase design renders them unfit for the MapReduce framework.

The simple approach for reducing partition skew [20] [21] [22] decoupled the Map and Shuffle phases. It only begins to shuffle data after all the mappers finish their jobs. Such an approach suffers from slow reduce job to start. An improved approach was based on sampling. Sampling can be executed by an independent process before Map phase [6]. Alternatively, it can be integrated into Map phase. [23] sampled the input data in the process of mapping and used an adaptive partitioner to generate balanced data at a specific time point in the midst of mapping. Similarly, [24] added a separate sampling thread to gain the distribution in the Oracle Loader for Hadoop. [25] selected parts of input splits as sampling data to evaluate the whole input set. The *Closer* system [9] [10] also inserted sampling into the map function. More recently, Libra [11] proposed an improved sampling method and used range partition to avoid the skewed partition, the core contribution is allocating one key to different reducers for part of applications. Complementary to these work, [7, 8] proposed to sample the input data by blocks instead of tuples.

However, the common drawback of the sampling-based techniques is that accurate sampling remains a challenging task especially when only a small part of input data can be sampled [26]. Further, allocation plan adjustment in the midst of mapping would delay the shuffle start point. Therefore, in practical implementation, it remains challenging to set an adjustment point that can optimize the trade-off between data load balance and early reduce start. The incremental approach we proposed in this paper can instead provide with a flexible mechanism to better manage their trade-off. Its effectiveness does not depend on accurate sampling either.

Workload unbalance among reducers is usually addressed in two phases. The first one is to ensure workload balancing if possible before the jobs start. Our work on partition skew falls into this phase. The second one instead concerns about reallocating the workload among computing nodes in the running process of jobs. For dynamic workload balancing in the midst of Reduce phase, the typical approach [26] [27] proposed to identify idle computing nodes and reallocate some workload on heavy nodes to the idle ones. Its effectiveness, however, to a large extent depends on accurate estimation of the remaining processing time on computational nodes.

There has been some work on [28] [29] studying how to handle data skew in parallel join on MapReduce. Since the proposed incremental approach aims to address partition skew, it can obviously be used to handle the skew of any reduce operation (including join). As demonstrated by these work, the join operation enables specific optimization techniques, which are however beyond the scope of this paper.

There is also some work complementary to ours. [10] proposed the method for estimating the cost of the tasks that are distributed to reducers based on a given cost model. The SkewReduce system [30] used user-defined cost functions to estimate the computational cost of partitions. To ensure workload balancing, the incremental approach also requires estimating the computational cost of micro-partitions. It can use these proposed estimation techniques to optimize performance in practical implementation.

3 Preliminaries

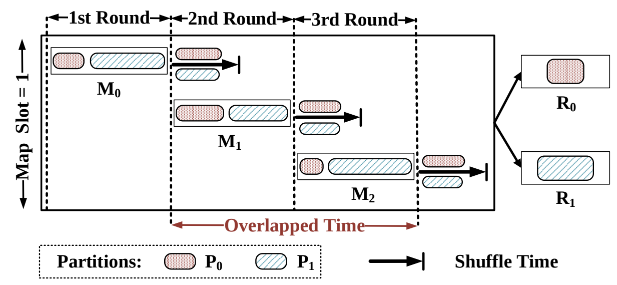


Fig. 2: Naive Approach on Hadoop

A MapReduce program consists of alternate Map and Reduce phases. In the Map phase, each mapper loads $Splits$ from local HDFS and maps them into $< key, value >$ pairs. The mapped data are divided by a hash function into partitions. Each partition corresponds to a reducer. Once a mapper finishes its job, Hadoop starts to shuffle partitions to their corresponding reducers. An example of data flow is shown in Fig.2, in which mapped data are divided into two partitions, P_0 and P_1. By default, the hash function is set to be *key%R*, in which R is the total number of reducers. Suppose that each *key* corresponds to a micro-partition. The default function can usually ensure that each reducer receives a balanced number of micro-partitions. However, due to size skew of micro-partitions, it does not necessarily result in balanced workload among reducers.

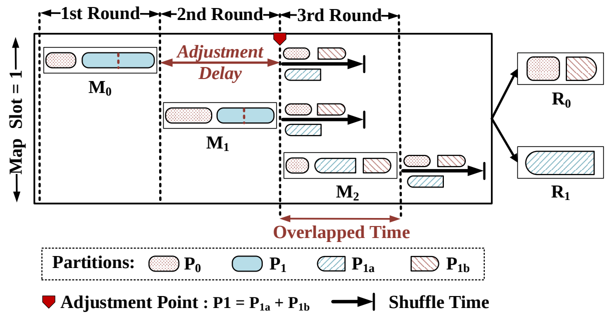


Fig. 3: Single-round Allocation Approach

An improved approach [9], as shown in Figure 3, schedules the allocation of mapped data in the process of mapping. Mapped data are first tentatively grouped into partitions which are assigned to reducers. The resulting plan is tentative in that it may be adjusted later. In the process of mapping, the statistics on partition sizes are simultaneously gathered. After the job of mapping progresses to a preset time point, which is also referred to as the adjustment point, the tentative plan is then adjusted based on the gathered distribution information. At this point, the default partition function would be amended as a new partition function to split the heavy partitions. Then, the finished mappers could shuffle after splitting the local heavy partitions. But for the unfinished mappers, they have to group tuples by the amended partition function. For instance, as shown in Figure 3, the partition P_1 is split into P_{1a} and P_{1b} at the adjustment point which is defined as the second round of mapper is finished. After that,P_{1b} is allocated to R_0 instead of its original destination R_1. The finished mappers M_0 and M_1 could begin to be shuffled to reducers. And for M_2, its data must be partitioned with the amended partition function. And with this rule, as shown in Figure 3, its data is partitioned into three parts: P_0, P_{1a} and P_{1b}.

4 Overview of The Incremental Approach

4.1 Overview

To enable flexible management on the trade-off between sampling accuracy and early shuffle start, we propose to allocate mapped data to reducers in multiple rounds. The process of

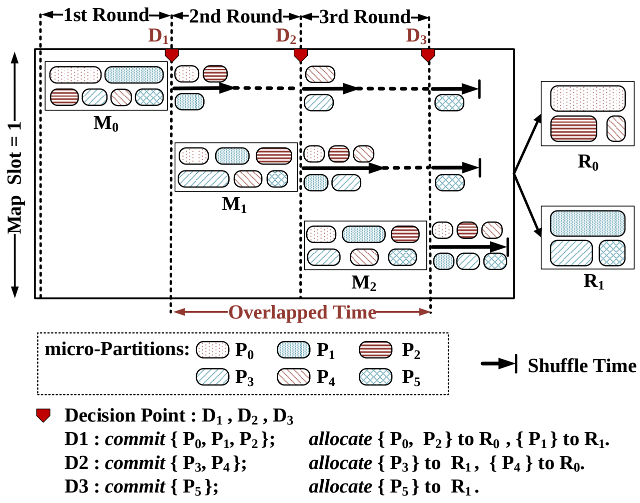

1st Round
2nd Round
3rd Round
D₁
D₂
D₃
Map Slot = 1
M₀
M₁
M₂
R₀
R₁
Overlapped Time
micro-Partitions: P₀ P₁ P₂ P₃ P₄ P₅
Shuffle Time
Decision Point : D₁ , D₂ , D₃
D1 : commit { P₀, P₁, P₂ }; allocate { P₀, P₂ } to R₀ , { P₁ } to R₁.
D2 : commit { P₃, P₄ }; allocate { P₃ } to R₁, { P₄ } to R₀.
D3 : commit { P₅ }; allocate { P₅ } to R₁.

Fig. 4: Incremental Allocation Approach

incremental allocation is sketched in Figure 4. The mapped data are first divided into many micro-partitions, the number of which is usually significantly larger than the number of reducers. The workload statistics of micro-partitions are continually gathered and sent to an allocation decision maker in the process of mapping. The generated micro-partitions are then allocated to reducers incrementally in multiple rounds. Instead of using a single allocation decision point, the incremental approach sets a series of discrete time points in the mapping process as allocation decision points. At each decision point, some of uncommitted micro-partitions are chosen and allocated to reducers. In the example shown in Figure 4, the micro-partitions, $\{P_0,P_1,P_2\}$, are allocated at the decision point D_1, P_3 and P_4 are allocated at D_2, and P_5 is allocated at D_3. The discrete decision points can be set according to mapping progress: each decision point corresponds to a percentage up to which the map job has been completed. For instance, the decision points can be set to be $\{T_1,...,T_{10}\}$, in which T_i represents the time point where $(i \cdot 10)\%$ of the map job has been completed.

The ultimate purpose of workload balancing on Hadoop is to optimize parallel performance. To this aim, the problem of incremental allocation has to consider the cost of data shuffling as well as workload balancing among reducers. Suppose that the intermediate data generated by mappers are divided into N micro-partitions. Also suppose that there are total M reducers ($N \gg M$) and there are total K discrete decision points in the process of mapping, which are denoted by $\{T_1,...,T_K\}$. The solution of incremental allocation consists of a series of allocation actions. Given a decision point T_i, an action allocates some uncommitted micro-partitions to reducers. Note that all the N micro-partitions should be allocated to reducers and each of them should be allocated only once. The objective function of the optimization problem can be represented by:

$$\min \max_{j} (f_s(R_j) + \sum_{i} (a_{ij} \cdot f_c(S_i))) \tag{1}$$

in which S_i denotes the size of the ith micro-partition, a_{ij} denotes the mapping between micro-partitions and reducers ($a_{ij} = 1$ if the ith micro-partition is allocated to the jth reducer, and $a_{ij} = 0$ otherwise), R_j denotes the jth reducer, $f_s(R_j)$ represents the total shuffling cost at the reducer R_j and $f_c()$ represents the computational cost of a micro-partition.

In this paper, we focus on handling the reduce-side partition skew, and thus assume that the workload of a micro-partition can be accurately estimated beforehand by its size. It can be observed that otherwise, workload skew *has to* be balanced in the midst of the Reduce phase. For ease of presentation, we use a micro-partition's size and its computational workload interchangeably in the rest of this paper. On MapReduce, the shuffle job interleaves with the map job. Therefore, the progress of a shuffle job at a reducer depends on the progress of the map job, the sizes of the micro-partitions allocated to the reducer, and the micro-partitions' shuffle start points. We also note that the statistics of micro-partition sizes (S_i) are incrementally garnered. Their estimations can fluctuate wildly in the process of mapping. Therefore, we propose to execute incremental allocation in two steps, micro-partition scheduling and micro-partition allocation. They are described as follows:

Problem 1. *Micro-partition Scheduling. Given a series of decision points and a set of micro-partitions, P, the problem of micro-partition scheduling refers to selecting a subset of micro-partitions in P for allocation commitment at each decision point. A feasible solution should satisfy that each micro-partition is committed once and only once.*

Problem 2. *Micro-partition Allocation. Given a decision time point T_i and a set of committed but unallocated micro-partitions, P_u, the problem of micro-partition allocation refers to allocating the micro-partitions in P_u to reducers. A feasible solution should satisfy that each micro-partition is allocated to one and only one reducer.*

The solution to Problem 1 chooses the micro-partitions for allocation commitment at decision points. Its objective is to optimize the trade-off between workload balancing and early shuffle start. The solution to Problem 2 allocates the committed micro-partitions to reducers. Its objective is to achieve balanced workload among reducers. We formulate them and study their optimization in Section 5 and 6 respectively.

5 Micro-partition Scheduling

In this section, we propose the algorithms based on Markov Decision Process (MDP) to solve the optimization problem of micro-partition scheduling.

5.1 The MDP Model

A standard MDP model is a 5-tuple (S, A, P, R, γ), where S denotes a finite set of states, A denotes a finite set of actions, P denotes the set of state transit probabilities upon actions, R denotes the state transit rewards and γ denotes the discount factor representing the difference in importance between present and future rewards. For ease of presentation, we use A_s to denote the finite set of actions available at a state s, $P_a(s_t, s_{t+1})$ denoting the probability of state s at time t transiting to state s_{t+1} at time $t+1$ as a result of the action a, $R(s_t, s_{t+1})$ denoting the immediate reward of the state transition from s_t to s_{t+1}. The core problem of MDP is to find a "policy" for the decision maker: a function that specifies the action that the decision maker will choose when in state s. The goal is to choose a policy that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon:

$$\sum_{t=0}^{\infty} \gamma^t \cdot R(s_t, s_{t+1}) \tag{2}$$

where $0 < \gamma \leq 1$.

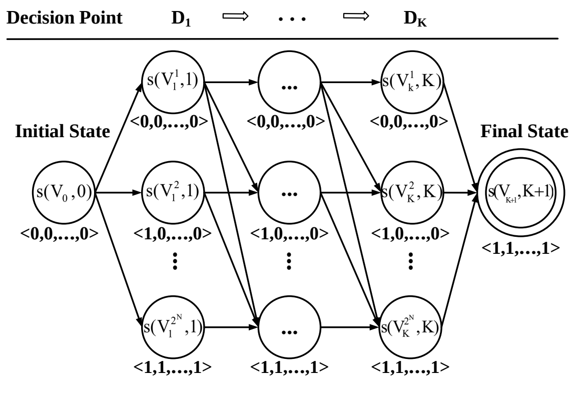


Fig. 5: The General MDP Model

We formulate the micro-partition scheduling problem as a MDP. Suppose that there are total N micro-partitions and K decision points. The state is represented by 2-tuple, $s(V_t,\ t)$,

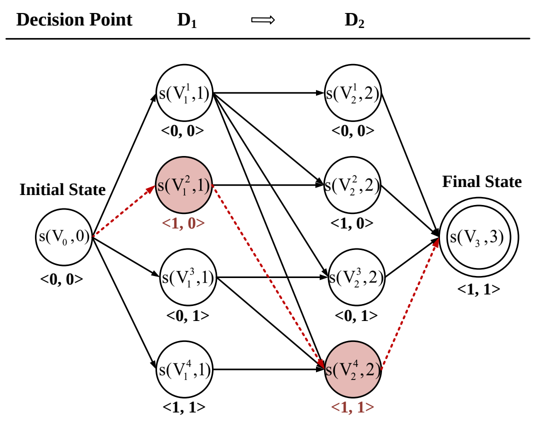


Fig. 6: An Example MDP

where V_t is a N-dimensional allocation vector and t represents a decision point. In V_t, $\{v_1,\ldots,v_N\}$, each dimension corresponds to a micro-partition, and $v_i = 1$ if its corresponding micro-partition has been committed for allocation at time t, otherwise $v_i = 0$. The action corresponds to commiting some uncommitted micro-partitions for allocation. The state transit can only occur from time t to time $t+1$. In a state transit from (V_t, t) to $(V_{t+1}, t+1)$, the value of v_i remains to be 1 at time $t+1$ if $v_i = 1$ at time t. Committing a micro-partition for allocation at time t would change the value of its v_i from 0 at time t to 1 at time $t+1$. As a result of a commitment action at time t, a state $s(V_t,\ t)$ would deterministically transit to another state $s(V_{t+1}, t+1)$.

The general MDP model is shown in Figure 5. The initial state is denoted by $s(V_0,\ 0)$. Since every micro-partition is initially uncommitted, every dimensional value v_i in V_0 is equal to 0. The final state is denoted by $s(V_{K+1},\ K+1)$. Since every micro-partition has to be committed for allocation at the end of the planning process, every dimensional value in V_{K+1} is equal to 1. At any decision point t_i $(1 \le i \le K)$, the total number of commitment states of V_i is equal to 2^N. A MDP example with 2 micro-partitions and 2 decision points is also shown in Figure 6. The colored path represents a feasible solution of state transfer, which allocates P_1 at D_1 and P_2 at D_2.

It is observed that committing a micro-partition for allocation would enable its immediate shuffling, but also reduce the opportunities for later balancing adjustment. Therefore, the definition of the reward function has to consider the trade-off between these two conflicting factors. We define the reward function as

$$R_{a_t}(s_t, s_{t+1}) = \frac{W_c^t}{W} \cdot \frac{N_u^t}{N} \tag{3}$$

in which W denotes the total size of all the micro-partitions, W_c^t denotes the total size of the micro-partitions committed at time t, N denotes the total number of micro-partitions and N_u^t denotes the total number of micro-partitions remaining uncommitted after t. In Eq. 3, the first part of (W_c^t/W), which corresponds to the size percentage of the micro-partitions committed at t to all the micro-partitions, measures the benefit of early shuffling. The second part of (N_u^t/N) represents the penalty of reduced balancing opportunities as a result of micro-partition commitment. Committing more micro-partitions at a decision point t would result in an increased value of W_c^t but a decreased value of N_u^t. It is worthy to point out that both extreme choices, committing no micro-partition or committing all the available micro-partitions at a decision point, would result in the minimal reward of 0.

We consider the discount of a future reward as a penalty of retrieving more accurate sampling results. In simple random sampling method, the sampling accuracy increases with the sample size. And the actual error is square root with the sample size [31]. In this paper, we borrow the definition of actual error and support sampling accuracy is square root with the sample size. We suppose the process speed is a constant for each mapper. And the interval time is the same between each adjacent decision point. So the total size of retrieved sample at the last decision point (t_K) is K times the sample size retrieved at the first decision point (t_1). Correspondingly, the sampling accuracy achieved at t_K is $\sqrt{K}$ times the sampling accuracy achieved at t_1. Combining formula Eq. 2 we have

$$\gamma^{K-1} \cdot \sqrt{K} = 1. \tag{4}$$

Correspondingly, the value of the discount factor (γ) is specified by

$$\gamma = K^{-\frac{1}{2(K-1)}}. \tag{5}$$

5.2 An Optimal Algorithm

The standard optimal algorithm for MDP repeats two types of computations in same order for all the states until no further state transit takes place. They are recursively defined as follows:

$$V(s) := \sum_{s'} P_{\pi(s)}(s, s')\,(R_{\pi(s)}(s, s') + \gamma V(s'))\,; \tag{6}$$

$$\pi(s) := \arg\max_a \left\{ \sum_{s'} P_a(s, s')\,(R_a(s, s') + \gamma V(s')) \right\}. \tag{7}$$

At the end of the algorithm, π will contain the solution and $V(s)$ will contain the discounted sum of the rewards to be earned (on average) by following that solution from state s.

In the proposed MDP, the number of probable states is $O(K \cdot 2^N)$, in which K and N denote the number of decision points and micro-partitions respectively. However, the number of the states probably traversed by an optimal solution can be shown to be only $O(K \cdot N)$. Given a decision point t_i, suppose that the total number of uncommitted micro-partitions is N_i. We can prove that the action taken at each decision point,$\{t_i,t_{i+1},..,t_K\}$, as specified by an optimal solution, would commit j top-sized uncommitted micro-partitions, in which $0 \leq j \leq N_i$. In other words, the optimal action always commits *the largest uncommitted micro-partitions* at any decision point. The optimal number of micro-partitions committed at each decision point, however, needs to be further determined. We have the following lemma:

Lemma 1. *In the proposed MDP model, the action is taken at any decision point as specified by an optimal solution always commits the set of micro-partitions consisting of the largest available ones.*

Proof. Suppose that in an optimal solution SL_1, the action at a decision point t_i commits a set of micro-partitions, P_{t_i},which are *not* the largest available ones. Then, there exists a micro-partition p uncommitted at t_i whose size is larger than a micro-partition p' in P_{t_i} ($|p| > |p'|$). We also suppose that p' is committed at t_j ($t_i < t_j$) in SL_1. We construct an alternative solution SL_2. The solution SL_2 takes the same actions as SL_1 except that it commits the micro-partition p' at t_i and p at t_j. According to the definition of the reward function, we have

$$\frac{R_{i,j}(SL_1)}{R_{i,j}(SL_2)} = \frac{W_c^{1i} \cdot N_u^{1i} + \gamma^{j-i} \cdot W_c^{1j} \cdot N_u^{1j}}{W_c^{2i} \cdot N_u^{2i} + \gamma^{j-i} \cdot W_c^{2j} \cdot N_u^{2j}} \tag{8}$$

in which $R_{i,j}(SL_1)$ represents the sum reward that SL_1 achieves at t_i and t_j, and W_c^{1i} represents the sum size of micro-partitions committed at t_i in SL_1, N_u^{1i} represents the number of micro-partitions remaining uncommitted after t_i in SL_1. Note that N_u^{1i}=N_u^{2i}, N_u^{1j}=N_u^{2j} and N_u^{1i}>N_u^{1j}. We also have

$$W_c^{1i} - W_c^{2i} = W_c^{2j} - W_c^{1j} = |p'| - |p| < 0. \tag{9}$$

Therefore, we have

$$R_{i,j}(SL_1) < R_{i,j}(SL_2). \tag{10}$$

Note that at any decision point other than t_i and t_j, the solutions SL_1 and SL_2 achieve the same rewards. Therefore, SL_2 achieves a total reward larger than that of SL_1. Contradiction. □

According to Lemma 1, the optimal algorithm only needs to consider committing the top-k largest uncommitted micro-partitions at each decision point. Therefore, we have the recursive reward function as follows:

$$R(s_{t-1}) = \max_{a_{t-1}} \{R_{a_{t-1}}(s_{t-1}, s_t) + \gamma \cdot R(s_t)\} \tag{11}$$

in which $R(s_{t-1})$ represents the maximal reward that the state s_{t-1} can receive at the decision point $t-1$, a_{t-1} represents the action of committing top-k largest available micro-partitions at $t-1$, s_t represents the state at t as a result of the action a_{t-1} at s_{t-1}. The optimal algorithm, based on dynamic programming, recursively computes $R(s_i)$ beginning with the initial state s_0. In the process, the maximal reward of every probable state is remembered to avoid redundant computations. We have the observation that the number of probable states at each decision point is bounded by $\mathbf{O}(N)$. As a result, the total number of probable states are bounded by $\mathbf{O}(K \cdot N)$. We also observe that in Eq. 11, $R_{a_{t-1}}(s_{t-1}, s_t)$ can be computed in an incremental way. The computation of Eq. 11 therefore takes $\mathbf{O(N)}$ time. Therefore, we have the following theorem:

Theorem 1. *The optimal dynamic programming algorithm has the space complexity of* $\mathbf{O(K \cdot N)}$ *and the time complexity of* $\mathbf{O(K \cdot N^2)}$

The purpose of the designed MDP model is to choose a series of micro-partition commitment actions at decision points such that the cumulative sum of rewards is maximal. An optimal MDP plan made at the first decision point t_1, is based on the retrieved sampling results up to t_1. In principle, it can also specify the optimal actions taken at every other decision point besides t_1. However, its effectiveness depends on the assumption that the estimated sizes of micro-partitions remain stable throughout the mapping process. Unfortunately, their estimation sizes may instead fluctuate wildly in practice. Therefore, the optimal MDP plan needs to be recomputed at each decision point after t_1. At a decision point t_i, the planner would commit the micro-partitions as specified by the computed optimal plan at t_i. The state at t_i would then transit to another state at t_{i+1} correspondingly.

5.3 A Greedy Algorithm

To reduce the MDP optimization overhead, we propose a greedy but more efficient algorithm for micro-partition scheduling.

At each decision point t, the greedy algorithm always chooses to commit the top-k largest uncommitted micro-partitions such that the immediate reward of state transit from

s_t to s_{t+1} is maximized. In addition, the sorting would take $O(NlnN)$ time. The algorithm is sketched in Algorithm 1. P_u denotes the set of uncommitted micro-partitions before the decision point t. P_c denotes the set of micro-partitions in P_u selected for allocation commitment at t.

Algorithm 1 A Greedy Algorithm

```
Require: $t$,$P_u$(the set of uncommitted micro-partitions)
Ensure: $P_c$(a subset of $P_u$)
 1: Sort the $h$ micro-partitions in $P_u$, $\{p_1,\ldots,p_h\}$, by size in
    the decreasing order;
 2: $P_c = \emptyset$; //the set of committed micro-partitions
 3: w=0; // the total weight of the committed micro-
    partitions
 4: k=1; // the numbers of committed micro-partitions at
    each decision point
 5: R=0; // total rewards of the committed micro-partitions
    at each decision point
 6: while $k \leq h$ do
 7:     w=w+$|p_k|$;
 8:     R'=$\frac{w}{W} \cdot \frac{h-k}{N}$;
 9:     // commit the micro-partition whose reward could in-
        crease the total reward
10:     if $R' \geq R$ then
11:         Insert $p_k$ into $P_c$;
12:         R=R';
13:         k=k+1;
14:     else
15:         Break;
16:     end if
17: end while
```

On the time complexity of Algorithm 1, we have the following theorem:

Theorem 2. *The greedy algorithm as presented in Algorithm 1 has the time complexity of $O(K \cdot NlnN)$.*

5.4 Empirical Validation

This subsection empirically validates the effectiveness of the proposed MDP model. We use the synthetic data satisfying the Zipf-γ distribution in empirical evaluation. Its generator [32] uses the parameter of γ, which presents the value of the exponent characterizing the distribution, to control the skew extent: a larger value means a more skewed distribution. We totally run 5 rounds of mappers and each round simultaneously runs 10 mappers. Each mapper processes 64MB data.

The detailed experimental results are presented in Figure 7. The X-axis represents the decision points. The Y-axis represents the number of micro-partitions committed at each decision point. It can be observed that the optimal algorithm and the greedy one perform very similarly. In both cases, most micro-partitions are committed at several initial decision points. Generally (but with exceptions), the number of committed micro-partitions decreases with time. The difference is that the plan generated by the greedy algorithm is slightly more aggressive: it commits more micro-partitions in the first two rounds. According to the experimental results, most micro-partitions can begin to be shuffled at the early stage of the map phase and only some smallest-sized micro-partitions would be committed at the later stage. These results demonstrate that the MDP plan is effective in balancing the trade-off between reducing partition skew and early reduce start.

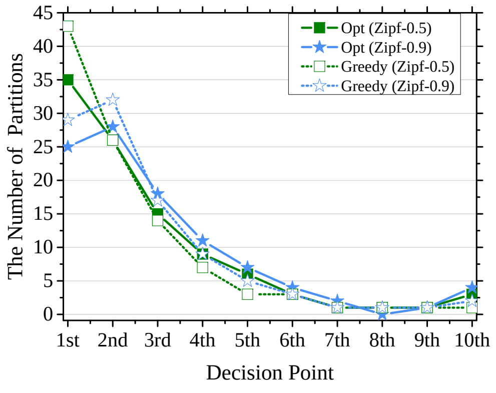


Fig. 7: Empirical Validation of MDP

It is also interesting to observe that the MDP model has a desirable property: as the value of s increases, which means the distribution becoming more skewed, the scheduling plan would become more conservative. In other words, in the case of more skewed distribution, the MDP plan would reserve more micro-partitions for later commitment. For instance, when $s = 0.5$, the optimal MDP plan commits 35 micro-partition at the first decision point. In comparison, if $s = 0.9$, it commits only 25 micro-partitions at the first decision point.

6 Micro-Partition Allocation

We formulate the problem of micro-partition allocation at a decision point t as a generalized multiprocessor scheduling problem [33]. Suppose that at t, the initial workload already allocated to the ith ($1 \leq i \leq M$) reducer is L_i. We also suppose that according to the MDP plan, there are total h micro-partitions supposed to be committed for allocation at t, whose set is denoted by P_c. At the decision point t, the optimization problem of micro-partition allocation is to allocate these h micro-partitions to M reducers such that each reducer achieves a balanced load. We use the variants x_{ij} to denote the mapping between the micro-partitions in P_u and reducers: $x_{ij} = 1$ if the ith micro-partition is allocated to the jth reducer, and $x_{ij} = 0$ otherwise. The micro-partition allocation problem is an optimization problem which can be formulated as follows:

$$\begin{aligned} &\min \max_j \{L_j^f\} \qquad (12)\\ &s.t.\ \forall 1 \le i \le h, \sum_{1\le j\le M} x_{ij} = 1,\\ &\qquad \forall 1 \le j \le M, L_j^f = L_j + \sum_i (x_{ij} \cdot e_i^c) \end{aligned}$$

in which e_i^c denotes the estimated size of the ith micro-partitioin in P_u and L_j^f denotes the estimated total size of micro-partitions allocated to the jth reducer.

Theorem 3. *The micro-partition allocation problem is NP-complete.*

Proof. Multiprocessor scheduling problem (MSP) is an optimization problem, that was proved NP-complete by J.D. Ullman in 1975 [33]. This problem is expressed as that nonpreemptively scheduling n independent tasks on m identical parallel processors with the objective of minimizing the "makespan" T. We take an instance of multiprocessor scheduling as: Set S of tasks, number $m \in Z$ of processors, length $l(s) \in Z$ for each $s \in S$, and a deadline $D \in Z$. The question is "Is there an m-processor schedule for S that meets the overall deadline D ?".

And we also take an instance of micro-partition allocation problem (MAP) as: Set S' contains n' micro-partitions, number $m' \in Z'$ of Reducers, the weight for each micro-partition is $l'(s') \in Z$ for each $s' \in S'$, and the maximal weight of Reducers is limit as D'. The question is "Is there an m'-Reducers allocation plan for S' micro-partitions, which make the maximal weight of Reducers to meet the limit weight D' ?" .

First, the micro-partition allocation problem is NP. Because we could verify the maximal weight of any solution whether it is larger than D' in polynomial time.

Next, we are going to reduce the known NP-complete problem MSP to the new one MAP in polynomial time. We transform the tasks to the micro-partitions and Reducers to the processors. And the length function $l(s)$ can be linearly transformed to the weight function $l'(s')$. Therefore, we could transform the input of MSP to the input of MAP in polynomial time. Under the inputs, supposed there was a solution using the cost function $l(s)$ in MSP, that the solution must be a solution using the cost function $l'(s')$ in MAP. And the reverse is true. So we reduced the instance of MSP to MAP.

According to the above, we conclude that the micro-partition allocation problem is NP-complete. □

Therefore, we use the classical LPT (Largest Processing Time first) algorithm, which was originally proposed for the multiprocessor scheduling problem, to solve it. With the approximation ratio of 4/3, the LPT algorithm has been empirically shown to be able to achieve good performance [34, 35]. It first ranks the micro-partitions in the decreasing order of their sizes, and then allocates a micro-partition to a reducer with the smallest data load one by one.

7 Implementation on Hadoop

In this section, we describe the details of implementing our method on the native Hadoop.

The main implementation of allocating partitions to reducers on the native Hadoop is shown as the blocks without the dotted-line in Fig.8. On the native Hadoop, after a mapper is finished, store the temporary data and write each partition's index-address in *MapOutputFile* on local. And send the "finished" message to *JobTracker* in *Heartbeat*. After *JobTracker* gets this message, and randomly allocates one partition to a reducer by sending one index-address message in *Heartbeat*. For each launched reducer, it can acquire the allocated partition's message and store it in *LocalPartition* on local and calls *Read()* function to pull the finished mapper's data. As reading is multi-thread, each mapper's index is stored in *MapOutput-Locations*. Once all mappers are finished, reducers can run the *Reduce()* function.

For our method, we need to develop three function modules and four data structures on the native Hadoop, which are the dotted-line blocks in Fig.8 . Compared with the native implementation, our multiple allocation rounds strategy is more complex. While a mapper is executing, instead of writing $< key, value >$ pairs into buffer directly, each pair would be processed by *Sample()*. *Sample()* is used to describe the distribution of micro-partition by summarizing the total number of *keys* which have the same micro-partition value. After that, the distribution is stored in *LocalSamplingTable* in memory. At each decision point, *TaskTracker* submits its own *Local-SamplingTable* to *JobTracker* in *Heartbeat*. And then, *Job-Tracker* would update *GlobalSamplingTable* by summering all *LocalSamplingTable*. Now, the distribution of all pro-

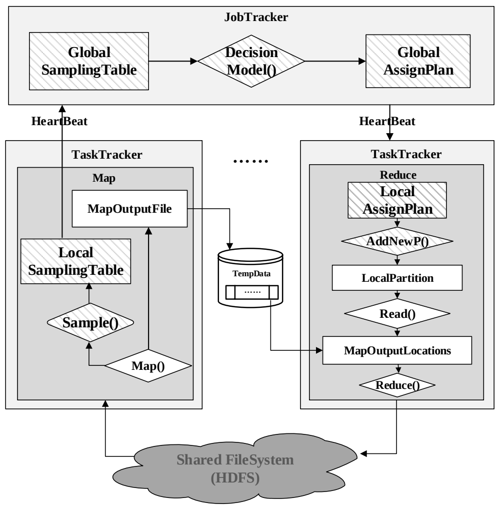


Fig. 8: The Implementation in Native Hadoop

cessed data is stored in *GlobalSamplingTable*. So *JobTracker* can run *DecisionModel() to make an allocation plan* by running Alg. 1 and store the plan in *GlobalAssignPlan*. In the next *Heartbeat*, *JobTracker* pushes the current *GlobalAssignPlan* to *TaskTracker*. When *TaskTracker* receives a new plan, it can only sift micro-partitions allocated to it and store them in *LocalAssignPlan* in local memory. At each decision point, *AddNewP()* would add new allocated micro-partitions into *LocalPartition*. After that, like the native Hadoop, the reducers would wait for the finished mapper's data to run *Reduce()* function.

8 Experimental Evaluation

In this section, we empirically evaluate the performance of the proposed incremental approach, which is denoted by *IPS* (*I*ncremental micro-*P*artition *S*cheduling), by the comparative study. We compare its performance with that of three alternatives: the native approach implemented on the open-source *Hadoop* , the *Closer* approach [9] and *Libra* approach [11]. The *Closer* approach essentially allocates micro-partitions to reducers in a single round based on partial size estimation results. In this comparative study, we set the adjustment point of *Closer* to be the time points when 20% or 80% of mapper jobs have been completed. The points of 20% and 80% represent the early and later adjustment points commonly used in practical implementation respectively. In *Libra*, we sample 20 percent of the map tasks. In *IPS*, we set the default *amplification coefficient*, which refers to the times the number of micro-partitions is to the number of reducers (or N/M), to be 50. And the number of decision points is 10. Since the optimal and the greedy algorithms for micro-partition scheduling, as presented in Section 5, perform very similarly, we present the performance of *IPS* with the greedy scheduling algorithm in our comparative study.

We use one synthetic data set and two real data sets, whose details are presented in Table 1. The synthetic data set satisfies the *Zipf* distribution. Its generator [32] uses the parameter of γ, which presents the value of the exponent characterizing the distribution, to control the skew extent: a larger value means a more skewed distribution. The two real data sets are BTS [36] and UK-2002 [37]. The BTS data set records the departure and arrival times reported by U.S. air carriers. The UK-2002 data set records a web page graph resulting from a 2002 crawl of the UK domain performed by UbiCrawler. We run the simple WordCount algorithm on the *Zipf* and BTS data sets, and the PageRank algorithm on the UK-2002 data set.

The approaches are evaluated on three metrics: (1) coefficient of variation (cv) of data loads, σ/μ, in which σ and μ represent the standard deviation and the mean respectively; (2) maximum of data loads; (3) maximum of running time. Note that the data load of a reducer is measured by the total size of micro-partitions allocated to it. All the experiments were executed on a cluster consisting of one master machine and ten slave machines. Each machine has an AMD processor with 16 2.20GHz cores, 16GB RAM, and 500GB hard disks. Each machine was installed with the 64-bit Ubuntu Linux 10.04 and Hadoop 1.1.2. Each machine can run up to 16 mappers or reducers simultaneously. We repeated each experiment 3 times and reported the averaged running time.

The rest of this section is organized as follows: In Subsection 1, we briefly evaluate the performance of *Closer*. In Subsection 2, we briefly evaluate the performance of *IPS*. In Subsection 3, we present the comparative results on the synthetic data. In Subsection 4, we present the comparative results on the real data set.

8.1 Performance of *Closer*

This subsection evaluates how the performance of *Closer* varies with the chosen adjustment point. We use the synthetic data and set 9 adjustment points varying from 10% of mapping progress to 90%. The mapping phase is executed in totally 10 rounds. The detailed evaluation results are presented in Figure 9. It can be observed that reducer data loads tend to become more balanced as the point percentage increases. The exceptions result from sampling fluctuation.

Table 1: Details of the Test Datasets

Dataset	Algorithm	Size(GB)	Tuple(Billion)	Description
Zipf-γ	WordCount	4	1	The standard Zipf distributions
BTS	WordCount	60	19	Departure and arrival times reported by U.S. air carriers. [36]
UK-2002	PageRank	2.5	0.18	A 2002 crawl of the uk performed by UbiCrawler [37]

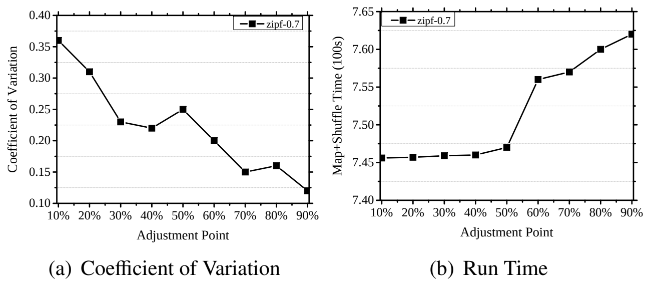


Fig. 9: The evalution results of *Closer* on *Zifp*-0.7

On the consumed map and shuffle time, the performance of *Closer* instead deteriorates as the point percentage increases due to slower shuffle start points.

8.2 Performance of *IPS*

In this section, we evaluate the effect of amplification coefficient and the number of decision points on the performance. We experiment this part on two synthetic data sets $Zipf-0.3$ and $Zipf-0.7$, which contain 1,000 kinds of tuples to run the WordCount instance.

Amplification Coefficient

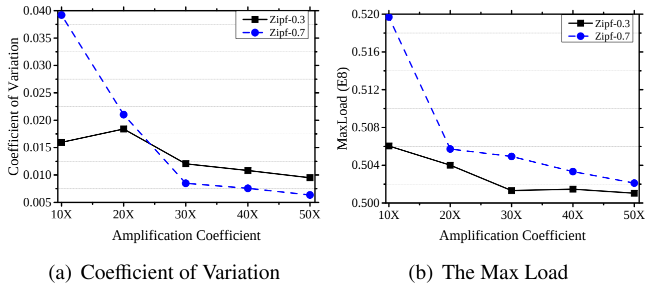


Fig. 10: Evaluate amplification coefficient on $Zipf-0.3/0.7$

The granularity of micro-partition is mainly affected by its total number. In this paper, we use the hash partition function and renew it to $key\%(M*\lambda)$. M is set to 20 and λ is denoted as amplification coefficient. And we vary λ from 10 to 50, increased by 10. When it is 50, each micro-partition contains only one kind tuple who has the same value.

The experimental results are presented in Figure 10. When the value of λ is increased from 10 to 30, both coefficient of variation and the max load are reduced obviously on two data sets. And the results of $Zipf-0.7$ are more significant. Continually increasing its value, the changing curve of the coefficient of variation and the max load is smooth. For $Zipf-0.7$, the value of coefficient of variation is smaller than $Zipf-0.3$'s from 30 to 50, while its max load is bigger. This is caused by the distributions of the data set. In comparison, the performance of IPS would be better with the large λ. And we can increase the value of it for more skew datas. But for different distributions, the effect can be smooth until some value. Moreover, the larger the number of micro-partitions is, the larger the load would be while shuffling. In our practice, it would be fine if the value of λ is between 30 and 50. In the rest of this paper, we set the value of λ to 50.

The Number of Decision Points

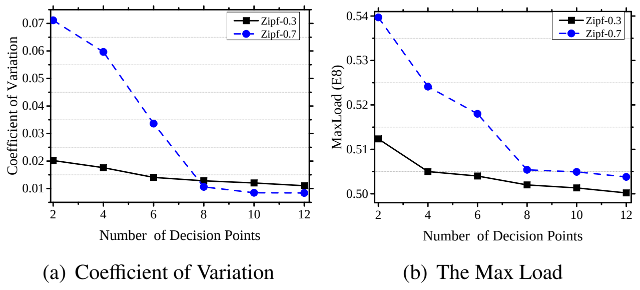


Fig. 11: Evaluate decision numbers on $Zipf-0.3/0.7$

The number of decision points impacts how many times should all the micro-partitions be allocated on reducers. In this paper, we define the decision point in term of mappers' processing ratio. And we mark the first decision point as the first mapper's finished time, and do the same for the last. Then we divide equally all the stage as some parts.

We do six groups of experiments and the results are presented in Figure 11. When the number of decision points is increased from 2 to 12 , both coefficient of variation and the max load are reducing constantly, which are obvious when the argument is changing from 2 to 8. And both of them are stable after 8. The reason of which is the decision points are in the Map phase. By adding the decision points, we can correct the deviation caused by the earlier allocation. But, for each data set, there would be an optimal amount of decision points, which is caused by the total number of micro-partitions. In our practice, we find the number of decision points would be better set as 10. Thus we set it to 10 in the rest of this paper.

8.3 Comparative Evaluation On Synthetic Data

We generate 11 sets of $Zipf$-γ data, where the value of γ varies from 0 to 1. The detailed evaluation results are presented in Figure 12. Figure 12 (a) reports the coefficient of variation among reducer data loads. It can be observed that the native Hadoop performs poorly even in the case of $\gamma = 0$, which means that the micro-partition sizes are the least skewed. Its coefficient of variation increases with the value of γ. The *Closer* and *Libra* methods perform better than the native approach. It achieves well-balanced reducer workloads when $\gamma \leq 0.3$. However, when $\gamma \geq 0.4$, the data loads among reducers become considerably less balanced. It can also be observed that *IPS* achieves better performance than both *Native Hadoop*, *Closer* and *Libra*. The results showed that *IPS* achieved well-balanced reducer data loads when $\gamma \leq 0.7$. In the cases of $0.8 \leq \gamma \leq 1$, its coefficient of variation increases dramatically but is still smaller than those of *Native Hadoop* and *Closer*.

To get a closer look at the performance of *IPS*, we also present the distribution of reducer data loads in the case of $\gamma = 0.8$ as shown in Figure 12 (d). The X-axis denotes the reducer IDs and the Y-axis denotes their data loads. The numbers above the rectangles in histogram represent the total numbers of micro-partitions allocated to reducers. It can be observed that the reducer with the largest load contains only one micro-partition. The data loads on other reducers are well balanced. Experimentally, the result achieved by *IPS* can be said to be the best possible with the predefined micro-partitions.

The evaluation results on the maximum of data load and run time are presented in Figure 12 (b) and (c) respectively. On data load maximum, *IPS* performs better than colorall of *Native Hadoop*, *Closer* and *Libar*. Generally, its performance advantage increases with the size skew of micro-partitions. The experimental results on run time are similar.

8.4 Comparative Evaluation On Real Data

We plot the distribution of key values in the BTS data in Figure 13 (a). The X-axis denotes the key values and the Y-axis denotes the occurrence frequency of each key. It can be observed that the distribution is skewed: the majority of key values have low or moderate frequencies and only a few have high frequencies.

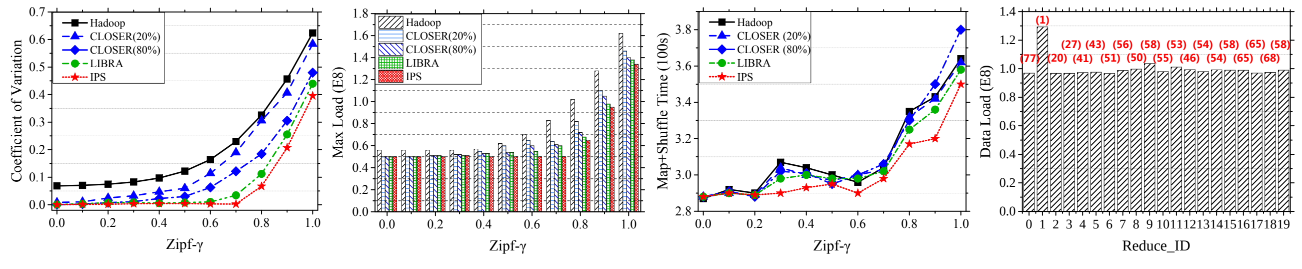
The experimental evaluation results on BTS in term of coefficient variation, maximum data load and run time are presented in Figure 13 (b), (c) and (d) respectively. The X-axis denotes the number of reducers. On coefficient variation, it can be observed that *IPS* performs significantly better than both *Native Hadoop*,*Closer* and *Libra*. On maximum load, IPS also consistently outperforms them. The evaluation results on run time are similar to what was observed on maximum load.

The evaluation results on the UK-2002 data set are also presented in Figure 14. Note that the PageRank algorithm has to be executed iteratively on Hadoop. We suppose that the mapping between micro-partitions and reducers remains unchanged at each iteration. Figure 14 reports the results of the first iteration. The experimental results are similar to what is observed on the BTS dataset. *IPS* consistently outperforms *Native Hadoop* and *Closer*. Note that on maximum load, plotting all the comparative results in a figure would make the performance of the three approaches appear very similar. Therefore, in Figure 14 (b), we only present the comparative result with the number of reducers set to be 10. The evaluation results on other numbers of reducers are similar.

With the number of reducers set to be 10, we also present the four approaches' performance difference on the time consumed by the map, shuffle and reduce phases in Figure 14 (d). It can be observed that they consume roughly the same time on the map phase. All of *Native Hadoop*, *IPS* and *Libra* can start to shuffle mapped data once the first mapper finishes its job. Therefore, they share the same shuffle starting point. Since *IPS* reserves some micro-partitions for later allocation commitment, it consumes slightly more time than *Native* on the shuffle phase. Even though *Closer* may consume less time on the shuffle phase, its shuffle phase has a later end point. *Libra* using the range partition method is more unbalanced than the hash partition method on UK-200. On the reduce phase, *IPS* consumes the least time because it has the smallest maximum load. *IPS* also achieves the best overall performance among them.

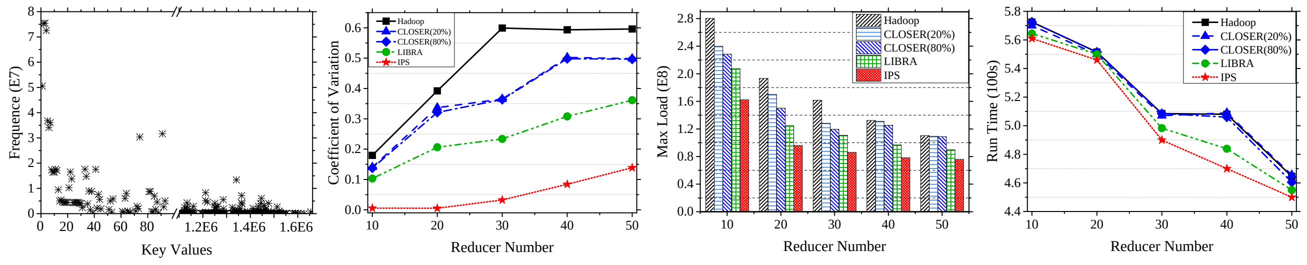
9 Conclusion

In this paper, we propose an incremental allocation approach to reduce partition skew on MapReduce. The approach consists of two steps: micro-partition scheduling and micro-partition allocation. We proposed effective and efficient solutions for both problems. Finally, our extensive experiments on synthetic and real data have shown that compared with the



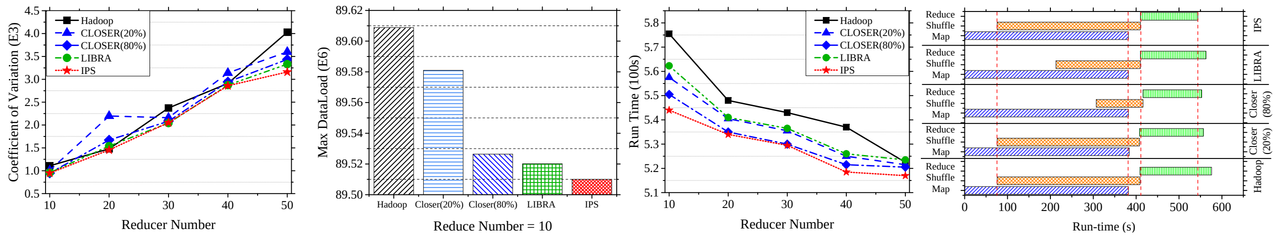
(a) Coefficient of Variation (b) Max Load (c) Run Time (d) Results on *Zipf*-0.8

Fig. 12: The Evaluation Results on *Zifp*-γ



(a) Distribution of Key Values (b) Coefficient of Variation (c) Max Load (d) Run Time

Fig. 13: The Evaluation Results on BTS



(a) Coefficient of Variation (b) Max Load (c) Run Time (d) Run Time(ReduceNum=10)

Fig. 14: The Evaluation Results on uk-2002

state-of-the-art solutions, the incremental approach achieved considerably better data load balance as well as overall better parallel performance.

Acknowledgements This work was supported by the Ministry of Science and Technology of China, National Key Research and Development Program (2016YFB1000703), the National High Technology Research and Development Program 863 of China (2015AA015307), the Natural Science Foundation of China (Nos. 61332006, 61672432, 61472321, 61502390).

References

1. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *To appear in OSDI*, page 1, 2004.
2. Feng Li, Beng Chin Ooi, M Tamer Özsu, and Sai Wu. Distributed data management using mapreduce. *ACM Computing Surveys (CSUR)*, 46(3):31, 2014.
3. Apache Hadoop. Hadoop, 2009.
4. Jimmy 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*, volume 1, 2009.
5. Kai Ren, Garth Gibson, and YongChul Kwon. Hadoop's adolescence; a comparative workloads analysis from three research clusters. In *SC Companion*, page 1452, 2012.
6. Sharat Chandra Racha. Load balancing map-reduce communications for efficient executions of applications in a cloud. 2012.
7. Lars Kolb and Andreas Thor. Block-based load balancing for entity resolution with mapreduce. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2397–2400. ACM, 2011.
8. Lars Kolb, Andreas Thor, and Erhard Rahm. Load balancing for mapreduce-based entity resolution. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 618–629. IEEE, 2012.
9. B Gufler, N Augsten, A Reiser, and A Kemper. Handing data skew in mapreduce. In *Proceedings of the 1st International Conference on Cloud Computing and Services Science*, volume 146, pages 574–583,

2011.

10. Benjamin Gufler, Nikolaus Augsten, Angelika Reiser, and Alfons Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 522–533. IEEE, 2012.
11. Qi Che, Jinyu Yao, and Zhen Xiao. Libra: Lightweight data skew mitigation in mapreduce. volume 26, pages 2520–2533, Sept.2015.
12. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
13. David DeWitt and Michael Stonebraker. Mapreduce: A major step backwards. *The Database Column*, 1, 2008.
14. YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. A study of skew in mapreduce applications. *Open Cirrus Summit*, 2011.
15. Alexander Rasmussen, Michael Conley, George Porter, Rishi Kapoor, Amin Vahdat, et al. Themis: an i/o-efficient mapreduce. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 13. ACM, 2012.
16. Kai Ren, YongChul Kwon, Magdalena Balazinska, and Bill Howe. Hadoop's adolescence: an analysis of hadoop usage in scientific workloads. *Proceedings of the VLDB Endowment*, 6(10):853–864, 2013.
17. Juwei Shi, Jia Zou, Jiaheng Lu, Zhao Cao, Shiqiang Li, and Chen Wang. Mrtuner: A toolkit to enable holistic optimization for mapreduce jobs. *Proceedings of the VLDB Endowment*, 7(13):1319–1330, 2014.
18. Behrooz A Shirazi, Krishna M Kavi, and Ali R Hurson. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Society Press, 1995.
19. Veeravalli Bharadwaj. *Scheduling divisible loads in parallel and distributed systems*, volume 8. John Wiley & Sons, 1996.
20. Shadi Ibrahim, Hai Jin, Lu Lu, Song Wu, Bingsheng He, and Li Qi. Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 17–24. IEEE, 2010.
21. Shadi Ibrahim, Hai Jin, Lu Lu, Bingsheng He, Gabriel Antoniu, and Song Wu. Handling partitioning skew in mapreduce using leen. *Peer-to-Peer Networking and Applications*, 6(4):409–424, 2013.
22. Prateek Dhawalia, Sriram Kailasam, and Dharanipragada Janakiram. Chisel: A resource savvy approach for handling skew in mapreduce applications. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 652–660. IEEE, 2013.
23. Rares Vernica, Andrey Balmin, Kevin S Beyer, and Vuk Ercegovac. Adaptive mapreduce using situation-aware mappers. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 420–431. ACM, 2012.
24. Smriti R Ramakrishnan, Garret Swart, and Aleksey Urmanov. Balancing reducer skew in mapreduce workloads using progressive sampling. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 16. ACM, 2012.
25. Raman Grover and Michael J Carey. Extending map-reduce for efficient predicate-based sampling. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 486–497. IEEE, 2012.
26. YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 25–36. ACM, 2012.
27. Prateek Dhawalia, Sriram Kailasam, and Dharanipragada Janakiram. Chisel++: handling partitioning skew in mapreduce framework using efficient range partitioning technique. In *Proceedings of the sixth international workshop on Data intensive distributed computing*, pages 21–28. ACM, 2014.
28. Ahmed Metwally and Christos Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. *Proceedings of the VLDB Endowment*, 5(8):704–715, 2012.
29. M Al Hajj Hassan, Mostafa Bamha, and Frédéric Loulergue. Handling data-skew effects in join operations using mapreduce. *Procedia Computer Science*, 29:145–158, 2014.
30. YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skew-resistant parallel processing of feature-extracting scientific user-defined functions. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 75–86. ACM, 2010.
31. William G Cochran. *Sampling techniques*. John Wiley & Sons, 2007.
32. Kenneth J. Christensen. http://www.csee.usf.edu/ christen/tools/genzipf.c.
33. J. D. Ullman. Np-complete scheduling problems. j comput syst sci. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.
34. Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.
35. RL Graham. Bounds on the performance of scheduling algorithms. *Computer and job scheduling theory*, pages 165–227, 1976.
36. BTS. http://www.transtats.bts.gov.
37. The University of Florida Sparse Matrix Collection. http://law.di.unimi.it/webdata/uk-2002/.

Zhuo Wang is a Ph.D. candidate at School of Computer Science and Technology, North Western Polytechnical University, China. He received his master degree from Tianjin University of Technology in 2011. His research interests include big data analysis, data management.

Qun Chen is a professor at School of Computer Science and Technology, North Western Polytechnical University, China. He is a member of China Computer Federation. He received his MS and Ph.D. degrees from NUS. His research interests include data management and big data analysis.

Bo Suo is a Ph.D. candidate at School of Computer Science and Technology, North Western Polytechnical University, China. He received his master degree from NWPU in 2010. He is a student member of China Computer Federation. His current research interests include big data analysis and big graph.

Wei Pan is an associate professor at School of Computer Science and Technology, North Western Polytechnical University, China. He is a member of China Computer Federation. His current research interests include big data analysis and in memory database.

Zhanhuai Li is a professor at School of Computer Science and Technology, North Western Polytechnical University, China. His research interests include data management and data mining.