# Towards Location-aware Joint Job and Data Assignment in Cloud Data Centers with NVM

Xin Li*†‡, Jie Wu§, Zhuzhong Qian†‡, Shaojie Tang¶, and Sanglu Lu†‡
*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
†State Key Laboratory for Novel Software Technology, Nanjing University
‡Collaborative Innovation Center of Novel Software Technology and Industrialization
§Center for Networked Computing, Temple University
¶Naveen Jindal School of Management, The University of Texas at Dallas
Email: lics@nuaa.edu.cn, jiewu@temple.edu, {qzz, sanglu}@nju.edu.cn, shaojie.tang@utdallas.edu

***Abstract*—In this paper, we investigate the joint job and data assignment problem in cloud data centers with non-volatile memory (NVM) for makespan minimization. Through extensive analysis, we find that there is an indicator variable that characterizes the hardness of the problem. Depending on the value of the indicator variable, we classify our problem into three cases: *inf-case*, *opt-case*, and *nph-case*. We first show that there is no feasible assignment under the *inf-case*. For the *opt-case*, we present an optimal algorithm. We show that a mixed data assignment with diversified popularity achieves high memory utilization. For the *nph-case*, we first prove the problem's NP-hardness and then propose a heuristic algorithm and a 2-approximation algorithm to tackle it. We conduct extensive simulations, and we find that the performance of the heuristic algorithm is better than the 2-approximation algorithm and that it is nearly the same as the theoretical optimal solution.**

***Index Terms*—cloud data center, data replication, job assignment, makespan, resource allocation.**

## I. Introduction

Timely data analysis is important because it can support better predictions while help companies to make better decisions. This motivates more firms to harness the power of cloud computing to reduce data-processing time. As data-intensive cloud computing systems like MapReduce [1] and Dryad [2] have gained popularity, a strong need has developed to share resources and run batched jobs over a common data set in the same cloud data center. Job scheduling is the primary issue in data analysis, and data placement is becoming more important [3] [4] as data-intensive applications develop. Hence, makespan optimization is the core objective of the joint job and data assignment problem [5], and researchers believe data locality is an efficient method of reducing makespan [1] [6].

Recent literature proposes many approaches to optimize job scheduling [7] [8] [9], but speeding up data analysis is difficult due to the limited capacity of typical DRAM. Though DRAM is widely used for main memory, researchers think DRAM is approaching scalability limits [10]. In contrast to DRAM, non-volatile memory (NVM) technology will achieve storage-class memory capacity and is expected to be equipped in future data centers [11]. With the increased capacity of NVM, "memory is the new disk" [12] and data can be directly deployed in memory. This motivates us to improve data analysis so it can keep up with faster data access. However, the traditional data replica placement and job scheduling strategy is no longer pratical. It is necessary to reconsider the job and data scheduling issue for the new memory system.

978-1-5090-6468-7/17/$31.00 ©2017 IEEE

In this paper, we investigate the joint job and data assignment problem for makespan minimization in cloud data centers with the new memory system. We focus on how many data replicas will be applicable to batched jobs and how to assign the jobs and data replicas. For data-intensive jobs, it is intuitive to schedule the job to the server that contains its input data in memory. Hence, the joint job and data assignment strategy must guarantee full data locality to take advantage of memory. We formalize the assignment problem for the new scenario, and we prove that the general case of the problem is NP-hard. We also find that there is an indicator variable that can characterize the hardness of the problem quantitatively. This indicator is determined by the relationship between the memory capacity and the expected data load for each server; it has no relationship with the number of jobs. According to the value of the indicator variable, we classify our problem into three cases: *inf-case*, *opt-case*, and *nph-case*. For the *inf-case*, there is no feasible assignment for given inputs. For the *opt-case*, we propose an optimal algorithm and prove its feasibility. For the *nph-case*, we prove the problem's NP-hardness and propose both a heuristic algorithm and a 2-approximation algorithm.

We first use a *tuple* to model the association of the jobs and their input data blocks. For each data block, we define *degree* to represent the popularity of the data block as input. Then, our algorithms are developed based on the modeled tuples and the degrees. Our contributions are summarized as follows.

- We formalize the new problem and demonstrate that its hardness is decided by an indicator variable. This variable is related to the number of available memory slots, the number of servers, and the number of data blocks. But it does not depend on the number of jobs.
- For the *opt-case*, we develop an optimal algorithm, creating a mixed assignment of data blocks with diversified degrees for servers in a load-balanced manner.

- For the *nph-case*, we develop a 2-approximation algorithm and a heuristic algorithm. We first sort the data blocks according to their degrees. In the 2-approximation algorithm, the data block with the least degree will be assigned first; in the heuristic algorithm, the data block with the greatest degree will be assigned first.
- We conduct extensive simulations, and the simulation results show that the heuristic algorithm performs better than the 2-approximation algorithm, and has nearly the same performance as the theoretical optimal solution.

The remainder of this paper is organized as follows. We give the problem statement in Section II. In Section III, we present our algorithms under various cases and analyze their performance theoretically. We evaluate our algorithms in Section IV, and we give conclusions in Section VI.

## II. Problem Statement

### A. Scenario and Notations

For a data center that consists of uniform servers, jobs share both a data set and resources. We assume that each server hosts one job per time slot [13], but it is easy to extend our work to a case with multiple jobs per time slot. We further assume that each job has the same execution time when data locality is guaranteed since the map tasks or reduce tasks of a job in MapReduce have similar execution times [14]. Let $\mathcal{N}$ be the number of uniform servers and $\mathcal{M}$ be the number of memory slots in each server used for storing data blocks. For the common data set, we use $\mathcal{K}$ to represent the number of data blocks. We should note that $\mathcal{K}$ is just the number of different data blocks, not the total number of data block replicas. There could be multiple replicas for a data block to preserve data locality. Therefore, the total number of data block replicas is larger than or equal to $\mathcal{K}$.

We use a two-tuple to represent a job profile. For example, $\langle \mathcal{J}_0, \mathcal{D}_0 \rangle$ represents job $\mathcal{J}_0$ and its input data block $\mathcal{D}_0$. We use $f$ to represent the input data of jobs, i.e., given a job profile $\langle \mathcal{J}_i, \mathcal{D}_j \rangle$, we have $f_i = \mathcal{D}_j$. Hence, the job profile can also be described as $\langle \mathcal{J}_i, f_i \rangle$. Any assignment decision can be represented by the following two indicators:

$$\boldsymbol{\pi}(\mathcal{J}_i, \mathcal{S}_j) = \begin{cases} 1, & \text{Job } \mathcal{J}_i \text{ is assigned to server } \mathcal{S}_j; \\ 0, & \text{otherwise.} \end{cases}$$

$$\boldsymbol{\pi}(\mathcal{D}_i, \mathcal{S}_j) = \begin{cases} 0, & \text{there is no replica of } \mathcal{D}_i \text{ on } \mathcal{S}_j; \\ 1, & \text{otherwise.} \end{cases}$$

$\mathcal{S}_j (1 \leq j \leq \mathcal{N})$ refers to the $j^{th}$ server.

### B. Problem Formulation

**Joint Job and Data Assignment Problem**. Given a data center consisting $\mathcal{N}$ uniform servers with a memory capacity of $\mathcal{M}$ slots and a set of jobs $\{\langle \mathcal{J}_i, f_i \rangle | \exists k, D_k = f_i, 1 \leq i \leq \mathcal{L}, 1 \leq k \leq \mathcal{K}\}$, the joint job and data assignment problem can be formulated as follows:

$$\begin{aligned} \textit{min.} \quad & \max_{1 \leq j \leq \mathcal{N}} \left\{ \sum_{i=1}^{\mathcal{L}} \boldsymbol{\pi}(\mathcal{J}_i, \mathcal{S}_j) \right\} \\ \textit{s.t.} \quad & (1) \ \sum_{i=1}^{\mathcal{K}} \boldsymbol{\pi}(\mathcal{D}_i, \mathcal{S}_j) \leq \mathcal{M}, 1 \leq j \leq \mathcal{N} \\ & (2) \ \boldsymbol{\pi}(\mathcal{J}_i, \mathcal{S}_j) \leq \boldsymbol{\pi}(f_i, \mathcal{S}_j), 1 \leq i \leq \mathcal{L}, 1 \leq j \leq \mathcal{N} \end{aligned}$$

The first constraint indicates that the number of data blocks placed on each server should not exceed the memory capacity $\mathcal{M}$. The second constraint enforces data locality; there should be one replica of the input data for each job on its host server. To achieve a smaller makespan, it is critical to balance the load among all of servers. However, this method may easily violate the data locality constraint. To this end, we develop several locality-aware job assignment strategies that nearly achieve the minimum makespan.

### C. Hardness

Our problem is similar to the *subset-sum* problem. To facilitate our study, we first introduce the *equal-size subset-sum* problem. The equal-size subset-sum problem has one more constraint than the classic subset-sum problem: the size of each subset must be identical. We prove that the *equal-size subset-sum* problem is also NP-hard.

*Lemma 1:* The equal-size subset-sum problem is NP-hard.

*Proof:* The classic subset-sum problem can be formulated as follows: given a multiset of integers $B = \{F_1, \cdots, F_n\}$, determine whether there exists a subset $B^c$ of numbers from $B$ such that $\sum_{F_i \in B^c} F_i = \sum_{F_i \in B} F_i/2$. Then, we construct an *equal-size subset-sum* problem as by constructing a new set of integers $A = \{I_1, \cdots, I_n, \cdots, I_{2n}\}$ where $\forall i \in [1,\ n],\ I_i = F_i + 1$, and $\forall i \in (n,\ 2n],\ I_i = 1$. The problem is to determine whether there exists a subset $A^c$ such that $\sum_{I_i \in A^c} I_i = \sum_{I_i \in A} I_i/2$, and $|A^c| = n$.

If there exists a subset $B^c$ of numbers from $B$ such that $\sum_{F_i \in B^c} F_i = \sum_{F_i \in B} F_i/2$, we let $|B^c| = m$, and we have $m \leq n$. Then, we can construct subset $A^c$ as follows: $A^c = A_1^c \cup A^1$ where $A_1^c = \{I_i | \forall F_i \in B^c,\ I_i = F_i + 1\}$ and $A^1 = \{I_j | I_j = 1, n < j \leq 2n - m\}$. Hence, we have $|A^c| = n$ and $\sum_{I_i \in A^c} I_i = \sum_{F_i \in B^c} F_i + n$. Because $\sum_{F_i \in B^c} F_i = \sum_{F_i \in B} F_i/2$, we have $\sum_{I_i \in A^c} I_i = \sum_{I_i \in A} I_i/2$.

On the other hand, if subset $A^c$ does exist for set $A$, we can pick the items whose value is larger than 1 and construct subset $B^c$ as $B^c = \{F_i | \forall I_i \in A^c,\ F_i = I_i - 1,\ F_i \neq 0\}$. It is easy to prove that $\sum_{F_i \in B^c} F_i = \sum_{F_i \in B} F_i/2$ using the deduction above. Therefore, we conclude that the *equal-size subset-sum* problem is NP-hard. ■

Now we can prove that our assignment problem is NP-hard by reducing it from the equal-size subset-sum problem.

*Theorem 1:* The joint job and data assignment problem is NP-hard.

*Proof:* We prove this theorem by showing that a special case of the problem is NP-hard. We assume that $\mathcal{N} = 2$, $\mathcal{K} = 2 \cdot \mathcal{M}$. Consequently, there is only one replica for each

data block, which indicates that the jobs with the same input data block will be assigned to the server that contains their input data block. And, there are $\mathcal{M}$ data blocks on each server. For data block $\mathcal{D}_i$, we define its *degree* $d_i$ as the number of jobs that take it as their input data. Then, we show that the special case can be reduced from the *equal-size subset-sum* problem.

Based on Lemma 1, given a multiset of integers $A = \{I_1, \cdots, I_n\}$ ($n$ is even) for the *equal-size subset-sum* problem, we construct an assignment problem. In the constructed case, we have $\mathcal{K}$ ($\mathcal{K} = n$) data blocks, and let $d_i = I_i$ ($1 \leq i \leq n$). Hence, the minimized makespan should be $\sum_{I_i \in A} I_i/2$.

If there exists a subset $A^c$ of numbers from $A$ such that $\sum_{I_i \in A^c} I_i = \sum_{I_i \in A} I_i/2$, then the data blocks, whose degree equals some number from $A^c$ i.e. $\exists i, I_i \in A^c, d_i = I_i$, and their associated jobs are selected to place on the first server. The remaining data blocks and jobs will be placed on the second server. This assignment ensures that the two servers have exactly the same load. On the other hand, if there exists an assignment with the minimum makespan, we can extract degree the $d_i$ of the data blocks from the same server. The union of the exacted degrees must be a feasible subset $A^c$ for the *equal-size subset-sum* problem. According to Lemma 1, we conclude that our assignment problem is NP-hard. ■

## III. Algorithms and Performance Analysis

### A. Overview

In general, the assignment problem is similar to the minimum makespan scheduling problem, but the memory capacity constraint distinguishes it. The hardness of the assignment problem also varies with the difference in memory capacity $\mathcal{M}$. Specifically, if the total number of memory slots is too limited to host one replica for each data block (i.e. $\mathcal{M} \cdot \mathcal{N} < \mathcal{K}$ or $\mathcal{M} < \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$ since $\mathcal{M}$ is an integer), then there is no feasible solution to assign all jobs. This is the *inf-case* in this paper. On the other hand, if the memory capacity is large enough to host all of the data blocks (i.e. $\mathcal{M} \geq \mathcal{K}$), then data locality is trivially preserved by creating one replica for all data blocks on each server. The assignment problem is equivalent to the minimum makespan scheduling problem, yet, it can be solved optimally. (*Round-Robin* is a well-known optimal method for solving the scheduling problem when every job has the same execution time.) Therefore, in the rest of this paper, we focus on the case where $\lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil \leq \mathcal{M} < \mathcal{K}$.

### B. opt-case when $\lceil \frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}} \rceil \leq \mathcal{M} < \mathcal{K}$

In this section, we first investigate *opt-case* when $\lceil \frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}} \rceil \leq \mathcal{M} < \mathcal{K}$. Note that under *opt-case*, the number of available data replicas is constrained by memory capacity. Simply creating data replicas in a greedy manner easily violates the data locality constraint. Therefore, we decide to group the jobs according to their input data, i.e., all jobs that have the same input data block are classified into the same group. Later, we focus on deciding the assignment of data blocks and the number of replicas needed for each data block. To optimize the memory slot utilization, we conduct the following preprocess.

**Grouping**. We group the jobs according to their input data. The jobs that have the same input data block are classified into the same group. We then have $\mathcal{K}$ groups. For each group, we denote the number of items (jobs) in the group as its *degree*. A group is represented as $g_i = \langle D_i, d_i \rangle$ ($1 \leq i \leq \mathcal{K}$) where $d_i$ indicates the degree of group $g_i$.

**Sorting**. We sort the groups in ascending *degree* order. In the uniform case, each job has the same execution time. We only consider the *degree* for each group, and the jobs contained in the group are not treated separately.

**Selecting**. We select groups for each server according to certain principle (the principle will be specified later). The selection will be carried out step by step. During the selection procedure, some group is partitioned into two or more sub-groups to distribute the workload (jobs) and decrease makespan. We concretely discuss the selection principle and group partitioning issues later in this section.

**Inserting**. When a group is divided into multiple parts, the unselected parts are inserted into the unselected group sequence while preserving the ascending *degree* order.

It is easy to realize grouping, sorting, and inserting; we mainly discuss how to select the groups and which groups should be divided. Since the best assignment achieves absolute load balancing, we then obtain a lower bound of the optimal solution, i.e. the minimum makespan, represented by $opt$.

$$opt \geq \varpi = \left\lceil \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{K}} d_i \right\rceil \tag{1}$$

where $\varpi$ is the makespan when jobs are assigned in an absolutely load-balanced manner.

To achieve the optimal assignment, there are two constraints that should be satisfied for each server: the memory constraint and the workload constraint. The workload constraint requires that the degree sum of the groups assigned to a server do not exceed $\varpi$. This motivates us to develop an algorithm that can *fully* utilize memory slots and ensure that the workload for each server equals $\varpi$.

Since the degree sum of the selected groups does not always equal $\varpi$, when the degree sum exceeds $\varpi$, we introduce the *partition* operation to cut the amount over $\varpi$. The basic idea of partitioning is to divide one group into two sub-groups with the same input data but with smaller degrees. The two sub-groups will be treated separately. For example, group $\langle D_1, 25 \rangle$ can be partitioned into $\langle D_1, 17 \rangle$ and $\langle D_1, 8 \rangle$.

Following this principle, we list three conditions that serve as the basis of our approach. In the following equations, $p$ and $q$ are indices indicating the first and last items of the unselected group sequence. At the beginning, we have $p = 1$ and $q = \mathcal{K}$.

**Condition 0**:
$\sum_{i=p}^{q} d_i \leq \varpi$, $q - p + 1 \leq \mathcal{M}$

**Condition 1**: $\Omega_1(n)$
$\sum_{i=p}^{p+n-1} d_i - \varpi = s^* \geq 0$, $\sum_{i=p}^{p+n-2} d_i - \varpi < 0$, and $n \leq \mathcal{M}$

**Algorithm 1** Condition-based Selection Algorithm: *csa*

**Input:** $\mathcal{G}$: sorted group sequence

1: $p \leftarrow 1$, $q \leftarrow \mathcal{K}$;
2: $selected \leftarrow \emptyset$, $\mathcal{O}_u \leftarrow NULL$, $\mathcal{O}_s \leftarrow NULL$;
3: **for** $r = 1 \rightarrow \mathcal{N}$ **do**
4: **if** $\exists n, \Omega_1(n)$ **then**
5: $\mathcal{O}_u \leftarrow \langle D_n, s^* \rangle$, $\mathcal{O}_s \leftarrow \langle D_n, d_n - s^* \rangle$;
6: $selected \leftarrow \bigcup_{i=1}^{n-1} g_i + \mathcal{O}_s$, $\mathcal{G} \leftarrow \mathcal{G} - \bigcup_{i=1}^{n} g_i$;
7: **else if** $\exists m, n, \Omega_2(m, n)$ **then**
8: $\mathcal{O}_u \leftarrow \langle D_q, s^* \rangle$, $\mathcal{O}_s \leftarrow \langle D_q, d_q - s^* \rangle$;
9: $selected \leftarrow \bigcup_{i=1}^{m} g_i + \bigcup_{j=q-n+1}^{q-1} g_j + \mathcal{O}_s$;
10: $\mathcal{G} \leftarrow \mathcal{G} - \bigcup_{i=1}^{m} g_i - \bigcup_{j=q-n+1}^{q} g_j$;
11: **else**
12: $selected \leftarrow \mathcal{G}$, $\mathcal{G} \leftarrow \emptyset$;
13: **end if**
14: insert $\mathcal{O}_u$ into $\mathcal{G}$, $q \leftarrow |\mathcal{G}|$;
15: **end for**

**Condition 2**: $\Omega_2(m, n)$

$$\sum_{i=p}^{p+m-1} d_i + \sum_{j=q-n+1}^{q} d_j - \varpi = s^* \geq 0,$$

$$\sum_{i=p}^{p+m} d_i + \sum_{j=q-n+2}^{q} d_j - \varpi < 0, \text{ and } m + n = \mathcal{M}$$

Based on these conditions, we present the condition-based selection algorithm whose framework is shown in Algorithm 1. We interpret the algorithm and the conditions jointly. We start with the basic condition; Condition 0 indicates that the unselected group sequence has no more than $\mathcal{M}$ items, and the sum of the degrees of these items is no more than $\varpi$. This case only appears at the last round of the selection procedure. Before that, Condition 1 or Condition 2 always holds. Condition 1 implies that the sum of the degrees of the least $\mathcal{M}$ items is greater than $\varpi$. We select the first $n$ items, where $n$ is the smallest number, which ensures that the degree sum of the first $n$ items is greater than $\varpi$. When the first $\mathcal{M}$ items cannot meet Condition 1, we perform a *swap* operation to increase the degree sum until the sum is equal to or greater than $\varpi$. In detail, when performing the *swap* operation, we take the item with the largest degree from the unselected groups to replace the item with the largest degree in the selected groups. For example, in the first round of *swap*, we use $g_q$ to replace $g_{\mathcal{M}}$. We repeat the *swap* operation until the degree sum is equal to or greater than $\varpi$, and Condition 2 holds true at this time. We will prove that before each round of selection, there must exist some condition that is true.

Once the correct condition is known, the selection procedure is determined and the degree sum of the selected groups may be greater than $\varpi$; this does not result in the optimal makespan. To achieve the minimum makespan, we partition the item with the greatest degree into two parts, $\mathcal{O}_u$ and $\mathcal{O}_s$, as shown in Algorithm 1. For Condition 1, the divided item is $g_n$. $\mathcal{O}_u$ and $\mathcal{O}_s$ have the same input data $D_n$. $s^*$ is the amount that needs to be cut. So, the newly created group $\mathcal{O}_u$ is assigned with a degree equals to $s^*$, while the degree of $\mathcal{O}_s$ is $d_n - s^*$. In the final decision, $\mathcal{O}_s$ is selected, and $\mathcal{O}_u$ is not. Consequently, the degree sum of the selected items is exactly equal to $\varpi$. A similar explanation applies to Condition 2 when $g_q$ is cut by $s^*$. Finally, the selected groups are assigned to one server. $\mathcal{O}_u$ is inserted into the unselected group sequence and the procedure repeats until all groups are assigned to a server.

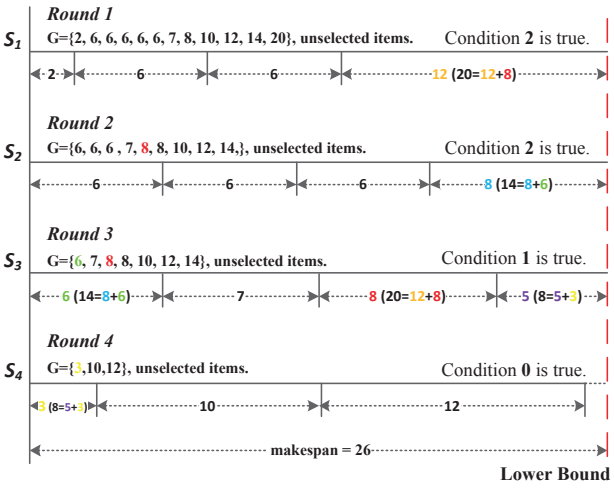



Fig. 1. An example with optimal assignment

The input of the algorithm is the *sorted* group sequence. We will illustrate the selection procedure via an example. Given sorted group sequence $\mathcal{G}$= $\{\langle D_1, 2\rangle$, $\langle D_2, 6\rangle$, $\langle D_3, 6\rangle$, $\langle D_4, 6\rangle$, $\langle D_5, 6\rangle$, $\langle D_6, 6\rangle$, $\langle D_7, 7\rangle$, $\langle D_8, 8\rangle$, $\langle D_9, 10\rangle$, $\langle D_{10}, 12\rangle$, $\langle D_{11}, 14\rangle$, $\langle D_{12}, 20\rangle\}$, and $\mathcal{N} = 4$, we then have $\mathcal{K} = 12$, $\varpi = 26$, and let $\mathcal{M} = 4$. According to our algorithm, we make the following selection:

- **Round 1**. Condition 2 is true; $m = 3$ and $n = 1$. $\langle D_{12}, 20\rangle$ is partitioned into $\langle D_{12}, 12\rangle$ and $\langle D_{12}, 8\rangle$. $\{\langle D_1, 2\rangle, \langle D_2, 6\rangle, \langle D_3, 6\rangle, \langle D_{12}, 12\rangle\}$ is selected.
- **Round 2**. Condition 2 is true; $m = 3$ and $n = 1$. $\langle D_{11}, 14\rangle$ is partitioned into $\langle D_{11}, 8\rangle$ and $\langle D_{11}, 6\rangle$. $\{\langle D_4, 6\rangle, \langle D_5, 6\rangle, \langle D_6, 6\rangle, \langle D_{11}, 8\rangle\}$ is selected.
- **Round 3**. Condition 1 is true; $n = 4$. $\langle D_8, 8\rangle$ is partitioned into $\langle D_8, 5\rangle$ and $\langle D_8, 3\rangle$. $\{\langle D_{11}, 6\rangle, \langle D_7, 7\rangle, \langle D_{12}, 8\rangle, \langle D_8, 5\rangle\}$ is selected.
- **Round 4**. Condition 0 is true. $\{\langle D_8, 3\rangle, \langle D_9, 10\rangle, \langle D_{10}, 12\rangle\}$ is selected.

The assignment is also shown in Fig. 1, where each slot refers to one group (or sub-group). The number under each slot represents the degree of this group. According to the memory constraint, the number of slots should not exceed memory capacity $\mathcal{M} = 4$.

This example motivates us to select a "diversified" group set with both large and small degrees that is more likely to meet both memory and workload constraints. This also shows the tradeoff. In fact, the group degree implies the *hotness* or *popularity* of the data block; we should make a mixed assignment with diversified *popularity*.

According to Algorithm 1, there is at most one *partition* operation in each round. The partition operation will create one replica for some data block. Therefore, the total number of data block replicas may exceed $\mathcal{K}$, but the total number of replicas is at most $\mathcal{K}+\mathcal{N}-1$. The total memory capacity is $\mathcal{M}\cdot\mathcal{N}$. If $\mathcal{K}+\mathcal{N}-1>\mathcal{M}\cdot\mathcal{N}$, the algorithm cannot give the optimal solution. Hence, we adopt $\mathcal{K}+\mathcal{N}-1\leq\mathcal{M}\cdot\mathcal{N}$ as the necessary condition to use Algorithm 1 to achieve the minimum makespan. Because $\mathcal{M}$ is an integer, the condition is equivalent to $\lceil\frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}}\rceil\leq\mathcal{M}<\mathcal{K}$, which refers to the *opt-case*. Then, we are ready to prove the optimality of Algorithm 1 under *opt-case*.

*Theorem 2:* For the *opt-case*, i.e. $\lceil\frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}}\rceil\leq\mathcal{M}<\mathcal{K}$, Algorithm 1 gives the optimal assignment.

*Proof:* It is easy to show that if this theorem is true when $\mathcal{M}=\lceil\frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}}\rceil$, and it also holds when $\mathcal{M}>\lceil\frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}}\rceil$ because the same assignment decision can be adopted. Hence, we only need to prove that Algorithm 1 is optimal when $\mathcal{M}=\lceil\frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}}\rceil$.

First, we assume that the sum of all degrees can be divided into $\mathcal{N}$ parts with exactly equal loads, i.e. $\sum_{i=1}^{\mathcal{K}}d_i=\mathcal{N}\cdot\varpi$. In fact, we can always achieve this by increasing the largest degree: $d_{\mathcal{K}}=d_{\mathcal{K}}+(\mathcal{N}\cdot\varpi-\sum_{i=1}^{\mathcal{K}}d_i)$. Through this transformation, the optimal makespan is still $\varpi$. Thus, this assumption holds.

We then give the proof by contradiction. Without loss of generality, we assume that the $j^{th}$ round selection does not meet any of the three conditions. This means that

$$\sum_{i=q_j-\mathcal{M}+1}^{q_j}d_i<\varpi,\ q_j-\mathcal{M}+1>1$$

where $q_j$ is the index of the last item of the unselected group sequence of the $j^{th}$ round selection. $p_j$ indicates the first item, as mentioned in Algorithm 1. We add a subscript for $p$ and $q$ to distinguish different rounds.

Condition 1 occurs under two cases, $n=\mathcal{M}$ and $n\leq\mathcal{M}-1$; we use $C_1^{\mathcal{M}}$ and $C_1^n$ to represent the two cases respectively. Depending on condition $C_1^n$, we analyze our algorithm under two cases.

1. $C_1^n$ never occurs in the previous $j-1$ rounds. The number of unselected groups can be described as $q_j-p_j+1=\mathcal{K}-\mathcal{M}\cdot(j-1)+(j-1)$.

Because $\mathcal{M}\geq\lceil\frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}}\rceil$, we have

$$\mathcal{M}\cdot\mathcal{N}\geq\mathcal{K}+\mathcal{N}-1\Rightarrow\mathcal{K}\leq\mathcal{M}\cdot\mathcal{N}-\mathcal{N}+1$$

Hence, we have

$$\begin{aligned}q_j-p_j+1&=\mathcal{K}-\mathcal{M}\cdot(j-1)+(j-1)\\&\leq\mathcal{M}\cdot\mathcal{N}-\mathcal{N}+1-\mathcal{M}\cdot(j-1)+(j-1)\\&\leq\mathcal{M}\cdot(\mathcal{N}-j+1)\end{aligned}$$

It is easy to show that the average value of the largest $\mathcal{M}$ degrees is greater than the average value of the unselected group degrees, i.e.,

$$\frac{1}{\mathcal{M}}\cdot\sum_{i=q_j-\mathcal{M}+1}^{q_j}d_i\geq\frac{1}{q_j-p_j+1}\cdot\sum_{i=p_j}^{q_j}d_i$$

According to the above assumption, we have

$$\begin{aligned}\sum_{i=p_j}^{q_j}d_i&\leq\frac{q_j-p_j+1}{\mathcal{M}}\cdot\sum_{i=q_j-\mathcal{M}+1}^{q_j}d_i<\frac{q_j-p_j+1}{\mathcal{M}}\cdot\varpi\\&\leq\frac{q_j-p_j+1}{\mathcal{M}}\cdot\frac{1}{\mathcal{N}-j+1}\cdot\sum_{i=p_j}^{q_j}d_i\leq\sum_{i=p_j}^{q_j}d_i\end{aligned}$$

We then have $\sum_{i=p_j}^{q_j}d_i<\sum_{i=p_j}^{q_j}d_i$, which is a contradiction. This case also implies that one of the three conditions must be true at the first round of selection.

2. $C_1^n$ occurs in the previous $j-1$ rounds. We assume the first appearance of $C_1^n$ is in the $i^{th}(i<j)$ round. So we have

$$\sum_{l=p_i}^{p_i+n-1}d_l\geq\varpi,\ n\leq\mathcal{M}-1$$

We use $s_i^*$ to indicate the degree of the newly created group, as also mentioned in Algorithm 1 ($\mathcal{O}_u$). We use $d_{imax}$ to indicate the maximum degree among the selected groups. When the selection of round $i$ is finished, the degree of the unselected groups is greater than $d_{imax}$ except $s_i^*$, i.e. $\forall a\in[p_{i+1}+1,\ q_{i+1}],\ d_a\geq d_{imax}$, and $d_{p_{i+1}}=s_i^*$.

Then, we prove our conclusion by induction. *For round $r(i<r\leq j)$, we have*

$$\begin{cases}\varphi_r=d_{p_r}+\sum_{i=q_r-\mathcal{M}+2}^{q_r}d_i\geq\varpi\\\forall a\in(p_{r+1},\ q_{r+1}],\ d_a\geq d_{imax}\end{cases}$$

Base Case: $r=i+1$.

$$\begin{aligned}\varphi_r&=d_{p_r}+\sum_{l=q_r-\mathcal{M}+2}^{q_r}d_l=s_i^*+\sum_{l=q_{i+1}-\mathcal{M}+2}^{q_{i+1}}d_l\\&\geq s_i^*+(\mathcal{M}-1)\cdot d_{imax}\geq s_i^*+\sum_{l=p_i}^{p_i+n-1}d_l\geq\varpi\end{aligned}$$

We use $s_{i+1}^*$ to indicate the degree of the newly created group. The degrees of the other unselected groups, which are inherited from the last round (round $i$), are still no less than $d_{imax}$. Thus, regardless of whether $s_{i+1}^*$ is greater than $d_{imax}$ or not, we always have $\forall a\in(p_{i+2},\ q_{i+2}],\ d_a\geq d_{imax}$.

By the induction hypothesis, we let $r=i+k$.

$$\begin{cases}\varphi_r=d_{p_r}+\sum_{i=q_r-\mathcal{M}+2}^{q_r}d_i\geq\varpi\\\forall a\in(p_{i+k+1},\ q_{i+k+1}],\ d_a\geq d_{imax}\end{cases}$$

$r = i + k + 1$.

$$\begin{aligned}\varphi_r &= d_{p_r} + \sum_{l=q_{i+k+1}-\mathcal{M}+2}^{q_{i+k+1}} d_l \\ &\geq d_{p_r} + (\mathcal{M}-1)\cdot d_{imax} \ \ (by\, I.H.) \\ &\geq d_{p_r} + \sum_{l=p_i}^{p_i+n-1} d_l \geq \varpi\end{aligned}$$

In a similar way, whether $s^*_{i+k+1}$ is greater than $d_{imax}$ or not, the degree of the other unselected groups is no smaller than $d_{imax}$. Therefore, we have

$$\forall a \in (p_{i+k+2},\ q_{i+k+2}],\ d_a \geq d_{imax}$$

Based on the above proof, we have

$$\sum_{i=q_j-\mathcal{M}+1}^{q_j} d_i \geq d_{p_j} + \sum_{i=q_j-\mathcal{M}+2}^{q_j} d_i = \varphi_j \geq \varpi$$

which is contradictory to the assumption.

Therefore, we conclude that one of the conditions is true for each round selection, which means the selection procedure is feasible. According to the conditions, we know that the workload on each server is $\varpi$ and we have $opt \geq \varpi$. Hence, Algorithm 1 gives the optimal assignment under *opt-case*. ■

### C. *nph-case when* $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$

In the previous two subsections, we have discussed the *inf-case* and the *opt-case*. In this subsection, we study the *nph-case* when $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$. Note that *nph-case* belongs to *opt-case* when $\lceil \frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}} \rceil = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$. Therefore we focus on the case when $\lceil \frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}} \rceil \neq \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$ or $\lceil \frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}} \rceil = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil - 1$ (so as to avoid trivial cases).

*Theorem 3:* The joint job and data assignment problem under the *nph-case* is NP-hard.

*Proof:* According to Theorem 1, the general joint job and data assignment problem is NP-hard. We have proved that *opt-case* allows for a polynomial-time optimal algorithm and that there is no feasible solution under the *inf-case*. Hence, the joint job and data assignment problem under *nph-case* is NP-hard. ■

We propose a 2-approximation algorithm, as detailed in Algorithm 2, to tackle this problem under the *nph-case*. To assure that all groups can be assigned to the servers, the memory capacity must be large enough to host at least one replica for each data block, which implies that $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$.

In Algorithm 2, the function $assign(g_1, \mathcal{S}_j)$ refers to assigning the first group ($g_1$) of $\mathcal{G}$ to the $j^{th}$ server of the sorted server set $\mathcal{S}$. Function $sorting(\mathcal{S})$ refers to sorting the servers in descending workload order, i.e., the sum of the degrees of the groups assigned to this server.

The condition-based selection algorithm (Algorithm 1) selects a group set for one server at each selection round. Different from Algorithm 1, the approximation algorithm selects one group for each server in one round. There are $\mathcal{M}$ assignment rounds and only one replica for each data block. The assignment procedure can be viewed as assigning the first $\mathcal{N}$ groups of sorted $\mathcal{G}$ to the $\mathcal{N}$ servers, which have been sorted in descending order.

**Algorithm 2** Approximate Algorithm: *app*

**Input:** $\mathcal{G}$: sorted group sequence

```
for i = 1 → M do
    S ← sorting(S);
    for j = 1 → N do
        assign(g_1, S_j), G ← G − g_1;
        if G = ∅ then
            break;
        end if
    end for
end for
```

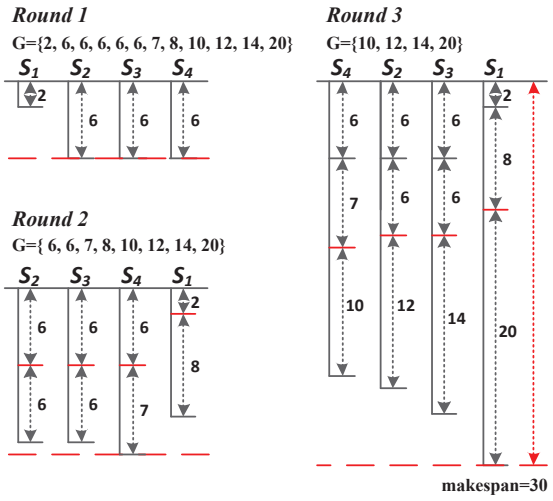



Fig. 2. An example of approximation algorithm

To gain a better understanding of the approximation algorithm, we take the input of Fig. 1 as an example. In this case, $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil = 3$. There are 3 selection rounds, and one item is selected for each server at each round. Fig. 2 shows the selection procedure.

There may still exist some unused memory slots after the assignment; partition operations can be adopted to move some jobs from the server with the highest workload to one that has an available memory slot. We can get a better solution via some partition and movement, but we cannot guarantee an optimal makespan because the number of unused memory slots is strictly less than $\mathcal{N}$. The remainder of this section is devoted to proving that the approximation ratio of Algorithm 2 is 2.

We first introduce a notation $W_u^j$ to indicate the workload of server $S_u$ after $j$ assignment rounds. $W_u^j$ can be defined recursively:

$$W_u^j = W_u^{j-1} + d_u^j$$

where $d_u^j$ indicates the degree of the group assigned to server $S_u$ at the $j^{th}$ assignment round. Let $W_u^0 = 0$. We first give the following lemma.

*Lemma 2:* $\forall u, v, u \neq v$, we have $W_u^{j+1} \geq W_v^j$, where $1 \leq u,\ v \leq \mathcal{N},\ and 1 \leq j < \mathcal{M}$.

*Proof:* First, we give a basic fact: $\forall u, v, d_u^{j+1} \geq d_v^j$. Because the groups are sorted in ascending order and they are assigned from smallest to largest, the degrees of the groups selected in the $(j+1)^{th}$ round must be greater than the items in the $j^{th}$ round.

We prove this lemma through the induction of $j$.

Base Case: $j = 1$. $\forall u, v, u \neq v$, $W_u^2 = d_u^1 + d_u^2, W_v^1 = d_v^1$. Because $d_u^2 \geq d_v^1$, we have $W_u^2 \geq W_v^1$.

By the induction hypothesis, we let $W_u^{k+1} \geq W_v^k$.

$\boldsymbol{j = k+1}$ It is easy to show that $W_u^{k+2} = W_u^{k+1} + d_u^{k+2}$ and $W_v^{k+1} = W_v^k + d_v^{k+1}$. According to the $I.H.$ condition, we have $W_u^{k+1} \geq W_v^k$, and we reach the basic conclusion $d_u^{k+2} \geq d_v^{k+1}$. Then we have $W_u^{k+2} \geq W_v^{k+1}$.

Therefore, $\forall u, v, u \neq v$, we have $W_u^{j+1} \geq W_v^j$. This concludes the proof of this lemma. ∎

*Theorem 4:* For *nph-case*, i.e. $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$, Algorithm 2 achieves an approximation ratio of 2.

*Proof:* From $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$, we have $\mathcal{K} \leq \mathcal{M} \cdot \mathcal{N}$. We assume $\mathcal{K} = \mathcal{M} \cdot \mathcal{N}$. Actually, if $\mathcal{K} < \mathcal{M} \cdot \mathcal{N}$, we can add some groups with degree $d = 0$, until $\mathcal{K} = \mathcal{M} \cdot \mathcal{N}$.

Let $opt$ indicate the minimum makespan. As mentioned above, we have a lower bound of $opt$, as shown in Eq. 1. Makespan should not be less than the degree of any of the groups, so we have $opt \geq d^*$, where $d^*$ refers to the greatest degree of the groups..

Without loss of generality, we assume that server $\mathcal{S}_u$ has the greatest workload $W_u^{\mathcal{M}}$, which also indicates the correspondent makespan. Meanwhile, we assume server $\mathcal{S}_v$ has the least workload $W_v^{\mathcal{M}}$. It is easy to show that

$$W_v^{\mathcal{M}} \leq \left\lceil \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{K}} d_i \right\rceil \leq opt$$

Let $\Delta = W_u^{\mathcal{M}} - W_v^{\mathcal{M}}$. According to Lemma 2, we conclude that $\Delta \leq d_u^{\mathcal{M}}$. (If not ($\Delta > d_u^{\mathcal{M}}$), we have $W_u^{\mathcal{M}-1} = W_u^{\mathcal{M}} - d_u^{\mathcal{M}} > W_u^{\mathcal{M}} - \Delta$. Because $\Delta = W_u^{\mathcal{M}} - W_v^{\mathcal{M}}$, this implies that $W_u^{\mathcal{M}-1} > W_v^{\mathcal{M}}$, which contradicts Lemma 2.) Therefore, $\Delta = W_u^{\mathcal{M}} - W_v^{\mathcal{M}} \leq d_u^{\mathcal{M}} \leq d^* \leq opt$. Then, we conclude that $W_u^{\mathcal{M}} = W_v^{\mathcal{M}} + \Delta \leq 2 \cdot opt$. ∎

In addition to the approximate algorithm (Algorithm 2), we propose another heuristic algorithm to solve the assignment problem when $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$. The framework of the algorithm is given in Algorithm 3. In the algorithm, the function $selectServer(\mathcal{S})$ returns the server with minimal workload at the current time. The available memory slots of the returned server should be no less than 1, and its value should be cut by 1 before returning. The variable $j$ is an index to indicate which item should be assigned.

This algorithm is similar to the approximation algorithm for the minimal makespan scheduling problem, but it includes a memory capacity constraint. We believe that the theoretical performance of the heuristic algorithm $hec$ is similar to the approximate algorithm $app$, but $hec$ shows more potential. We will present this in the simulation results.

**Algorithm 3** Heuristic Algorithm: $heu$

**Input:** $\mathcal{G}$: sorted group sequence

```
j ← |G|;
for i = 1 → |G| do
    S* ← selectServer(S);
    assign(g_j, S*);
    j ← j − 1;
end for
```

## IV. Performance Evaluation

In this section, we focus on evaluating the Approximation Algorithm $app$ and Heuristic Algorithm $heu$ with respect to various parameters. The algorithm $csa$ (Algorithm 1) will not be evaluated since we have already prove that it provides the optimal solution for the problem.

### A. Simulation Settings

To evaluate our approximation algorithm and heuristic algorithm, we let $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$, which means that $\mathcal{M}$ is determined by the input parameters, jobs, and data center settings.

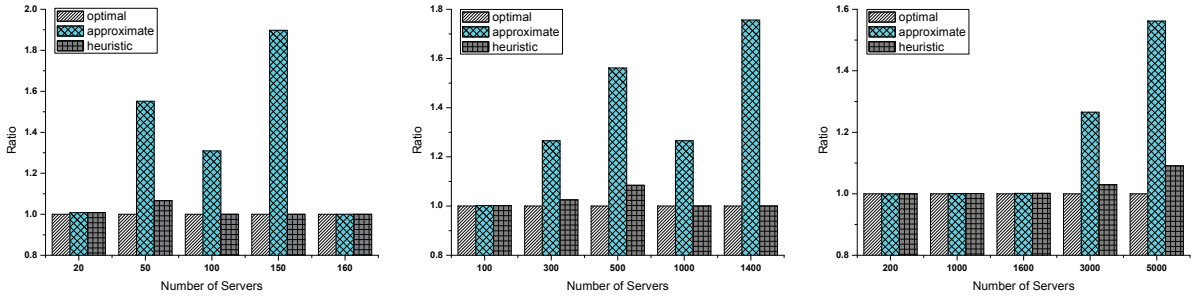
It is hard to achieve the optimal solution when $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$. It is impossible to traverse all the possible assignment cases. However, we can get the lower bound of the minimal makespan, represented by $opt$. There are three factors that indicate the lower bound: (1) $\varpi$, as shown in Eq. 1; (2) $d^*$, the greatest degree of the groups; and (3) $sum(\mathcal{M})$, the degree sum of the first $\mathcal{M}$ items of the sorted group sequence. Because $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$, we have $(\mathcal{M}-1) \cdot \mathcal{N} < \mathcal{K}$, which means that there is at least one server that hosts $\mathcal{M}$ items. Therefore, the optimal makespan should be no less than $sum(\mathcal{M})$. Therefore, we let $opt = \max\{\varpi, d^*, sum(\mathcal{M})\}$.

In the typical Hadoop file system, each data block size is 64MB [15]. We adopt the same setting in our simulation. For the shared common data set, we take into account 3 sizes: 10 GB, 100GB, and 1TB. Therefore, the value of $\mathcal{K}$ equals 160, 1600, and 16000 respectively. The degree of each data block is a random number belonging to the interval (0, 2000), and the expected value is 1,000. Meanwhile, we consider various values for $\mathcal{N}$.

### B. Simulation Results

We conduct extensive simulations based on the above settings. The simulation results are shown in Fig. 3. For each value of $\mathcal{K}$, which implies the size of a common data set, there is one sub-figure to exhibit the corresponding results. In each sub-figure, the x-coordinate represents the number of servers $\mathcal{N}$. To compare the performance clearly, we transform the makespan to ratio, i.e. the ratio of makespan determined by the algorithms to the value of $opt$.

According to the above analysis, the value of $opt$ is determined by $\varpi$ and $d^*$. When $\mathcal{M}$ is very large, there are lots of items on each server and no item, not even $d^*$, has a big impact on the makespan. The approximation algorithm and heuristic algorithm are more likely to reach the optimal solution. That is why the results are nearly the same in Fig. 3(c) when




(a) $\mathcal{K} = 160$ (b) $\mathcal{K} = 1600$ (c) $\mathcal{K} = 16000$

Fig. 3. Evaluation results

$\mathcal{N} < 3,000$ and this observation also explains some of the cases in Figs. 3(a) and 3(b). As a result, we adopt different $\mathcal{N}$ for each $\mathcal{K}$ in the experiments.

When $\mathcal{N} \geq \mathcal{K}$, there should be at most one item on each server to achieve the optimal solution. This explains the result when $\mathcal{N} = 160$ in Fig. 3(a). However, when the value of $\mathcal{M}$ is not too great, $d^*$ has a remarkable impact on the makespa, as shown by the other cases like $\mathcal{N} = 150$ in Fig. 3(a), $\mathcal{N} = 1,400$ in Fig. 3(b), and $\mathcal{N} = 5,000$ in Fig. 3(c). The result varies for different cases, but there is a solid upper bound for the results, i.e. $ratio < 2$, as declared in Theorem 4.

## V. Related Work

Job scheduling is one of the most hot topics in cloud data center, the works are conducted from all kinds of aspects, like energy [16], resource sharing [17], data locality [6], etc. With the development of data-intensive applications, data placement becomes more and more important [3] [4]. This motivates us focus on the joint job and data assignment problem. Makespan optimization is one of the most important issues for both job scheduling and data placement problem [5], and many researchers have pointed that preserving data locality is an efficient method to reduce makespan [7].

The authors in [7] addressed the conflict between data locality and fairness, and proposed a simple *delay scheduling* algorithm to make a tradeoff between locality and fairness. In [3], the authors focus on multiple data items for one traction, and proposed an associated data placement scheme. The capacity is taken into account in [4] for heterogeneous clusters to achieve data placement.

## VI. Conclusion

In this paper, we investigate the joint job and data assignment problem for data centers with the new memory system. We classify the problem into three cases and propose an optimal algorithm when $\mathcal{M} \geq \lceil \frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}} \rceil$. When $\mathcal{M} = \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil$, we present a 2-approximation algorithm and a heuristic algorithm. The experimental results show the performance of our algorithms. Because $\lceil \frac{\mathcal{K}+\mathcal{N}-1}{\mathcal{N}} \rceil - \lceil \frac{\mathcal{K}}{\mathcal{N}} \rceil \leq 1$, the optimal algorithm works most of the time.

## Acknowledgement

This work is supported in part by the National High Technology Research and Development Program of China under Grant No. 2015AA015303; National Natural Science Foundation of China under Grant No. 61472181, 61373015 and 61402225; Jiangsu Natural Science Foundation under Grant No. BK20160813, BK20151392 and BK20140832; Project Funded by China Postdoctoral Science Foundation; and NSF grants CNS 1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, and CNS 1439672.

## References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *USENIX OSDI*, 2004.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *EuroSys*, 2007, pp. 59–72.

[3] B. Yu and J. Pan, "Location-aware associated data placement for geo-distributed data-intensive applications," in *IEEE INFOCOM*, 2015.

[4] B. Wang, J. Jiang, and G. Yang, "Actcap: Accelerating mapreduce on heterogeneous clusters with capability-aware data placement," in *IEEE INFOCOM*, 2015.

[5] Y. Zhu, Y. Jiang, W. Wu, L. Ding, A. Teredesai, D. Li, and W. Lee, "Minimizing makespan and total completion time in mapreduce-like systems," in *IEEE INFOCOM*, 2014.

[6] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in *IEEE INFOCOM*, 2013.

[7] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys*, 2010.

[8] F. Ahmad, S. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Shuffle-watcher: Shuffle-aware scheduling in multi-tenant mapreduce clusters," in *USENIX ATC*, 2014.

[9] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *ACM SIGCOMM*, 2015.

[10] T. Hirofuchi and R. Takano, "RAMinate: Hypervisor-based virtualization for hybrid main memory systems," in *ACM SoCC*, 2016.

[11] L. Wang, C. Yang, and J. Wen., "Physical principles and current status of emerging non-volatile solid state memories," *Electronic Materials Letters*, vol. 11, no. 4, pp. 505–543, 2015.

[12] S. Robbins, "RAM is the new disk," InfoQ, June 2008.

[13] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing cluster," in *ACM Symposium on Operating Systems Principles (SIGOPS)*, 2009.

[14] A. Verma, L. Cherkasova, and R. H. Campbell, "Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance," in *IEEE MASCOTS*, 2012.

[15] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *IEEE MSST*, 2010.

[16] J. Fu, J. Guo, E. W. Wong, and M. Zukerman, "Energy-efficient heuristics for job assignment in processor-sharing server farms," in *IEEE INFOCOM*, 2015.

[17] J. Tan, X. Meng, and L. Zhang, "Coupling task progress for mapreduce resource-aware scheduling," in *IEEE INFOCOM*, 2013.