Creating Condor Pool-On-Demand by Using a Loosely-Coupled Peer-To-Peer Network

University of Florida. Advanced Computing and Information Systems Lab.
Kyungyong Lee (klee@acis.ufl.edu)

1. Introduction and Motivation

Many research institutions build local computing clusters to perform scientific experiment or analysis of the experiment results. The clusters in the institutions are usually connected through a high speed network under a strict control of the institution. One step further from the local computing pool resource, Grid computing allows multiple computing sites to share their computing resources when available. In Open Science Grid (OSG) project, 80 sites in US share their resources to achieve high computing throughput. TeraGrid and Grid5000 are also examples of grid computing environment. As the need for sharing idle computing resources increases, the need for efficient management of those idle resources also increase. The management systems should be scalable, fault-tolerant, and secure in the presence of abnormal or malicious behavior of a user or churn. These systems must be able to distribute jobs evenly amongst nodes to achieve maximum utilization and alleviate hot stops. Middleware solutions for managing distributed resources, such as Condor and Boinc, often rely on a central master server that maintains global view of available resources for optimal scheduling decisions. Worker nodes, in turn, publish their resource information to the central master node, so that submitted jobs' requirements can be matched against available resources to determine the most suitable nodes for executing the submitted jobs.

2. Problem Statement

The central server approach is simple and straightforward to manage a local cluster. In grid computing, where many local clusters are connected to share idle resources, central nodes in each small cluster can publish their own view of local cluster nodes to share their local cluster computing power. However, the centralized master server approach often imposes significant administrative, uptime, and scalability issues, and the failure of the master node can render all its managed resources unusable. To prevent the single point of failure problem, the system administrator can run multiple duplicated central servers while backing-up up-to-date central server state which might cause extra expenses and state synchronization problem. For scalability issue, well-maintained management software might not cause much overhead for a mid-sized clusters. However, even an efficient management software imposes O(N), where N is total number of nodes in a pool, overhead for memory, CPU, and network bandwidth consumption for the periodic resource information update even no job is submitted to the cluster. Grid computing management software CorralWMS and MyCluster rely on pilot-based approach by using each central node's local resource pool view. They still have shortcomings of central management server approach, and it shows inefficiencies due to over-booking resources by a proxy. To overcome the above shortcomings of central management approach, I present a decentralized grid computing resource discovery method and its application for creating a small computing cluster pool when a node submits jobs to a pool.

3. Approach

I built a self-organizing tree based multicast mechanism on a structured Peer-To-Peer (P2P) network. The method spreads a message over a sub-region of a P2P network by recursively partitioning the address range of P2P network. It is currently implemented on top of Brunet, which implements Symphony, a 1-d Kleinberg's small-world network. I will not discuss about the multicast method in this work, because it is not a vital component for this

work. By using the self-organizing multicast tree, a node can submit resource discovery query by specifying the resource requirement and sorting method. The discovery system is currently available as a web-service at http://www.acis.ufl.edu/~klee/discovery . At the resource discovery system, there is no central server to keep the entire view of the machine state. Instead the multicast is used to disseminate query and aggregate matchmaking results. In the web page resource discovery system, there are about 600 PlanetLab nodes deployed in the world. I use Condor ClassAd for matchmaking purpose and condor_startd to get local resource information. Using the decentralized resource discovery system, I can build a closely coupled resource pool where every node in the pool can run a job immediately. In Figure 1, I draw an on-demand pool creation by using a loosely-coupled decentralized P2P connection. In the center of the figure, you can see a loosely-coupled cluster connected by a P2P network. At the top of bottom of figure, you can see two created clusters based on the job characteristics. Each node in the pool shows its operating system (Linux, Windows, or Mac) and location (USA, Europe, or Asia). I will describe a step to form an On-Demand pool. At the explanation, I will assume every node in the pool has condor software installed.

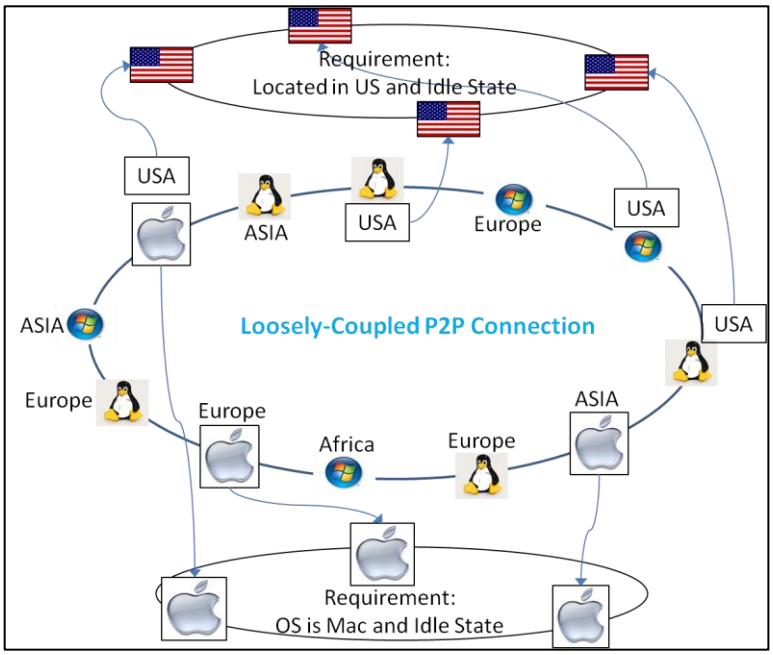

Figure 1 Pool creation by using a loosely-coupled P2P network

1. A node submits a job with a requirement and number of machines to run the job using condor_sched.
2. P2P module intercepts the job submission file and creates a resource discovery query based on the requirement and number of jobs to be executed and disseminate the query.
3. When each node gets the query, it executes matchmaking module to see whether the node satisfies the requirement or not. If a node satisfies the requirement, it registers itself to the newly created condor pool. There are two methods to join the pool. One is inserting condor_startd ClassAds to the target condor_collector by using Condor Soap interface. The other is setting CONDOR_HOST variable as the IP address of condor_collector to that of job submission node.
4. After joining an On-Demand pool, job scheduling and execution is processed by condor software.

In this system, there is no central server which keeps the status of entire machines in the pool. When a job is submitted, a pool is created On-Demand by using a resource discovery method and condor command. By adding only resource discovery method in the middle of job execution, we could inherit most characterstics of condor scheduler, which is proven to be reliable and efficient through a long run time and many usage in the HTC field.

4. Application

In a typical P2P file sharing system, such as BitTorrent, Gnutella, and LimeWire, over a million people in US are connected at the same time to share their files by showing a high join and leave rate. To share computing resources using a P2P network, a project called Archer provides a method to connect nodes in a pool by using a P2P network. By using Archer's Virtual machine instance, a user can easily join and leave the cluster pool. It uses

condor software intact, so users who are familiar with condor interfae can easily submit jobs. However, Archer still relies on a central master server, which might not handle well for high churn rate and a huge number of nodes. Our system will show better performance when the number of node in a pool is very large and churn rate is high, which is not a likely scenario in a strictly managed cluster but a likely scenario on a loosely-coupled P2P environment.

5. Lessons from the OSG Summer School and Acknowledgements

At the school, I could broaden my knowledge on condor. As a condor user, I used only a simple command or fuction before the school. After the school, I could use what I learned in the school to efficiently run jobs on a conodr pool. As a researcher on a grid computing middleware, courses on a Condor-G, glild-in, and security were very helpful to broaden my eyes in the field. In addtion, discussion about the overall condor architecture and grid software with Alain Roy, Tim CartWright, Greg Thain, and Mats Rynge was very fruitful. I appreciate all your efforts at the school.