Providing Services in a Resource Shared Distributed Environment

Introduction

In the following document we propose not a generic high-level system as "The Grid" is supposed to be but instead a simple system to solve a specific problem: sharing resources with finer granularity - resources which are available in individual machines. This means such system would have to deal with resources on a much larger scale (many more of them), very unstable and in very dynamic environments.

Our proposal differs significantly from "The Grid" because we don't view resources on a higher-level. We view the reader personal machine playing a possibly important role in the global system. Therefore, our view of this system is somewhere between a peer-to-peer system where all network nodes are equal and a Grid system where we have a greater level of organization and distribution.

Architecture

Requirements

One of the major requirements for the system is being able to handle large amounts of data in a distributed environment with a high degree of reliability. Therefore we choose a peer-to-peer approach which is known for being very fault-tolerant. Also since the main service we'd like to use is Data Management where peer-to-peer has had high success.

The other major requirement is to try to avoid the use of centralized servers. The system is design to be decentralized. Also, we pretend the applications that use this system to interact with it in such a way that movement of data shouldn't be necessary.

Architecture Components

In terms of functionalities, these can be split in four components: worker nodes, aggregator nodes, root node and user interface.

Worker - The worker node is responsible for performing a service.

Aggregator - The aggregator is responsible for issuing service requests to worker nodes and dealing with the results. Since the request is usually made to various nodes simultaneously in a peer-to-peer fashion, this node major functionality is aggregating results. In a typical use case, the aggregators are responsible for performing requests to the worker nodes they know of. They are also responsible for joining results from several peer sources and provide compiled (joint) information to the nodes consuming them (those are usually UI nodes).

Root Node - The root node is just a starting point for establishing a peer-to-peer network.

User Interface - The user interface is the human starting point for the system. It interacts mostly with aggregators but can also consume directly services provided by worker nodes if it is necessary – i.e. to perform a data transfer from a set of worker nodes to the user interface machine.

Therefore, a network node has either the role of worker, aggregator or user interface. For each role, it has a set of services available: i.e. data management or job submission.

Communication

All communication between the components of the system is done using SOAP web services. The use of

web services allows easy development of new services and consumers for those services besides facilitating communication through nodes (firewalls). The problem is the performance penalty in using web services. Although we realize this penalty we believe that interoperability is more important in this design.

One service that in current design is not decentralized is the root node, or peer discovery mechanism. When a new peer joins the network it must find other peers. Therefore, it is the function of the root node to give out this information to new peers. Still, this source of information is extremely small - usually just a list of IP address -, easy to replicate and there is not really a problem if some of it is out of date.

Storage System

All information about the distributed system is stored in metadata. Metadata was used because in order to have a useful system there would have to be a persistent way of storing and querying information. In an attempt to make the system extensible and interoperable RDF was chosen. Currently we use XML/RDF as the standard for this definition and storage. Therefore we have defined schemas for describing information about the nodes. There is an RDF file to represent a network node in terms of the role it has – aggregator, worker or user interface - and of the services it has installed – data management, job submission ... There may also be other RDF files for describing information specific to a service a node has – i.e. a file dm_worker.rdf might exist to describe worker metadata information for the data management service in a network node with role 'worker node'.

Example

An example using the data management service can be. A UI asks an aggregator to search all known nodes for a given file ('higgs001.dat'). The aggregator receives the request and forwards it to all worker nodes with data management role. It then aggregates all the results – typically a list of peer nodes which have the requested file. Afterwards it passes that list to the UI machine the request was originally made from. That UI machine would receive the compiled list of nodes and initiate a data transfer by communicating directly with the worker nodes.

The nodes are distributed in groups so that when a request is made to an aggregator – or to a collection of aggregators – only specific worker nodes known by those aggregators will receive the request.

The P2P system uses the concept of 'overlay network' and implements what is known as a second-generation P2P network.

Current Implementation Status

The current implementation handles the basic functionality described in this document. It is being built as a prototype on which further implementations will be based from.

Some problems such as node discovery – currently using a root node – and node groups – currently supporting just one level of hierarchy – have been addressed using the simplest solution. For instance, currently a worker node is only known by a single aggregator. There is no support for redundancy or caching of common requests. In the Future Directions section we discuss how these problems will be addressed in the near future.

Comparison with typical Grids

This system presents some advantages in comparison to classical grids. The Resource Brokers in typical grids must know the global status of the system - who has what and what each worker node can actually do – besides having to be aware where all data is stored and so forth. This global publication of information poses huge problems – it is a single point-of-failure even though there are some distributed architectures for

the system. In the model we are presenting there is no need for a global server of information – there is no global knowledge. This is not an issue in the situations we wish to address but manifestly limits the application of the system to other domains. Therefore, comparing to classical grid systems, while a P2P approach is more fault-tolerant it fails in providing persistent and multipurpose infrastructure, which always comes with a high cost for system stability. These infrastructures are something that is hard to accomplish in a P2P system because these tend to focus on integration of simple resources (individual PCs) instead of large and complex services and resources. So, this is something that should be balanced in a P2P system – maintain the system simple but structured.

Future Directions

To keep the use of the system as simple and user-friendly as possibly, the entire system should be packaged in a single tarball or other convenient packaging method. It can be extended by installing new services but the core should be just one package. We believe that having this installation schema in the future will prove crucial for the use of the system. One of the great characteristic of P2P system is their apparent simplicity either in installation or in use. We intend to mimic this characteristic.

We also propose to use of some proven grid-technologies. The security for this model should be provided by GSI. P2P approaches are very prone to security vulnerabilities and many times aren't used for that matter. A strong security methodology should be implemented. Data transfers for instance should also be performed using GridFTP protocol.

An end-user of the system would typically use a Web Portal for performing all actions. This portal should be tightly coupled with the aggregator. Since by design the UI is just an interface for the system, we can build either a web portal or other OS-specific interface. Having a web portal would enable the use of interesting features in cross-platform environment.

The RDF files for representing each network node should be stored in a database for faster access and manipulation. A HTTP GET request can provide a complete RDF file, but this could be generated on demand from a database and not always be kept as a separate RDF file.

A more advanced node discovery mechanism can also be thought. For that we are thinking to use some king of hash key based protocol like Kadmelia.

The usage of agents is another component which can be very helpful. These agents could be useful for walking through the system doing specific tasks.