# Dynamic Resource-Critical Workflow Scheduling in Heterogeneous Environments

Yili Gong[1*], Marlon E. Pierce[2], and Geoffrey C. Fox[3]

[1] Computer School, Wuhan University, Wuhan, HuBei, P.R.China 430079
yiligong@whu.edu.cn,
[2] Community Grids Lab, Indiana University, Bloomington, IN 47404
mpierce@cs.indiana.edu
[3] Community Grids Lab, Department of Computer Science, School of Informatics, Indiana University, Bloomington, IN 47404
gcf@indiana.edu

**Abstract.** Effective workflow scheduling is a key but challenging issue in heterogeneous environments due to the heterogeneity and dynamism. Based on the observations that not all tasks can run on all resources and acquired data transferring and queuing for a resource can be concurrent, we propose a dynamic resource-critical workflow scheduling algorithm which take into consideration the environmental heterogeneity and dynamism. We evaluate its performance by simulations and show that it outperforms another selected widely used approach.

**Key words:** Dynamic Scheduling, Resource-Critical Scheduling, Workflow, Heterogeneous Environments.

## 1 Introduction

Heterogeneous distributed systems are widely deployed for executing computation and/or data intensive parallel applications, especially scientific workflows [1]. A workflow is a set of ordered tasks that are linked by logic or data dependencies and a workflow management system is employed to define, manage and execute these workflow applications [12]. The efficient execution of workflows in this kind of environments requires an effective scheduling strategy which decides when and which resources the tasks in a workflow should be submitted to and run on.

The environment includes both heterogeneous resource and policy. The software installation and configuration on resources are different as well as their physical computing capabilities. On the other side, the administration policies, such as access control policies, are autonomous and diverse. The dynamism means that the resource status, e.g. load, waiting time in the queue, availability, etc., changes over time are often uncontrollable. Thus the environment requires

* This work is supported by 973 Program (Grant No. 2005CB321807) and the National Natural Science Foundation of China (Grants No. 60773058 and No. 60672051).

that the workflow scheduling take into consideration both heterogeneity and dynamism, which make the problem very unique and challenging.

Concerning heterogeneity, we find that in practice due to access control policy, software version incompatibility or special hardware requirement, it is common that some tasks can not run on certain resources. With this observation, the tasks which can run on every resource are more flexible for scheduling than the resource-critical ones which can only run on just a few resources. For a resource-critical task, considering the more resource-flexible tasks before and after it as a group when scheduling should be better than scheduling them individually. This is the key idea of our resource-critical algorithms.

In terms of the timing of scheduling, there are two categories of workflow scheduling approaches: static scheduling and dynamic scheduling. A static scheduling system makes a schedule before the workflow starts to run based on available resource and environment information; while a dynamic scheduling approach schedule a workflow realtime. The static approach is comparatively simpler and easier to implement. However, its performance heavily relies on the accuracy of the resource and environment information. Unfortunately it is difficult to precisely predict this information due to resource autonomy and free will user behavior. To make full advantage of the known and predicted information as well as to adapt to dynamics of environment, dynamic scheduling is introduced. After initially scheduling, the schedule can be re-assigned according to the hitherto workflow execution progress and resource status at runtime. Thus we use the resource-critical mapping algorithm as a base, but when resource status changes, we using the base algorithm to reschedule the unfinished part of a workflow.

With respect to the architecture of a scheduling system, it could be either centralized or distributed. In a centralized workflow scheduling system, all the scheduling is fulfilled by a central scheduler. While in a decentralized scheduling system, there are many distributed brokers. The cooperation among the brokers is a tough problem and makes the system complicated. Since generally speaking, the calculation overhead of a dynamic scheduling algorithm is far less than the execution cost of a workflow, we still prefer a centralized approach.

Analyzing the makespan of a workflow, it can be seen that it is composed of tasks' execution time, data transferring time and waiting time in resource queues. To reduce any of these three items is beyond the reach of a workflow management system, but it is possible that the data transferring time and the waiting time can be concurrent, i.e. a task can be inserted into a resource waiting queue though its required data are not transferred to the resource yet. As long as the data are available when the task can actually get to use the resource, it works. This is also a principal distinction between our work and other existing work.

In this paper, our main contributions include that we propose a dynamic resource-critical workflow scheduling approach and prove that it outperforms the other selected widely used approach by simulations.

The rest of the paper is organized as follows. The related work is discussed in Section II. In Section III, the proposed dynamic resource-critical workflow scheduling algorithm is described. We elaborate the design of experiments and evaluate the performance of our algorithm in Section IV. The conclusion is shown in Section V.

## 2 Related Work

Extensive work has been done in the field of workflow scheduling in distributed environments. The key differentiators of our work in this paper from the related work lies in that (1) we do not assume that a task can run on all resources, which greatly extends the meaning of heterogeneity; (2) we assume that the data transferring and the waiting time for resources can be concurrent.

HEFT(Heterogeneous Earlier Finish Time) [10] is one of the most popular static heuristic and proven to be superior to other alternatives. Thus we select it as a base algorithm for comparison. In [4], Yu et al. proposes a HEFT-based adaptive rescheduling algorithm, AHEFT. It assumes the accuracy of estimation, i.e. communication and computation cost is estimated accurate and task starts and finishes punctually as predicted. In contrast, our proposed algorithm, DRCS, does not assume this. On the other side, in the AHEFT algorithm, a task can not start without all required inputs available on the resource on which the task is to execute; while we take advantage of the fact that data transferring and waiting in a queue for a resource can be concurrent. In AHEFT, if a task has not finished by *clock*, it will be rescheduled; while in DRCS, the unfinished tasks will be rescheduled when the resource's waiting time changes. [9] is a HEFT-based algorithm for dynamically created DAG scheduling.

The authors of [7] present a decentralized workflow scheduling algorithm which utilizes a P2P coordination space to coordinate the scheduling among the workflow brokers. It is a static scheduling approach and focuses on the scheduling coordination.

In [6] a distributed dynamic scheduling is proposed and it needs to collect resource information from local resource monitor services. Since the calculation overhead of a scheduling algorithm is far less than the execution duration of a workflow and resource information is available from existed third party sercies, we still adopt a centralized approach to avoid additional resource information propagation and synchronization.

Besides using makespan as the single criteria, there are some work on multi-criteria workflow scheduling. [8] proposes a bi-criteria scheduling heuristic based on dynamic programming. [5] presents a bi-criteria approach to minimize the latency given a fixed number of failures or the other way round.

## 3 Dynamic Resource-Critical Workflow Scheduling Algorithm

In this section, we give the details of our dynamic resource-critical workflow scheduling algorithm.

### 3.1 Task Status

During the execution of a workflow, a task is in one of the five possible statuses: unmapped, mapped, submit, running and finished, shown in Figure 1.

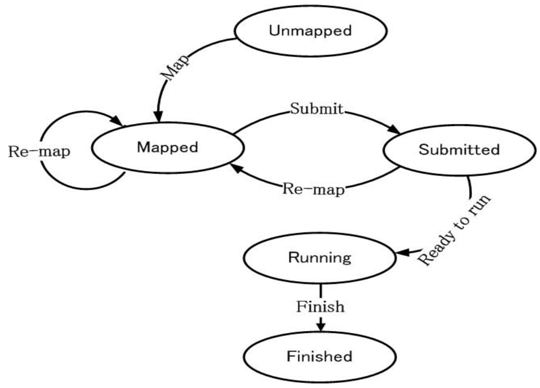


**Fig. 1.** A task's possible statuses and their transitions.

– Unmapped: The task has not been mapped yet.
– Mapped: The task is assigned with a resource but has not been submitted.
– Submitted: The task has been submitted to the resource and is in the waiting queue.
– Running: The task is running.
– Finished: The task has finished and the result is ready for use or transfer.

If it is unmapped, mapped or submitted, a task is called in unfixed status or *unfixed* for short, and we consider it could be rescheduled; if it has started running or is finished, it should not.

### 3.2 Revised Resource-Critical Mapping

In [2], we proposed a Static Resource-Critical workflow Mapping heuristic, referred as SRCM here. Its key idea is that it is better to map neighboring resource-critical tasks as a group than to map them individually. In this paper, we adapt the static approach to dynamic scheduling.

Given a DAG (Directed Acyclic Graph) of a workflow application, $G = (V, E)$, $V = \{v_1, \ldots, v_N\}$ is the set of nodes in the DAG, i.e. tasks in the workflow, $N$ is the total number of nodes. Hereafter, we use the two terms – node and task interchangeably. $vol_{ij}$ denotes the volume of data generated by node $i$ and required by node $j$, $i, j \in V$ and $ij \in E$.

Let the set of resources be $R = \{r_1, \ldots, r_M\}$ and $M$ be the number of resources. $c_{ij}$ is the computation cost of task $i$ on resource $j$. If task $i$ can not run on resource $j$, $c_{ij}$ is infinity.

In a batch system, after submitted, a task typically has to wait for some time in a queue before actually get started. Due to the load on the resource, the waiting time varies with time. $w_{ij}(t)$ is the waiting time for task $i$ on resource $j$ at time $t$. Since in most heterogeneous environments, resources are shared among a lot of autonomous users, it's impossible to know the exact waiting time in future. So far we use QBETS [3] to predict the waiting times, represented as $w'_{ij}(t)$, which might be different from the actual waiting time of a task.

$tr_{kl}$ is the transfer rate from resource $k$ to $l$, $k, l \in R$. $t_{ij}^{kl}$ is the communication cost between task $i$ and $j$ when $i$ is executed on resource $k$ and $j$ on $l$, and $t_{ij}^{kl} = \frac{vol_{ij}}{tr_{kl}}$, $i, j \in V$, $k, l \in R$. When task $i$ and $j$ are executed on the same resource $k$, the communication cost is zero, i.e. $t_{ij}^{kk} = 0$.

Let $parent(v)$ be the parent(s) of task $v$ and $child(v)$ be the child(ren) of task $v$, $v \in V$. These functions can be inferred from the DAG. We assume that the DAG has a single start node $v_0$ which has no parent, i.e. $parent(v_0) = \phi$ and a single end node $v_{N-1}$ which has no child, i.e. $child(v_{N-1}) = \phi$; any of the other nodes has at least one parent and one child.

The main difference of the dynamic scheduling from the original static mapping is that the assignment of a task to a resource might be changed during the workflow execution, thus a variable, time $t$, is introduced. The function $map(v, t) : V \rightarrow R$ is the resource mapping of the task $v$ at time $t$. When scheduling, the new mapping is only related to the last time scheduling result. $t$ represents the current time and $t'$ is the last scheduling time, correspondingly $map(v, t)$ is the current mapping and $map(v, t')$ is the last time mapping.

Let $EST(v, r, t)$ and $EFT(v, r, t)$ be the earliest start time and the earliest finish time of task $v$ on resource $r$ at time $t$ respectively by estimation. $AST(v)$ is the actual start time and $AFT(v)$ is the actual finish time of task $v$.

To calculate the makespan of a workflow, we set $EST(v_0, r, 0) = 0$, $r \in R$, which means that the entry task $v_0$ can run on any satisfactory resource at time 0. For a task $v$, $EST(v, r, t)$ means calculated at time $t$, the earliest time at which all data that $v$ requires have been transferred to resource $r$ and $v$ gets the right to run on $r$. Here, we make an important assumption that the waiting for the data and the resource be concurrent. Thus $EST(v, r, t)$ is defined as

$$EST(v, r, t) = \begin{cases} \max\left(drt(v, r, t), rat(v, r, t)\right), & v\ is\ unfixed, \\ AST(v), & otherwise. \end{cases}$$

Wherein $drt(v, r, t)$ is task $v$'s data ready time and $rat(v, r, t)$ is its resource available time, which are described in detail later.

$EFT(v, r, t)$ is the earliest finish time of task $v$ on resource $r$ and

$$EFT(v, r, t) = \begin{cases} EST(v, r, t) + c_{vr}, & case\ 1, \\ AST(v) + c_{vr}, & case\ 2, \\ AFT(v), & case\ 3, \\ Infinity, & otherwise, \end{cases}$$

Wherein,
case 1: task $v$ is unfixed;
case 2: task $v$ is running and $map(v, t) = r$;
case 3: task $v$ is finished and $map(v, t) = r$.

When a task finishes, its output will be transferred to its child(ren)'s assigned resource(s) immediately. Thus when calculating the data ready time for a parent-child pair, if the previously arranged data transfer is no longer valid (either or both of the mappings for the parent and the child change, we need arrange a new transfer. The earliest data ready time for data from parent $u$ to child $v$ on resource $r$ at time $t$, $edrt(u, v, r, t)$, is as follows:

$$edrt(u, v, r, t) = \begin{cases} t + t_{uv}^{map(u,t),r}, & case\ 1, \\ EFT(u) + t_{uv}^{map(u,t),r}, & otherwise, \end{cases}$$

Wherein,
case 1: task $u$ is finished and either $map(u, t') \neq map(u, t)$ or $map(v, t') \neq r$ or both.

The data ready time for all data that task $v$ requires, $drt(v, r, t)$, is the maximum of the data ready times for all parents, i.e. $drt(v, r, t) = \max_{\forall u \in parent(v)} edrt(u, v, r, t)$.

The resource available time for task $v$ on resource $r$ at time $t$ is the earliest time that $v$ can get $r$ and start to run. If a task has been submitted to its resource's waiting queue, as long as its data can arrive before it finishes waiting and gets the resource, the submission is valid. Otherwise, we need to resubmit the task at time $t$. Here, we assume that resources are FIFO batch systems and jobs submitted earlier should get resources no later than those submitted later.

$$rat(v, r, t) = \begin{cases} rat(v, r, t'), & case\ 1, \\ t + w'_{vr}(t), & otherwise, \end{cases}$$

Wherein,
case 1: $v$ is submitted and $map(v, t') = r$ and $rat(v, r, t') > t$ and $drt(v, r, t) < rat(v, r, t')$.

The makespan, the overall execution time of the workflow, is the actual finish time of the end node, $v_{N-1}$, i.e. $AFT(v_{N-1})$.

In Algorithm 1, we show the revised resource-critical mapping (RRCM) algorithm. The key idea is to consider resource-critical jobs with their resource-flexible neighbors together as a group for mapping is better than mapping them individually.

Since a job may not run on all resources, we define $MR(v)$ as the *match ratio* of the number of resources on which the job $v$ can run and the number of all resources, $v \in V$. By checking the computation cost array, it is easy to get $MR(v)$ by calculating the number of $c_{vr}$ which is not equal to infinity, $r \in R$.

The algorithm has three steps: ranking, grouping and group scheduling.

In the first step, each node and edge of the DAG is given the mean value of all its non-infinite values. The weight of a node is the mean of its computation cost on all matched resources. The weight of an edge is the mean of the maximum

**Algorithm 1** The revised resource-critical mapping (RRCM) algorithm.

```
// ranking
Set weights of nodes and edges with mean values.
Compute the rank of nodes by traversing the DAG upward, starting from the end
node.
Sort the nodes in a non-ascending order of the rank values.
// grouping
G_0 ← φ; i ← 0.
repeat
    Get a node v in the order of nodes' rank values.
    if v's mapping is unfixed and it is ungrouped then
        G_i ← G_i + {v}.
        for all u such that u is v's descendants do
            if all ancestors of u have been grouped, all nodes on the path from v to u
            is in G_i and MR(u) ≤ α then
                G_i ← G_i + {u}.
            end if
        end for
        i ← i + 1; G_i ← φ.
    end if
until there are no more nodes.
// mapping
for all group G_i, in ascending order of i. do
    Schedule the jobs in G_i.
    Choose the schedule with the smallest finish time.
end for
```

of the communication cost and the waiting time of all possible combinations of resources.

With the weights, upward ranking is computed and a rank value is given to each node. The rank value, $rank_i$, of a node $i$ is recursively defined as follows: $rank_i = nw_i + \max_{\forall j \in children(i)} (ew_{ij} + rank_j)$, where $nw_i$ is the weight of node $i$, and $ew_{ij}$ is the weight of the edge connecting node $i$ and $j$.

In the second step, nodes are grouped. First of all, nodes are sorted in the non-ascending order of their rank values. Tie-breaking is done randomly. Mark all fixed nodes as grouped. The first ungrouped node with the highest rank value is added to a group numbered 0. Check each of the node's children if its ancestors are grouped and its match ratio is below a certain valve $\alpha$. If so, add the child node into the group, mark it as grouped and check its children further on. If no additional such node is found, make the next ungrouped node with the highest rank value as the first node of a new group, and so on. The outcome of this process is a set of ordered group, each of which consists of a node and its descendants on the path to which the match ratios of the nodes are all lower than the valve $\alpha$.

In the third step, the node groups are mapped, where any algorithm for scheduling a DAG could be used. Since when scheduling a group, the mapping is probably incomplete, the makespan of the whole workflow is not a proper metric to value different assignments. Given a mapping, an end node is defined as a node which either has no children or whose children have not all yet been mapped. The *finish time* of a schedule is defined as the largest EFT of all end nodes in the group. Comparing two mappings, the one with the smaller finish time is preferred; if they have the same finish time, i.e. the same largest EFT, the one with the smaller second largest EFT is better; and so on. If all EFTs of the end nodes are the same, choose one of them randomly. So far, we adopt an enumerative algorithm to try all combinations of resources for a group and choose the one with the best EFTs of all end nodes.

The main difference between RRCM and SRCM lies in:

- Grouping nodes: on line 9, RRCM requires that if a node is fixed, it will not be grouped.
- EFT calculation: on line 21 and 22, RRCM's method to calculate EFT is different as described above.

### 3.3 Dynamic Resource-Critical Scheduling

In this section, we will introduce the dynamic resource-critical scheduling algorithm, which is based on RRCM. Specifically we use RRCM to schedule the unfinished workflow tasks, shown in Algorithm 2.

When a workflow is first submitted for execution, an initial resource schedule is generated. When some triggering events happen, such as the resource waiting time changing, the tasks would be rescheduled.

**Algorithm 2** The dynamic resource-critical scheduling (DRCS) algorithm.

```
S ← φ
while (((S == φ) OR (triggering event happens)) AND (v_{N−1} is not finished)) do
    update the resource statuses
    update the task statuses
    call the revised resource-critical mapping (RRCM) algorithm
    update mapping() and schedule submit and/or data transfer events
end while
```

## 4 Experiments

In this section, we evaluate the performance of our dynamic resource-critical workflow scheduling algorithm. First, we introduce the experimental environment, followed by the metrics that we select. Then, we compare our DRCS with three other algorithms: AHEFT [4], HEFT [10] and SRCM [2].

### 4.1 Simulation Setup

1. *DAG Generator*
We generate parameter sweep DAGs, whose structure is shown in [2]. Every DAG has one start node and one end node. Tasks on the same level in different branches have same resource requirements and similar execution time. We vary the branch number and the depth respectively from 4 to 12 and from 8 to 24, correspondingly the number of nodes varies from 34 to 290.
2. *Heterogeneity Model*
The heterogeneity model we adopt is based on the loosely consistent heterogeneity model, also called the proportional computation cost model in [11]. Instead of generating the resource computing power randomly, we use the practical numbers from TeraGrid.
The baseline execution time of a task is chosen by using a random uniform distribution over the interval [10, 100]. The computing cost of a task on a resource is a random between 95% and 105% of the quotient of its baseline time divided by the resource's computing power number.
3. *Match Ratio*
This is a factor used in SRCM and DRCS introduced by the factor that some tasks can never run on certain kinds of resources. The match ratio for a task is the ratio of the matching and total resource numbers. The ratios are generated randomly among (0, 1] and a task can run on at least one resource.
4. *Communication Bandwidth*
The communication bandwidth between any two resources is a random number between 5M/s and 300M/s, which are the bandwidth range we measured on TeraGrid.
5. *Communication-to-Computation-Ratio (CCR)*
CCR of a workflow is defined as its average communication cost divided by its average computation cost for all resources. If a workflow's CCR is low, it would be considered as a computation intensive application; while if the CCR is high, it is data intensive.
6. *Waiting-to-Computation Ratio (WCR)*
WCR is the ratio of the average resource waiting time to the workflow computation time.
7. *Match Ratio Threshold (MRT)*
This value is used by SRCM and DRCS to decide what kind of nodes should be grouped together for mapping. If MRT is so small that no node's match ratio below it and every node is an individual group, the SRCM and DRCS will degenerate to HEFT and AHEFT respectively. If MRT is large, the group size grows, it is time-consuming to find the best solution for a big group. In our experiments, we set MRT between 0.1 to 0.5.
8. *Parameters for Dynamic Changing of Resources*
We use two parameters to represent the changing of resources:
   – Resource Change Period (RCP) – the interval of the resource waiting time change;

– Resource Fluctuation Indicator (RFI) – the waiting time fluctuation percentage from the initial value.

### 4.2 Metrics

To compare the performance of the four algorithms, the main metric we use is average makespan difference ratio, which is based on two metrics: makespan and average makespan difference ratio.

1. *Makespan*
   Makespan is the complete time needed to finish a workflow under a certain workflow scheduling algorithm.
2. *Makespan Difference Ratio*
   We use the makespan of HEFT algorithm as a base, and the performance of other algorithms is compared with HEFT's. Thus the average makespan difference ratio of HEFT is always 0.
3. *Average Makespan Difference Ratio*
   For any given branch number and depth, we generate 200 DAGs with their own task computation costs, communication cost, resource matchings and resource bandwidths, each of which is called a case. With each combination of the branch number, depth, CCR and MRT, these four algorithms will run on the 200 cases.
   The average makespan difference ratio is the average of the makespan difference ratios for the 200 cases under the same environmental setting.

### 4.3 Results

In our simulation, we vary the factors introduced above to evaluate their influence on the four workflow scheduling approaches.

Except in the experiment 3, which deals with how the DAG shape of the parameter sweep applications affects the scheduling, the DAG branch number and depth are fixed at 8 and 16 respectively.

1. *Communication-Computation-Ratio (CCR)*
   To analyze the influence of CCR on the scheduling performance, we set $WCR = 1.0$, $RCP = 5000$, $RFI = 0.2$, and $MRT = 0.3$ for the two resource-critical algorithms. The makespans and the average makespan difference ratios under various CCR values are shown in Figure 2 and Figure 3 respectively. Since we set computation cost fixed, bigger CCR means bigger communication cost, thus for all the algorithms, the overall makespan gets longer.
   When CCR is small, the two static approaches, HEFT and SRCM, and the two dynamic approaches, AHEFT and DRCS, perform almost the same. As CCR grows, the performance of SRCM and DRCS get better and when CCR is over 3, the static approach SRCM even outperforms the dynamic approach AHEFT. This surpassing depends on the fact that most benefit of

the resource-critical algorithms comes from the communication time saving. As the weight of the communication time in the makespan gets higher, the benefit gets bigger. Therefore, SRCM and DRCS are more suitable for the data intensive applications.

Figure 3 presents the improvement of AHEFT, SRCM and DRCS over HEFT, from which we can notice more clearly the tendency that AHEFT approaches HEFT and SRCM approaches DRCS. In further on simulation, when $CCR = 100$, the difference between HEFT and AHEFT is about 0.69% and the difference between SRCM and DRCS is about 1.19%. This is because as CCR increases, the dynamic scheduling algorithms have less opportunity to re-assign the tasks, since the cost of moving data gets bigger.

2. *Waiting-Computation-Ratio (WCR)*

   Here $CCR = 1.0$, $RCP = 5000$, $RFI = 0.2$, and $MRT = 0.3$. Figure 4 and Figure 5 present the results. For all four algorithms, the WCR increasing causes the increasing of the waiting time cost, correspondingly the increasing of the makespan.

   When WCR is small (= 0.1), the two resource-critical algorithms performs almost the same and better than HEFT and AHEFT. While as WCR grows, the two dynamic algorithms are much less affected than the static ones. It shows that dynamic scheduling can adjust the schedule when the waiting time changes to shorten the overall execution time and the longer the waiting time, the more obvious the advantage. It can be seen that DRCS is always performs better than the other three, including AHEFT.

   From Figure 5, it can been seen that the performance of HEFT and AHEFT tends to close to that of SRCM and DRCS respectively. When $WCR = 10$, the average makespan difference ratio of SRCM over HEFT is only 0.65%, and the difference between AHEFT and DRCS is 0.92%. This shows again that the benefit of SRCM and DRCS are from the communication cost reduction, once the waiting time gets longer, the weight of the communication cost decreases, thus the performance improvement decreases.

3. *DAG branch number and depth*

   In this set of experiments, $CCR = 1.0$, $WCR = 1.0$, $RCP = 5000$, $RFI = 0.2$, $MRT = 0.3$. When the branch number varies, the depth is fixed at 16; while when the depth varies, the branch number is 8.

   As the branch number varies from 4 to 12, the makespan of four algorithms increases (refer to Figure 6). This happens due to the reason that the branch number growth causes more tasks are ready to run at approximately the same time, since the capacity of resources is limited, some of the tasks have to wait longer to actually acquire the resources.

   Figure 7 presents the performance improvement of the two dynamic algorithms decreases with the branch number. For instance, when the branch number is 4, the makespan difference ratios of AHEFT and DRCS are 22.69% and 30.96 respectively; while when the branch number is 12, the ratios are 18.96% and 23.59%.It shows that when the resource competition is fierce, there is little room for the dynamic approaches to reschedule the tasks to get

better waiting time. In contrast, the difference ratio of SRCM over HEFT does not change much with the different branch numbers.
It is evident that the makespan increases approximately linearly as the depth varies from 8 to 24 (see Figure 8 and Figure 9), since more tasks should be executed sequentially. The deeper the depth, the bigger the improvement ratio of the two resource-critical algorithms than the corresponding HEFT or AHEFT algorithms. The improvement ratio of SRCM over HEFT increases from 4.00% to 8.69% and that of DRCS over AHEFT increases from 5.26% to 10.82%. This shows that deeper depth allows the resource-critical algorithms group more nodes together to achieve better schedule.

4. *Resource Change Period (RCP) and Resource Fluctuation Indicator (RFI)*
   To measure how the resource changing affect the algorithms, we introduce two factors: Resource Change Period and Resource Fluctuation Indicator, which depict when and by what degree resources change.
   In Figure 10, the setting is $CCR = 1.0$, $WCR = 1.0$, $RFI = 0.2$, and $MRT = 0.3$. We can see that the resource change period has no influence on the performance of the dynamic approaches. In contrast, as the period grows, the makespan of the static ones decreases. The static approaches decide the schedule of the workflow before it starts, and will not change during the its execution duration. Thus when the resources change, i.e. the waiting times change, the initial schedule will become unsuitable and the performance suffers. If the resource change period is long, it would change less times during the workflow execution and the suffering would be less, correspondingly the makespan improves. As a result, the dynamic scheduling methods are adapted to the dynamic resource environment. In Figure 11, $CCR = 1.0$, $WCR = 1.0$, $RCP = 5000$, and $MRT = 0.3$. It shows that the resource fluctuation percentage does not affect the performance of workflow scheduling much. This is because the resource status fluctuation makes some jobs finish earlier than predicted and some later, and the influence is balanced out.
5. *Match Ratio Threshold (MRT)*
   Match ratio threshold is only used in the resource-critical algorithms. Here, we set $CCR = 1.0$, $WCR = 1.0$, $RCP = 5000$, and $RFI = 0.2$.
   In Figure 12, as the MRT increases from 0.1 to 0.5, the makespan of SRCM and DRCS decreases from 56404.30s to 55122.44s and from 44122.46s to 42841.48s respectively. This is because with a bigger MRT, the algorithms could group more nodes together and try all the combinations to select the best out them.

## 5 Conclusion

In this paper we have presented DRCS, an efficient workflow scheduling approach for heterogeneous and dynamic systems based on the resource-critical algorithm. Aiming at heterogeneity, the algorithm combines the resource-critical

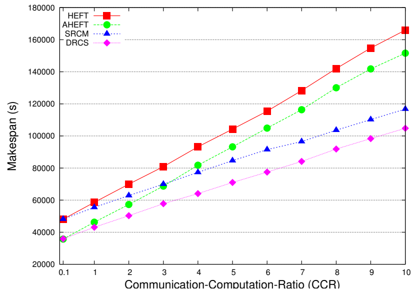


**Fig. 2.** Makespans under various CCRs.

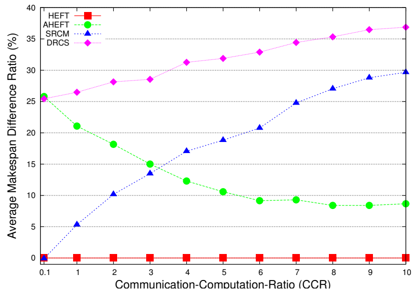


**Fig. 3.** Average makespan difference ratios under various CCRs.

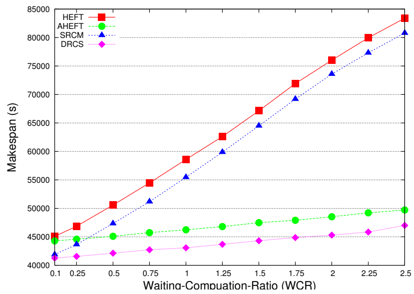


**Fig. 4.** Makespan under various WCRs.

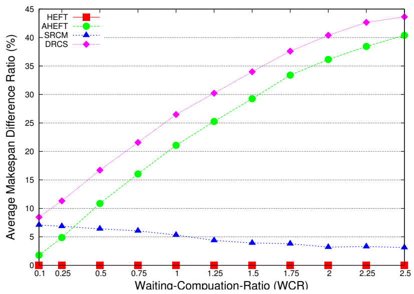


**Fig. 5.** Average makespan difference ratios under various WCRs.

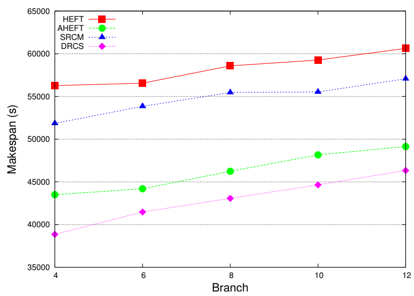


**Fig. 6.** Makespan under various branch numbers.

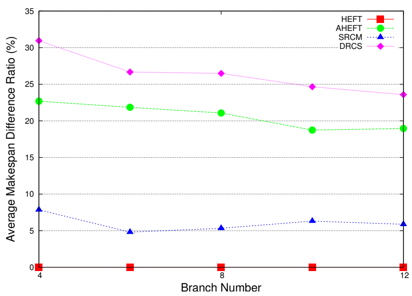


**Fig. 7.** Average makespan difference ratios under various branch numbers.

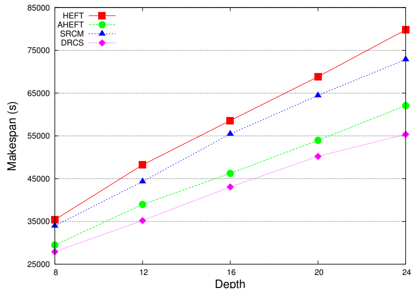


Fig. 8. Makespan under various depths.

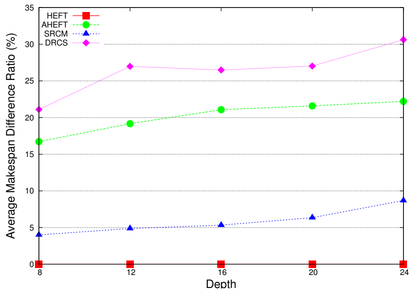


Fig. 9. Average makespan difference ratios under various depths.

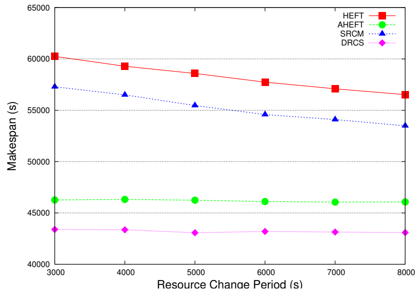


Fig. 10. Makespan under various resource change periods.

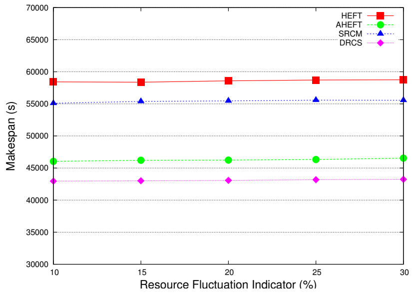


Fig. 11. Makespan under various resource fluctuation percentages.

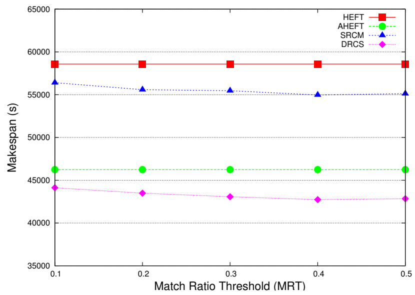


Fig. 12. Makespan under various match ratio thresholds.

tasks with their ancestors and/or descendants together and finds the best schedule for them as a group. For dynamism, it reschedules the unfinished tasks according to the current resource status. To evaluate the performance of DRCS, simulation studies were conducted to compare it with other competitors in the literature, HEFT, AHEFT and SRCM. It is shown that DRCS outperforms HEFT, AHEFT and SRCM in almost all environments in terms of makespan. Especially, the two resource-critical idea based algorithms, DRCS and SRCM are suited for data-intensive applications. The two dynamic scheduling algorithm, DRCS and AHEFT are superior in the long waiting time systems.

To further on adapt to the unreliable, dynamic and heterogeneous environment, we plan to investigate the effect of resource liability and task failure on the scheduling performance.

## References

1. The QuakeSim Project Website, http://quakesim.jpl.nasa.gov/
2. Gong, Y., Pierce, M.E., Fox, G.C.: Matchmaking Scientific Workflows in Grid Environments. In: 20th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'07), Cambridge, MA, Nov. 2007
3. Nurmi, D., Brevik, J., Wolski, R.: QBETS: Queue Bounds Estimation from Time Series. In: 13th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'07), Seattle, WA, June 2007
4. Yu, Z., Shi, W.: An Adaptive Rescheduling Strategy for Grid Workflow Applications. In: 21st IEEE International Parallel & Distributed Processing Symposium (IPDPS'07), Long Beach, CA, March 2007
5. Benoit, A., Hakem, M., Robert, Y.: ault Tolerant Scheduling of Precedence Task Graphs on Heterogeneous Platforms. In: 22nd IEEE International Parallel & Distributed Processing Symposium (IPDPS'08), Miami, FL, April 2008.
6. Dong, F., Akl, S.G.: Mobile Agent Based Workflow Rescheduling Approach for Grids. In: 20th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'07), Cambridge, MA, Nov. 2007
7. Ranjan, R., Rahman, M., Buyya, R.: A Decentralized and Cooperative Workflow Scheduling Algorithm. In: 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08), Lyon, France, May 2008
8. Wieczorek, M., Podlipnig, S., Prodan, R., Fahringer, T.: Bi-criteria Scheduling of Scientigc Workflows for the Grid. In: 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08), Lyon, France, May 2008
9. Hunold, S., Rauber, T., Suter, F.: Scheduling Dynamic Workflows onto Clusters of Clusters Using Postponing. In: 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08), Lyon, France, May 2008
10. Topcuouglu, H., Hariri, S., Wu, M.Y.: Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems. Vol. 13, No. 3, pp. 260-274, March, 2002
11. Kwok, Y., Ahmad, I.: Dynamic Critical Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Transactions on Parallel and Distributed Systems. Vol. 7, No. 5, pp. 506-521, May, 1996
12. Yu, J., Buyya, R.: Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing. Vol. 3, No. 3-4, pp. 171-200, Sept. 2005