Integrating Multiclusters for Efficient Application Execution
Zhifeng Yun, Zhou Lei, Daniel S. Katz, Jagannathan Ramanujam, Gabrielle Allen, Tevfik Kosar, Shantenu Jha
Center for Computation & Technology, Louisiana State University

I. PELECANUS OVERVIEW

Fig. 1. Pelecanus architecture.

The design goal of Pelecanus is to reduce execution time, improve application monitoring, and increase resource utilization in multiclusters grid environment. It adopts the DA-TC [2] model so that it can employ dynamic load balancing policy for application execution. Pelecanus is designed to execute in the user space, therefore there is no specific requirement for system configuration and software installation on participating clusters.

The architecture of Pelecanus is demonstrated in Figure 1. It is built on the top of local cluster schedulers, playing key roles in resource management, workflow control, user interface, and high availability. The Grid Execution Management Service (GEMS) engine provides the core services, including cluster interoperability, workflow management, and execution implementation, based on the DA-TC execution model. There are three interfaces: standalone Graphic User Interface (GUI), Web Interface (i.e., grid portal), and High Availability (HA) Interface. This architecture employs web server and HA server for web user access and reliability improvement, respectively.

The functionality of Pelecanus has five major perspectives: 1) Security service. A single entry point is provided for a user to access clusters and grid services, based on various existing technologies on credential and access management; 2) Resource monitoring and discovery. Grid monitoring and discovery technologies, such as Globus [1] trigger service, Ganglia [3] cluster toolkit, and MonALISA [4], are evaluated and integrated into the Pelecanus; 3) Application specification. A user specifies application executable, resource requirement, dataset location, and which execution pattern will be adopted by an application. There is an application workflow template for each execution pattern; 4) Application monitoring and steering. Once an application is submitted, a user is able to monitor execution progress of the whole application and any particular component on a cluster. Meanwhile, the user is allowed to steer the workflow to adapt runtime status of resources and execution progress. 5) Data management. Data manipulation is provided to support application deployment on remote clusters and massive input dataset operation.

An application researcher accesses the services provided by the Pelecanus via standalone GUI or web interface (i.e., grid portal). The first thing to run an application is to obtain authorization and authentication of available resources. Then, the user needs to specify which execution pattern the application will adopt. The Pelecanus provides corresponding application specification interface for each execution pattern, including executable location, data source, and workflow description. The GEMS engine, the core of the Pelecanus, intelligently manages application components and submits them onto proper clusters. Meanwhile, the user transparently monitors execution progress. A user will be allowed to steer application execution by re-organizing application workflow under dynamic load balance strategies adopted by GEMS engine.

II. EXECUTION MODEL

The GEMS engine is based on Dynamic Assignment with Task Container concept. It is designed to improve application execution in a multi-cluster grid environment. In the DA-TC circumstance, an application consists of a number of tasks. It is also assumed that there is no inter-task communication.

There are two components adopted to execute an application: application execution agent (AEA) and task containers (TC). AEA is the local gateway to the Grid. It is in charge of deploying and submitting task containers to the participating clusters; monitoring container status; dynamically orchestrating workflow and assigning application tasks; steering application, task, and container executions; etc. An application execution typically employs multiple TCs, depending on user configuration. A TC is submitted to a participating cluster...
as a normal job, waiting for scheduling by local resource management system. Once a TC obtains resource allocation, it is used to execute the tasks assigned dynamically by AEA. TCs take responsibilities of holding resources until application execution accomplished, task stage in/out, invocation, status monitoring, communication with AEA, etc.

Figure 2 shows the interaction diagram between AEA and TC. To carry out an application execution, the first thing for AEA to do is to submit TCs to participating clusters. The submitted TCs are placed as normal jobs at the end of the scheduling queues on participating clusters, waiting for resource allocation by local resource management systems. One participating cluster may host multiple task containers, according to different load balancing strategies adopted by AEA. After a TC obtains the required computing resources from a local scheduling system, it communicates with AEA for task assignment. First, the TC sends AEA a message to claim that it is ready to run a task. Second, AEA updates TC status table and then a task (or more) is selected, based on application workflow management strategies. Third, task stage in, execution, and stage out are performed, and the status tables associated with tasks and TCs on AEA are updated. After a task is completed successfully, AEA and TC are ready for the execution of next task.

![Interaction Diagram](image)

**Fig. 2.** The interaction diagram between AEA and TC. “R” denotes running and “Q” queuing. “Other” delegates the jobs submitted by other users.

This execution model can significantly reduce execution turnaround time, since dynamically assigned tasks can get immediate execution once a task container get the needed resource. Besides, it can improve resource utilization since *fast* clusters take more responsibility of task execution. Although *slow* clusters take less work for the total task execution, it can still make contribution, instead of bottleneck in normal execution way.

### III. Experimental Evaluation

Without lack of generality, our multicluster Grid testbed consists of five Linux clusters with local scheduling system PBS, connected by the Internet. Each cluster can be accessed by Globus GRAM or SSH.

Turnaround time of application execution is investigated as performance metric. Two execution ways are compared. The one is the traditional way, by which tasks are directly assigned to the participating clusters, and the number of tasks on a cluster is based on its relative speed and its number of CPUs. The other execution way is based on the DA-TC execution model.

![Experimental Results](image)

**Fig. 3.** Experimental results to compare turnaround time of automatic history matching under two different submission strategies: DA-TC and traditional way.

Figure 3 demonstrates the experimental results to compare the turnaround time under two different submission strategies: DA-TC and traditional way. We observe that the turnaround time increases much faster under traditional way than under DA-TC when the number of iterations increases. The major reason is that task containers in DA-TC never release the resources until the whole automatic history matching process accomplished, but for traditional way, tasks in each iteration are submitted to the end of the local scheduling queues. In the DA-TC, all runs in a container has only one queue wait and dynamic load balancing also speeds up the execution.

### REFERENCES


