Yu ZF, Shi WS. Queue waiting time aware dynamic workflow scheduling in multicluster environments. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 25(4): 864–873 July 2010. DOI 10.1007/s11390-010-1067-6

Queue Waiting Time Aware Dynamic Workflow Scheduling in Multicluster Environments

Zhi-Feng Yu (余志峰) and Wei-Song Shi (施巍松), *Senior Member, IEEE*

Department of Computer Science, Wayne State University, Detroit, MI 48202, U.S.A.

E-mail: zhifeng.yu@wayne.edu; weisong@wayne.edu

Received August 1, 2009; revised March 17, 2010.

Abstract Workflows are prevailing in scientific computation. Multicluster environments emerge and provide more resources, benefiting workflows but also challenging the traditional workflow scheduling heuristics. In a multicluster environment, each cluster has its own independent workload management system. Jobs are queued up before getting executed, they experience different resource availability and wait time if dispatched to different clusters. However, existing scheduling heuristics neither consider the queue wait time nor balance the performance gain with data movement cost. The proposed algorithm leverages the advancement of queue wait time prediction techniques and empirically studies if the tunability of resource requirements helps scheduling. The extensive experiment with both real workload traces and test bench shows that the queue wait time aware algorithm improves workflow performance by 3 to 10 times in terms of average makespan with relatively very low cost of data movement.

Keywords workflow management, scheduling, multicluster

1 Introduction

Workflow applications have been prevailing in scientific computation recently. Most of them can be represented in a form of a direct acyclic graph (DAG), whose performance is measured by *makespan*, the time difference between the start and finish of a workflow. The term workflow and DAG are used interchangeably hereafter.

Nowadays, as infrastructure technology matures rapidly, multicluster environments emerge to meet increasing computation demands. Clusters are connected with a high speed backbone to build high performance and throughput computing platforms, where each cluster has its own independent workload management system. TeraGrid[1] and DAS-2[2] are two typical examples in the US and Europe respectively.

In a multicluster environment, each cluster manages its local workload independently. A job submitted to a cluster is subject to the local scheduling policy. Jobs may have to wait in a queue for long time before getting executed. However, traditional workflow scheduling approaches, majority of which are static, always assume that a job will get executed immediately once it is scheduled. On the other hand, it is a challenge to predict queue wait time for any job. Recently, there has been a breakthrough on the queue wait time prediction practice[3], which has been implemented on TeraGrid[1] as QBETS Network Weather Service[4] querying all queues of multiple clusters.

In addition, there has been more research on scheduling mixed-parallel application on Grids[5-6], which provides a suitable degree of multiprocessor job parallelism[7]. The progress can help a workflow request more resources to improve its performance and maximize resource efficient utilization, depending on the resource availability.

These advancements in grid computing challenge traditional workflow schedulers. Specifically, the traditional workflow scheduling heuristics cannot effectively benefit from significantly increased resource provision. Most of them do not consider the job queue wait time in a mutilcluster environment, even worse, the fundamental assumptions made by static ones are not valid at all in this situation. For example, static heuristics unrealistically assume that resources are fixed and are always available with full capacity. The misconceptions and unrealistic assumptions are analyzed in the work[8].

We argue that the increasing resource provision provided by multicluster environments can be practically and efficiently utilized by augmenting the planner guided dynamic scheduling approach[9-10]. With a hierarchical scheduling scheme, the *Global Scheduler*, with queue wait time awareness, makes scheduling decisions

Regular Paper
This work is in part supported by the US National Science Foundation CAREER Grant No. CCF-0643521.

by balancing the queue wait time, possible execution time reduction thanks to more available resources, and possible extra data movement time, with a goal of minimizing the workflow *makespan*.

The contributions of this paper are: 1) augment the scheduling strategy[9-10] for multicluster environments; 2) evaluate the performance of the proposed algorithm with real workload trace and published workflow test bench; and 3) study the impact of tunable resource requirements, e.g., the same job can be executed on different numbers of processors, in the context of dynamic environments.

The rest of the paper is organized as follows. We describe the system design and algorithm in Section 2. Section 3 presents the experiment design. The experimental results are evaluated in Section 4. Section 5 discusses the related work and finally Section 6 summarizes the paper and lays out the future work.

2 Scheduling Strategy

In this section we introduce the system design, define resource requirement and performance models used in this study, describe the queue wait time prediction, and finally present the detailed algorithm.

2.1 System Design

The system designed to support scheduling workflows in multicluster environments is illustrated by Fig.1.

Similar to the system design described in [10], we employ the scheme of planner guided dynamic scheduling:

1) *Meta Scheduler* manages job admission, enforces the inter-job dependencies, prioritizes jobs, constructs appropriate resource requirements and selects suitable resources to achieve minimal overall makespan. It is further broken into three major components:

a) *Planner* ranks each individual job of a workflow and ensures that jobs submitted to the *Job Pool* are ready to schedule. The upward ranking algorithm defined in [11] is used to calculate the rank value recursively, starting from the exit job n_{exit} of a workflow:

$$rank_u(n_{\text{exit}}) = \overline{w_{\text{exit}}}. \tag{1}$$

For other non-exit jobs, the rank values are computed recursively defined by

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} (\overline{c_{i,j}} + rank_u(n_j)) \tag{2}$$

where $succ(n_i)$ is the set of immediate successors of job n_i, $\overline{c_{i,j}}$ is the average communication cost jobs between n_i and n_j, and $\overline{w_i}$ is the average computation cost of job n_i.

Basically, the $rank_u(n_i)$ is the length of the critical path from job n_i to the exit job, including the computation cost of job n_i. It also guarantees that the rank value of n_i is always higher than any successor. Furthermore, the job with higher rank is more critical than

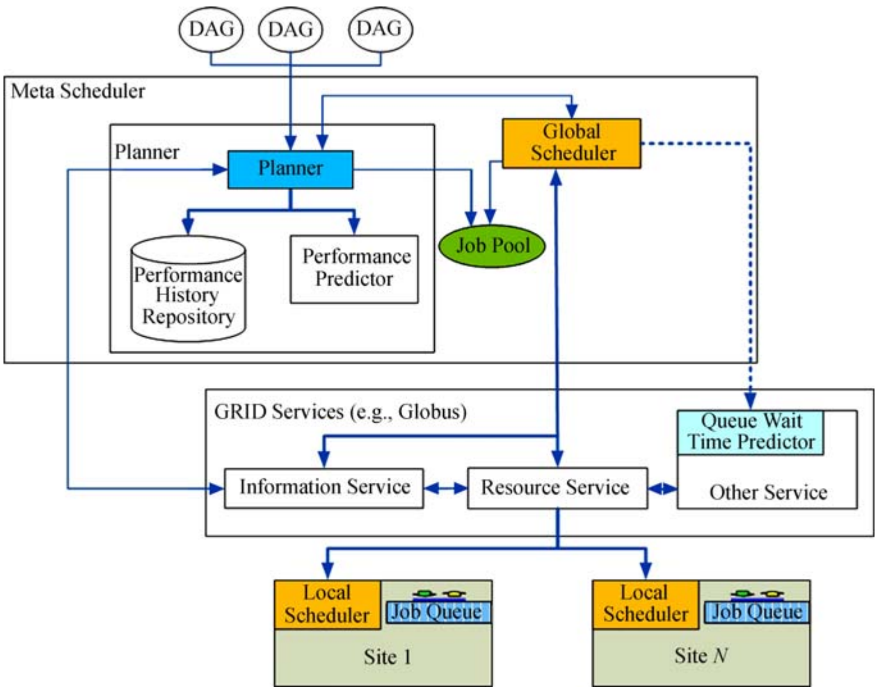

Fig.1. Conceptual system design of DAG scheduling on multicluster.

the ones with lower rank. It should be assigned with the better resource collection to minimize the total execution time.

In addition, the *Planner* is responsible for specifying the resource requirements and estimating the execution performance accordingly. As a result, each job is submitted with a rank value and possibly multiple resource requirements. The requirements may be different on the number of resources to request.

b) *Job Pool* contains all jobs ready to be scheduled in an unsorted manner. A job is ready to be scheduled when all its predecessors complete successfully. However, the job actually starts until all required input data is available on the executing cluster where the job runs.

c) *Global Scheduler* is the most vital component in this system. It picks up the job with the highest priority and selects the most suitable resource. In a multicluster environment, the *Global Scheduler* does not have direct control over resources. Instead, it only dispatches jobs to an independent cluster with appropriate resource requirements accordingly. Therefore, it is very crucial to schedule jobs based on priority and be able to predict the queue wait time on targeted cluster.

2) *Grid Service* includes software and components supporting security, information infrastructure, resource management etc. As one of the grid service software in production use, the Globus Toolkit[12] allows users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when. One important component is the "Queue Wait Time Predictor", which can relatively accurately predict how long a job has to wait in the queue for resource requirement fulfillment. A real world implementation of such a component is the QBET Network Weather Service[4] deployed on TeraGrid[1], which can predict the upper bound of wait time given the resource requirement.

3) *Local Scheduler* refers to the independent workload management system on each individual cluster. It receives jobs dispatched by the *Global Scheduler* and treats these jobs as any other local job submitted by the users directly to this cluster. No different from local jobs, the workflow jobs are subject to the same local scheduling policy.

2.2 Resource Specification and Performance Model

While recent years are witnessing Grid platforms in growth, the single processor based heuristics hardly leverage the resource abundance. However, when an application can access more resources, it is still difficult for a user to describe the resource requirements. [13] presents an empirical model that allows a DAG based workflow application to generate appropriate resource specifications, including the number of resources, the range of clock rates among the resources and network connectivity. This is referred to as *Tunable Requirement* in this paper.

While [13] proposes a "Size Prediction Model" that predicts the best resource number to use in resource requirement specifications sent to a scheduler, it admits that predicting the "best size" is challenging. To simplify the performance modeling, we assume that each processor has the same clock rate[13].

Furthermore, we refer to Amdahls' law discussion in [14] which assumes efforts that devote r resources will result in sequential performance $\sqrt{r}$. In other words, if $w_{i,j}$ is the estimated time to complete job n_i on processor r_j, and job n_i will be scheduled to n processors with the same capacity as r_j, the execution time of job n_i will be $w_{i,j}/\sqrt{n}$. Thus, the performance can be doubled with four time numbers of processor assigned. It is worth noting that [14] tried other similar functions (for example, $\sqrt[1.5]{r}$) but found no important changes to their results. This simplified performance model will be used in the forthcoming discussion and experiment design.

2.3 Queue Wait Time Prediction

Queue wait time prediction technology did not show significant impact on scheduling strategy design until the recent deployment of QBETS[3] on more than a dozen super computing sites, offering two types of online queue delay predictions for individual jobs:

1) Given the job characteristics, QBETS can predict a statistical upper bound on how long the job is likely to spend waiting in the queue prior to execution;

2) Given the job characteristics and a start deadline, QBETS can calculate the probability that the job begins execution by the deadline.

As QBETS provides an API and web service for external integration, we can safely assume an on-line queue time prediction service readily available and the *Global Scheduler* can inquiry the service of queue wait time for any resource requirement. The query result is the input for *Global Scheduler* to decide where to dispatch the job and with what resource requirement, more specifically, the number of processors to request.

2.4 Scheduling Algorithm

As Fig.1 shows, the *Global Scheduler* picks up the job with the highest priority from the *Job Pool*, i.e., the highest rank value, to schedule first. Different from traditional heuristics, which assume a job is only to execute on a single processor, the *Planner* in this system potentially defines multiple appropriate resource requirement specifications for any individual job. In reality, many parallel programs specify the number of processors as a command line argument. In this study we assume that, if a job has multiple appropriate resource requirements, the requirements differentiate each other solely on the number of processors to request.

The scheduling algorithm is to schedule jobs on the "best" resource collection, which is measured by the smallest earliest finish time (EFT). The *makespan* of a workflow is actually the maximum EFT value of all exit jobs. In this context, a resource collection is qualified if it meets all the resource requirements, such as operating system type, storage space, network bandwidth. The best one minimizes the EFT with all potential costs and benefits being taken into account, including the performance adjustment with request of different numbers of processors, consequently resulted different queue wait time and possible data movement cost, described as below:

$$\begin{aligned}EFT(t) = clock() + exec(t, q, p) + \\ comm(t, S) + QBETS(S, q, p).\end{aligned} \quad (3)$$

For each appropriate resource requirement, the *Global Scheduler* checks all qualified resource collections to determine the EFT for each by estimating:

• *Execution Time* $exec(t, q, p)$. This is estimated based on the performance model defined in Subsection 2.2. We assume the estimated execution time for job t on single processor in queue q is $W_{t,q}$ and $exec(t, q, p) = \frac{W_{t,q}}{\sqrt{p}}$.

• *Communication Cost* $comm(t, S)$. It is very common for a local cluster to have a centralized share data storage. If a job and all its parent jobs execute on the same cluster S, the communication cost will be zero, otherwise there is a cost for data movement between clusters.

• *Queue Wait Time Prediction* $QBETS(S, q, p)$. This service returns the predicted queue wait time for a job if it is scheduled to queue q on cluster S with request for p processors. It is worth noting that a typical cluster S may contain multiple queues. As a common sense, a request for larger number of processors may cost longer wait time in the queue. So a trade off has to be made between gain on performance and wait time in the queue.

The proposed algorithm is naturally augmented from the planner guided dynamic scheduling[9-10]. However, how to schedule multiple workflows is not discussed here for two reasons: first, as a matter of fact, each cluster may receive workflows locally and schedule them within its domain, which is equivalent to scheduling multiple workflows; second, here we focus on how to schedule workflows with dynamic background workload across multiple clusters and the algorithm presented here shares the same philosophy with the one defined in [10].

3 Experiment Design

The simulation based experiment is performed with DAS-2 workload trace[2] and workflow test bench[15], and developed with a workflow simulation framework GridSim[16]. As an explorative study, the following assumptions are made in the simulation:

• Each job may request various numbers of processors as part of the resource requirements. Once scheduled, a job cannot change its resource requirement over time.

• Each individual job is scheduled to a single cluster.

• The local scheduler implements FIFO-ordered First Fit policy. When there are free resources available, the first job in the queue will be evaluated and scheduled if there is a sufficient number of processors available. Otherwise, the job stays at its original location in the queue until sufficient resources become available. However, the next job can backfill if its request can be accommodated. So on and so forth for the remaining jobs in the queue.

• If a job's parent(s) execute on different cluster(s), it is required to transfer the parent job's data output to the cluster to where the job is scheduled. The data transfer starts right after the job arrives. It is assumed that the Grid service transfers file(s) from the remote cluster(s) as requested.

• We assume the queue wait time prediction capability in the system design. The experimental predicts the queue wait time for a job by simulating the execution sequence of the jobs in the queue. With regard to prediction accuracy, we are not able to run a statistical comparison against QBETS due to unavailability of actual data of QBETS. However, the prediction accuracy is insignificant in the context of workflow scheduling and not a focus here.

3.1 Workload Simulation

The DAS-2 supercomputer consists of 5 clusters located at 5 Dutch universities. The clusters are interconnected by the Dutch university Internet backbone and the nodes within a local cluster are connected by high speed Myrinet as well as Fast Ethernet LANS. All clusters use openPBS[17] as local batch system (one and only one queue is configured for each cluster). One may notice that most assumptions made earlier are rightfully valid with the DAS-2 environment. A comprehensive analysis of the workload on DAS-2 is available in [2].

From the original DAS-2 workload traces, we randomly extract four 24-hour-period (from 14:00 to 14:00 next day) traces with various intensity and add a special zero workload trace for the simulation. Table 1 provides workload trace details.

Table 1. Simulation Trace Details

Trace	No. Jobs					Total
No.	fs0	fs1	fs2	fs3	fs4	No. Jobs
0	0	0	0	0	0	0
1	3 823	857	1 170	277	431	6 558
2	2 166	850	819	261	801	4 897
3	4 191	656	1 632	483	219	7 181
4	4 701	18	17	3 505	1 911	10 152

Each job entry in the workload trace contains the following information: the job number, the job arrival time, the job start time, the job run time, the job completion status, the number of processor requested and allocated, the cluster assigned to, and so on. All jobs in the traces are exactly simulated according to these information except for the job start time. The jobs from traces may experience longer wait time in the queue as there are extra jobs from workflows to compete resources.

3.2 Performance Metrics

The following performance metrics are used in the simulation to evaluate the effectiveness of the proposed workflow scheduling strategy:

• *Makespan*, which is the total execution time for a workflow application from start to finish. It is used to measure the performance of a scheduling algorithm from the perspective of workflow applications.

• *Data transfer time*, which is a measure of total time of data transfer across clusters. Data movement occurs when the child jobs are scheduled to different clusters which either accommodate the requests or have better resource provision. But this consumes network resources and should be minimized.

• *Queue wait time*, which is a measure of the total time workflow jobs wait in the queue. The *queue wait time* for a job is the time difference between when it arrives in the queue and when it gets executed. A job waits in the queue because: it observes the FCFS policy to wait its turn to get executed and/or it waits for the required data being transferred to the cluster which it is scheduled to.

• *Resource effective utilization*, which is a measure of the fraction of used processor cycles with respect to its best possible usage during the workflow execution. Similar to [18], it is defined in (4).

$$\mathit{Effective\ Utilization} = \frac{\sum_i \mathit{ProcessorUsed}_i \times \mathit{Runtime}_i}{\mathit{ProcessorsAvailable} \times \mathit{Makespan}}. \tag{4}$$

The *resource effective utilization* also takes the background workload into account but is only measured for the period of workflow execution in this simulation.

3.3 Workflow Simulation

The published test bench[15] for workflow applications is used to evaluate the proposed algorithm as well. It consists of randomly generated DAGs and is structured according to several DAG graph properties[15]:

• *DAG Size*: the number of nodes in a DAG.

• *Meshing Degree*: the extent to which the nodes are connected with each other.

• *Edge-Length*: the average number of nodes located between any two connected nodes.

• *Node- and Edge-Weight*: two parameters describe the time required for a job's computation and communication costs, related to the communication to computation ratio (CCR).

The test bench also assumes that each of the available computing nodes executes just one job at a time and that we have accurate estimates for the computation time and communication time of the corresponding DAG scheduling problems[15].

3.4 Scenarios to Evaluate

As the previous work[9-10] already demonstrate that the planner guided dynamic scheduling well outperforms the popular FIFO and Random algorithms, we do not compare against them again. In this experiment, we simulate different scheduling scenarios defined in Table 2 with a focus on understanding how the proposed algorithm benefits from the queue wait time awareness in a multicluster environment and the impact of tunable resource requirements.

Table 2. Scheduling Scenarios to Evaluate

Scenario Name	No. Processors to Request	Queue Wait Time Predictable
16_no	16	No
32_no	32	No
48_no	48	No
16_yes	16	Yes
32_yes	32	Yes
48_yes	48	Yes
Tun_yes	16, 32 or 48	Yes

For the first six scenarios, each job has a single resource requirement which specifies the number of processors to request. As the name suggests, 16_no always requests 16 processors and is blind to queue wait time.

In other words, in (3), the $QBETS(S(q), q, p)$ function always returns value of 0 in scenarios of 16_no, 32_no and 48_no. In another scenario Tun_yes, each job has multiple requirements, i.e., requesting for 16, 32 and 48 processors respectively. To accommodate the tunable requirements, the algorithm should be able to predict the queue wait time. It is worth noting that jobs observe the local FCFS scheduling policy in all clusters in these scenarios.

4 Evaluation Results and Analysis

Now we are in a position to depict the evaluation results. The published test bench[15] for workflow applications is used to evaluate the algorithm. The DAG group with most numbers of jobs (175 to 249 jobs) in chosen in the simulation. The time unit used in the following discussion is in seconds.

4.1 Experimental Results

We first simulate all scenarios with the 5 workload traces described in Table 1 and 50 DAGs with all characteristics (Meshing degree, Edge-length, Node- and Edge-weight) being random. Fig.2 illustrates workflow performance with regard to average *makespan* in different scenarios. With workload trace 0, a special case of zero background workload, the algorithm performs almost identically with all scenarios. With the real workload in presence, measured by the average *makespan* the algorithm performs much better with queue wait time awareness than without. Overall, with the same number of processors requested, the algorithm improves average *makespan* 3 to 10 times when prediction is enabled.

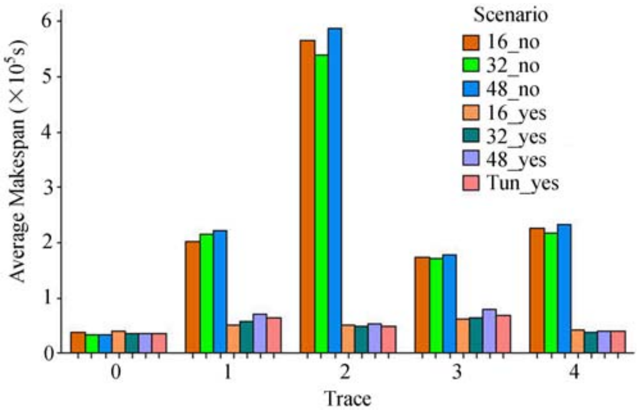

Fig.2. Average workflow *makespan* in various scenarios.

In terms of total *queue wait time*, as shown in Fig.3, jobs spend way more time waiting in queues when the algorithm is not able to predict, regardless of how many processors jobs get allocated. This contributes in a considerable portion to the long *makespan*. It is understandable that the queue wait time awareness helps reduce job queue wait time significantly.

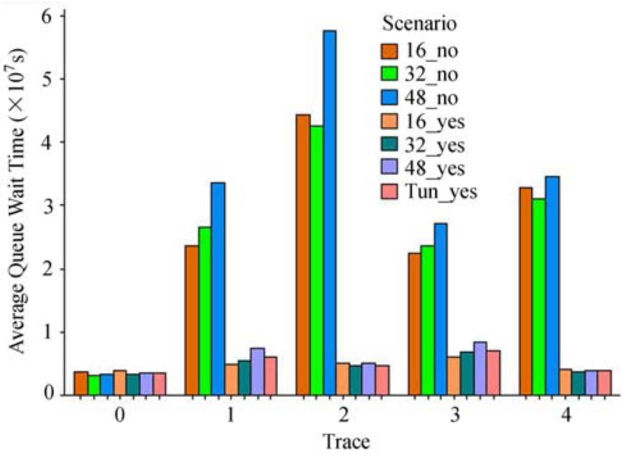

Fig.3. Average job queue wait time in various scenarios.

However, good performance is achieved at the cost of extra time spent on data movement across clusters, as shown by Fig.4. Data movement is more active in the scenarios with prediction enabled. This is the trade off between performance gain and extra data movement.

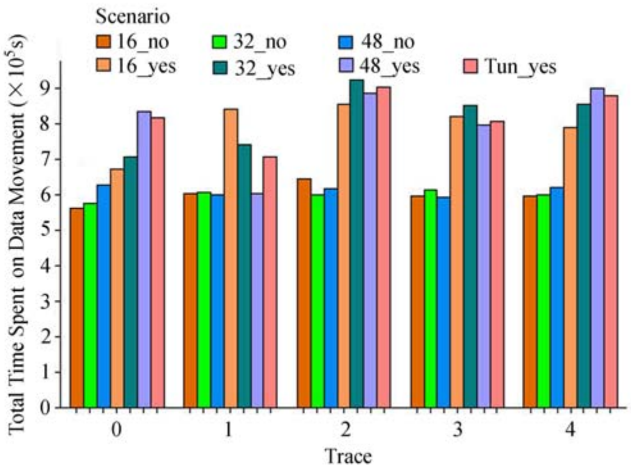

Fig.4. Total time spent on data movement in various scenarios.

To better understand how the algorithm performs in different scenarios with various workload traces, [19] provides an overview of performance measurements on average *makespan*, total *queue wait time*, total *data transfer time* and total number of data transfer requests. It is observed that:

1) There is an evidently strong correlation between average *makespan* and the total queue wait time. The shorter the queue wait time, the less the average *makespan*. This leads to an obvious conclusion that to reduce the queue wait time is more crucial then to

minimize *makespan*. It is worth noting that we are not able to break down the *makespan* to the degree at which we can tell exactly how much queue wait time contributes to overall *makespan* for two reasons: first, the *makespan* measures overall workflow execution time but queue wait time is tracked at individual job level; second, a job waits in a queue for various reasons as specified in Subsection 3.2.

2) Average *makespan* is reduced at the cost of data movement. Except for trace 0, the scenario with the smallest *makespan* always has very high total data transfer time and the number of data transfer requests. The data movement happens if the total reduction of execution time and queue wait time can offset the extra time spent on data movement.

3) Queue wait time prediction is critical when the background dynamic workload is considered. For trace 0, which is a special case of zero background workload, the algorithm performs almost identically with or without prediction. Particularly, the scenario of 32_no has the best performance. This does not cast any doubt on the necessity of queue wait time prediction. Contradictorily, it shows that existing heuristics, which do not consider any of resource competition and dynamic workload, are not justified realistically.

4) Dynamic scheduling is inherently nearsighted. The dynamic scheduling decision is to minimize the EFT of each individual job. With this guidance, a job can be simply dispatched to a different cluster with data movement required to reduce the EFT for this job only, but the decision may hurt the overall performance of entire workflow. This observation is supported by: no single scenario outperforms others in all cases; high volume data movement helps reduce *makespan* but does not warrant that the scenario with the highest total time of data transfer always has the least *makespan*. Even so, the dynamic scheduling is the only viable choice in real world when both dynamic resource and dynamic workload are considered. The key to improve its effectiveness is the collaboration between the *Planner* and the *Executor*.

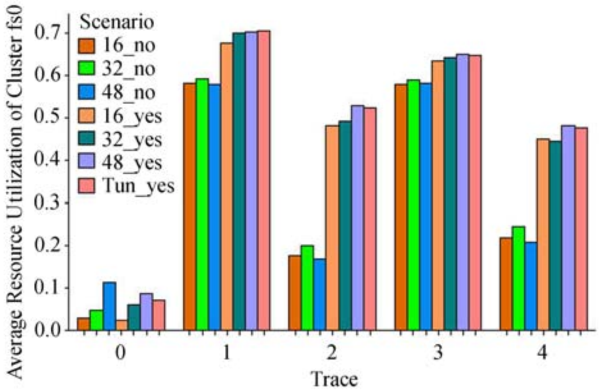

Fig.5. Average *resource effective utilization* of cluster fs0 in various scenarios.

With the queue wait time prediction, the average *resource effective utilization* improves as well for all clusters. As shown in Fig.5, the average *resource effective utilization* of cluster fs0 is better with queue wait time awareness than without. The observation holds true for other clusters as well. One may note that the average *resource effective utilization* is extremely low in all scenarios with trace 0 simply because there is no background workload at all. As traces 1 and 3 present intensive workload on cluster fs0, it is relatively high for all scenarios.

4.2 Cumulative Distribution Analysis

We further study how these performance metrics distribute in the experiment. For this purposes, the cumulative distribution function (CDF) figures of the *makespan* and the *queue wait time* are created for traces 0 and 4, as shown in Fig.6. Trace 0 is a special case with no other workload involved, and trace 4 is a representative one for real workload traces with intensive background workload.

It is noticed that all CDF curves fit well with linear form, indicating the validity of evaluating overall performance by the average values of corresponding performance metrics in Subsection 3.2. For trace 0 and trace 4, 99% of average *makespan* are significantly reduced compared with non predictable scenarios.

Fig.6 once again shows that, with intensive dynamic workload, the queue wait time prediction helps reduce the *queue wait time* drastically which results in very big improvement on *makespan* at relatively very low cost of extra *data transfer time*.

These CDF figures also help better understand how tunable resource requirement impacts scheduling. Even though the Tun_yes is not the best performer in all cases with respect to average *makespan*, it actually leads in steepness of cumulative distribution curves, evidently in Figs. 6(a) and 6(c). If we take out the lower 10% and upper 10% of the test cases to minimize possible abnormality in simulation, Tun_yes is actually the steepest scenario with respect to *makespan* and *queue wait time* in both traces, as shown in Fig.6. As all prediction-enabled scenarios perform closely if measured by average *makespan*, the steepness actually places Tun_yes as a leading performer in terms of *makespan* and *queue wait time*.

Furthermore, we try to understand how CCR impacts the scheduling effectiveness. Fig.7 shows that in terms of the average *makespan* the proposed algorithm

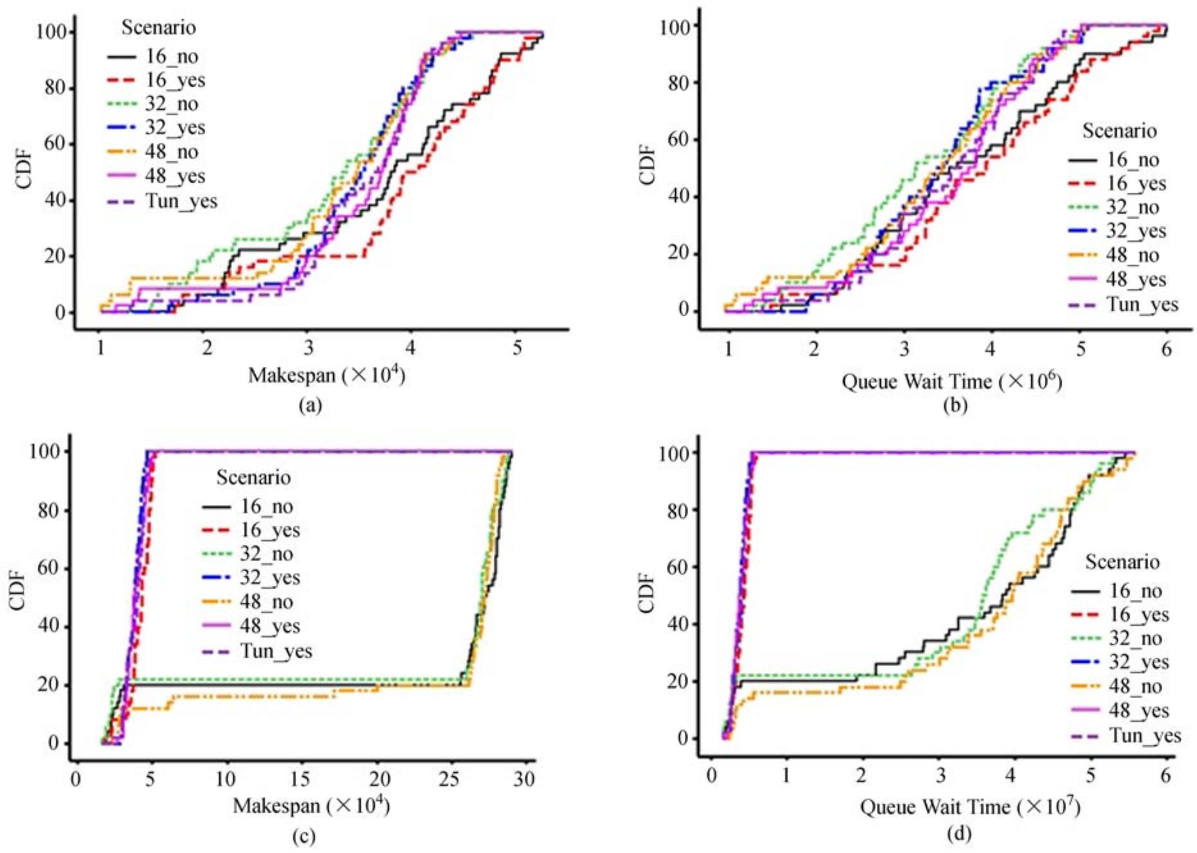

Fig.6. CDF of *makespan* and *queue wait time* with traces 0 and 4. (a) CDF of *makespan* with trace 0. (b) CDF of *queue wait time* with trace 0. (c) CDF of *makespan* with trace 4. (d) CDF of *queue wait time* with trace 4.

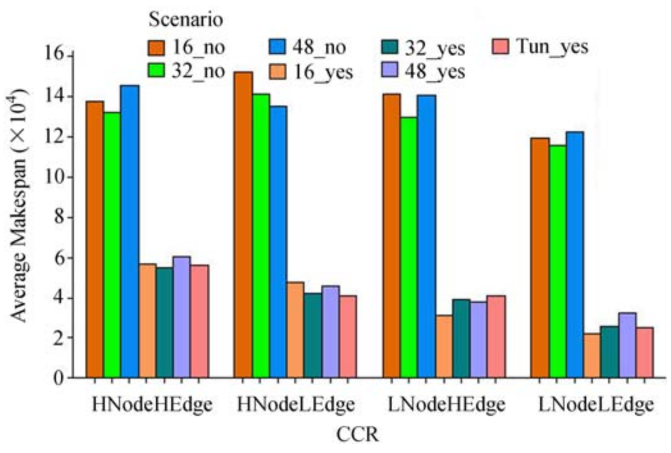

Fig.7. Average *makespan* vs. Node- and Edge-weight ratio in various scenarios.

performs noticeably best with scenario Tun_yes when CCR is high (HNodeLEdge, the 2nd case), i.e., computation intensive, and worst when CCR is low (LNodeHEdge, the 3rd case), compared with other scenarios. This indicates that when a workflow has high CCR, the *Planner* intends to request more processors. The observation also suggests that the resource requirement specification should consider the workflow characteristics, e.g., CCR, intelligently.

4.3 Discussion of Tunable Requirements

It is interestingly observed that Tun_yes is not the best performer as we thought. It actually has similar performance as 16_yes, 32_yes and 48_yes. We further study how Tun_yes requests the number of processors when it has options of 16, 32 or 48 processors. Table 3 shows that the algorithm requests 48 processors at most of time. We believe this partially attributes to nearsightedness of dynamic scheduling, which always makes local decisions. On the other hand, we admit that the combination of 16, 32 and 48 is only an educated guess. Ideally, the resource requirements of a

Table 3. Distribution of Processor Number Request

Trace	Request for 16 Processors (%)	Request for 32 Processors (%)	Request for 48 Processors (%)
Trace 1	8.2	19.5	72.2
Trace 2	1.3	5.6	93.1
Trace 3	8.6	18.5	72.9
Trace 4	1.0	1.9	97.1

workflow are determined specifically by its application characteristics. However, the experiment here is performed with DAGs randomly generated and we use one set of resource options for all cases. In reality, scientific workflows are well studied and the recently emerging studies, such as [13], will help generate "best" resource specifications.

The experimental results conclude that queue wait time awareness is crucial for scheduling workflows in a multicluster environment and the proposed algorithm performs extremely well with prediction awareness. It also suggests that future progress on how to better generate workflow resource requirements can further improve the proposed algorithm.

5 Related Work

While most DAG scheduling heuristics in literature assume that each computational resource is a processing node of a single processor, the study[8] demonstrates that if the resource is actually a cluster with multiple processing nodes, this assumption will misperceive the jobs' execution time and execution order which results in the scheduler inefficiency. An empirical study[13] is performed on automatically generating resource specification for workflow applications in order to get the "best" resource collection. But the work is still at an early stage.

How to dynamically schedule and distribute load in a local distribution system is not new, but these works[20-21] primarily focus on independent workload in a homogeneous environment. The concept can apply to workflow scheduling directly only for the pool of jobs ready to execute if we consider they are independent. However in a heterogenous environment, we have to ensure that in a workflow the job with higher priority, determined by its job dependence and critical path, being scheduled to a better resource to achieve minimal overall makespan of the entire workflow.

Recently, RePA[22] is proposed as a dynamic scheduling algorithm to reduce the communication and redistribution costs by mapping child jobs to processors which are assigned to parent jobs (reuse processors). However, the algorithm makes an unrealistic assumption that jobs are always mapped to a single cluster. Another algorithm, DMHEFT[23], uses "postponing" strategy to schedule a workflow onto a cluster of clusters. The heuristic postpone some ready to execute jobs if it thinks more resource will be available after a certain mount of wait time. However this heuristic does not consider that a job submitted to a cluster has to compete with other jobs already in the queue.

To consider the resource competition in the real world scenario, it is crucial to be able to predict queue wait time for a given job. A prediction approach named QBETS[3] is recently developed and deployed on more than a dozen super computing sites, offering on-line queue delay predictions for individual jobs. The study[24] evaluates a static scheduler by incorporating the queue wait time prediction into the process of generating a static "plan". However, it does not consider that the local workload is dynamic and it is impossible to foresee the future workload when the plan is generated.

6 Conclusions

To answer the challenges of scheduling workflows in a multicluster environment, the proposed algorithm incorporates the queue wait time awareness and tunability of resource requirements. It allows dispatching jobs to multiple clusters to utilize extra resources while balancing the queue wait time, possible performance gain and possible data movement cost to achieve minimal overall workflow *makespan*. A comprehensive simulation based experiment is conducted with a published workflow application test benchmark and real workload traces. The evaluation results show that the proposed queue wait time aware algorithm improves workflow performance significantly in terms of average *makespan* with relatively low cost of data movement. However, data movement is only measured in terms of time duration in this paper. It actually also consumes other network resources. As data movement is not trivial for data intensive workflows, its impact on resource consumption needs further study.

References

[1] NSF teragrid. http://www.teragrid.org/.

[2] Li H, Groep D, Wolters L. Workload characteristics of a multi-cluster supercomputer. In *Proc. the 10th International Workshop on Job Scheduling Strategies for Parallel Processing* (*JSSPP 2004*), New York, USA, June 13, 2004, pp.176-193.

[3] Nurmi D, Brevik J, Wolski R. QBETS: Queue bounds estimation from time series. In *Proc. the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (*SIGMETRICS 2007*), San Diego, USA, June 12-16, 2007, pp.379-380.

[4] QBETS web service. http://spinner.cs.ucsb.edu/batchq/.

[5] Aida K, Casanova H. Scheduling mixed-parallel applications with advance reservations. In *Proc. the 17th International Symposium on High-Performance Distributed Computing* (*HPDC 2008*), Boston, USA, June 23-27, 2008, pp.65-74.

[6] N'Takpe T, Suter F, Casanova H. A comparison of scheduling approaches for mixed-parallel applications on heterogeneous platforms. In *Proc. the Sixth International Symposium on Parallel and Distributed Computing* (*ISPDC 2007*), Hagenberg, Austria, July 5-8, 2007, p.35.

[7] Rauber T, Runger G. M-task-programming for heterogeneous systems and grid environments. In *Proc. 19th International Parallel and Distributed Processing Symposium* (*IPDPS 2005*), Los Alamitos, USA, April 4-8, 2005, p.178b.

[8] He L, Jarvis S, Spooner D, Nudd G. Performance evaluation of scheduling applications with DAG topologies on multiclusters with independent local schedulers. In *Proc. the 20th International Parallel and Distributed Processing Symposium* (*IPDPS 2006*), Rhodes Island, Greece, April 25-29, 2006, pp.8-15.

[9] Yu Z, Shi W. An adaptive rescheduling strategy for grid workflow applications. In *Proc. the 21st International Parallel and Distributed Processing Symposium* (*IPDPS 2007*), Long Beach, USA, March 26-30, 2007, p.115.

[10] Yu Z, Shi W. A planner-guided scheduling strategy for multiple workflow applications. In *Proc. the Fourth International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems, in conjunction with ICPP 2008*, Portland, USA, Sept. 8-12, 2008, pp.1-8.

[11] Topcuouglu H, Hariri S, Wu M. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distribution Systems*, 2002, 13(3): 260-274.

[12] The Globus Alliance. http://www.globus.org/.

[13] Huang R, Casanova H, Chien A. Automatic resource specification generation for resource selection. In *Proc. the 2007 ACM/IEEE Conference on Supercomputing* (*SC 2007*), Reno, USA, Nov. 10-16, 2007, pp.1-11.

[14] Hill M, Marty M. Amdahl's law in the multicore era. *Computer*, 2008, 41(7): 33-38.

[15] Hönig U, Schiffmann W. A comprehensive test bench for the evaluation of scheduling heuristics. In *Proc. the 16th International Conference on Parallel and Distributed Computing and Systems* (*PDCS 2004*), Cambridge, USA, Nov. 9-11, 2004, pp.437-442.

[16] Sulistio A, Cibej U, Venugopal S, Robic B, Buyya R. A toolkit for modelling and simulating data grids: An extension to gridsim. *Concurr. Comput.: Pract. Exper.*, 2008, 20(13): 1591-1609.

[17] Portable Batch System. http://www.openpbs.org, Dec. 2008.

[18] Sabin G, Kettimuthu R, Rajan A, Sadayappan P. Scheduling of parallel jobs in a heterogeneous multi-site environment. In *the 9th International Workshop of Job Scheduling Strategies for Parallel Processing* (*JSSPP 2003*), Seattle, USA, June 24, 2003, pp.87-104.

[19] Yu Z. Toward practical multi-workflow scheduling in cluster and grid environments [Ph.D. Dissertation]. Wayne State University, 2008.

[20] Shivaratri N, Krueger P, Singhal M. Load distributing for locally distributed systems. *Computer*, 1992, 25(12): 33-44.

[21] Maheswaran M, Ali S, Siegel H, Hensgen D, Freund R. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.*, 1999, 59(2): 107-131.

[22] Hunold S, Rauber T, Runger G. Dynamic scheduling of multiprocessor tasks on clusters of clusters. In *Proc. 2007 IEEE International Conference on Cluster Computing*, Austin, USA, Sept. 17-21, 2007, pp.507-514.

[23] Hunold S, Rauber T, Suter F. Scheduling dynamic workflows onto clusters of clusters using postponing. In *Proc. the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid* (*CCGRID 2008*), Lyon, France, May 19-22, 2008, pp.669-674.

[24] Nurmi D, Mandal A, Brevik J, Koelbel C, Wolski R, Kennedy K. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *Proc. the 2006 ACM/IEEE Conference on Supercomputing* (*SC 2006*), Tampa, USA, Nov. 11-17, 2006, Article No.119.

Zhi-Feng Yu received the B.S. degree in applied mathematics and M.S. degree in economics and management from Tongji University, Shanghai, China, in 1990 and 1993 respectively and the Ph.D. degree in computer science from Wayne State University, Detroit, USA in 2009. He is currently Manager of Application Development and Integration of T-Systems North America, Rochester Hills, MI. His research interests include computer systems and enterprise computing.

Wei-Song Shi is an associate professor of computer science at Wayne State University. He received his B.S. degree from Xidian University in 1995, and Ph.D. degree from the Chinese Academy of Sciences in 2000, both in computer engineering. His current research focuses on mobile computing, distributed systems and high performance computing. Dr. Shi has published more than 80 peer-reviewed journal and conference papers in these areas. He is the author of the book "Performance Optimization of Software Distributed Shared Memory Systems" (Higher Education Press, 2004). He has also served on technical program committees of several international conferences, including WWW, ICPP, MASS. He is a recipient of Microsoft Fellowship in 1999, the President Outstanding Award of the Chinese Academy of Sciences in 2000, one of 100 Outstanding Ph.D. Dissertations (China) in 2002, "Faculty Research Award" of Wayne State University in 2004 and 2005, the "Best Paper Award" of ICWE'04 and IPDPS'05. He is a recipient of the NSF CAREER award and Wayne State University Career Development Chair award.