An adaptive algorithm for scheduling parallel jobs in meteorological Cloud

Yongsheng Hao¹, Lina Wang², Mai Zheng³

¹Network Center, Nanjing University of Information Science & Technology, Nanjing, 210044, China

²School of electronic & information engineering, Nanjing University of Information Science & Technology, Nanjing, China

³Computer Science Department, New Mexico State University, Las Cruces, NM, USA

E-mail: hyslove@163.com

Abstract: From traditional clusters to cloud systems, job scheduling is one of the most critical factors for achieving high performance in any distributed environment. In this paper, we propose an adaptive algorithm for scheduling modular non-linear parallel jobs in meteorological Cloud, which has a unique parallelism that can only be configured at the very beginning of the execution. Different from existing work, our algorithm takes into account four characteristics of the jobs at the same time, including the average execution time, the deadlines of jobs, the number of assigned resources, and the overall system loads. We demonstrate the effectiveness and efficiency of our scheduling algorithm through simulations using WRF (Weather Research and Forecasting model) that which is widely used in scientific computing. Our evaluation results show that the proposed algorithm has multiple advantages compared with previous methods, including more than 10% reduction in terms of execution time, a higher completion ratio in terms of meeting soft deadlines, and a much smaller standard deviation of the average weighted execution time. Moreover, we show that the proposed algorithm can tolerate inaccuracy in system load estimation.

Keywords: Distributed computing; cloud computing; parallelism; job scheduling; cluster;

1. Introduction

Data-intensive scientific applications rely heavily on parallelism to achieve high performance. By splitting the whole application into multiple tasks and executing them simultaneously, the performance may be improved by orders of magnitude compared with sequential execution. With the rapid growing of data sets, as well as the mature of large-scale cloud infrastructures (such as Amazon Web Services, Microsoft Azure, and Rackspace Clouds), more and more applications, especially those scientific applications handling huge data sets, will be parallelized to take advantage of the large-scale distributed computing environment. As in the traditional Grid computing environment, job scheduling in this new era is still one of the most critical factors affecting the performance of the parallelized applications.

To achieve high performance in the distributed environment, parallel jobs must be scheduled efficiently. Generally, these parallel jobs require certain amount of resources (e.g., CPU cycles) to execute, and they exclusively occupy the resources assigned until completion. There are two main objectives for scheduling: first, reducing the average execution time of the jobs; second, improving the ratio of the jobs that can be finished before the deadline. However, the two objectives may conflict with each other. For example, if we assign more resources to a job, the job can finish in a shorter time. On the other hand, others jobs have to wait for more time. In the worst case, the other jobs cannot meet their deadlines. As a result, it is challenging to design an optimal scheduling algorithm to meet both

goals. Indeed, previous studies have proved that optimizing task scheduling in the distributed environment is a NP-hard problem [15].

Much work has been done on different aspects of the scheduling problem [1~9, 11~14] and on different platforms [5, 8, 10]. F. Ramezani et al. [15] try to consider the four conflicting objectives at the same time: minimizing task transfer time, task execution cost, power consumption, and task queue length. They designed a MOPSO (Multi-Objective Particle Swarm Optimization) /MOGA (Multi-Objective Genetic Algorithm) based algorithm to find the optimal solution for the proposed multi-objective task scheduling problem and evaluated the method on Cloudsim. L. Jing et al. [16] proposes the global earliest deadline first (GEDF) scheduling policy, where each job can be represented by a directed acyclic graph (DAG) with nodes representing computational work and edges representing dependences between nodes. Y. Hao et al. [17] analyze the scheduling target and propose a 0-1 integer programming for the target. Z. longxin et al. [18] propose to schedule parallel jobs from two perspectives: reliability and energy conservation. They design a RMEC (Reliability Maximization with Energy Constraint) algorithm to keep a balance between reliability and energy consumption. Besides, for real-time workloads with intra-task parallelism, F. David et al. [19] proposes a prototype scheduling service called RT-OpenMP based on OpenMP. While these approaches are excellent for their original design goals, they only focus on one single characteristic of the jobs (e.g., deadline *or* reliability). They do not consider multiple characteristics jointly (e.g., the speedup of jobs *and* the deadlines). Moreover, they do not consider the execution environment (e.g., the system load), which is also critical for efficient scheduling. As a result, these methods may lead to some potential problems when scheduling non-linear parallel jobs. For example, if we always assign the maximum resources to jobs with the shortest estimated execution time, the system may run into the starvation problem, i.e., some low-privilege jobs may never be able to finish before their deadlines. Similarly, if we always assign resources to jobs with the minimum resource needs, some resources may be underutilized. Thus, focusing only on one aspect of the job or ignoring the system load is unlikely to achieve high scheduling efficiency. Moreover, because it is difficult, if not impossible, to estimate the system load precisely, the scheduling algorithm must be able to tolerate inaccuracy in terms of system load calculation.

In this paper, we propose an adaptive scheduling algorithm which takes into account multiple characteristics of the jobs as well as the execution environment, including the average execution time, the deadlines of jobs, the number of assigned resources, and the overall system loads. Our approach is based on the key observation that when the system load is light, we can give more resources to every job than the time when the system has a large system load. With the increase of the arrival rate of jobs, however, we should decrease the number of resources allocating to every job in order to reserve the computing capability of the whole system for more jobs. Also, for a job that has been waiting for a long time, we may give more resources to the job to ensure that it can be finished before its deadline. In one word, we take into account the number of resources as well as the execution time under different system loads all together. More specifically, we consider how to assign resources when the number of assigned CPUs has a non-linear influence on the execution time of the parallel jobs. In most systems, assigning more resources to a job does not always reduce the execution time [20], especially when the number of resources is more than a certain threshold. Thus, when assigning the resources, our algorithm not only takes the different system load into consideration, but also takes the non-linear relation between the execution time and the number of assigned resources. Meanwhile, most of the jobs should be finished before their soft deadlines [21]. The soft deadline means the latest finishing time of

the job without incurring negative effects. If the job cannot finish before the deadline, there will be a negative effect on the system. Note that the job may keep executing even after the deadline, so the negative effect may last. Thus, our algorithm strives to make sure that most jobs can meet their deadlines.

In summary, the main contributions of this paper are:

- An indepth analysis on the correlation among the performance, the average execution time of jobs, the deadline of jobs, and the number of the assigned CPUs under different system loads. A detailed example based on the WRF (Weather Research and Forecasting) Model is provided in Section 3 and Section 7 to illustrate the performance implications of the various factors.
- An adaptive scheduling algorithm taking into account the system load, the speedup under different parallelisms, the soft deadline, and the execution time. The policy makes the decision of resource allocation based on the attributes of the jobs (e.g., deadlines) as well as the system load. Every job is assigned a number of resources based on the system load. If the system load is light or the remaining time of the job is close to the deadline, the job will be given more resources to meet the deadline. Moreover, the scheduling policy is based on the speedup of the jobs. While calculating the speedup of a job precisely is difficult, we make the observation that we can estimate the overall speedup based on some critical points of the speedup function, and use the estimated speedup to improve the scheduling efficiency.
- A thorough evaluation of the proposed scheduling algorithm based on simulation. Particularly, we evaluate our algorithm under different system loads. In practice, it is almost impossible to predict the system load precisely [2, 17, 18]. Our method does not rely on a precise estimation of the system load. We show that our scheduling algorithm can achieve good performance even with only 80% accuracy (i.e., 20% error) of the system load.

The rest of the paper is organized as follows: Section 2 describes related work. Section 3 shows an example of WRF. Section 4 illustrates the framework of our system. Section 5 discusses the scheduling algorithm in details. Section 6 evaluates the algorithm under different system loads based on simulations. Section 7 is the theoretical analysis and discussion. Section 8 concludes the paper.

2. Related work

L. Fan et al. classifies the existing scheduling algorithms for malleable parallel jobs [19] into three categories, including:

- List algorithm. List algorithm schedules the job in arbitrary order. If there are enough resources for the first job, the job is scheduled immediately;
- LPT (Longest Processing Time) algorithm. The only difference between LPT and list algorithm is that the jobs are listed in non-increasing order according to the execution time;
- OM (Optimizing the Middle algorithm) algorithm. OM starts from the schedule produced by LPT and follows by a series of iteration. Through each round of the iteration, OM tries to find a scheduling method which has the smallest execution time of all the scheduling method. The iteration is stopped by some defined conditions.

Compared with malleable parallel jobs, we find that most parallel jobs need more resources when they have higher parallelism. Researchers have proposed some heuristics for the scheduling of parallel tasks. The EQUI (Equi-partitioning) algorithm [27] simply divides the total number of processors evenly among all active job sets at any time. HEFT (Extended-Heterogeneous Earliest Finish Time) [28] has been derived from a list-scheduling algorithm for standard sequential tasks [29]. In these

algorithms, the parallel platform is modeled as a set of configurations where each configuration consists of a set of identical processors. In each step of the algorithm, an unscheduled parallel task is selected and scheduled to the configuration that minimizes its finish time. The task is selected based on the length of the longest path to an exit node where the length of a path is computed as the sum of the computation and communication costs of the nodes and edges along this path. HEFT is similar to the Min-min method [30] in the Grid and in the Cloud (Cloud min-min), except that Min-min does not consider the parallelization of the job. Falco et al. extend the min-min algorithm to the scheduling of parallel tasks in [31].

Papazachos et al. take the parallel tasks as a Gang [32] and they have evaluated AFCFS (Adapted first come first served) and LGFS (Largest gang first served) under different environments. According to AFCFS, a job is scheduled whenever resources assigned to its tasks are available. When a job whose tasks are waiting in front of the queue cannot start its execution, AFCFS policy schedules other Gangs whose tasks are behind the tasks of the aforementioned job. In contrast to AFCFS, according to the LGFS policy, large sized Gangs (refer to the parallelism of the job) place their tasks in processor queues ahead of any other tasks that belong to those Gangs with a smaller size. Y. Hao et al. [17] propose a 0-1 integer programming for the Gang scheduling. Their proposed method tries its best finishing more jobs and minimizing the average waiting time.

Considering the deadline of jobs, some methods are proposed for the scheduling of parallel tasks. L. Jing et al. [16] use DAG (directed acyclic graph) to express a general task model: each task is represented as a DAG and where each node represents a sequence of instructions (thread) and each edge represents a dependency between nodes. The model works in this way: the scheduler first tries to schedule as many jobs with the earliest deadline as it can; then it schedules jobs with the next earliest deadline, and so on, until either all nodes are busy or no more nodes are empty. L. Xiao et al. [33] consider the scheduling of parallel tasks when the job has a soft deadline. They introduce the novel definitions for throughput constraints and candidate constraint points and propose a novel throughput consistency model which can measure the probability of on-time completion at any given constraint point. Based on those works, they propose a novel deadline assignment strategy which is based on the queuing model.

Some methods also try to reduce energy consuming under the satisfaction of other requirements. L. Wei et al. [14] propose a novel two-phase AES (Adaptive Energy-efficient Scheduling). AES improves the system flexibility by justifying threshold automatically. They propose an adaptive threshold-based task duplication strategy in the first phase, and in the second phase, they schedule the groups on DVS-enabled processors to reduce processor energy whenever tasks have slack time due to task dependencies. F. Ramezani et al. [15] consider four targets in the scheduling, not only include the power consumptions, but also include other three aspects: minimizing task transfer time, task execution cost, and task queue length. They develop a MOPSO/MOGA-based algorithm to solve the proposed multi-objective task scheduling problem, and evaluate two evolutionary algorithms on Cloudsim. Z. Zhong et al. [35] try to find the balance between schedule lengths and energy savings by judiciously replicating predecessors of a task if the duplication can aid in performance without degrading energy efficiency. They propose two energy-efficient duplication-based scheduling algorithms—EAD (Energy-Aware Duplication) scheduling and PEBD (Performance-Energy Balanced Duplication) scheduling.

Others researchers also give more work to parallel tasks in different aspects, such as fairness [11, 12], efficiency [11], energy consuming [13, 14]. They are not the target of our paper and we do not

give more information about those studies. Different to previous work, this paper focuses the condition when the job are modular no-linear speedup jobs, and every job has a deadline.

3. An example of WRF

The Weather Research and Forecasting (WRF, <http://www.wrf-model.org/>) Model is the next-generation numerical weather prediction system designed to serve both atmospheric research and operational forecasting needs. It features two dynamical cores, a data assimilation system, and a software architecture allowing for parallel computation and system extensibility. The model serves a wide range of meteorological applications across scales ranging from meters to thousands of kilometers. Figs. 1~4 are the simulation results for different parameters under different numbers of CPUs. Figs. 1~4 shows the execution time (ET), the Speedup (SP), the efficiency (EP) and the consumed resources (CR) of WRF, respectively. T_1 is the execution time when the job only has one resource. T_p is the execution time when the job y has p resources.

The formulas for those parameters are calculated as:

$$SP = T_1/T_p \quad (1)$$

$$EP = SP/p \quad (2)$$

$$CR = P \times T_p \quad (3)$$

Based on the simulation results, we observe that:

- (1) The execution time drops with the increase of the number of CPUs when the number is less than 32;
- (2) The speedup increases with the increase of the number of CPUs when the number is less than 32;
- (3) There are only a little change of the execution time and SP when the number of CPUs is changed from 32 to 48;
- (4) EP always drops with the increase of the number of CPUs.

Moreover, we find that there are three zones according to the relation between the number of assigned CPUs and the execution time:

- (1) 1~32: adding CPUs obviously enhances the speedup, and at the same time, the execution time reduces dramatically. We call this zone [1, 32] as the ideal zone (IZ). In other words, IZ means the scope of the number of the assigned resources that more resources can enhance the speedup and reduce the execution time.

Fig. 1. ET of WRF

Fig. 3. $Speedup$ of WRF

Fig. 2. EP of WRF

Fig. 4. CR of WRF

(2) 32~48: even we assign more CPUs to the job, there is no obvious change in the performance of execution time and speedup. We call (32, 40] as the middle ideal zone (MIZ).

(3) 48~64: if more CPUs are assigned to the job, the execution time of the job will be longer. The reason is that we need more time to exchange information from different CPUs.

In addition, WRF also support checkpoints. The checkpoints ensure that: (1) if the algorithm is stopped for some reasons, the job can be re-submitted to the system and be executed from the checkpoint that is finished last; (2) the checkpoint also bans some large jobs hold some resources for a long time and every job can be stopped from every checkpoint; (3) if there are errors in a job, the user can check after the last checkpoint when the job has errors. Fig. 5 provides an example. Suppose the job applies for three CPUs and gets the permission of the system. There are two checkpoints between $t_{1,1}$, $t_{1,2}$ and $t_{1,3}$, and the same to other tasks. Every task ($t_{i,j}$, i is the i th tasks, j is the j th slot) in the figure needs 4 hours and every slot in the system only gives the job 6 hours. Slot is a constant time in every system and it is the longest serial time that the job can use the resource. We only give 6 hours to every job in a succession time (time slot) to prevent some jobs from holding the resource for a long time. The first slot, the tasks ($t_{i,1}$, $i=1, 2, 3$) have been finished. Though the tasks ($t_{i,2}$, $i=1, 2, 3$) have been begun executing in the first slot, the CPU will be re-restart executing the tasks ($t_{i,2}$, $i=1, 2, 3$) again in the second slot. Until the third slot, all of the tasks can be completed.

Even the job gets four CPUs in the second slot, the job only can use three of the four CPUs. The reason is when the tasks ($t_{i,2}$, $i=1, 2, 3$) have been finished, after the first checkpoint, the data about the cache, memory are stored in the hard disk, even we assigned more CPUs to the job, the system cannot work with more CPUs, expect we restart all the tasks that belong to the job.

Fig. 5. An example of job scheduling on three CPU

4. System model

This study employs a simulation model to address performance issues associated with the job that can be parallelized by different numbers of tasks with different execution times. We do not take into

account other middlewares that is involved in the various platforms. We only pay more attention to the modules that which is more related to our scheduling of parallel tasks. Generally, there are three phases for the scheduling of parallel tasks [26]: submission, mapping or task-to-node allocation and task scheduling. But under our environment, one additional phase must be added into the system before the second phases-named deciding parallelism. The reason is our scheduling target is modular non-linear parallel jobs. Those jobs can not change the parallelism once they have begun. So, deciding parallelism decides the number of assigned resources according to the load of the system, the deadline of jobs, and other requirements. The role of the phrase is to decide the parallelism to ensure the system: (1) to maximize the number of jobs that which have been finished before the deadline; (2) to minimize the average execution time. The two targets are ensured by the cooperation of the following models: *Accounting*, *CNS* (Cluster Node Status Monitoring), *GS* (Global Scheduler), *PTS* (Parallel Tasks Scheduler).

Fig. 6. The System model

Accounting: a user who needs running a job, through the *Accounting* tool to submit the job to the system. In fact, under our environment, the user not only submits the code of the job, but also submits the relation between the number of CPUs and the execution time of the job. Some users may not know any more about the relation, and they submit a null relation to the system. In practically weather models systems, because the number of weather models is a limited number (about 30), the relation can be saved in *GS* (Global Scheduler) and the system can get information from the *GS*. Even some weather models cannot support parallel computing. *Accounting* also has the responsibilities such as ensuring every job is a secure job.

CNS: *CNS* gets the system load information from *LS 1* (Local Scheduler 1) and *LS 2* (Local Scheduler 2), then, according to the system load, to decide whether permit the job to enter into the *GS* (Global Scheduler), in other words, it is in change of arrival rate. Though to get the accuracy system load is difficult, we can use some methods to forecast the system load. For example, from the web application provider's point of view, J. Jiang et al. [38] try to predict future resource demands through uncovering the features of seasonal time patterns by analysing the history data of web application requests.

GS: *GS* manages the system globally. It decides the job that will be paralleled with a certain number of tasks according to two factors: (1) the system load, with the system load increasing, the value drops; (2) the relation between the execution time and the number of assigned CPUs, which is decided by the job.

PTS: after the job has been submitted to *GS*, the parallelism of the job has been decided, *PTS* gets the parallelism of the job from *GS*, and then according to the load information of *LS 1* and *LS 2*, *PTS* decides assign the task to which cluster center. *PTS* uses scheduling methods such as HEFT, EQUF, AFCFS and LGFS. Of course, we can use the method that we will introduce in the following section.

Under our model, the four phrases (submission, deciding parallelism, mapping or task-to-node allocation, task scheduling) work together to schedule resources. At first, we estimate the system load (submission), and then according to the speedup of parallel jobs and other requirements to the jobs, to decide the number of resources that we would allocate to the job (deciding parallelism). The speedup of parallel jobs influences the scheduling result. It gives the relation between the number of resources and the execution time. After we get the parallelism of a parallel job, we decide to assign tasks to resources (mapping) and schedule them accordingly (task scheduling).

5. Scheduling method

As discussed in Section 3, the paper focuses on jobs that have the following attributes:

- (1) The job can be paralleled by different number of CPUs and the system (or the user) can assign different numbers of CPUs to every job;
- (2) Every CPU is assigned to one task and the task collaborates with others and stops when the execution time is more than the slot ;
- (3) The tasks on every CPU have checkpoints, every checkpoint stores data of prior work; the job can be re-launched from the checkpoint where it has been stopped;
- (4) If the job has been assigned to a specific number of CPUs, the number cannot be changed after the job has been begun execution;
- (5) Every job has a soft deadline [21].
- (6) The job has its IZ and MIZ. The scope of IZ and MIZ is determined by the job.

Generally, we can model our system as:

$$\text{Clu}(n, \lambda, q, m, \text{IZ}, \text{MIZ})$$

Where,

- n is the number of kinds of the jobs;
- λ is the average arrival rate of every slot;
- q is the slot of the system;
- m is the number of CPUs;
- IZ is the ideal zone of i th jobs, $\text{IZ}=[bl_i, bh_i]$;
- MIZ is the middle ideal zone of i th jobs, $\text{MIZ}=[ml_i, mh_i]$.

We assume the time between two adjacent checkpoints is less than the slot. We give every job an upper limit because: (1) we try to keep the fairness between jobs to prevent some jobs from holding the resource for a very long time; (2) some algorithms may have errors or exceptions in the execution, we can find them as soon as possible; (3) the user can get phased result of every checkpoint. Suppose the arrival rate of jobs is λ (CPUs per. slot) and the system has m CPUs in the scheduling. There are l jobs that which belong to the i th jobs. For the job J_{ij} , it will be finished in d_{ij} slots if the job has got k_{ij} CPUs. The slot of the system is q hours (suppose $q < 24$). The scheduling result must follow the requirement:

$$\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (k_{ij} * d_{ij}) \leq 24m$$

$$\frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (k_{ij} * d_{ij})}{24m} \leq 1 \quad (5)$$

We must know the load of the system, and then according to the load and the speedup function of jobs, to decide the scope that the number of resources that which is assigned to a job. For the selected scope, a detailed number of resources also is given in the following. We will take account of three cases according to the load of the system.

We will assign CPUs whose number is in IZ when the system has a higher load. Extremely, the job J_{ij} has been assigned bh_i CPUs. and under this condition, the load of the system ∂ must:

$$\text{Case 1: } \partial \leq \frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (bh_i * d_{ij})}{24m} \quad (6)$$

And if every job is assigned bh_i CPUs, the system has a load of 1, in other words:

$$\frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (bh_i * d_{ij})}{24m} = 1$$

And suppose we assign more CPUs ($A_i > bh_i$) to one job, according to the attributes of CR (Fig. 4), the system needs more resources (d_{ij} in the left and in the right of the following formula has different value), then

$$\frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (A_i * d_{ij})}{24m} > \frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (bh_i * d_{ij})}{24m}$$

$$\frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (A_i * d_{ij})}{24m} > 1$$

Then, the system is overload, and that can never happen. So, the number of CPUs that is assigned to a job is in IZ.

The number of assigned CPUs in MIZ when the system has a relatively lower load. Extremely, the job J_{ij} has been assigned mh_i CPUs and it has ck_i checkpoints. The load of the system must in the scope:

$$\text{Case 2: } 1 \geq \frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (mh_i * d_{ij} * k_{ij})}{24m} \geq \partial > \frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (bh_i * d_{ij} * k_{ij})}{24m} \quad (7)$$

Under case 2, the load of the system is 1 when we assign mh_i CPUs to the job that belongs to i th jobs, then:

$$\frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (mh_i * d_{ij} * k_{ij})}{24m} = 1$$

If we assign the number of CPUs (B_i) to the job is more than mh_i , according to the attributes of CR (Fig. 4),

$$\frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (B_i * d_{ij} * k_{ij})}{24m} > \frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (mh_i * d_{ij} * k_{ij})}{24m}$$

$$\frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (B_i * d_{ij} * k_{ij})}{24m} > 1$$

Then, the system is overload, and that can never happen. So, the number of CPUs that is assigned to a job is less than mh_i . In fact, the number of resources that assigned to the job can be less than ml_i , but that makes the job needs more time to be finished and wastes resources. So the number of CPUs is in the scope of MIZ.

If the load of the system is really low and we can assign CPUs as many as possible until the job has the smallest execution time. For the specialty of every job, we can give mh_i CPUs to the job that belongs to i th jobs. The load of the system:

$$\text{Case 3: } \partial \leq \frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (mh_i * d_{ij} * k_{ij})}{24m} \quad (8)$$

Under Case 3, the number of assigned CPUs can be a random number in IZ and MIZ, but the best selection is mh_i , because it has the shortest execution time.

Suppose:

$$t_1 = \frac{\frac{24}{q} * \lambda * \sum_{i=1}^n p_i * \sum_{j=1}^l (bh_i * d_{ij} * k_{ij})}{24m} \quad (9)$$

$$t_2 = \frac{\lambda \frac{24}{q} * \sum_{i=1}^n p_i * \sum_{j=1}^l (mh_i * d_{ij} * k_{ij})}{24m} \quad (10)$$

Besides case 3, we only know that the scope of the number of assigned CPUs in the scheduling. For the i th jobs, we will assign the ideal number of CPUs c_i as follows:

$$\begin{cases} \text{Case 1: } c_i = bl_i + (bh_i - bl_i) * e^{t_1 - \partial} \\ \text{Case 2: } c_i = ml_i + (mh_i - ml_i) * e^{t_2 - \partial} \end{cases} \quad (11)$$

We select exponential function as the ideal number because the value drops more quickly when the value of the load ∂ is increasing. From Figs. 1~4, we find that with the increasing of the speedup, the resources need increasing quickly. Our selection is a tradeoff between the system load and the speedup.

Then how to keep the fairness between jobs is the next problem that which waits for our solution. In this paper, we calculate the fairness by defining the execution time of different jobs. The execution time not only includes the processing time, but also includes the waiting time. We hope a job has the same average execution time when the job has different number of instructions. At the same time, we try to keep the execution time at a relatively low value. Suppose the average weighted execution time is Δe of all jobs, it means the average execution time of the standard task that which can be finished on one CPU. The value of Δe can get from the past scheduling result. For the job J_{ij} , the expected execution time is :

$$etsd_{ij} = \Delta e \times J_{ij} \quad (12)$$

For the three cases, we have given the ideal number of assigned CPUs from the above analysis. Algorithm 1 gives the method about how to give the detailed number of CPUs in the system. Fig. 7 is the flow chart of Algorithm 1:

Algorithm 1 :

Input: J_{ij}, sd_{ij}, w_{ij} /* w_{ij} is the waiting time of the job J_{ij} , sd_{ij} is the soft deadline of the job J_{ij} */

Output: mh_j -the number of CPUs that have been assigned to the job J_{ij} ;

1. Get the average load of the system from past execution ∂ ;
2. $T = w_{ij} + exp(J_{ij}, bh_i)$; /* $exp(j_{ij}, bh_i)$ returns the execution time of J_{ij} with bh_i CPU */
3. If $T \geq sd_{ij}$ /* the job cannot be finished before the soft deadline */
4. Switch ∂ :

```

5.      Case 1:
6.          assign  $bh_i$  CPUs to the job  $J_{ij}$ ;
7.      Case 2:
8.          assign  $mh_i$  CPUs to the job  $J_{ij}$ ;
9.      Case 3:
10.         assign  $mh_i$  CPUs to the job  $J_{ij}$ ;
11.      EndSwitch
12.  Else
13.      Switch  $\partial$ :
14.          Case 1:
15.               $idec = bl_i + (bh_i - bl_i) * e^{t_1 - \partial}$ ;
16.               $T_1 = w_{ij} + exp(j_{ij}, mh_i)$ ;
17.               $ascpu1 = mh_i$ ;
18.              While  $T_1 \leq sd_{ij}$ 
19.                   $ascpu1 = ascpu1 - 1$ ;
20.                   $T_1 = w_{ij} + exp(j_{ij}, mh_i)$ ;
21.              Endwhile
22.              SelectCPUnum( $bl_i, ascpu1, sd_{ij}, w_{ij}$ );
23.          Case 2:
24.               $idec = ml_i + (mh_i - ml_i) * e^{t_2 - \partial}$ ;
25.               $T_2 = w_{ij} + exp(j_{ij}, mh_i)$ ;
26.               $ascpu2 = mh_i$ ;
27.              While  $T_1 \leq sd_{ij}$ 
28.                   $ascpu2 = ascpu2 - 1$ ;
29.                   $T_2 = w_{ij} + exp(j_{ij}, mh_i)$ ;
30.              Endwhile
31.              SelectCPUnum( $ml_i, ascpu2, sd_{ij}, w_{ij}$ );
32.          Case 3:
33.              Assign  $mh_i$  CPUs to the job  $J_{ij}$ ;
34.      EndSwitch
35.  EndIf

```

First of all, we find out whether the job can be finished before its deadline if we assign the number of CPUs according to the lowest value of the zone that the system load belongs to (line 2). If the job cannot be finished before the deadline, we will assign as more as possible of CPUs only under the permission of the system load (lines 4-11). When the system load is in Case 1, we will assign bh_i CPUs to the job (lines 5-6); under other cases, we will assign mh_i CPUs to the job (lines 7-10). If the job can be finished before the deadline, we will give a more detailed analysis of the number of CPUs according to the system load (lines 13-33). When the system load is in Case 1, we get the upper number of CPUs according to formula (11) (lines 15), at the same time, we ensure the job can be finished before the deadline (lines 18-21); when the system load is in Case 2, we get the upper number of CPUs according to formula (11) (line 24) and ensure the job can be finished before the deadline (lines 27-31); when the system load is in Case 3, we assign mh_i CPUs to the job J_{ij} (line 33). For the Case 1 and Case

2, we will give more requirements: (1) it keeps the fairness of the execution time; (2) and it reduces the resource fragments [29].

Fig. 7. Flow chart of Algorithm 1

SelectCPUnum (Algorithm 2) takes those responsibilities in the system. Fig. 8 is the flow chart of algorithm 2. Algorithm 2 gives the details:

Algorithm 2: SelectCPUnum(*lcpu*, *hcpu*, *sd_{ij}*, *w_{ij}*)

Input: *lcpu*, *hcpu*, *sd_{ij}*, *w_{ij}* /* *lcpu* and *hcpu* are the upper and lower limit of the number of CPUs, *w_{ij}* is the waiting time of the job *J_{ij}*, *sd_{ij}* is the soft deadline of the job *J_{ij}* */

Output: *acpu_{ij}*-the number of CPUs that has been assigned to the job *J_{ij}*;

1. Get the expected execution time *et_{ij}* according to formula (12)
2. *beginpos* = *lcpu*;
3. Get the number of the time slots when *J_{ij}* has assigned *beginpos* CPUs;
4. Calculate the resource fragments: *minrefr* = *beginpos* * [*ns*] - *beginpos* * *ns*
5. *etsd* = $\Delta e \times J_{ij}$; /* get the standard execution time of the job; */
6. *selectpos* = *lcpu*;
7. While *beginpos* <= *hcpu*
8. *ettemp* = *exp(J_{ij}, beginpos)* + *w_{ij}*;
9. If (*ettemp* ≥ *etsd*) or (*ettemp* ≥ *sd_{ij}*)
10. Break;
11. Else
12. *beginpos* = *beginpos* + 1;
13. Calculate the resource fragments: *refr* = *beginpos* * [*ns*] - *beginpos* * *ns*;
14. Endif
15. If (*refr* ≤ *minrefr*)

16. $minref = treft;$
 17. $selectpos = beginpos;$
 18. Endif
 19. Endwhile
 20. Assign $selectpos$ CPU to the $J_{ij};$
-

Fig. 8 Flow chart of Algorithm 2

Algorithm 2 gives a detailed method to select the number of CPUs when we get a scope of the number of CPUs from Algorithm 1. First of all, we calculate: (1) the resource fragments when we assigned the number of CPUs as algorithm 1 (line 4); (2) the expected execution time according to formula (12) (Line 5); then we select the number of CPUs to meet the two requirements: the execution time is less than the deadline of the job and the expected execution time (Line 9). To reduce the resource fragments, we select the number of CPUs which has the minimum of resource fragments (Lines 7-19).

As mentioned above, the job J_{ij} has been assigned a specific number of CPUs and it has ck_i checkpoints, it has $ck_i * mh_i$ tasks. The instructions of those tasks are listed:

$$J_{ij} = \left\{ \begin{matrix} t_{1,1}^{ij}, & t_{1,2}^{ij}, & t_{1,3}^{ij}, & \dots & t_{1,ck_j}^{ij} \\ t_{2,1}^{ij}, & t_{2,2}^{ij}, & t_{2,3}^{ij}, & \dots & t_{2,ck_j}^{ij} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{mh_j,1}^{ij}, & t_{mh_j,2}^{ij}, & t_{mh_j,3}^{ij}, & \dots & t_{mh_j,ck_j}^{ij} \end{matrix} \right\} \quad (13)$$

In our model, the tasks $t_{x,temp}^{ij} (x = 1 \dots mh_j)$ formulate a Gang [27] which is a column of J_{ij} . In fact, it is really difficult to make the tasks in the same column of J_{ij} , have the same instructions and can

be finished in the same time, so in the true system, we set the maximum of every column as the instructions of the column ($ck_j \geq temp \geq 1$):

$$t_{temp}^{ij} = \max\{t_{x,temp}^{ij} | mh_j \geq x \geq 1\} \quad (14)$$

J_{ij} also can be denoted as:

$$J_{ij} = \{t_1^{ij}, t_2^{ij}, \dots, t_{temp}^{ij}, \dots, t_{ck_j}^{ij}\} \quad (15)$$

Under the permission of every task only can get one CPU and only has a maximum q hours in a slot. We organize the task in a group as the Algorithm 3:

$exet$ is the execution time of the current group and gid is the label of the current group. First of all, we decide the number of assigned CPUs according to algorithm 1 (line 3). Then we calculate the sum of the execution time of the task t_{pos}^{ij} and the leaving time before the task (line 5). If the value is less than the time slot ($24q$) (line 6), we will add the task to the group gid (line 7); otherwise, we organize the task to the next group (lines 9-11). It must be pointed out that the execution time of every job is less than the slot, otherwise, the task never be finished in the system.

Algorithm 3:

Input: J_{ij}

Output: organize t_{temp}^{ij} in Group G ;

1. $exet=0$; /* the execution time of the current group */
 2. $gid=1$; /* the label of the current group */
 3. Get the number of CPUs C that would be allocated to J_{ij} from Algorithm 1.
 4. For $pos=1$
 5. $exet=exet+Exe(t_{temp}^{ij}, C)$; /* $Exe(t_{temp}^{ij}, C)$ returns the execution time of the task Et_{temp}^{ij} with C CPUs */
 6. If $exet < 24q$;
 7. Add t_{pos}^{ij} to group gid ;
 8. Else
 9. $exet=Exe(t_{temp}^{ij}, C)$;
 10. $gid=gid+1$;
 11. Add t_{pos}^{ij} to group gid ;
 12. EndIf
 13. EndFor
-

6. Performance evaluation

We evaluate the performance of our scheduling algorithm using simulation in this section.

6.1 Simulation environments

In our simulation, there are two clusters, each of which has 2000 computing nodes. All nodes have the same computing capacity. We suppose that the ratio between the executing time of every job and the number of assigned computing nodes (CPUs) has the same value to WRF. Table 1 gives detailed ratios of them and they are the average value of 50 times of WRF under different numbers of CPUs. Suppose the length of the job is obeyed uniform distribution in [500, 9500] minutes, it is also the execution time (in minute) when the job gets 8 CPUs. We suppose every job at least has 8 CPUs in

the system. Every job has a deadline and the deadline is a random number in $[0.75, 4]$ times of the execution time of the job when the job has 8 computing nodes.

Table 1. The relation between the number of CPUs and the execution time of WRF

CPU number	8	16	24	32	40	48	56	64
Execution time (<i>mins</i>)	410	281	231	190	187	184	217	246

To get the accurate system load is difficult in most of real systems [17, 18], so we will investigate our scheduling method in different accuracies of the forecasting system load. “AS100” means the system has 100% accuracy of the system load. “AS90” means that the system has 90% accuracy to the system load. For example, for AS90, when the system load is 100, we can suppose we can forecast the system load is a random number in $[90, 110]$. Similar to “AS90”. “AS80” means that the system has 80% accuracy to the system load. λ is the average arrival rate and it is the average number of jobs that arrive in a slot.

As discussed in the Section 5, the slot value is also essential to the scheduling result. λ is the average arrival rate of every slot. We will set the slot as three different values: 360 minutes, 480 minutes and 720 minutes. The value satisfies the following formula (16). Otherwise, the system is over load and there are jobs can never be finished.

$$\lambda * \left(\frac{500+9500}{2} \right) * 8 < 4000 * ns \quad (16)$$

ns is the time slot of the system (minutes). We get different arrival rate scopes when the system has different slot values:

- when $ns = 360, \lambda < 36$;
- when $ns = 480, \lambda < 48$;
- when $ns = 720, \lambda < 72$.

Because we focus on the system which has a relative high value of load, we select the arrival rate in $[30, 32.5]$, $[40, 45]$ and $[60, 70]$ when the time slot is 360, 480, and 720 (minutes), respectively.

Suppose all the finished jobs are in the set Jl , the total finished job number is $ffend$. The average weighted execution time is AVE and the standard deviation of execution time is $SDWET$.

$$Jl = \{job_1, job_2, \dots, job_{ffend}\} \quad (17)$$

$$AVE = \frac{\sum_{jobid=1}^{ffend} (job_{jobid}.leng * job_{jobid}.et)}{\sum_{jobid=1}^{ffend} job_{jobid}.leng} \quad (18)$$

$$SDWET = \sqrt{\frac{\sum_{jobid=1}^{ffend} job_{jobid}.leng * (job_{jobid}.et - AVE)^2}{\sum_{jobid=1}^{ffend} job_{jobid}.leng}} \quad (19)$$

Where,

- $job_{jobid}.leng$ is the number of the instructions of the job;
- $job_{jobid}.et$ is the execution time of the job.

We compare our method with HEFT, AFCFS, LGFS and EQUI under different values of accuracy of the forecast load. Though HEFT and EQUI are not proposed for the parallel tasks, we can extend them to schedule the parallel task. F. Suter et al. have extended HEFT to the scheduling of parallel tasks in [28]. We suppose EQUI always assigned the resources under the maximum number of resources that the system can provide and the scheduling target can execute. All the values in our simulation are the average values of 50 times of simulations.

6.2 Analysis the average execution time and JBD under different slots

Figs. 9, 11 and 13 are the execution time of different methods. AS100 always has the smallest value of execution time in all those methods. AS100 average decreases 1852.4, 1870.5, 1889.7, 1870.5 minutes to HEFT, AFCFS, LGFS and EQUI. With the drop in the accuracy of the system load, though AS80 has an increase in the value of execution time to AS100, the execution time of AS80 is lower than other methods. To HEFT, AFCFS, LGFS and EQUI, even AS80 average reduces 1157.6, 1175.7, 1194.9, and 1175.7 (minutes) when the time slot is 360, 480 and 720 minutes, respectively.

Fig. 9. AET when slot=360

Fig. 10. The average JBD when slot=360

Fig. 11. AVE when slot=480

Fig. 12. The average JBD when slot=480

Fig. 13. AET when time slot=720

Fig. 14. The average JBD when slot=720

Figs. 10, 12 and 14 are the JBDs (the ratio of the jobs that cannot be completed before their deadlines) under different values of slots and arrival rates. HEFT always has the largest value and followed by LGFS. AS100 keeps the smallest value of JBD under different conditions and followed by

AS90. When the system has a small value of the slot, the difference of all the methods is very obvious. With the increase of the time slot, the difference of JBDs between all the methods becomes small. The reason is the system is more close to the upper limit of the system load when the system has a larger slot. Similar to JBD, the execution times of all the cases also have the same trends.

AS100 has the lowest value in JBD under different environments. Even AS80 (Fig. 10), also is smaller than others four methods. When the slot equals to 360 (*ms*), average JBD of AS100, AS90, AS80 are 12.74%, 11.73%, and 10.77%, respectively; to HEFT, AFCFS, LGFS and EQUI, even AS80 reduces 56.51%, 14.92%, 18.55%, and 14.97%, respectively. When the slot equals to others values, AS also has better performance than other methods.

The average execution times of HEFT, AFCFS, LGFS and EQUI do not show much difference to each other, because they do not consider the relation between the number of assigned CPUs, the execution time and the system load. HEFT tries to minimize the finish time of the job, but it postpones the subsequent jobs, so HEFT has the largest value in JBD.

6.3 Analysis on different accuracies of the prediction system loads

Section 6.2 has shown that our method AS has reduced the value of execution time and JBD under different accuracies of the prediction of system load. In this section, we will compare our method under different arrival rates and different slots to check the influence of the forecast accuracy of the system load. Figs. 15, 17 and 19 are the execution times when the time slots are 360, 480, and 720 minutes, respectively, Figs. 16, 18 and 20 are the JBDs under different cases. The values of X axis are the forecast accuracy of the system loads. The values of Y axis are the arrival rates.

Fig. 15. AET of AS when slot=360 minutes

Fig. 16. JBD of AS when slot=360 minutes

Fig. 17. AET of AS when slot=480 minutes

Fig. 18. JBD of AS when slot=480 minutes

Fig. 19. AET of AS when slot=720 minutes

Fig. 20. JBD of AS when slot=720 minutes

Figs. 15, 17 and 19 show that our method has an increase in the average execution time (AET) with the increase of the arrival rate and the forecast accuracy of the system load. The increase trend enhances more quickly when the time slot increases. In Fig. 15, AS100 always has the lowest value in AET; the average AET of AS100, AS90, AS80 is 3734, 3941 and 4428. To AS90 and AS80, AS100 average reduces 5.25% and 11.68%. Fig.15 and Fig.17 also have the same trends.

Figs. 16, 18 and 20 also show the JBD has the same tendency to the value of execution time under different conditions. AS100 always has the lowest value in JBD. In Fig. 16, the average JBD of AS100, AS90 and AS80 are 12.74%, 11.73% and 10.77%; to AS90 and AS80, AS100 average reduces 8.20% and 13.20%. Fig. 18 and Fig. 20 also have the same trends to Fig. 14.

Figs. 15~20 show the accuracy of the system plays an important role in the performance of JBD and the average execution time. More accurate system load can reduce the execution time and JBD of our method. So, we hope to get a more accurate system load when we use our method.

Though our method is dependent on the accuracy of the system load, Section 6.1 also shows our method also has a lower value in execution time and JBD even we only have 80% accuracy to the system load

6.4 standard deviations of the weighted execution time

Fig. 21. SDWET of different methods when time slot=360 minutes

Fig. 22. SDWET of different methods when time slot=480 minutes

Figs. 21, 22 and 23 are the SDWETs of the five methods under different slots. In general, the simulation results in the three figures show the same trends: (1) the values of SDWETs listed from the largest to the lowest are: AFCFS, LGFS, EQUI, HEFT, AS80, AS90 and AS100; (2) all values of

different methods increase with the increase of the arrival rate; (3) Compared with other methods, AS has a higher value even when the accuracy of the forecasting load is low.

Fig. 23. SDWET of different methods when time slot=720 minutes

We consider the data in Fig. 23 when the time slot equals to 720. The SDWET of AFCFS, LGFS, EQUI, HEFT, AS80, AS90 and AS100 are 9441, 8566, 8945, 8208, 6937, 6658 and 5716, respectively. AS100 always has the lowest value which is only 60.55% of the largest one-AFCFS.

Figs. 21~23 show our method has the lowest value of SDWET even we only have an accuracy of 80% to the forecasting system load. The reason is that AS ensures most of the jobs can be finished near the average execution time. So, AS has proven to be a fair scheduling method to the jobs. It ensures the weighted execution time of every job more closely to each other.

7. Discussion and theoretical analysis

Most existing work assumes that the total resources consumed by parallel tasks is a constant value. In other words,

$$T_i(P_i) \times P_i = T_i(P_i + 1) \times (P_i + 1) \quad (20)$$

-Where $T_i(P_i)$ is the computing time of the job T_i on P_i standard resources, $P_i = 1, 2 \dots N$ [11]. But for most practical systems [15], the following formula holds:

$$T_i(P_i) \geq T_i(P_i + 1) \quad (21)$$

$$T_i(P_i) \times P_i \leq T_i(P_i + 1) \times (P_i + 1) \quad (22)$$

From formula (22), we can find that the consumed resources (CR) increase with the number of the assigned resources. Suppose:

$$k = \frac{T_i(P_i+1) \times (P_i+1)}{T_i(P_i) \times P_i} \quad (23)$$

According to formula (22), $k \geq 1$. For example, in Section 3, the value (k) of WRF (Weather Research and Forecasting model) is more than 1. It means that we consume more CRs for a job if we assign more resources (the number of resources) to the job.

There are two basic laws describing the relation between the speedup and the number of resources that have been assigned to the job [20, 21]: Amdahl's Law and Gustafson's Law.

Amdahl's Law: the maximum percentage of parallel execution time of each resource is par , the percentage of serial execution time is $1-par$, and the number of resources is p . A simple formulation of Amdahl's Law to estimate speedup sp of a parallel application on p cores is given here:

$$sp \leq \frac{1}{(1-par) + \frac{par}{p}} \quad (24)$$

Amdahl's Law has been criticized for ignoring real-world overheads such as communication,

synchronization, and so on [36]. One of the strongest criticisms of Amdahl's Law is that as the number of resources increases, the amount of data handled is likely to increase as well, seeing the example of WRF in Section 3.

Gustafson's Law: also known as the law for scaled speedup. It takes into account an increase in the data size in proportion to the increase in the number of cores and computes the (upper bound) speedup of the application, as if the larger data set could be executed in serial. The formula for scaled speedup is as follows:

$$sp \leq p + (1 - p)s \quad (25)$$

-where, p is the number of cores, s is the percentage of serial execution time in the parallel application for a given data set size.

In very rare circumstances, the speedup of an application exceeds the number of resources. This phenomenon is known as super-linear speedup. In most of true systems, to get the speedup is very difficult. Most of time, we can get the scope of speedup under a certain condition. Amdahl's Law and Gustafson's Law give the upper limit from different views. But for the two laws, they also have difficulty to get the value of par and s . So, it is difficult to get the speedup, and that makes the problem of scheduling of parallel tasks more difficult in the true system.

To know the detail of the speedup is very important for the scheduling. But from the two laws and the example of WRF, it is difficult to get the accurate value. So, in the paper, we only know the approximate value of speedup, even some special points (such as, bl_i , bh_i , and mh_i in the paper), even under this environment, our proposed method work well. We will discuss the speedup of WRF (simplify WRF) in the following paper.

a	a	b	c	e			
a	a	b	c	e			
b	b	b	c	e			
c	c	c	c	e			
e	e	e	e	e			

Fig 24. An example based on the division of zone

Table 2. Execution under different parallelisms

Data size (parallelism)	Execution time	Exchanged Data size	Transferred time per. data block					
			0.25T		0.50T		0.75T	
			TTD	TT	TTD	TT	TTD	TT
8*8(1)	64T	0	0	64T	0	64T	0	64T
4*4(4)	16T	36D	9T	27T	18 T	3T	27T	43T
3*3(9)	9T	80D	20T	29T	40 T	49T	60T	69T
2*2(16)	4T	132D	33T	37T	66 T	70T	99T	103T

WRF is paralleled by the division of zone. Fig. 24 is a simple example of this kind of parallel computing. Suppose every square grid need $1T$ to be finished, the total execution time is $64T$ when we do not parallel executed the job (at the time, the data size is $8 * 8$ in Table 2). At the same time, we suppose that the data in every square grid is $1D$. When the job is paralleled, every task needs the data near to the data (We can think in this way, the weather is always influenced by the weather near the zone). For example, when the parallelism of the job is 9 (the data size is $3 * 3$), the job (which has marked with “a” and “b”) need the data in the square grid which has marked with “c”, the exchanged data is $7D$ (Table 2). The total exchanged data is $80D$. Suppose the transferring of data and the computing of tasks are working sequential, when the transferring time of every data block is $0.25T$, $0.50T$, $0.75T$, the total execution time is $49T$, $60T$ and $69T$ (Table 2). Table 2 and Fig. 25 also give other data under different environment, including the parallelism (plism), the time of transferring data (TTD), the total execution time (TT, including TTD). We find that when the parallelism is increasing, the transferring data size increases greatly, especially when the transferring time of $1D$ is $0.75T$. Some times, even we give more resources to the job, it may not reduce the execution time.

Fig. 25 The TTD and TT under different environments

Generally, if there is only jobs that belong to WRF, we can model our system as:

$$\text{Clu}(\lambda, q, m, \text{IZ}, \text{MIZ})$$

Where,

- λ is the average arrival rate;
- q is the slot of the system;
- m is number of CPUs;
- IZ is the ideal zone, $\text{IZ}=[1, 32]$
- MIZ is the middle ideal zone, $\text{MIZ}=[32, 40]$.

Suppose we assign k_{ij} CPUs to the job and the job need d_{ij} slots, no matter which case the system is, the load of the system must follow ($i=1$):

$$\rho = \frac{\frac{24}{q} * \lambda * \sum_{j=1}^l (k_{ij} * d_{ij})}{24m} \leq 1$$

For Case 1:

When we assign 32 CPUs to the job, the system has the largest value of the load. The upper load of the system is 1, so:

$$\partial = \frac{\frac{24}{q} * \lambda * \sum_{j=1}^l (32 * d_{ij})}{24m} = 1$$

If we assign more CPUs to every job, then $\partial \geq 1$, that can never happen, because it is overload.

Algorithm 2 gives the details of how to select the number of CPUs in the scope of [1,32].

For Case 2 ($i=1$):

$$1 \geq \frac{\frac{24}{q} * \lambda * \sum_{j=1}^l (48 * d_{ij} * k_{ij})}{24m} \geq \partial > \frac{\frac{24}{q} * \lambda * \sum_{j=1}^l (32 * d_{ij} * k_{ij})}{24m}$$

When we assign 48 CPUs to every job, the system has the maximum of the load and it equals to 1. We also can select the number of CPUs in [1, 32], but if we do that, some resources are not working and the job needs more time to be finished. So the best scope of the number of selected CPUs is [32, 48]. Algorithm 2 also gives the details of the number of assigned CPUs.

For Case 3: the best number of the assigned CPUs is 48, which can make the job has the shortest execution time.

8. Conclusion and future work

In this paper, we analyze the correlation between the performance and the execution time and the system load such as the number of assigned resources. From the WRF model, we find that there are three zones for the scheduling. For different zones, we propose different scheduling details. Our proposed method is based on the estimated system load. So, in the simulation, we evaluate our method under different accuracies of the estimated system load. The simulations show our method has relatively good performance even if we only have an accuracy of 80% of the system load.

This paper analyzes from different system load. If the cloud resource can auto-scaling, the scheduling of problem may be more difficult. There are three main methods for the auto-scaling resource in the Cloud [38]: reactive mechanism, predictive-based method, hybrid method. As a future work, we also hope we can analyze the scheduling of parallel tasks when the resource providers can change resource provision under the requirement of the parallel tasks.

In the paper, we only evaluate WRF model in the meteorological Cloud, and our simulation is based on WRF. In fact, most meteorological models can be paralleled in the Cloud, so we hope we can evaluate more models in the future. Some weather models cannot be paralleled. We hope to evaluate environments with both parallel and non-parallel jobs in the same system. We only evaluate our method on a simulation environment, we also hope we can evaluate our method on some new simulation environment, such as Cloudsim[15] or Gridsim [37]. With the Cloud computing becomes more popular in the life, the scheduling of parallel tasks in Cloud also is a hot topic that we will pay more attention to in the future.

Acknowledgements

The work was partly supported by the National Natural Science Foundation of China (NSF) under grant (NO. 41475089), and Open Fund Project (No. NSS1403) of State International S&T Cooperation Base of Networked Supporting Software, Jiangxi Normal University.

References

1. K. Jorissen, F.D. Vila, J.J. Rehr, A high performance scientific cloud computing environment for materials simulations, *Computer Physics Communications*, Volume 183, Issue 9, September 2012, Pages 1911-1919, ISSN 0010-4655, <http://dx.doi.org/10.1016/j.cpc.2012.04.010>.
2. S. Akioka, Y. Muraoka, Extended forecast of CPU and network load on computational Grid, *Cluster Computing and the Grid*, 2004. CCGrid 2004. IEEE International Symposium on , vol., no., pp.765,772, 19-22 April 2004 doi: 10.1109/CCGrid.2004.1336711.
3. F. Nadeem, T. Fahringer, Optimizing execution time predictions of scientific workflow applications in the Grid through evolutionary programming, *Future Generation Computer Systems*, Volume 29, Issue 4, June 2013, Pages 926-935, ISSN 0167-739X, <http://dx.doi.org/10.1016/j.future.2012.10.005>.
4. Y. Hao, G. Liu, An Evaluation of Nine Heuristic Algorithms with Data-intensive Jobs and Computing-intensive Jobs in a Dynamic Environment, *IET software*, Value 9, No. 1, 2015, Page 7-16
5. M. Zhu, Q. Wu, Y. Zhao, A cost-effective scheduling algorithm for scientific workflows in clouds , *Performance Computing and Communications Conference (IPCCC)*, 2012 IEEE 31st International , 2012 , Page(s): 256 – 265. doi: 10.1109/PCCC.2012.6407766.
6. C. Wan, C. Wang, J. Pei, A QoS-awared scientific workflow scheduling schema in cloud computing, *Information Science and Technology (ICIST)*, 2012 International Conference on , vol., no., pp.634,639, 23-25 March 2012 doi: 10.1109/ICIST.2012.6221722.
7. C. Liu, X. Zhang, C. Yang, J. Chen, CCBKE-Session key negotiation for fast and secure scheduling of scientific applications in cloud computing, *Future Generation Computer Systems*, Volume 29, Issue 5, July 2013, Pages 1300-1308, ISSN 0167-739X, <http://dx.doi.org/10.1016/j.future.2012.07.001>.
8. F. Chen, T. Xiang, Y. Yang, Privacy-preserving and verifiable protocols for scientific computation outsourcing to the cloud, *Journal of Parallel and Distributed Computing*, Available online 8 December 2013, ISSN 0743-7315, <http://dx.doi.org/10.1016/j.jpdc.2013.11.007>.
9. D. Yuan, Y. Yang, X. Liu, J. Chen, A data placement strategy in scientific cloud workflows, *Future Generation Computer Systems*, Volume 26, Issue 8, October 2010, Pages 1200-1214, ISSN 0167-739X, <http://dx.doi.org/10.1016/j.future.2010.02.004>.
10. M. Wang, L. Zhu, J. Chen, A QoS-awared scientific workflow scheduling schema in cloud computing, *Information Science and Technology (ICIST)*, 2012 International Conference on , Page(s):634 – 639, ISBN: 978-1-4577-0343-0, doi: 10.1109/ICIST.2012.6221722
11. L. Fan, F. Zhang, G. Wang, Z. Liu, An effective approximation algorithm for the Malleable Parallel Task Scheduling problem, *Journal of Parallel and Distributed Computing*, Volume 72, Issue 5, May 2012, Pages 693-704, ISSN 0743-7315, <http://dx.doi.org/10.1016/j.jpdc.2012.01.011>.
12. H. Sun, W. Hsu, Y. Cao, Competitive online adaptive scheduling for sets of parallel jobs with fairness and efficiency, *Journal of Parallel and Distributed Computing*, Volume 74, Issue 3, March 2014, Pages 2180-2192, ISSN 0743-7315, <http://dx.doi.org/10.1016/j.jpdc.2013.12.003>.
13. L. Wang, S.U. Khan, D. Chen, J. Kołodziej, R. Ranjan, C. Xu, A. Zomaya, Energy-aware parallel task scheduling in a cluster, *Future Generation Computer Systems*, Volume 29, Issue 7, September 2013, Pages 1661-1670, ISSN 0167-739X, <http://dx.doi.org/10.1016/j.future.2013.02.010>.
14. W. Liu, W. Du, J. Chen, W. Wang, G. Zeng, Adaptive energy-efficient scheduling algorithm for parallel tasks on homogeneous clusters, *Journal of Network and Computer Applications*, Available online 31 October 2013, ISSN 1084-8045, <http://dx.doi.org/10.1016/j.jnca.2013.10.009>.

15. F. Ramezani, J. Lu, J. Taheri, F. K. Hussain, (2015). Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments. *World Wide Web-internet & Web Information Systems*, 18(6), 1737-1757.
16. J. Li, Z. Luo, D. Ferry, K. Agrawal, C. Gill, C. Lu, Global EDF scheduling for parallel real-time tasks, *Real-Time Systems*, July 2015, Volume 51, Issue 4, pp 395-439
17. Y. Hao, G. Liu, R. Hou, Y. Zhu, J. Lu. Performance Analysis of Gang Scheduling in a Grid. *Journal of the Network and Systems Management*, July 2015, Volume 23, Issue 3, pp 650-672
18. L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, *Information Sciences*, Volume 319, 20 October 2015, Pages 113-131, ISSN 0020-0255, <http://dx.doi.org/10.1016/j.ins.2015.02.023>.
19. D. Ferry, J. Li, M. Mahadevan, K. Agrawal, C. Gill, and C. Lu, A Real-Time Scheduling Service for Parallel Tasks, *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013 IEEE 19th, 2013, Pages: 261 - 272, DOI: 10.1109/RTAS.2013.6531098
20. K.B.P. Banerjee, Approximate algorithms for the partitionable independent task scheduling problem, in: *Proceedings of the 1990 International Conference on Parallel Processing*, vol. I, 1990, pp. 72–75.
21. M. Tao, S. Dong, Two-tier policy-based consolidation control for workload with soft deadline constrain in virtualized data center, *Control and Decision Conference (CCDC)*, 2013 25th Chinese , vol., no., pp.2357,2362, 25-27 May 2013.
22. M. Matijaš, J.A.K. Suykens, S. Krajcar, Load forecasting using a multivariate meta-learning system, *Expert Systems with Applications*, Volume 40, Issue 11, 1 September 2013, Pages 4427-4437, ISSN 0957-4174, <http://dx.doi.org/10.1016/j.eswa.2013.01.047>.
23. Y. Yang, X. Zhang, Y. Song, An Effective and Practical Performance Prediction Model for Parallel Computing on Nondedicated Heterogeneous NOW, *Journal of Parallel and Distributed Computing*, Volume 38, Issue 1, 10 October 1996, Pages 63-80, ISSN 0743-7315, <http://dx.doi.org/10.1006/jpdc.1996.0129>.
24. L. Fan, F. Zhang, G. Wang, Z. Liu, An effective approximation algorithm for the Malleable Parallel Task Scheduling problem, *Journal of Parallel and Distributed Computing*, Volume 72, Issue 5, May 2012, Pages 693-704, ISSN 0743-7315, <http://dx.doi.org/10.1016/j.jpdc.2012.01.011>.
25. J.L. Gustafson. —Reevaluating Amdahl’s Law. || *Communications of the ACM*, Vol. 31, pp. 532-533, 1988.
26. M. J. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2004.
27. A.Tucker, A. Gupta, Process control and scheduling issues for multiprogrammed shared-memory multiprocessors, *ACM SIGOPS Oper. Syst. Rev.* 23(5) (1989)159–166.
28. F. Suter, F. Desprez, H. Casanova, From Heterogeneous Task Scheduling to Heterogeneous Mixed Parallel Scheduling, In: Danelutto, M., Vanneschi, M., Laforenza, D. (eds.) *Euro-Par 2004. LNCS*, vol. 3149, pp. 230–237, Springer, Heidelberg (2004)
29. H. Topcuoglu, S. Hariri, M.Y. Wu, Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing, *IEEE Trans. Parallel Distrib. Syst.* 13(3), 260–274(2002)
30. S. Anousha, M. Ahmadi, An Improved Min-Min Task Scheduling Algorithm in Grid Computing, *Grid and Pervasive Computing Lecture Notes in Computer Science* Volume 7861, 2013, pp 103-113.
31. I.D. Falco, U. Scafuri, E. Tarantino, Two new fast heuristics for mapping parallel applications on cloud computing, *Future Generation Computer Systems*, Volume 37, July 2014, Pages 1-13, ISSN 0167-739X, <http://dx.doi.org/10.1016/j.future.2014.02.019>.

32. Z. C. Papazachos, H.D. Karatza, Performance evaluation of bag of gangs scheduling in a heterogeneous distributed system, *Journal of Systems and Software*, Volume 83, Issue 8, August 2010, Pages 1346-1354, ISSN 0164-1212, <http://dx.doi.org/10.1016/j.jss.2010.01.009>.
33. X. Liu, D. Wang, D. Yuan, Y. Yang, A Novel Deadline Assignment Strategy for a Large Batch of ParallelTasks with Soft Deadlines in the Cloud, *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC)*, 2013 IEEE 10th International Conference on, 2013, Pages: 51 - 58, DOI: 10.1109/HPCC.and.EUC.2013.17
34. Z. Zong, A. Manzanares, X. Ruan, X. Qin, EAD and PEBD: Two Energy-Aware Duplication Scheduling Algorithms for Parallel Tasks on Homogeneous Clusters, *IEEE transactions on computers*, Volume 60, NO. 3, pp. 360-374,2011.
35. S. Dutta, A. Verma, Service deactivation aware placement and defragmentation in enterprise clouds,"*Network and Service Management (CNSM)*, 2011 7th International Conference on , vol., no., pp.1-9, 24-28 Oct. 2011
36. Gillespie, Matt. "Amdahl's Law, Gustafson's Trend, and the Performance Limits of Parallel Applications." Online].WWW-sivu:http://software.intel.com/sites/default/files/m/d/4/1/d/8/Gillespie-0053-AAD_Gustafson-Amdahl_v1__2_.rh.final.pdf (2008).
37. Y. Hao, G. Liu, N. Wen, An enhanced load balancing mechanism based on deadline control on GridSim, *Future Generation Computer Systems*, Volume 28, Issue 4, April 2012, Pages 657-665, ISSN 0167-739X, 10.1016/j.future.2011.10.010.
38. J. Jiang, J. Lu, G. Zhang, G. Long, (2013). Optimal Cloud Resource Auto-Scaling for Web Applications. *Cluster Computing and the Grid*, IEEE International Symposium on (pp.58-65). IEEE.