Hummer: Mitigating Stragglers with Partial Clones

Jia LI, Changjian WANG, Dongsheng LI
National Laboratory for Parallel and Distributed Processing, School of Computer Science
National University of Defense Technology, Changsha, China
{josephlijia@163.com, lds1201@163.com, c_j_wang@yeah.net}

***Abstract*—Stragglers can delay jobs and decrease cluster efficiency seriously. Many efforts have been devoted to the solution, such as Blacklist [8], speculative execution [1, 6], Dolly [8]. Blacklist causes a waste of resources; speculative execution is too late to detect stragglers; Dolly launches copies for each task and consumes too many resources. In this paper, we propose a new approach for straggler mitigation in MapReduce, named Hummer, which launches copies only for high-risk tasks. To support Hummer, a risk model for MapReduce jobs is proposed and an algorithm is designed to compute the number of task clones. Comprehensive experiments have been performed and results show that Hummer can decrease the job delaying risk with fewer resources consumption. For small jobs, Hummer also improves job average duration by 48% and 10% compared to LATE and Dolly.**

Keywords-MapReduce, straggler mitigation, task clones

I. INTRODUCTION

Straggler is an important problem in MapReduce since it can decrease MapReduce performance obviously. Lots of research efforts [1, 2, 5, 8, 9, 10, 25, 26] have been devoted to the solution of stragglers in MapReduce and some important approaches have been proposed, such as Blacklist [6], speculative execution [1, 6], Dolly [8], GRASS [9]. In Blacklist, node delaying is regarded as a main reason of stragglers. Failed or slow nodes will be put into a blacklist and no new tasks will be assigned to them. Speculative execution detects straggler tasks. For straggler task, speculative execution launches multiple copies and selects the earliest completed one as result. For small jobs, Dolly clones each task to mitigate stragglers. Each map task in small job has multiple back-ups at the beginning of the job. The earliest completed copy will be adopted.

Though the above-mentioned approaches are effective for the stragglers in MapReduce, there exist some problems:

- The nodes in the blacklist do not always lead to stragglers. Stragglers rarely appear in a node persistently and they are evenly distributed in the cluster [1]. Therefore, it will cause a waste of resources to put some nodes into a blacklist.
- Speculative execution needs gather enough execution information to detect a straggler task. Waiting and then making speculative executions mean that the job will be delayed by stragglers and this approach is just able to alleviate stragglers.
- In Dolly, launching multiple copies for each task is simple and effective, but it needs so many cluster resources. When resources are limited in cluster, Dolly will reduce job throughput of cluster.

In this paper, we propose a new approach for stragglers in MapReduce, named Hummer. In Hummer, the tasks with high delaying risk will be cloned. Under Hummer, the risk of stragglers in a job will decrease obviously when resources are limited. To support Hummer, the risk of a task to be straggler should be able to compute and the number of tasks to clone could be able to know.

The main contributions of this paper are as follows:

- A new approach for stragglers is proposed, named Hummer. It can mitigate the impact of stragglers in a job by cloning only high-risk tasks.
- A risk model of stragglers for MapReduce jobs is established. According to the model, the delaying risk of either a task or a job can be computed.
- An algorithm is designed to compute the number of tasks which need to be cloned in one job.

The rest of this paper is organized as follows. Section II gives an overview of related works. Section III introduces the risk models in Dolly and Hummer. Section IV presents the design of Hummer. Section V shows the experimental results of Hummer. Section VI concludes the paper.

II. RELATED WORKS

Stragglers are slow tasks which complete obviously far behind most of the tasks of the same job [1, 25], which widely exist in large-scale data intensive computing systems [3, 4]. Although the number of stragglers in a job is small [2, 7, 8, 9], they can delay the job completion seriously. Frequently, tasks in one job often need synchronizations in data intensive computing systems [3, 4], the nodes that have finished their tasks must wait for the unfinished tasks, such as stragglers, which will result in a low cluster efficiency [12, 14]. For example, in MapReduce [5, 11, 15], only when all map tasks finish, reduce tasks will begin. If some map tasks in certain nodes become stragglers, other nodes where tasks have completed will have to remain idle until the stragglers also finish [11]. Therefore, it is important to improve cluster efficiency by mitigating stragglers in jobs [6, 10].

There are lots of methods to mitigate stragglers. It is mainly divided into two categories: active method and

passive method. LATE [6] and Mantri [1] are the most popular ones in passive methods (§A, §B), also speculative execution ones. Dolly [8] is an active one (§C).

A. LATE

LATE would detect the straggler task. After finding stragglers, it will launch multiple copies for them, and then dispatch these tasks to the nodes to run. When selecting nodes to execute copies of tasks, select the fastest nodes to run. LATE would set a maximum number of copies to prevent thrashing.

LATE uses progress score to detect stragglers. Progress score is the proportion of processed data accounting for the entire input data. If progress score of one task is up to 2 x slower than the median task in the job, we regard it as straggler. Before launching multiple copies of this task, we examine whether the number of copies is above the *SpeculativeCap*. If not, we clone this task to run.

LATE mitigates stragglers to some extent, but due to the speculation time, it still causes delay of job finishing.

B. Mantri

Mantri is another approach to mitigate straggler. What differs from LATE is that Mantri concerns about the global resources.

Mantri adopts a conscious trade-off resource allocation scheme. For stragglers, Mantri estimates the time required to finish a new copy for this straggler task t_{new} and the remaining time t_{remain} to decide whether launching a new copy or not.

Mantri adopts a trade-off between continuing and launching copy. It uses a continuous observation and avoids redundant copies. Mantri pays more attention to global resources view compared to LATE. But Mantri also needs a certain time to detect stragglers, which also leads to a waste of time and job delaying.

C. Dolly

To avoid wasting time in speculative execution, Dolly [8, 16] adopts an active way to mitigate stragglers.

Dolly eliminates the waiting time in speculative execution. All tasks will be scheduled to clone at the beginning of the job, and then scheduler assigns these tasks and their copies to execute on nodes. For each task, Dolly selects the fastest copy as a result of the task. Then release the resources occupied by other copies for the same task immediately. Dolly mitigates stragglers effectively. For the short, real-time jobs [8, 22], Dolly improves the job average response time by 46% and 34%, compared to LATE and Mantri.

However, the resources for cloning in cluster are limited (typically 5 to 10% of the total cluster resources) [8]. Dolly starts multiple copies for all tasks. This approach occupies too many resources in cluster and few jobs can get resources to run. Studies show that, there are only few stragglers in a job (typically 9% to14% of job' tasks) [2, 7, 8]. Obviously, there is no need to clone all tasks. Thus, Dolly causes a serious waste of clone resources.

III. RISK MODEL

Speculative execution and Dolly both have their shortcomings for mitigating stragglers. So we propose a new approach for straggler mitigation. We start this section by presenting the job delaying risk model of Dolly (§A). However, considering the impact of node difference on job delaying, we propose a trade-off approach: Hummer (§B).

A. Job Delaying Risk Model

Traditional Dolly model considers job delaying is due to the emergence of straggler task. Thereby, traditional Dolly model takes the probability of each task delaying as fixed value p (turn to be 10% to 14%). In Dolly, c represents the number of copies for a task, and n represents the number of tasks in a job. The risk of job delaying is [13, 20, 21]:

$$\Pr ob_{Dolly} = 1 - (1 - p^{c})^{n} \quad (1)$$

Fig.1 shows that how does Dolly eliminate stragglers actively. Task dispatched to the slow node is one of the main reasons for task delaying. Therefore, considering each task have the same delaying risk is not correct. Task delaying risk is always related with the possibility of the node being slow [1].

There exit two main shortages in Dolly model:

- Regard the delaying risk of each task as fixed value;
- Dolly launches clones for all tasks in a job. Since resources are limited, it will lead to a waste of clone resources.

B. Task Delaying Risk Model

For the shortcomings of LATE and Dolly, we propose a trade-off approach: Hummer. Hummer launches clones for part of tasks in one job to mitigate stragglers proactively, just as they are submitted. Hummer picks the earliest clone as the result. There exist differences between nodes in cluster. One task dispatched to high risk node (stragglers appear more often) turns out to be a straggler with higher probability. Thus, in Hummer model, different nodes have different risk of straggler. Hummer eliminates the waiting time in speculative execution and saves more resources compared to Dolly, improving job average duration. Fig.2 shows how Hummer launches clones.

Task delaying risk is related with the node it's running on. We assume that task delaying risk is equal with node delaying risk. Assuming there are N nodes in cluster, each node has different computing ability [18]. Each node has different delaying risk. We calculate the risk of each node delaying P_i ($1 \leqslant i \leqslant N$) according to historical information.

For task-i with its clones, its delaying risk is $1 - q_{i1}q_{i2}\ldots q_{ic}$. Task-$i$ delaying risk without clones is $1 - q_i$. And q_i represents the risk of task-i being straggler, $q_i \in \{P_1, P_2, \ldots P_N\}$. q_{ij} denotes the risk of copy-j of task-i delaying, $q_{ij} \in \{P_1, P_2, \ldots P_N\}$, $1 \leq i \leq n$, $1 \leq j \leq c$. We get the

Hummer model as follows [14, 22, 23]. The job delaying risk with Hummer:

$$Prob_{Hummer} = 1-(1-q_{11}q_{12}...q_{1c})(1-q_{21}q_{22}...q_{2c}) \cdots (1-q_{m1}q_{m2}...q_{mc})(1-q_{m+1})(1-q_{m+2})\cdots(1-q_n) \quad (2)$$

In Hummer, we don't consider task delaying risk as fixed value any more, but related with the node which task is running on.

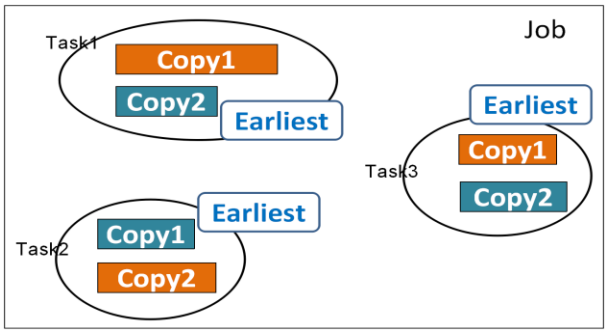


Figure.1 Dolly launches multiple copies for all tasks

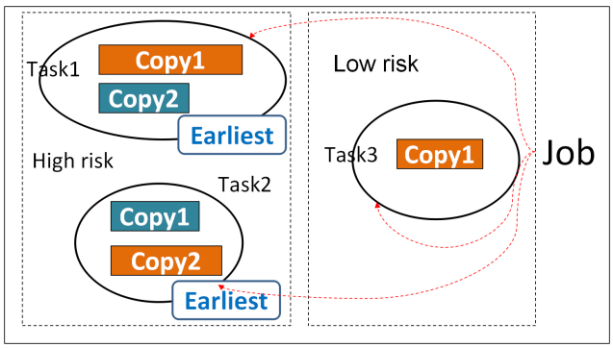


Figure.2 Hummer clones partial tasks in a job

IV. DESIGN OF HUMMER

According to Section III, we know Hummer launches partial clones of job tasks and mitigates stragglers probabilistically. However, there are still some problems for Hummer to solve. They are mainly divided into three categories:

- How we know the task delaying risk;
- The number of tasks which get clones in one job;
- The architecture of Hummer.

A. Risk of Task Delaying

According to Section III, task delaying risk is related with the node which task will run on. Risk of each node appearing straggler is dynamic changing. So we need to estimate the risk of each node appearing straggler periodically. Every time Δ (generally 3s), TaskTracker sends heartbeats to JobTracker, and then JobTracker will re-calculate the node delaying risk according to the information. EWMA is a common method of processing sequence data. It can smooth out the random fluctuations. We use EWMA (Exponentially Weighted Moving-Average) model [24] to estimate node delaying risk:

$$P_i^{new} = \alpha P_i^{measure} + (1-\alpha)\ P_i^{old} \quad (3)$$

α (Hummer uses 0.2) is the smoothing factor. It represents the weight coefficient of history measured value. P_i^{old}, P_i^{new} each represents the old and new risk of node i appearing straggler ($1 \le i \le N$). $P_i^{measure}$ represents the possibility of stragglers in the new arrival tasks within Δ. Algorithm for JobTracker about how to update the task delaying risk is shown as Pseudocode 1.

Pseudocode 1 Task Risk Update Algorithm

```
1: procedure GetTaskDelayingRisk()
2:   if time is up to Δ then
       ▷ every time Δ, update the node delaying probability P_i^new
3:     return UpdateNodeInformation()
         ▷ Update node delaying probability
4:   M= {}
5:   else
6:     for all P_i ∈ {Set of NodeDelayingProb} do
7:       Q_i = min { P_i^new , P_i^measure }
           ▷ Use Q_i before node update, within Δ
8:       M = MUQ_i
9:     end for
10:    return M
11:  end if
12: end procedure
13: procedure UpdateNodeInformation()
14:  for all P_i ∈ {set of NodeDelayingProb} do
15:    P_i^new = α P_i^measure + (1-α) P_i^old
16:    return {set of P_i^new }
17:  end for
18: end procedure
```

Pseudocode 1 shows how to get task delaying risk. The update time Δ means interval between two updates.

We use Q_i to calculate node delaying risk for JobTracker before the next update. Once time is up to Δ, we use P_i^{new} instead of Q_i.

A smaller Δ leads to more accurate estimations. However, if Δ is too small, it will be a burden for JobTracker. We set Δ = 3s, which suits for heartbeat intervals (3 seconds) in existing Hadoop framework.

B. Risk of Job Delaying

According to sub-section A, every time Δ, JobTracker will re-calculate node delaying risk. That is to say, we

update the node delaying risk every time Δ. We use Hummer model in Section III to get the job delaying risk.

If a job comes to the head of job queue before next update, we use min { P_i^{old} , $P_i^{measure}$ } of corresponding node as the risk of each task delaying in the job. Firstly JobTracker makes plan for the dispatching of job' tasks. Then JobTracker calculates the risk of each node delaying and regards this value as task delaying risk that runs on this node. If a job comes right on time Δ, we use the latest update P_i^{new} to calculate the risk of node delaying. Job delaying risk is as follows.

$$\begin{aligned} \Pr ob_{Hummer} &= 1-(1-q_{11}q_{12}...q_{1c})(1-q_{21}q_{22}...q_{2c}) \\ &\cdots(1-q_{m1}q_{m2}...q_{mc})(1-q_{m+1})(1-q_{m+2})\cdots(1-q_n) \end{aligned} \quad (4)$$

C. Number of Tasks Cloned in One Job

It is another key problem to solve how many tasks should get clones in one job. In order to make sure the number of tasks getting clones in one job, we should do some simplifies. We modify the model in Section III as follows (value of c is 3):

$$\begin{aligned} \Pr ob_{Hummer} &= 1-(1-q_{11}q_{12}q_{13})(1-q_{21}q_{22}q_{23})\cdots\cdots \\ &(1-q_{m1}q_{m2}q_{m3})(1-q_{m+1})(1-q_{m+2})\cdots\cdots(1-q_n) \end{aligned} \quad (5)$$

It can be represented by: $1-\prod_{i=1}^{m}(1-\prod_{j=1}^{3}q_{ij})\prod_{i=m+1}^{n}(1-q_i)$

To solve the value of m, we define a function as below:

$$f(m) = 1 - \prod_{i=1}^{m}(1 - \prod_{j=1}^{3} q_{ij}) \prod_{i=m+1}^{n}(1 - q_i) \quad (6)$$

We need to solve the value of m to let $f(m)$ get its minimum value. We take $\prod_{j=1}^{3} q_{ij}$ and q_i as P_1 and P_2. So job delaying risk is:

$$f(m) = 1 - (1 - P_1)^m (1 - P_2)^{n-m} \quad (7)$$

We know that larger P_1 and P_2, larger *f(m)*. So we should clone more tasks by setting larger *m* to decrease *f(m)*. We set P_1 and P_2 as their maximum value p_{max} and p'_{max}. In this case, we get the maximum *m* in order to decrease *f(m)*. In order to get the number of tasks cloned in a job, we design the algorithm as Pseudocode 2.

Pseudocode 2 Resource Dispatching Algorithm

1: **procedure** DispatchResource()
2: M=Cluster resources for clones
▷ Total resources for clones
Q=0
3: $f(m) = 1 - \prod_{i=1}^{m}(1 - \prod_{j=1}^{3} q_{ij}) \prod_{i=m+1}^{n}(1 - q_i)$
▷ Job delaying probability
4: Set maximum value of q_i and $\prod_{j=1}^{3} q_{ij}$ as $p_{\max}$ and $p'_{\max}$
5: $f'(m) = 1 - (1 - p'_{\max})^m (1 - p_{\max})^{n-m}$
▷ Job delaying probability
6: **for all** jobs in cluster **do**
7: Calculate each job delaying probability $f'(m_i)$
8: **end for**
9: According to $f'(m_i) \le \rho$, solve m_i
▷ ρ is the threshold of the job delaying risk
10: **if** $Q \le M$ **then**
11: $Q=Q+ m_i$
12: **else**
do not clone job
13: **end if**
14: **end procedure**

As Pseudocode 2 shows, each job *i* has n_i tasks. Each job *i* can get m_i tasks for cloning. The total resources for cloning are M. Our goal is to let the risk of job delaying be under the threshold by allocating cloning resources. We assume each job delaying risk as $f'(m_i)$. We solve m_i by letting $f'(m_i) \le \rho$.

D. Architecture of Hummer

We implement Hummer prototype by modifying the Hadoop framework, and deploy the prototype on a 30-nodes cluster. We modify the JobTracker of Hadoop MapReduce. Now we introduce the main components of Hummer. Fig.3 shows the architecture of Hummer.

JobTracker JobTracker is responsible for resource monitoring and job scheduling. It monitors all TaskTrackers and status of all jobs. It is different from the original Hadoop JobTracker. When scheduling a job, Hummer JobTracker will make decisions about how to dispatch each task in the job. After developing allocation plan [17, 18, 19], JobTracker collects and updates the straggling risk of nodes which tasks will run on according to the information by TaskTracker. Then after the calculation and decision by Hummer JobTracker, it will launch copies for the high risk tasks at the beginning of tasks execution starting. As for these low risk tasks, Hummer do not launch any copy.

TaskTracker TaskTracker collects the information of the node it monitors and then reports to JobTracker by heartbeats periodically. TaskTracker also executes the commands of JobTracker and does appropriate actions (start new tasks, kill finished or failed tasks, and start copies). TaskTracker uses slots to calculate the resources of local node. A task execution must have a slot.

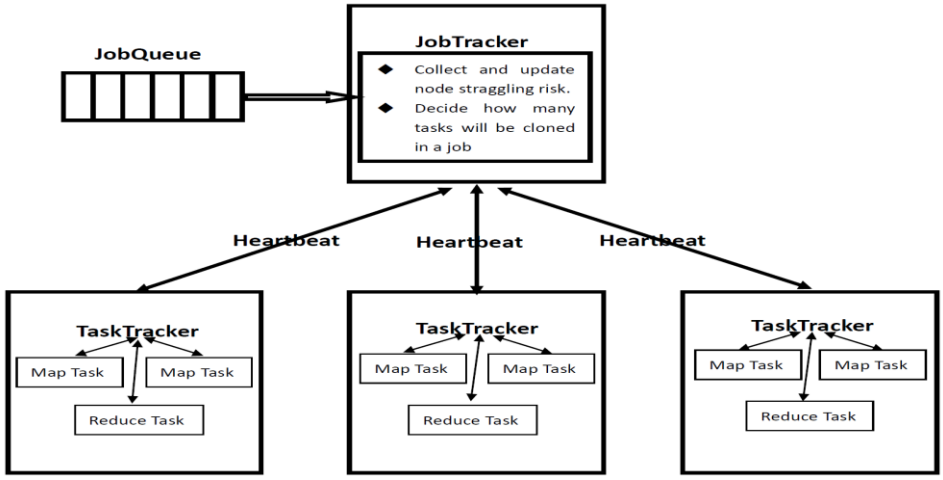


Figure.3 Hummer architecture

V. EXPERIMENTAL EVALUATION

We evaluate Hummer using traces from 100 nodes in our experiment. We deploy our prototype on a 30-node cluster. Data trace is from a Hadoop cluster called OpenCloud [23], which is a research cluster at Carnegie Mellon University (CMU). The traces include 20-month logs, containing 52675 successful jobs, 4423 failed jobs and 1857 killed jobs. There were 78 users in total that submitted jobs in the logs. We analyze the arrival time and finishing time of each job and get the information about how many stragglers exist.

A. The Job Average Completion Time

As table 1 shows, we get job bins by the number of tasks. Fig.4 shows the improvement of job completion time in Hummer.

For only one job, cloning resources in cluster are enough. Dolly can get the resources for all tasks in the job. Although Hummer speeds up the single job completion time, the improvement is not so good as Dolly. We can see that when cloning resources are enough, Hummer's performance is almost as good as Dolly.

However, when the number of jobs increases, cloning resources are not sufficient any more. In this case, Hummer reduces the job average completion time by 32% and 10% compared to LATE and Dolly overall. Hummer is better than LATE and Dolly.

From the results, we can see that Hummer is more suitable for small jobs. Hummer reduces the job average completion time by 46% and 18% compared to LATE and Dolly.

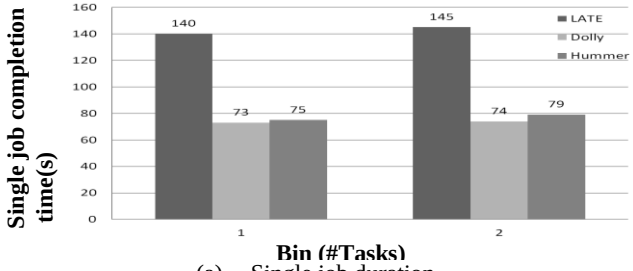
B. Resources Consumption

The most significant advantage of Hummer is that it saves cluster cloning resources. In this experiment, we bin jobs as Table 1. We find that the speed of resources consumption in Hummer is 33% slower than Dolly, as Fig.5 shows. The total resources used for cloning in Hummer are fewer than LATE and Dolly as Fig.6 shows. The reduction of cloning resources in Hummer is almost 50% compared to Dolly.

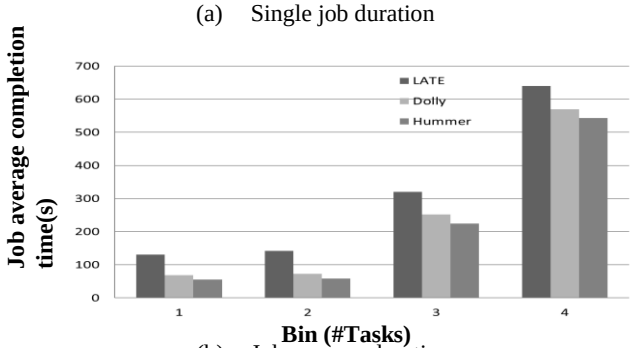
We find small jobs (0-100 tasks) occupy almost 80% cloning resources consumption in cluster. Results show that no matter in Hummer or Dolly, small jobs have priority for resources. Larger job size, fewer cloning resources job consumes. Because JobTracker in Hummer has preference for small jobs. When there exist idle resources, JobTracker will dispatch them to small jobs.

Table 1. Job bins by the number of tasks

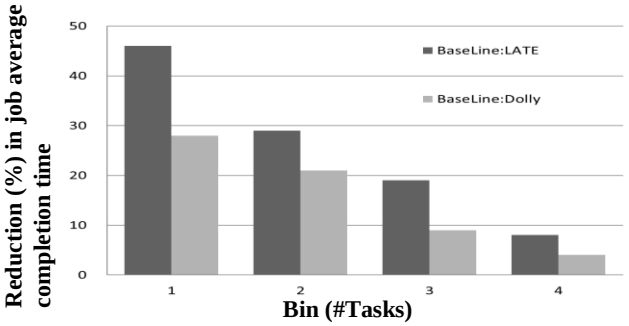
Bin	1	2	3	4
Tasks	0-50	51-100	101-200	201-500



(a) Single job duration



(b) Job average duration



(c) Reduction in job average completion time

Figure.4 Single and Average Job Duration and Reduction (%) of Job Average Completion Time in Hummer and Dolly. We bin jobs of different sizes and calculate the average result of each bin. (a) compare Hummer with Dolly in single job completion time. (b) show the job average duration. (c) show the reduction of job average completion time with baseline LATE and Dolly.

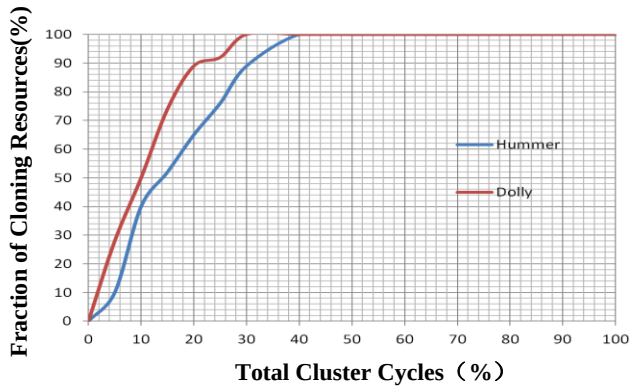
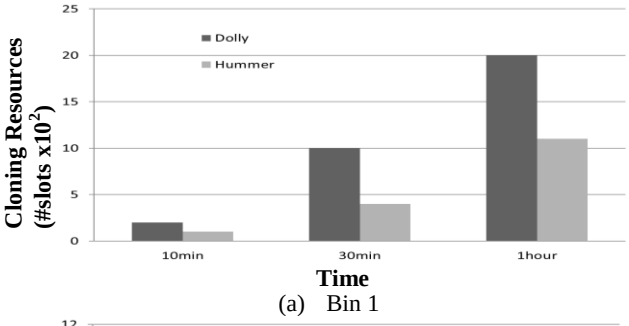
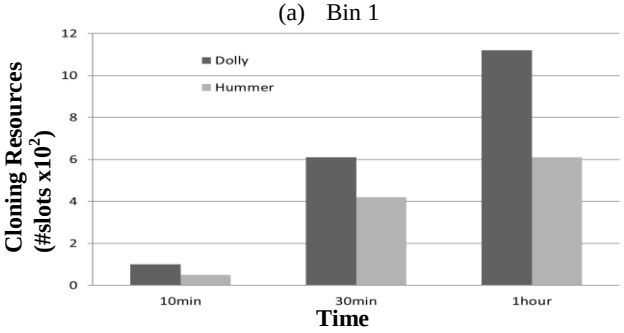


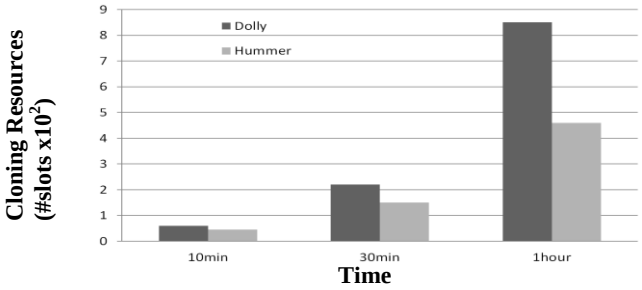
Figure. 5 Speed of Cloning Resources Consumption in Hummer and Dolly. Cloning resources is 5% of the total.



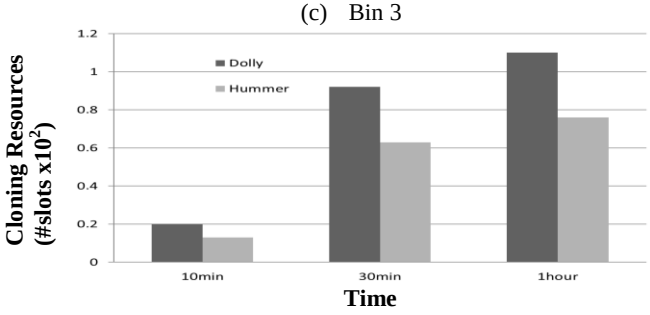
(a) Bin 1



(b) Bin 2



(c) Bin 3



(d) Bin 4

Figure.6 Cloning Resources Consumption

C. *Probability of Job Delaying*

As Fig.7, the y-axis represents the risk of job delaying, the x-axis is the number of tasks in one job.

We can see that as the number of tasks in one job increases, the risk of job delaying also increases. LATE increases fastest. Dolly almost tends to be stable, nearly zero. Hummer is very close to the Dolly [8]. Dolly launches clones for all tasks in a job, so Hummer performs not so well as Dolly in decreasing job delaying risk. But Hummer decreases the possibility of job delaying obviously compared to LATE [6].

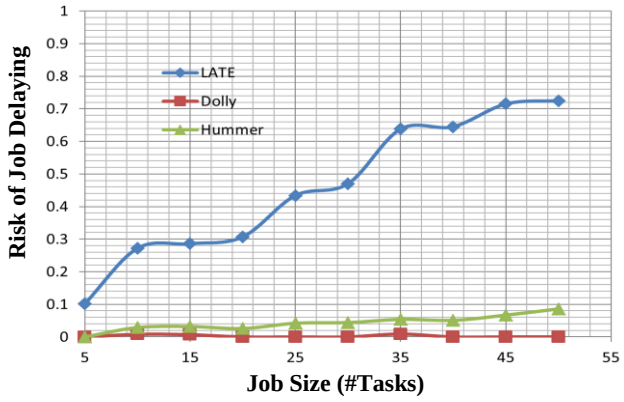


Figure.7 Risk of job delaying in LATE, Dolly and Hummer.

D. *Number of Stragglers*

In this experiment, we use 5% cluster resources for cloning. We run jobs of different size and get the number of stragglers. If one task is up to 2 x slower than the median task completion time in the job, we take it as

straggler. Fig.8 shows the number of stragglers in one job with different size.

We find that when the number of tasks in a job increases, the number of stragglers increases. The average percent of stragglers in a job is 40% in LATE. In Dolly, the average percent is 1%. As we know, Dolly is the better one due to its complete cloning. Hummer is 2%, which is very close to Dolly. As the result shows that Hummer can reduce the number of stragglers effectively. Though Hummer is not good as Dolly in reducing the number of stragglers, Hummer has its advantages over Dolly in job average completion time and saving cloning resources.

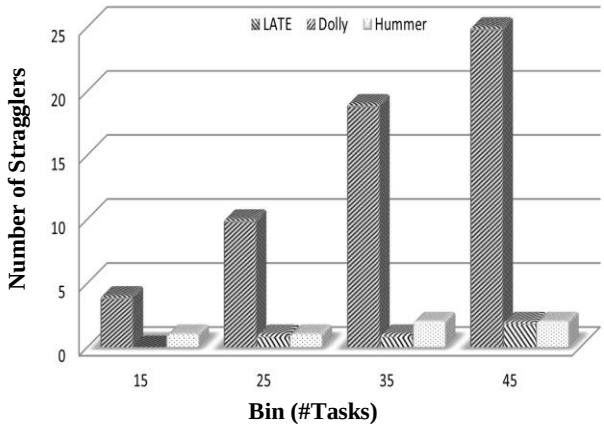


Figure.8 Number of Stragglers in LATE, Dolly and Hummer. We use jobs with different size and calculate the average number of tasks in different job size.

E. Number of Clones for One Task

We know that how many clones for a task are related with resources consumption. Less clones and lower job delaying risk is our goal. Fig.9 shows how job delaying risk changes with the number of copies for a task.

When the number of copies is more than 3, risk of job delaying changes slowly and tends to be stable. So in our experiment, we set the value c as fixed 3.

We use slots a job using for execution to represent the resources consumption. A job consists of *n* tasks, and one task occupies one slot. So, we get the consumption of resources for a single job as follows:

- If we don't clone any tasks, we need the resources Q_1= n;
- If we use Dolly, the resources are Q_2=c×n;
- When we use Hummer, the resources Q_3=(c-1)×m+n.

We observe that LATE needs the minimum resources, but due to the large amount of stragglers, it needs too much time. Though Dolly has lower risk of job delaying, Hummer uses fewer resources than Dolly. In distributed computing, saving cluster resources are very critical for improving the efficiency of cluster. So Hummer is a trade-off between LATE and Dolly.

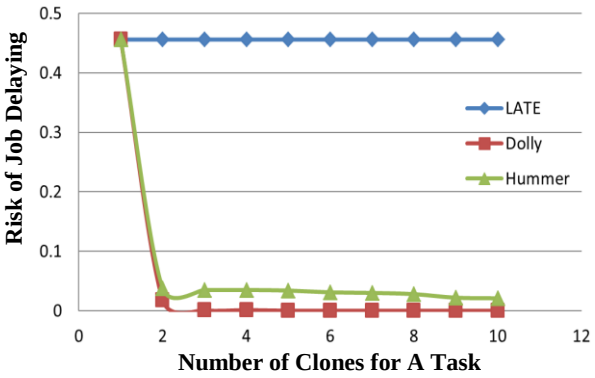


Figure.9 Risk of job delaying changes with the number of copies.

VI. CONCLUSIONS

This paper presents a proactive approach to solve the straggler problem, called Hummer, which just select partial tasks of a job to launch copies.

For which tasks to choose, Hummer makes use of the node differences. Hummer calculates the risk of each node appearing straggler and updates the risk of each node delaying periodically. Hummer selects tasks which are most possible to be stragglers to launch multiple copies. By building job delaying risk model in Hummer, we can calculate the number of tasks which can get clones.

Experimental results show that Hummer reduces the job average completion time by almost 46% and 18% compared to LATE and Dolly for small jobs. Hummer saves the cluster cloning resources by almost 50% compared to Dolly.

ACKNOWLEDGMENT

This work is sponsored in part by the National Basic Research Program of China (973) under Grant No.2014CB340303, the National Natural Science Foundation of China under Grant No. 61222205, the Program for New Century Excellent Talents in University, and the Fok Ying-Tong Education Foundation under Grant No. 141066.

REFERENCES

[1] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, E.Harris, and B. Saha. Reining in the Outliers in Map-Reduce Clusters using Mantri. Proc. of the USENIX OSDI, 2010.

[2] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. SkewTune: Mitigating skew in MapReduce applications. In Proc. of the SIGMOD Conf., pages 25–36, 2012.

[3] LiK, TongZ, LiuD, AzghiT T, LiaoX. A PTS-PGATS based approach for data-intensive scheduling in data grids. Frontiers of Computer Science in China, 2011, 5(4): 513-525 doi: 10.1007/s11704-011-0970-5

[4] JianjinJ, GuangwenY. An optimal replication strategy for data grid systems. Frontiers of Computer Science in China, 2007, 1(3): 338-348 doi: 10.1007/s11704-007-0033-0

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. Proc. of the USENIX OSDI, 2004.

[6] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. Proc. of the USENIX OSDI, 2008.

[7] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. SkewTune in action (demonstration). Proc. of the VLDB Endowment, 5(12):1934–1937, 2012.

[8] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective Straggler Mitigation: Attack of the Clones. Proc. of the USENIX NSDI, 2013.

[9] Ganesh Ananthanarayanan, Michael Chien-Chun Hung, Xiaoqi Ren, Ion Stoica, Adam Wierman, Minlan Yu. GRASS: Trimming Stragglers in Approximation Analytics. In Proc. of the 11th USENIX NSDI, 2014.

[10] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. A Study of Skew in MapReduce Applications. In Proc. of the Open Cirrus Summit, 2011.

[11] Y. Chen, S. Alspaugh, D. Borthakur, R. Katz. Energy Efficiency for Large-Scale MapReduce Workloads with Significant Interactive Analysis. In Proc. of the ACM EuroSys, 2012.

[12] L.A.Barroso. Warehouse-scale computing: Entering the teenage decade. In Proc. of the ISCA, 2011.

[13] S. Resnick. Heavy-tail phenomena: probabilistic and statistical modeling. Springer, 2007.

[14] W. Cirne, D. Paranhos, F. Brasileiro, L. F. W. Goes, and W. Voorsluys. On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems. In Parallel Computing, Vol. 33, No. 3. (2007), pp. 213-234.

[15] Hadoop. http://hadoop.apache.org/.

[16] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Why Let Resources Idle? Aggressive Cloning of Jobs with Dolly. In Proc. of the HotCloud, 2012.

[17] K. Ousterhout, P. Wendell, M. Zaharia and I. Stoica. Sparrow: Distributed, Low-Latency Scheduling. In Proc. of the SOSP, 2013.

[18] A. Ghodsi, M. Zaharia, S. Shenker and I. Stoica. Choosy: Max-Min Fair Sharing for Datacenter Jobs with Constraints. In Proc. of the EuroSys, 2013.

[19] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In EuroSys '10: Proceedings of the 5th European conference on Computer systems, pages 265–278, New York, NY, USA, 2010. ACM.

[20] J. C. Gittins. Bandit Processes and Dynamic Allocation Indices. Journal of the Royal Statistical Society. Series B (Methodological), 1979.

[21] I. Sonin. A Generalized Gittins Index for a Markov Chain and Its Recursive Calculation. Statistics & Probability Letters, 2008.

[22] J. Dean. Achieving Rapid Response Times in Large Online Services. http://research.google.com/People/jeff/latency.html

[23] Kai Ren, YongChul Kwon, Magdalena Balazinska, Bill Howe. Hadoop's Adolescence: An Analysis of Hadoop Usage in Scientific Workloads. In Proc. of the VLDB, 2013.

[24] "EWMA Control Charts". NIST/Sematech Engineering Statistics Handbook. National Institute of Standards and Technology. Retrieved 2009-08-10

[25] Yaobin HE,Haoyu TAN,Wuman LUO, et al. MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data[J]. Front. Comput. Sci., 2014, 8(1): 83-99.