Design of a Dynamic Scheduling Engine of Grid Workflow

Li Xiang-ning
Laboratory of Electronic Equipment Structure
Xidian University
Xi'an, China
e-mail: zklieu@126.com

Yang Guang-ping, Gu Lin-ying
School of Computer Science & Technology
Xidian University
Xi'an, China
e-mail: 253557425@qq.com, 297048124@qq.com

***Abstract*—Scheduling engine is an important component in the grid workflow system, which assign different job to specific resource node for execution according to job description information under appropriate policy constraints, so as to achieve optimal allocation of grid resources. Aiming at heterogeneous and dynamic of grid environment, a kind of dynamic scheduling method based on heuristic algorithm was proposed to schedule grid services. Based on the method, the hierarchical overall structure of Grid Workflow Scheduler was designed. Design in detail functional modules as job management, resource matching, policy management and scheduling engines in interface layer, management layer and service provider layer were also provided, which lays foundation for implementation of the system.**

Keywords-grid; workflow; dynamic scheduling; heuristic algorithm

I. Introduction

As the distributed computing environment of next generation, grid computing provides a new computing model to solve complex computing problems. Large-scale grid applications are usually divided into several interrelated activities, the complex constraint relationships among which can be seen as a workflow [1-3]. Directed Acyclic Graph (DAG) is a common description of the workflow approach, which is widely used in grid application fields as e-science, e-business and etc. Workflow scheduling is to select the branch of activity and to allocate appropriate resources to meet different scheduling objectives. However, the mapping of activities and resources is essentially a complex optimization problem, usually the NP-Hard problem [4]. Many literatures have proposed heuristic scheduling algorithm based on minimum completion time and many grid workflow management system also take minimum completion time as scheduling target. The paper also took minimum completion time as the scheduling objective. A kind of grid service workflow scheduling algorithm based on genetic algorithm was proposed in [5] and another scheduling algorithm that combines genetic algorithm and simulated annealing algorithm was presented in [6]. These algorithms all considered dependencies between tasks in grid workflow and can be scheduled well planned, but these algorithms are static scheduling, does not taking into account the characteristics of heterogeneity and dynamic. This paper presented a dynamic scheduling algorithm, and designed the grid workflow scheduler based on the proposed scheduling algorithm. The paper is organized as follows: Section 2 gives grid workflow model and proposes dynamic scheduling algorithm based on heuristic algorithm; Section 3 presents hierarchical structure of Grid Workflow Scheduler; Section 4 designs main functional modules of each layer in detail; Section 5 concludes our work.

II. Grid Workflow Scheduling Model and Algorithm

A. Grid Workflow Model

A grid service workflow application can be expressed as a Directed Acyclic Graph (DAG). Define Γ as limited job collection $T_i(1 \le i \le n)$ in the workflow and Λ as the set of directed edges, such as (T_i, T_j) as a directed edge. Then we call T_i as the parent job of T_j and T_j as the child job of T_i. Suppose that a child job can not be executed before completion of all its parent jobs, such kind of grid workflow can be described as a pair $\Omega(\Gamma,\Lambda)$. Fig. 1 shows a grid workflow that represented with DAG, where there are 8 jobs of T_0 to T_7 need to be scheduled and executed.

In a workflow, the execution order of the relevant tasks is decided by their dependencies, namely one job can only be executed after all the completion of his parent nodes. However, many non-related tasks, as shown in Fig. 1, T_3 and T_4 may be assigned to the same service node. The priority of parallel services will greatly affect on the performance of workflow execution. Based on this, the paper used two-dimensional string to represent the scheduling result as shown in Fig. 2.

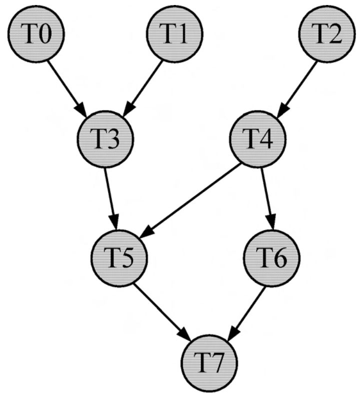

Figure 1. Workflow represented with DAG.

978-1-61284-486-2/11/$26.00 ©2011 IEEE

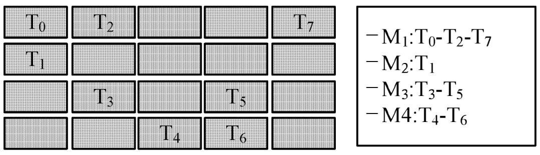

Figure 2. Scheduling plan.

B. Dynamic Scheduling Algorithm

Because the dynamic nature of grid environment, such as it is possible that at any moment resources join and exit, and for the reasons of internal or external resources performance can be constantly changing. In the grid environment, it is difficult to accurately estimate data transmission and execution time of each job in advance.

Most of the existing scheduling algorithms are static scheduling, which completes scheduling plan according to predictions before workflow execution and only schedule workflow one time. In the grid environment, static scheduling algorithm has the following disadvantages: (1) Accuracy of prediction. It is difficult to predict execution time and data transmission delay of child job in different resource, which is the key factor to determine static scheduling. (2) Adapt to dynamic environment. Most of the scheduling policies assume that resources in the grid are known and remain the same in the job execution process. In the static scheduling policy, once scheduling plan was developed, it can not be modified. (3) Separation of workflow scheduler and execution engine. These two shortcomings are due to lack of mutual communication between scheduler and execution engine. If the scheduler can timely access to dynamic information of grid resources and develop scheduling plan based on these dynamic information, and then submitted to the execution engine for implementation, and execution engine to execute the workflow state of timely feedback to the scheduler, such as a task to perform ultra or fail, so that not only increases the probability of a successful process execution, process execution time will be significantly reduced.

This paper presented a dynamic scheduling on the workflow, the idea of which is based on the current status of resources changes and events that trigger the execution engine to generate new scheduling program to ensure the smooth implementation of the workflow to complete, and to shorten execution time. Scheduling algorithm is as follows:

```
T-set of the jobs in the DAG
R-set of all available resources
P-performance estimation matrix
H-Heuristic employed by scheduler
S-Scheduler
1. set initial scheduler $S_0$=null
2. while ($s_0$ is empty OR grid services change AND DAG not finished) do
    #R is updated via the communication with Resource Manager
    3. update Resource Set R
    4. update Performance History Repository
    #Predicator component will update performance estimation matrix P
    5. call P=estimation (T, R)
    #New scheduler is made by applying the heuristics H on execution status snapshot of $S_0$ and P
    6. call $S_1$=scheduler ($S_0$, P, H)
    7. if ($S_0$==null OR $S_0$.makespan >$S_1$.makespan)
        8. $S_0$=$S_1$
        9. Submit $S_0$
    10. end if
11. end while
```

III. Overall Design of System

Grid workflow scheduler provides structure for the workflow engine to achieve dynamic binding and run of service. The architecture of scheduler is shown in Fig. 3, which based on the idea of hierarchical design. The advantages includes: (1) to reduce dependence between layers; (2) easy to replace new implementation with the original implementation, which depends on the level of the module is transparent; (3) in favor of standardization; (4) in favor of re-use of each logic; (5) developers can focus only one layer of the entire system.

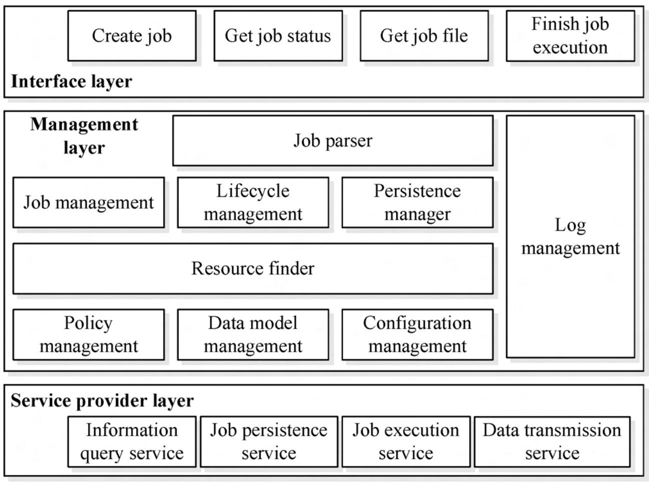

Figure 3. System architecture.

According to functional requirements, the scheduler is divided into the following specific levels:

(1) Interface layer. Scheduler provides service for workflow execution engine and achieves dynamic binding of services. It can also provide interface of grid job generation, grid job execution status query, grid job description file generation and terminate executing grid job.

(2) Management layer. It is the core layer of scheduler, the main components of which includes JSDL parser module, job management module, scheduling engine, policy management module, data model management module and configuration management module.

(3) Service provider layer. It includes many Service Provider Interface (SPI), each service provider can implement interface to provide appropriate service. The services provided here is to provide necessary functions support for the needed operations of management layer, such as data transmission SPI needed for data transmission, resource access SPI needed for resource access and matching operations, job execution SPI needed for job execution and etc.

IV. Detailed Design of Functional Modules

A. *Job Management Module*

(1) Job lifecycle management

As soon as the job that submitted by user to scheduler be created as Job object, the lifecycle of job began. The state transition of job is shown in Fig. 4.

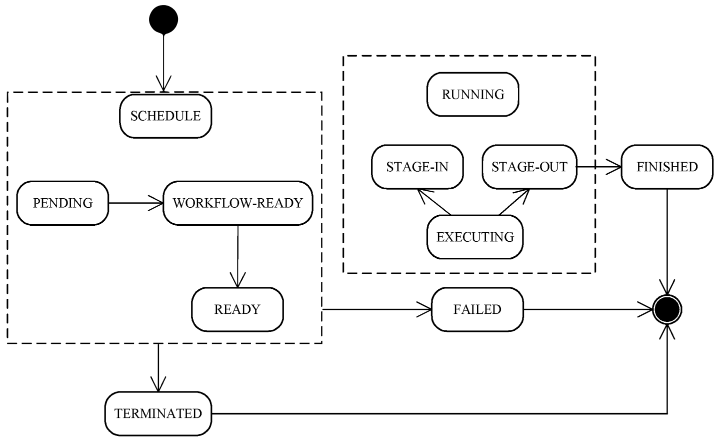

Figure 4. Job state transition diagram.

The job state is mainly divided into three stages: SCHEDULE, RUNNING and END. Each stage has its own child states. The child states of SCHEDULE include PENDING to wait for resource allocation, MATCHMAKING for resource matching and READY. The child states of RUNNING include STAGE-IN parameter input, EXECUTING for job execution and STAGE-OUT for execution result output. The child states of END include TERMINATED, FAILED and FINISHED.

(2) Job queue management

In the Grid Workflow Scheduler, there are three job queues: (1) pending queue, which is a thread-safe priority queue to store job of SHEDULE: PENDING state; (2) matchmaking queue, which is a thread-safe priority queue to store job of SCHEDULE: WORKFLOW-READY state; (3) ready queue, which is as thread-safe queue to store job of SCHEDULE:READY state.

The transition of job in the above queues is mainly the following two types:

(1) JobBuilder thread in Grid Workflow Scheduler created job object and put it in Pending Queue, and then Matchmaker thread in the resource search engine thread pool will take job from Pending Queue to perform resource matching, namely to add related information of appropriate executable node on job. After matching, scheduler put job in Workflow Matchmaking Queue.

(2) In case of scheduling events, the scheduler will firstly check Workflow Matchmaking Queue and move all entire child jobs in the Workflow into Ready Queue, while the remaining Workflow child job that has not complete searching will still remain in the Workflow Matchmaking Queue.

B. *Resource Matching Module*

The main function of Resource matcher is to check all executable node that can execute specific job and add them into job object as candidate resource, so as to provide basic data for scheduling operation of scheduling engine.

(1) Information service SPI

It mainly defines an interface specification. Aiming at different information service type, there are different implementation classes. The mechanism greatly improves scalability of scheduler system.

(2) Resource matching thread pool

Resource matching call information service SPI to query job resource. In order to reuse resource matching thread and reduce the cost of thread initialization, the scheduler use CachedThreadPool in Java 5.0 Concurrency Utility for thread pool display. In the program execution process, CachedThreadPool will create threads has same number as needed and dynamic adjust thread number. If the job was submitted too fast, new resource matching thread will dynamic created. If the idle time of some thread exceeds some threshold, it will be recovered by thread pool. The dynamically adjust ability can utilize scheduler system resource more reasonably, so as to improve system performance.

The flow of resource matching call information service to match candidate resource for job is as follows:

Step 1: Resource matching thread get job object from Pending queue. If the Pending queue is empty, the resource thread is blocked;

Step 2: Modify job state as SCHEDULE: MATCHMAKING;

Step 3: Traverse all information services supported by scheduler and add queried candidate resource to candidate resource list of job;

Step 4: If the found candidate resource after traversing is empty, modify job state as SCHEDULE: PENDING and re-insert it into Pending queue, otherwise go to Step 5;

Step 5: Order candidate resource list of job according to configured priority policy;

Step 6: Modify job state as SCHEDULE: WORKFLOW-READY;

Step 7: If all job of a workflow completed resource matching and the WORKFLOW-READY queue is not full, the job is insert into queue, otherwise the resource matching thread is blocked;

Step 8: Modify job state to SCHEDULE: READY;

Step 9: If READY queue is not full, insert job into it, otherwise the resource matching thread blocked.

C. *Policy Management*

It realized multiple scheduling algorithms of MinMin, MaxMin and GA and determine default scheduling algorithm of scheduler according to experiments. The user can specify scheduling algorithm.

D. *Scheduling Engine Module*

After scheduling events, the scheduling thread was activated. The scheduling flow is shown in Fig. 5.

Step 1: Firstly traverse MatchMaking queue to check if there is completed Workflow-task, if it exist, go to Step 2, otherwise go to Step 3;

Step 2: Move all completed Workflow-task in MatchMaking queue to Ready queue;

Step 3: Clear Ready queue and create to be scheduled array with all job objects in the queue;

Step 4: If the to be scheduled array is empty, the scheduling ends and waits or next scheduling event; otherwise go to Step 5;

Step 6: Create policy object from policy factory and set data model for it;

Step 7: Call scheduling algorithm for policy object and generate scheduling plan;

Step 8: Schedule job one by one according to scheduling plan and modify state of scheduled job to RUNNING.

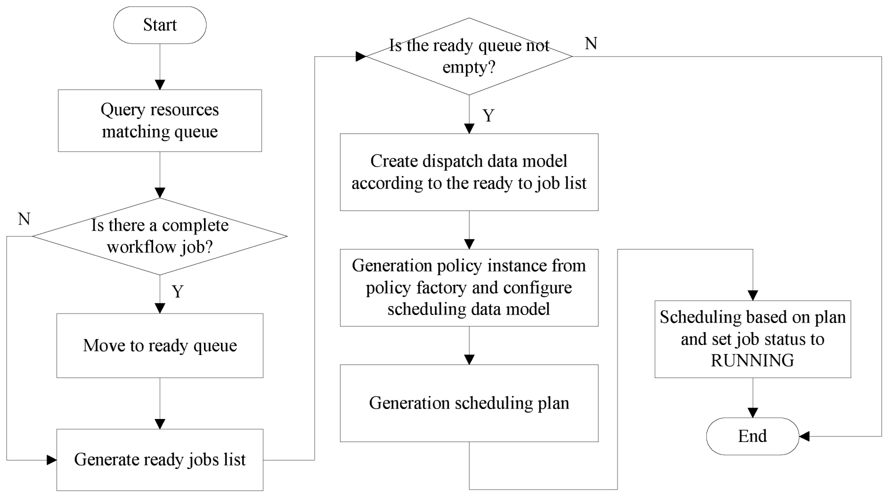
Start
Query resources matching queue
Is there a complete workflow job?
N
Y
Move to ready queue
Generate ready jobs list
Is the ready queue not empty?
N
Y
Create dispatch data model according to the ready to job list
Generation policy instance from policy factory and configure scheduling data model
Generation scheduling plan
Scheduling based on plan and set job status to RUNNING
End

Figure 5. Scheduling flow.

V. CONCLUSION

Workflow scheduling aims at select branch for activities and allocate appropriate resources, and to meet different scheduling target. Grid workflow management also takes he minimum completion time as scheduling target. Aiming at heterogeneous and dynamic of grid environment, a kind of dynamic scheduling method based on heuristic algorithm was proposed to schedule grid services. Based on the method, the hierarchical overall structure of Grid Workflow Scheduler was designed. Design in detail functional modules as job management, resource matching, policy management and scheduling engines in interface layer, management layer and service provider layer were also provided, which lays foundation for implementation of the system.

REFERENCES

[1] Foster I., KessmanC. and Nick J. M., “Grid Service for Distributed System Integration,” IEEE Computer, vol.35, 2002, pp. 37-46.

[2] Deelman E., Blythe J. and Gil Y., “Mapping Abstract Complex Workflows onto Grid Environments,” Journal of Grid Computing, vol. 1, 2003, pp. 25-39.

[3] Yu Jia and Buyya R., “A Taxonomy of Scientific Workflow Systems for Grid Computing,” ACM SIGMOD Record, vol. 34, 2005, pp. 44-49.

[4] Yu Jia and Buyya R., “Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms,” Scientific Programming Journal, vol. 14, 2005, pp. 217-230.

[5] Guo W. C. and Yang Y., “Scheduling of Grid Workflow for Grid Services Based on Genetic Algorithm,” Computer Applications, vol. 26, 2006, pp. 54-56.

[6] Ding Y. m. and Sun R. Z., “Scheduling of Grid Workflow for Grid Services Based on Genetic Simulated Annealing Algorithm,” Computer Applications, vol. 27, 2007, pp. 89-91.