HMHS: Hybrid Multistage Heuristic Scheduling Algorithm for Heterogeneous MapReduce System

Heng Chen, Yao Shen, Quan Chen, and Minyi Guo

Department of Computer Science and Engineering,
Shanghai Jiao Tong University Shanghai, China
{chenheng417,quanchen1986}@gmail.com, {yshen,guo-my}@cs.sjtu.edu.cn

Abstract. The scale of data in a MapReduce system is increasing quickly. Thus how to efficiently schedule a set of production jobs has become increasingly important. For a given set of jobs, a well-designed scheduling algorithm can significantly reduce makespan and increase the utilization of clusters. However, there exists very few studies that aim to construct a scheduler that minimizes the makespan of batch jobs in a heterogeneous environment. This paper proposes a heuristic scheduling algorithm called Hybrid Multistage Heuristic Scheduling (HMHS), which tries to solve the scheduling problem by breaking down it into two-subproblems: sequencing and dispatching. For sequencing, we develop a heuristic based on *Pri*(the modified Johnson's algorithm). For dispatching, we offer two heuristics *Min-Min* and *Dynamic-Min-Min*. Our simulation results on two kinds of workloads demonstrate that every heuristic employed in HMHS contributes to reducing the makespan. As a whole, HMHS improves the performance ranging from 51% to 77% compared to FIFO.

Keywords: task scheduling, MapReduce, makespan, heterogeneous system, heuristic algorithm.

1 Introduction

Large scale of data has been generated daily. To handle such huge amount of data quickly, large companies (e.g. Google) group large number of commodity computers together to construct a distributed cloud system for data processing. Parallel programming model MapReduce which is popularized by Google [5] is widely used in these systems for handling data.

Empirically, a typical MapReduce system is usually used for running thousands of jobs periodically for data processing(e.g. 10,000 jobs are processed daily by Facebook's data center [15]). Obviously, for a given set of independent MapReduce jobs in a heterogeneous environment, the less time the clusters cost to execute these jobs, the earlier the resources of clusters can be released. However, the processing time is affected by several factors, such as the precedence constraints between map and reduce tasks. Therefore, a study on how to schedule a given

J. Kołodziej et al. (Eds.): ICA3PP 2013, Part I, LNCS 8285, pp. 196–205, 2013.

set of independent MapReduce jobs in a heterogeneous environment to minimize the makespan is of great value.

Many researchers have proposed delicate job scheduling algorithms to improve the system performance of MapReduce. Chang et al. [2] abstracted the scheduling problem as a novel optimization problem. They focused on constructing an optimal scheduling algorithm that minimizes the weighted sum of the job completion times. However, they ignored the precedence relationships between map and reduce tasks. Verma et al. [17] proposed a heuristic algorithm to organize the order in which jobs are executed to minimize the completion time of a given set of MapReduce jobs in a homogeneous environment. Since heterogeneous environment will greatly affect the performance of the scheduling algorithm, a algorithm works well in a homogeneous environment may have poor performance in a heterogeneous environment.

In general, none of existing works design a scheduling algorithm to minimize the makespan of a given set of independent MapReduce jobs in a heterogeneous environment. To address this problem, we propose a Hybrid Multistage Heuristic Scheduling (HMHS) algorithm, which tries to solve the scheduling problem by dividing it into two sub-problems: sequencing and dispatching. For sequencing, we consider the precedence constraints of map and reduce, and then design a *Pri* based heuristic to get the order of jobs. (Here, *Pri* stands for the priority of a MapReduce job which is defined in section 3.1.) Meanwhile, for dispatching, we offer two heuristics: *Min-Min* and *Dynamic-Min-Min* to balance the load of machines in a heterogeneous environment. We compare performance benefits of HMHS with three scheduling strategies via simulation. The results demonstrate HMHS outperforms FIFO by reducing up to 51%-77% makespan. We also study how system heterogeneity will affect the performance of HMHS.

2 Problem Description

In this paper, we study how to schedule a set of independent MapReduce jobs in a heterogeneous system to minimize the makespan. A real MapReduce system is usually complex and affected by many factors. In this paper, we make several assumptions to simplify the scheduling problem. We discuss some of assumptions here. (1)We assume that all map (reduce) tasks of a given job are uniform. Thus, the processing times of these map (reduce) tasks are same. Meanwhile, we assume the processing times of tasks of a given job is known. This is not available in a real system at present, but some researchers [16] try to approximately estimate the processing time based on historical logs and job profiles. (2)We assume that one map (reduce) machine contains one map (reduce) slot. In real MapReduce system (e.g. Hadoop), each machine will contain a specified number of map slots and reduce slots, and each map (reduce) slot can be used to execute one map (reduce) task. Tasks executed at the same machine will preempt resources and affect the processing time of each other. As mentioned in 1, the processing time is supposed to be known and from statistical results. Thus, the assumption that each machine owns one slot will not have a significant impact on our scheduling

problem. (3) We ignore the shuffle phase between map and reduce, and then assume that, for a given job, reduce tasks can only be launched when all map tasks have been finished. This assumption is widely used in literatures [2],[11],[17] to simplify the scheduling problem.

According to above assumptions and the setting of real MapReduce systems, the scheduling problem considered in this paper will satisfy the following conditions: Precedence relationships exist between map and reduce stage; Each job contains a specified number of map and reduce tasks; The processing times of map tasks of a given job are same, as well as the processing times of reduce tasks; Multiple map machines and reduce machines exist; All jobs arrive at zero; Task processing time is deterministic and given in advance; Each machine can process only one task at a time and processing tasks can not be interrupted; All MapReduce jobs are independent.

2.1 Definitions

To describe the problem more clearly, we give the following definitions, which are used throughout the paper.
N: Number of MapReduce jobs.
M^m (M^r): Number of alternative machines at map(reduce) stage.
T_i^m (T_i^r): Number of map(reduce) tasks of ith job.
P_i^m (P_i^r): Normal task processing time of ith job at map(reduce) stage.
V_{ij}^m (V_{ij}^r): Speed factor of any map(reduce) task of ith job on jth machine.
$\overline{P}_i^m$ $(\overline{P}_i^r)$: Average total processing time of map(reduce) tasks of ith job.

$$\overline{P}_i^m = P_i^m * \left(\sum_{j=1}^{M^m} V_{ij}^m \right) / M^m * T_i^m \ , \ \overline{P}_i^r = P_i^r * \left(\sum_{j=1}^{M^r} V_{ij}^r \right) / M^r * T_i^r \ . \quad (1)$$

FM_{ij}: Finish time of jth map task of ith job.
A_i: Arriving time of ith job's reduce tasks.

$$A_i = \max_{1 \le j \le T_i^m} (FM_{ij}) \ . \quad (2)$$

FR_{ij}: Finish time of jth reduce task of ith job.
C : The completion time of all jobs, which can also be called makespan.

$$C = \max_{1 \le i \le N} \max_{1 \le j \le T_i^r} (FR_{ij}) \ . \quad (3)$$

2.2 Hardness of Our Scheduling Problem

Two-stage flexible flow shop scheduling problem (2-FFS) is similar to our scheduling problem. The main difference is that each job in 2-FFS only contains one task at each stage, while each job considered in our work contains multiple tasks. Gupta [7] proved that the 2-FFS is NP-complete even if the number of machines at one of the two stages is one. Obviously, the scheduling problem in our paper is also NP-hard according to [7].

3 Hybrid Multistage Heuristic Scheduling Algorithm

As discussed above (section 2.2), the scheduling problem of the MapReduce system is NP-hard. Thus, we intend to simplify the problem. It is common to break down the scheduling problem into smaller pieces. This enlightens us to divide the problem into two sub-problems: (a) sequencing the tasks allocated to each machine; (b) dispatching tasks of jobs to heterogeneous machines at map and reduce stages.

We show the details of sequencing and dispatching problems in section 3.1 and section 3.2, respectively.

3.1 Sequencing

During map and reduce stage, each machine will be allocated multiple tasks. Precedence relationships between map and reduce tasks may block the jobs' execution. A well-designed sequencing algorithm can help to organize map tasks to decrease waiting time of reduce tasks.

Johnson [9] proposed an classical optimal algorithm for the two flow shop problem (only one machine is available at each stage and each job contains only one task at each stage), which is similar to our scheduling problem. In Johnson's algorithm, each job contains three attributes M_i, R_i and V_i. M_i stands for the task processing time at map stage. R_i stands for the task processing time at reduce stage. $V_i = \min(M_i, R_i)$. For a job List L with N jobs, the steps of Johnson's algorithm are:

Step 1. Define two output lists L1= {}, L2 = {}.
Step 2. Order all jobs in List L by V_i in nondecreasing order.
Step 3. Process the ordered list from the beginning. For each job, if $M_i \leq R_i$, place it at end of L1; otherwise, place it at the beginning of L2.
Step 4. Add list L2 to the end of list L1. L1 is the ordered list.

We approximately evaluate the sum of processing times of map (reduce) tasks of a given job, indicated by $\overline{P}_i^m$ ($\overline{P}_i^r$), by combining the average processing time of a map (reduce) task and the number of map (reduce) tasks. Hence, we can replace the task processing times M_i and R_i in Johnson's algorithm by $\overline{P}_i^m$ and $\overline{P}_i^r$, respectively. This enables us to apply Johnson's algorithm to sequence jobs.

In order to describe the sequencing algorithm, we define the priority *Pri* which takes advantage of Johnson's algorithm to indicate the processing order of a job. The smaller *Pri* a job has, the earlier it will be executed during map stage. The expression is modified from Gupta [7]. For each job i,

$$Pri^i = Sgn\left(\overline{P}_i^m - \overline{P}_i^r\right) / min\left(\overline{P}_i^m, \overline{P}_i^r\right)$$
$$where \quad Sgn = \begin{cases} 1, & \text{if } \overline{P}_i^m > \overline{P}_i^r \\ -1, & \text{otherwise.} \end{cases} \tag{4}$$

We can easily obtain the ordered list of jobs by sorting the *Pri* in nondecreasing order.

3.2 Dispatching

The basic intention of dispatching tasks is to balance the work load of all machines. Since the dispatching exists at both map and reduce stage, we discuss it in two cases:Assigning map tasks to map machines and assigning reduce tasks to reduce machines.

Algorithm 1. Dynamic-Min-Min heuristic

The set of N MapReduce jobs, U; Selected jobs to be dispatched, Jw; Available time of the earliest free machine, EAT; The execution time of reduce tasks of ith job on machine m_j, E_{ij}; The minimum completion time of reduce task of ith job if it is mapped to m_j, C_{ij}; The time that ith reduce machine finishes all tasks assigned to it, Ava_i;

1: **while** $U \neq \varnothing$ or $Jw \neq \varnothing$ **do**
2: $EAT \longleftarrow min(Ava_i)$;
3: **for** each $j_i \in U$ **do**
4: **if** $A_i \leq EAT$ **then**
5: add j_i to Jw, remove j_i from U;
6: **end if**
7: **end for**
8: **if** $Jw == \varnothing$ **then**
9: add the job with minimum A_i to Jw, remove it from U;
10: **end if**
11: **for** each $j_i \in Jw$ **do**
12: **for** each machine m_j **do**
13: $C_{ij} \longleftarrow E_{ij} + max(Ava_j, A_i)$;
14: **end for**
15: **end for**
16: $C_{pq} \longleftarrow min(C_{ij})$;
17: assign one reduce task of j_p to m_q;
18: $Ava_q \longleftarrow C_{pq}$;
19: **if** all reduce tasks of j_p are assigned **then**
20: remove j_p from Jw;
21: **end if**
22: **end while**

Dispatching Heuristic of Map Stage. During map stage, we try to select a heuristic to make the whole map stage finished as soon as possible. We employ Min-Min as the dispatching rule, which outperforms most heuristics on dispatching a set of independent tasks onto heterogeneous systems [1]. The key steps of Min-Min are: In each cycle, for each unassigned map task, Min-Min calculates its minimum completion time by comparing the completion times of it on different map machines. Then, the map task with the minimum completion time is assigned to corresponding map machine. Therefore, the possibility of the tasks assigned to their best matched machines is relatively high. The more tasks that are assigned to their best matched machines, the smaller makespan our scheduler can obtain.

Dispatching Heuristic of Reduce Stage. As mentioned above, our algorithm is designed to obtain an optimized batch scheduling policy for a large number of jobs. Multiple jobs with similar $\overline{P}_i^m$ will be launched at approximately the same time according to our sequencing algorithm. Thus, reduce tasks of these jobs will have similar arriving time. Under such assumption, we propose a new heuristic Dynamic-Min-Min, which takes advantage of the idea of Min-Min to dispatch arrived reduce tasks to balance the work load of reduce machines. Dynamic-Min-Min works as follows: At each round of task assignment, it firstly updates the job set Jw which contains all jobs to be dispatched, and then dispatchs a reduce task. Algorithm 1 shows the pseudocode of Dynamic-Min-Min. (Some definitions are explained in section 2.1.)

3.3 Hybrid Multistage Heuristic Scheduling Algorithm

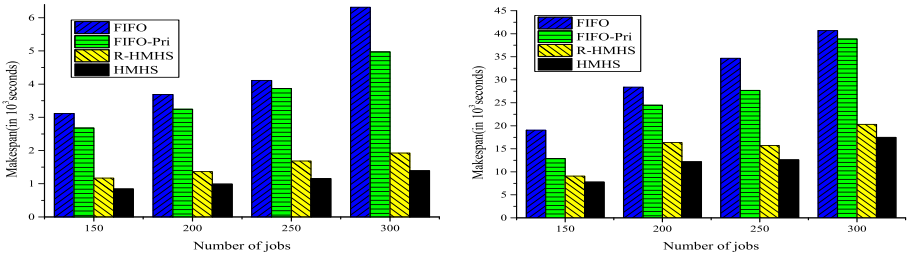
By combining the solutions for sequencing and dispatching together, we can draw the outline of Hybrid Multistage Heuristic Scheduling Algorithm.
1) Dispatch map tasks of all jobs into map machines by heuristic Min-Min.
2) Define T_i as the set of tasks assigned to map machine i. For each T_i, sequence tasks by Pri in nondecreasing order.
3) Dispatch reduce tasks of all jobs into reduce machines by heuristic Dynamic-Min-Min.

4 Evaluation

In this section, we evaluate the benefits of our algorithm via simulations. We compare HMHS with the following three scheduling strategies.
Default FIFO Scheduler: FIFO is the default scheduler used by Hadoop (a widely used MapReduce system). Once a job arrives, FIFO scheduler partitions it into individual tasks and then assigns tasks to free machines.
FIFO-Pri Scheduler: To investigate the effect of our Min-Min and Dynamic-Min-Min dispatching strategies, we combine FIFO Scheduler and modified Johnson's algorithm together. The FIFO-Pri sorts all jobs by priority Pri first, and then uses FIFO Scheduler to assign tasks to free machines in order.
Reverse-Hybrid Multistage Heuristic Scheduler (R-HMHS): R-HMHS is designed to analyze how deeply the priority Pri affects HMHS. The R-HMHS reverses the sequencing result of HMHS. In other words, tasks are sorted by Pri in descending order.

4.1 Simulation Setup

Since building a large distributed system with thousands of machines is beyond the scope of our ability, to evaluate the performance of HMHS, we design a simulator and generate some synthetic workloads according to statistical results in [10],[19].



(a) Single workload (b) Hybrid workload

Fig. 1. Effect of heuristics

In our simulations, a single job is constructed from two aspects: task duration and job size. We generate two types of workloads.(1) Single workload: for each job, P_i^m and P_i^r are drawn from uniform distributions U[5, 45] and U[15, 135] respectively. T_i^m and T_i^r are drawn from U[1, 300] and U[1, 40]. (2) Hybrid workload: in real industry system, a small number of large and long jobs exist [13]. Thus, we generate the hybrid workload in the following way. Normal jobs(80%) are constructed in the same way of jobs of single workload. For long(15%) jobs, P_i^m and P_i^r are drawn from U[100, 2000] and U[300, 6000]. T_i^m and T_i^r are drawn from U[1, 300] and U[1, 40]. For large(5%) jobs, P_i^m and P_i^r are drawn from U[5, 45] and U[15, 135]. T_i^m and T_i^r are drawn from U[2000, 5000] and U[100, 400].

Besides the parameters used to represent workload, the speed factors(V_{ij}^m and V_{ij}^r) are generated to indicate the heterogeneous system. V_{ij}^m and V_{ij}^r are drawn from uniform distribution U[0.1, 1.0].

4.2 Simulation Results

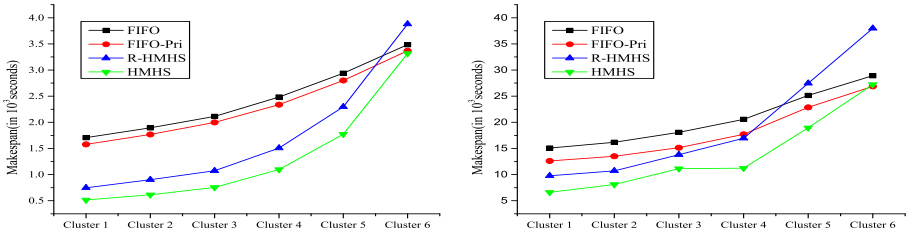
A. Comparison with Other Heuristics. In our first simulation, to evaluate the benefits of our algorithm, we compare the makespan of our scheduler with FIFO, FIFO-Pri and R-HMHS. We generate multiply workloads with different job sizes under Single and Hybrid distributions respectively. We also create a heterogeneous system of 100 map and 100 reduce machines. From Fig. 1, we can see that HMHS works the best among all other strategies. Compared to FIFO, HMHS decreases up to 77% of makespan. We also observe that there is a slightly upward trend of makespan improvements with the increase of number of jobs.

Effect of dispatching heuristics: To investigate the effect of dispatching heuristics Min-Min and Dynamic-Min-Min, we analyze the results of FIFO-Pri and HMHS. By observing the result of FIFO-Pri in Fig. 1, we can find that HMHS decreases up to 72% makespan of FIFO-Pri for single workload and 55% for hybrid workload. This results clearly illustrate that our dispatching heuristics achieve significant makespan improvements.

Effect of sequencing heuristic: As mentioned above, R-HMHS is designed to test the effect of our sequencing algorithm. Fig. 1 illustrates that, compared to R-HMHS, HMHS exhibits 10%-30% makespan improvements. We can see that

Table 1. Configuration of heterogeneous systems

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
slow slot(0.9-1)	0%	20%	40%	60%	80%	100%
random slot(0.1-1)	100%	80%	60%	40%	20%	0%



(a) Single workload (b) Hybrid workload

Fig. 2. Impact of heterogeneity

there is no strong association between the effect of our sequencing algorithm and the job sizes. From the view point of workload sets, single workload gets about 7% more performance benefits than hybrid workload.

The above results fully demonstrate that each of our heuristics is effective in reducing the makespan of a given set of independent MapReduce jobs in a heterogeneous environment.

B. Impact of Heterogeneity

In practice, a cloud system is usually combined by heterogenous machines. By changing the percentage of slow machines in the system, we create six kinds of heterogeneous systems as shown in Table 1. Meanwhile, We suppose slow machines' speed factors are drawn from U[0.9, 1.0] and the rest machines' speed factors are drawn from U[0.1, 1.0] randomly. We generate two kinds of workloads with 100 jobs as the test workloads. Each heterogeneous system contains 100 map machines and 100 reduce machines. Fig. 2 shows that the performance of HMHS is close to FIFO when the system is dominated by slow machines. One major conclusion from Fig. 2 is that the less slow machines the cluster has, the more improvements our algorithm gains.

5 Related Work

5.1 Foundational Work on Job Scheduling Problem

Job Scheduling is not a new problem. Indeed, a lot of foundational works exist in the literature [12]. However, the problem discussed in this paper can not be corresponded to any classical problem. To the best of our knowledge, the classical two-stage flexible flow shop (2-FFS) scheduling problem has the closest model to ours.

The 2-FFS has been studied extensively [12], [14]. Gupta [7] proved the 2-FFS with parallel processors to minimize makespan is NP-Complete and developed an

efficient heuristic algorithm for constructing an approximate solution. Haouari et al. [8] studied the 2-FFS with identical parallel machines at each stage. A tabu search heuristic and a simulated annealing algorithm are presented in their work.

5.2 Scheduling on MapReduce

Job scheduling in MapReduce environment is a new problem. With the rapid development of cloud computing, it has received much attention.

Many efforts(such as Fair Scheduler [18], Delay Scheduling [19], SAMR Scheduler [4] etc.) try to improve the FIFO scheduler, which is the origin strategy of Hadoop. While other works [2] [3] [6] [11] try to formalize the MapReduce scheduling problem and offer an offline scheduling algorithm for a given set of MapReduce jobs.

Chang et al. [2] gave an LP based lower bound of the MapReduce scheduling problem. Meanwhile, they designed a 3-approximation algorithm to minimize the sum of job completion times for offline case. However, they ignored the precedence relationships between map and reduce tasks. To improve Chang's work, Chen et al. [3] not only considered the precedence constraints, but also added the shuffle phase of MapReduce into their model. Similarly, they provided LP based lower bound and constant factor approximation algorithms to minimize the sum of job completion times, which is different from our goal.

The closest work to ours is by Verma et al. [17]. They offered an abstraction of the scheduling problem which is similar to ours in homogeneous system and aimed to minimize the completion time of a set of MapReduce jobs. They designed a heuristic, which extends the classical Johnson's algorithm [9]. In our work, we consider the scheduling problem in a heterogeneous environment. This encourages us to explore new algorithms to minimize the makespan.

6 Conclusion and Future Work

In this paper, we propose a novel algorithm, Hybrid Multistage Heuristic Scheduling(HMHS), which aims at minimizing the makespan of a given set of independent MapReduce jobs in a heterogeneous system. The simulation results show that the heuristics used in our algorithm exhibit significant makespan improvements. In the future, we plan to examine the performance of our algorithm by running the experiment in a real MapReduce cluster with larger input data. On the other hand, we plan to compare our algorithm with some advanced Hadoop schedulers, such as Fair Scheduler [18] and Delay Scheduling [19].

Acknowledgments. This work is supported by the 863 Program of China (No. 2011AA01A202), the Doctoral Fund of Ministry of Education of China (No. 20100073120022), Natural Science Foundation of China (No.61202025), Shanghai Excellent Academic Leaders Plan(No. 11XD1402900), the Program for Changjiang Scholars and Innovative Research Team in University of China (IRT1158, PCSIRT). Yao Shen is the corresponding author.

References

1. Braun, T.D., Siegel, H.J., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing 61(6), 810–837 (2001)
2. Chang, H., Kodialam, M., Kompella, R.R., Lakshman, T., Lee, M., Mukherjee, S.: Scheduling in mapreduce-like systems for fast completion time. In: 2011 Proceedings IEEE INFOCOM. IEEE (2011)
3. Chen, F., Kodialam, M., Lakshman, T.: Joint scheduling of processing and shuffle phases in mapreduce systems. In: 2012 Proceedings IEEE INFOCOM. IEEE (2012)
4. Chen, Q., Zhang, D., Guo, M., Deng, Q., Guo, S.: Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In: 2010 IEEE 10th CIT. IEEE (2010)
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
6. Fischer, M.J., Su, X., Yin, Y.: Assigning tasks for efficiency in hadoop. In: Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures. ACM (2010)
7. Gupta, J.N.: Two-stage, hybrid flowshop scheduling problem. Journal of the Operational Research Society, 359–364 (1988)
8. Haouari, M., M'Hallah, R.: Heuristic algorithms for the two-stage hybrid flowshop problem. Operations Research Letters 21(1), 43–53 (1997)
9. Johnson, S.M.: Optimal two-and three-stage production schedules with setup times included. Naval research logistics quarterly 1(1), 61–68 (1954)
10. Kavulya, S., Tan, J., Gandhi, R., Narasimhan, P.: An analysis of traces from a production mapreduce cluster. In: 2010 10th CCGrid. IEEE (2010)
11. Moseley, B., Dasgupta, A., Kumar, R., Sarlós, T.: On scheduling in map-reduce and flow-shops. In: Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures. ACM (2011)
12. Pinedo, M.: Scheduling: theory, algorithms, and systems. Springer (2012)
13. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing. ACM (2012)
14. Ruiz, R., Vázquez-Rodríguez, J.A.: The hybrid flow shop scheduling problem. European Journal of Operational Research 205(1), 1–18 (2010)
15. Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., Murthy, R., Liu, H.: Data warehousing and analytics infrastructure at facebook. In: The 2010 SIGMOD. ACM (2010)
16. Verma, A., Cherkasova, L., Campbell, R.H.: Aria: automatic resource inference and allocation for mapreduce environments. In: Proceedings of the 8th ACM International Conference on Autonomic Computing. ACM (2011)
17. Verma, A., Cherkasova, L., Campbell, R.H.: Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance. In: 2012 20th MASCOTS. IEEE (2012)
18. Zaharia, M., Borthakur, D., Sarma, J.S., Elmeleegy, K., Shenker, S., Stoica, I.: Job scheduling for multi-user mapreduce clusters. EECS Department, University of California, Berkeley, Tech. Rep. USB/EECS-2009-55 (2009)
19. Zaharia, M., Borthakur, D.: et al.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European Conference on Computer Systems. ACM (2010)