Dynamic Allocation of Servers to Jobs in a Grid Hosting Environment

Charles Kubicek, Mike Fisher, Paul McKee and Rob Smith

North East Regional e-Science Centre

Abstract

As computational resources become available for use over the Internet a requirement has emerged to reconfigure servers to an optimal allocation to cope with an unpredictable flow of incoming jobs. This paper describes an architecture that allows the dynamic reconfiguration of servers to process incoming jobs by switching servers between conceptual pools. The system makes use of heuristic policies to make close to optimal switching decisions. A prototype system which builds on existing resource management software has been developed to demonstrate some of the concepts described

1. Introduction

A resource hosting environment contains a large amount of computational resources which together serve requests for a number of hosted services [1]. Our work focuses on systems which offer services of different types, requiring dedicated servers. A server in this context is defined as a computational resource capable of processing jobs. Consider a collection of M (possibly distributed) servers partitioned into n subsets, such that subset i is dedicated to serving jobs of type i (i=1, 2, k). Examples of different job types are databases queries, long running simulations streamed media etc. To avoid ambiguity, we shall refer to the entire collection of servers as a "cluster", and to the dedicated subsets as "pools". If the allocation of servers to pools is fixed, changes in demand may cause some queues to become very long, while other servers remain under-utilised. This is clearly inefficient. It is desirable, therefore, to develop a policy whereby servers may be re-allocated from one type of service to another. That is the object of this paper.

A reconfiguration policy must take into account both the characteristics of demand (arrival rates and average service times of different types), and costs associated with keeping jobs waiting, and with server reallocation.

A method of dynamic server reconfiguration has been studied [2] in which servers are switched between services to cope with demand. Groups of servers or virtual clusters are monitored by a virtual cluster manager which makes decisions on when to switch a server from one virtual cluster to another. A request is made to add a new server to a virtual cluster when its queue has reached a given size, and the virtual cluster manger selects an idle server from another virtual cluster, or a pool of unconfigured machines to import. This approach only takes into account queue sizes for a service when making a decision to switch a server, but other factors are just as important. The time taken to run a job may differ between services, as may the time it takes for a server to be reconfigured. There may also be costs associated with keeping jobs waiting if the services in the system are honouring Service Level Agreements, so there are clearly factors other than queue sizes which must be considered when deciding to reconfigure a server.

We present the architecture and prototype implementation of a dynamic pooling system for reconfiguring servers to improve the efficiency of a resource hosting environment. The system partitions computational resources into pools and uses heuristics with current and past system information to make reconfiguration decisions which result in servers that are part of under utilised pools being switched to over utilised pools.

2. Pooling Architecture

The goal of the system architecture is to allow servers partitioned into conceptual pools to switch from one pool to another, where the decision to switch is based on heuristics and system metrics gathered over time.

Networking technology allows a Resource Management System (RMS) to control servers in geographically disparate locations, allowing servers to switch between pools regardless of geographically location and organisational boundaries if permitted. Three main components make up the pool partitioning architecture, and at least one component must be running on each machine.

The Server Manager is a daemon which runs on each machine capable of processing jobs to make adjustments to its host allowing it to switch pools.

The Pool Manager runs an instance of the RMS in a central manager mode, queuing and scheduling jobs on servers in its control. The pool manager keeps track of the servers in its pool, participates in coordinating switches of resources to and from other pools, and queries the RMSs central manager for data such as queue length to send to the cluster manager.

The Cluster Manager receives then dispatches submitted jobs to the appropriate pool based on job type, and stores information about job arrival to be used in switching heuristics. The cluster manager receives information from each pool manager which is used with stored data to make reallocation decisions. The cluster manager acts as a coordinator between two pools during a server switch, and does not need to run an instance of the RMS.

Jobs of type i arrive at the cluster manager and are dispatched to pool i. Each pool has k_i servers dedicated to run job type i, where the total servers in the system is given as:

$$k_1 + k_2 + \ldots + k_n = M$$

Figure 1 illustrates how instances of the tree components are organised to process jobs.

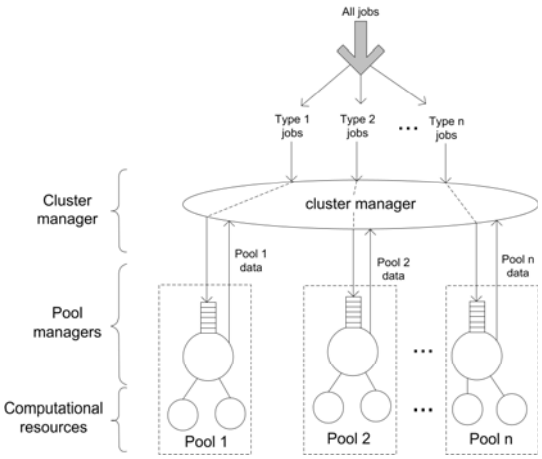

Figure 1: Architecture of a cluster partitioned into pools

Different job types are defined by administrators and may correspond to a request for a particular service or to a job dependency such as the required operating system. Information about the dependencies needed for each job type must be specified so each server is configured correctly after reconfiguration.

Dispatching a job to a pool is a trivial task and requires no scheduling as jobs are queued and scheduled at each pool manager by the RMS, so queues are not likely to build up at the cluster manager. The distributed queuing and scheduling amongst pool managers helps to disperse load on the whole system. Together the cluster manager and pool manager constantly monitor events in the system including the arrival times and processing time of each job type, which allows the cluster manager to make informed decisions on server switching between pools. Different switching policies may be used by the cluster manager provided the necessary system metrics can be gathered for the policy. Data may be obtained by the cluster manager from pool managers in an event driven or polling fashion. Events that may trigger the sending of data may include a change in the job queue or the completion of a server switch which means switching heuristics are calculated only when necessary. Each pool manager may also be polled at set intervals to send its current state to the cluster manager. A polling approach may not be efficient if a request causes a pool manager to run many monitoring systems, but if monitoring is performed continuously and saved, a poll from the cluster manager only involves reading a value from a data store.

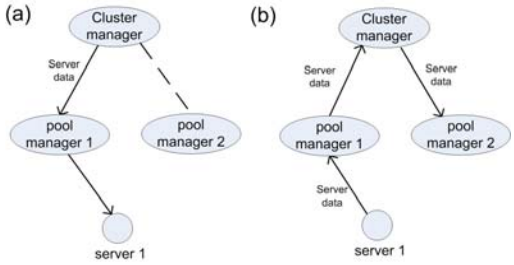

Figure 2: Communication during the switching of a server between two pools

The diagrams in Figure 2 show the communication between the components involved in a server switch. To make a switch the source pool manger (pool manager 1) is informed of the decision to remove one of its servers, and it is also told of the destination pool (pool manager 2) the server is to be reallocated to (a). The pool manager may then choose the most appropriate server to be removed, for example a currently idle

machine. Any jobs(s) currently running on the server are halted and put back onto the queue, then the source pool manager initiates the configuration alteration which may include downloading and installing software from a server. Upon successful reconfiguration of the server the source pool sends details of the server that is to be switched to the cluster (b), the details are then forwarded on to the destination pool manager which can make the necessary adjustments to allow the resource to join the pool. Adjustments made by a pool manager to accept a server include updating a list of current pool members and any execution of operations need to instruct the RMS accept the new resource. The source pool manager updates its list of servers upon successful completion of the switch.

3. Prototype System

A prototype implementation has been developed to test the effectiveness of the pooling system. The system implements enough functionality of the server manager, pool manager and cluster manager components described above to show how servers may be dynamically reallocated to pools based on a heuristic. Condor [3] is used to queue and schedule jobs at each pool, and the pool managers interface with instances of condor central managers deployed at each pool. Two pools are used to demonstrate the features of the system, as the heuristic used at the time of development supported two pools. The switching that occurs involves a server being taken from one pool added to another pool, with the only reconfiguration taking place on the server being adjustments to instances of Condor. To fully demonstrate server switching a graphical interface has been constructed, with a system of allowing a user to set system variables to observe the outcome. GridSHED software must be installed and running on each server in each pool, and each component may run on a single machine.

The cluster manager is stared via a GUI and an initial configuration of servers to pools is used to initialise each server and pool to processes different job types. It is not necessary for individual server information to be stored by the cluster manager as the cluster manager is only concerned with the number of servers in a pool, and pool information. After pool initialisation the cluster manager is started and starts processing job submissions.

A simple polling pattern is employed by the cluster manager to obtain data from each pool. The cluster manager makes a call twice every second to each pool manger which return their current queue and job details. Independently of the cluster manager each pool manager also uses the polling approach obtain queue data from the RMS, which is saved then retrieved when a call comes from the cluster manager. From the state of the queue the cluster manager may determine which jobs are running and waiting, and which jobs have completed. When a polling cycle completes and each pool has returned its current state, the switching policy is invoked with the necessary data and the heuristic is calculated. If the result determines that a server switch needs to take place, a switching protocol is invoked.

Condor was chosen to provide queuing and scheduling facilities to each pool because of its flexibility in allowing servers to switch between condor pools quickly and without extensive reconfiguration. Because of condors flexibility, all jobs running in the source and destination pool are not disturbed by a server switch and may continue executing while a switch is taking place. A job currently running on a server chosen to switch must be interrupted and placed back into the queue so the server can join the new pool and accept a queued job as soon as possible. This interruption may mean the job starts again when it gets processing time, or it may be check-pointed so it can start from where it left off at the time of interruption.

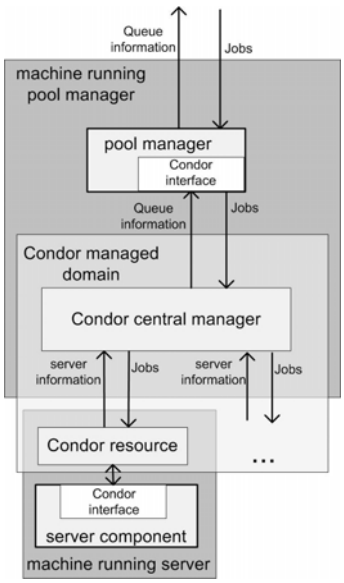

Figure 3: Integration of Condor with the pool manager and server components

Figure shows the integration of Condor with the rest of the system, the area labelled “Condor managed domain” represents components and communication internal to Condor which the prototype system employs. The pool manager and server components communicate with Condor through a proxy

made up of various scripts that are called to complete an operation. A Condor pool has one server running as a central manager which queues then matches job submissions to available servers. The actual dispatching of jobs from the pool manager to servers, including any file transfer is handled by Condor. The server component communicates with Condor only when a server is to be switched.

A graphical user interface has been developed for the cluster manager to show a user the states of various parts of the system. The GUI shows each pool currently running, with a queue of submitted jobs and a number of servers in each pool. A workload simulator is used to send test jobs to the cluster manager to observe how the system adapts to the incoming requests. The arrival rate and job time of each job type may be set through the GUI, and an exponential formula is applied to the values in an attempt to simulate unpredictable request patterns that may happen in a real resource hosting environment.

4. Design and Justification of Heuristic

The prototype system uses a heuristic policy [4] which calculates a close to optimal allocation of servers to pools in a two pool system with *M* servers. A reallocation decision is made using the average arrival rate and average job processing time of each job type and the time taken to switch a server between pools. Using the average values means past measured values may be taken into account so the heuristic produces a more meaningful result. The heuristic takes into account the cost of holding a job which may be measured in monetary terms, but serves to represent relative importance between the two job types. For example, if all measurements are equal but the holding cost of job type 1 is greater than type 2 jobs, more servers will be allocated to serve type 1 jobs than jobs of type 2. The process of switching a resource also has an associated cost as the server is unable to process jobs while in a state of reconfiguration.

A method of calculating the optimal server allocation for the system has been found, but its computation is complex. Therefore a heuristic policy is used which is practical and easy to implement, and has been shown to be almost as good as the optimal policy in simulations.

5. Conclusions and Further work

This paper presents an architecture and prototype implementation of a system which reallocates servers between conceptual pools in a close to optimal fashion to process incoming jobs with varied and unpredictable demand. By applying a heuristic policy to past and present system information, decisions are made to reconfigure servers to process jobs with differing requirements.

Future work will include more sophisticated means of monitoring and gathering data from resource pools to produce more details that may be fed into switching policies. The data could include the average amount of time particular resources are idle for and how often resources are switched. With detailed information a system for providing provisions to enable quality of service guarantees will be developed.

Acknowledgement
This work was carried out as part of the collaborative project GridSHED (Grid Scheduling and Hosting Environment Development), funded by British Telecom and the North-East Regional e-Science centre.

Contact e-mail: Charles.Kubicek@ncl.ac.uk

6. References

1. Livny M, Raman R. In *The Grid: Blueprint for a New Computing Infrastructure*, pp. 311 - 39: Morgan Kaufmann.
2. Chase JS, Irwin DE, Grit LE, Moore JD, Sprenkle SE. 2003. *Dynamic Virtual Clusters in a Grid Site Manager*. Presented at 12th IEEE International Symposium on High Performance Distributed Computing
3. Litzkow M, Livny M, Mutka M. 1988. *Condor - A Hunter of Idle Workstations*. Presented at 8th International Conference of Distributed Computing Systems
4. Mitrani I, Palmer J. 2003. *Dynamic Server Allocation in Heterogeneous Clusters*. Presented at The First International Working Conference on Heterogeneous Networks, Ilkley, UK