CHAPTER 2

BACKGROUND AND RELATED WORKS

This chapter is divided into two main parts namely Background and Related Works. These topics are described as follows.

2.1 Background

In order to have common understanding, theoretical background used in this research are briefly explained here. There are 4 topics described in this part which are Job Scheduling on a Cluster Computer, Parameter-Sweep Applications and Scheduling, Workflow and Directed Acyclic Graph, and Workflow Scheduling.

2.1.1 Job Scheduling on a Cluster Computer

Cluster is a collection of computing and storage resources connected with a high speed network. The most important concept of cluster is Single System Image (SSI). That mans a group of physically distributed computers are used as a single computer.

The cluster middleware is the most important component because it provides SSI mechanisms. It includes cluster file system such as Network File System (NFS) and Parallel Virtual File System (PVFS), programming environment such as Message Passing Interface (MPI), cluster-enabled Java Virtual Machine, job-manager and scheduling systems such as Condor [6], Sun Gird Engine (SGE) [7], and Portable Batch System (PBS) [8]. Currently, there are many software tools for building and managing a cluster; for example, Rocks [4], Oscar [5], etc.

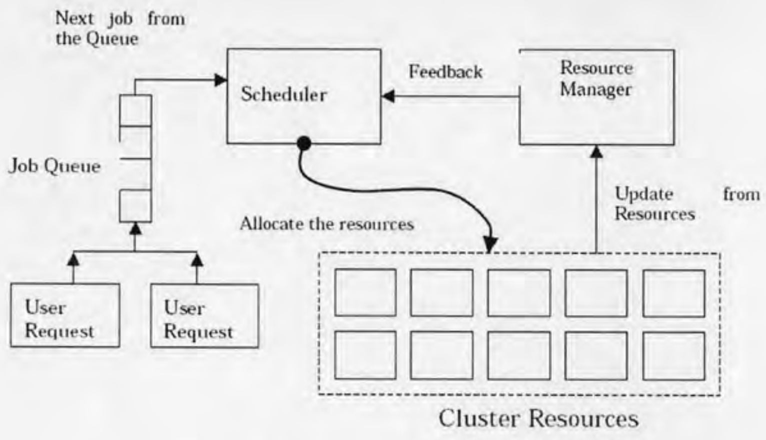


Figure 2-1 The scheduling system

In Figure 2-1, the execution on a cluster can be started in the scheduling system. When jobs or applications are submitted to the scheduling system, they are waiting in the job queue. A scheduler gets a job from the queue and sends it to an available compute node. When the execution is completed, the scheduler will be informed.

2.1.2 Parameter-Sweep Applications and Scheduling

A parameter-sweep application is an application structured as a set of multiple experiments, each of which is executed with a distinct set of parameters. There are many important parameter-sweep application areas, including bioinformatics, operations research, data mining, business model simulation, massive searches, Monte Carlo simulations, network simulation, electronic CAD, ecological modeling, fractals calculations, and image manipulation. Such an application consists of a set of independent coarse-grained tasks such that each task corresponds to computation for a set of parameters.

2.1.3 Workflow and Directed Acyclic Graph

A collection of jobs, called a workflow, is an orchestration of jobs that performs a complicated task. A workflow enables the structuring of an application as a directed acyclic graph (DAG) where each node represents a constituent task and edges represent inter-task dependencies of the application.

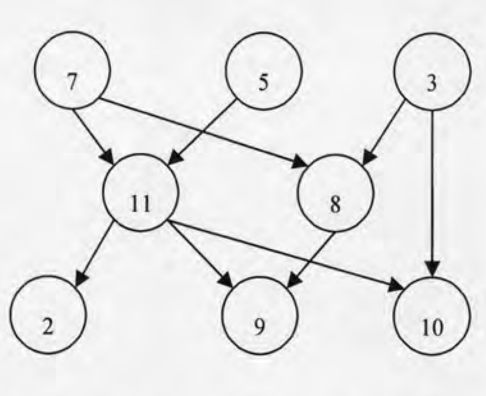


Figure 2-2 A simple directed acyclic graph

In Figure 2-2, DAGs can be considered as a generalization of trees in which certain subtrees can be shared by different parts of the tree.

2.1.4 Workflow Scheduling

Basically workflow scheduling is the process of scheduling individual tasks in a workflow. Additional complexity lies in the conditions that need to be satisfied including dependency among tasks, data transfer, and control over a workflow as a whole.

A workflow scheduler generally places individual task into a waiting queue with dependency conditions. When a processor is available, a task with satisfied dependency conditions is selected from the queue. The selection method depends on the scheduling policy by used. Usually the scheduling policies for single-task scheduling can be applied, for example, First In First Out, Shortest job First, and Fair Share.

2.2 Related Works

Nowadays, there are a lot of schedulers for distributing jobs into cluster. In this part, 3 types of schedulers are described which are batch schedulers, workflow schedulers, and parameter-sweep schedulers. The details of these types are expressed as follows.

2.2.1 Batch Schedulers

A batch scheduler is a system that manages resources, which are the nodes of the cluster. It is in charge of accepting jobs submissions (which are sequential or parallel computation required by the users of the cluster) and schedule the execution of these jobs on resources it manages. Thus, the objective of a batch scheduler is to allow users to use resources easily: users should not have to worry neither about the availability of the nodes nor about the interference of their job with some other jobs. Popular batch schedulers include PBS, Torque, SGE, SQMS, and Condor.

A classical batch scheduler architecture is pictured in Figure 2-3. This architecture is made of the following main parts: a client part which is a module for jobs submission that validates the query and a server part made of a scheduling module associated with a resources matching mechanism and an execution module which controls the actual execution of jobs. In addition, a module is in charge of the logging as well of accounting of the system activity and another of the monitoring of the resources.

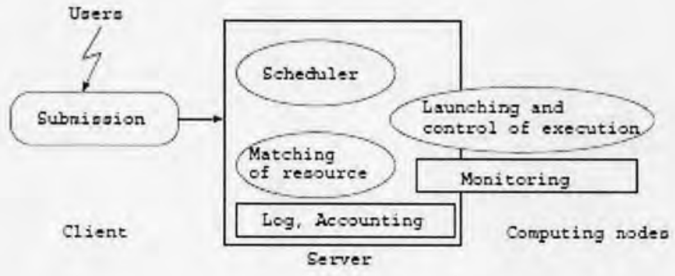


Figure 2-3 Classical batch scheduler architecture

For example, Condor is a scheduler for High-Throughput Computing (HTC) environment which is to efficiently harness the use of all available resources. Most batch schedulers operate on dedicated machines but Condor can operate with dedicated and non-dedicated machines. In case of non-dedicated machines, Condor utilizes those resources when they are idle.

2.2.2 Workflow Schedulers

There are many schedulers that can take a workflow as input and assign the subtasks onto separate nodes. Some common schedulers, such as Torque, can define dependency among tasks but do not fully support workflows.

- ***DAGMan***

When scheduling a workflow, Condor provides an efficient tool called Directed Acyclic Graph Manager (DAGMan). It represents a workflow or a set of jobs which have dependencies on one or more other jobs into DAG. The jobs and dependencies are described in DAGMan scripts. DAGMan interprets the script and submits jobs to Condor in the order represented by the DAG.

```
# Filename: diamond.dag
#
Job  A  A.condor
Job  B  B.condor
Job  C  C.condor
Job  D  D.condor
Script PRE  A top_pre.csh
Script PRE  B mid_pre.perl  $JOB
Script POST B mid_post.perl $JOB $RETURN
Script PRE  C mid_pre.perl  $JOB
Script POST C mid_post.perl $JOB $RETURN
Script PRE  D bot_pre.csh
PARENT A CHILD B C
PARENT B C CHILD D
Retry  C 3
```

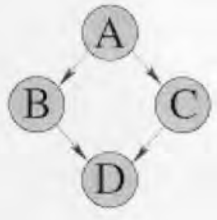


Figure 2-4(a) DAGMan script

Figure 2-4(b) Job dependency represented by DAG

Figure 2-4(a) is an example of a DAGMan script that describes the workflow in figure 2-4(b). The syntax of the description includes job, pre/post processing, and job dependency.

The job definition is a mapping between unique job name and executable program. This definition uses keyword *Job*. In the pre/post processing definition, they are usually used when a file has to be placed into staging area for the job to use and removed before computing a new job. This definition uses keyword *Script Pre* and *Post*.

When defining job dependency, parent job and child job are defined with keyword *Parent* and *Child*. The keyword is followed by one or more job names. The keyword Parent defines parent jobs and the keyword Child defines child jobs which execute after the parent job is finished.

However, Condor lacks the ability to manage workflows as a single entity.

- **TrellisDAG [9]**

The Trellis project aims to develop simple and effective techniques for forming a virtual supercomputer out of a geographically distributed collection of computing resource. TrellisDAG is designed to support any application with job dependencies and provides a framework for implementing different scheduling. However, the TrellisDAG focuses on the simple First-Come-First-Serve (FCFS) scheduling policy. It is able to define simple to complex workflows in DAG scripts. This system is implemented on Placeholder [10] which is a scheduler that provides automatic load balancing of computational jobs across different administrative domains. The TrellisDAG architecture consists of five components as listed below and as shown in the Figure 2-5.

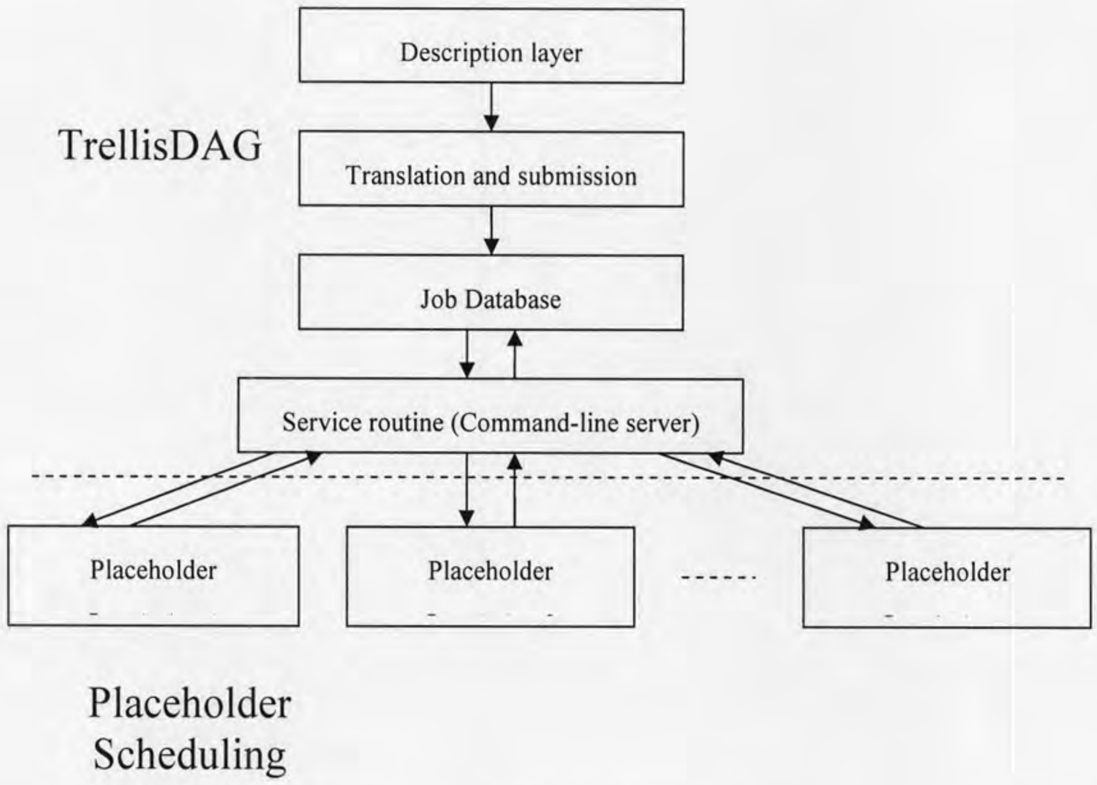


Figure 2-5 TrellisDAG architecture

TrellisDAG uses a concept of defining a workflow called Group model. The group model can define workflow dependency between groups. A group at the lowest level may either contain individual jobs which have no dependency or workflows. A group at a higher level is a supergroup. A supergroup may contain individual jobs, dependency between subgroups which is a group in at a lower level, and independent subgroups.

There are three ways for describing a workflow. The method depends on how complicated the workflow is. First, flat submission script is submitted workflow by mqsub utility. Second, using a Makefile because each rule in this Makefile can specifies workflow dependencies. Third, the DAG Description Script is designed to describe complicated workflow. This script is described with module coded using the Python scripting language. When submitting this script, it is transformed into Makefile and sends job information to job database for waiting for the placeholder to get a job to run.

There are many other interesting works that are less related to our work, including GrAD[11], Pegasus[12], GridAnt[17], and GridFlow[18].

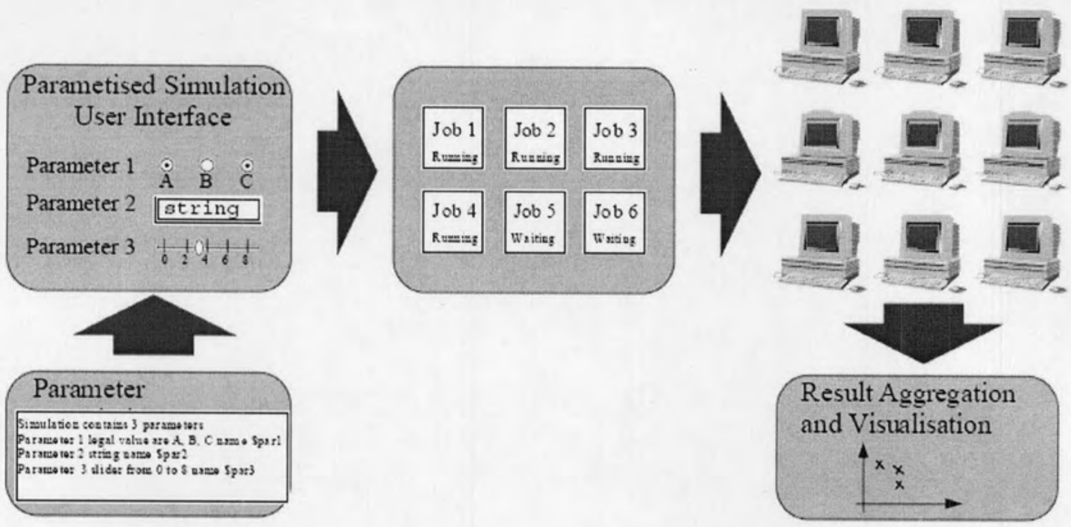
2.2.3 Parameter-sweep Schedulers

Parameter sweep applications require that a lot of similar jobs are submitted together. This may be difficult for the user of normal schedulers. There are some tools that specifically support parameter-sweep applications.

Nimrod [15] is a tool that combines the advantages of remote queue management systems with those of distributed applications. It provides the user with a problem oriented view of their application without requiring any modifications to their simulation code. After the user specifies the simulation parameters, Nimrod takes the cross product of these parameters and generates a job for each set. It then manages the distribution of the various jobs to machines, and organizes the aggregation of results.

By implementing log facilities Nimrod can be restarted at any time throughout an experiment. This attribute is important because a large modeling application may run for days. Over this time various workstations may fail, including the one executing Nimrod. It is also possible to run multiple copies of Nimrod concurrently, allowing simultaneous execution of experiments with different parameter settings, or even different modeling applications.

Rather than assuming a shared file system, Nimrod copies files between systems before and after execution of the program. Thus, when a job is started, the data files are copied to the target system. It is also possible to copy in the executable image for the application, thus it is not necessary to prepare the target system prior to use. When the program terminates, the output files are copied back to the host system.



Simulation contains 3 parameters
Parameter 1 legal value are A, B, C name $par1
Parameter 2 string name $par2
Parameter 3 slider from 0 to 8 name $par3

Figure 2-6 Overall structure of Nimrod