

Requirement for distributed computing

Goal: Build a robust, scalable, uniform and secure distributed environment for parallel computing.

Targets:

Architecture of hardware platform:

1. single node
 - a. single cpu desktop
 - b. SMP desktop
2. cluster
 - a. diskless cluster:
 - i. one or a few login nodes: each has the local file system and the connection to the outside world
 - ii. multiple computing nodes, which have no local file system and only have the connection to the login node(s)
 - b. PC cluster behind the firewall:
 - i. one or more login nodes: each has the local file system and the connection to the outside world
 - ii. multiple computing nodes, which have no local file system and only have the connection to the login nodes
 - iii. may or may not have one or more dedicated shared file system
 - c. cluster-of-workstation:
 - i. each has the individual file system and the connection to the outside
 - ii. may or may not have one or more dedicated shared file system

Software platforms:

1. Support Windows, Unix/Linux, and Mac OS
2. Support local services written in native languages, such as in C, C++, FORTRAN, etc
3. Support distributed services written in byte-code languages, such as in Python, Java, C#, etc.

Under the framework of this distributed environment, there are four roles:

1. Users:

They are the final users. They may sit before the desktop, edit and submit the service request, view the results or be notified the finishing of the work. They may choose to control the service via script. An email gateway would also be suitable for managing multiple long running jobs.
2. View developers:

They are the graphics software developers. They develop various view tools (maybe the GUI tools) to help the user to build the service request, and view the service results.
3. Model Developers:

They are the numeric software developers. They develop various computational-intensive services that will do the real computational work. The environment must provide a means for developing and debugging models locally, and debugging them remotely.

4. Service Managers:

They manage the whole distributed framework and the services, including the implementation of new parallel communication patterns and applications which use them.

Basic requirements:

1. Platform and languages independent:
 - a. Uniform interface on different hardware and software platform.
 - b. Developers don't need to deal with the details about the communication and authentication.
 - c. All the services and tools can be plugged into the framework seamlessly
2. Low Total Cost of Ownership:
 - a. Easy to deploy the framework, Single node installations should require no more sophistication beyond the usual practices for installing third party software on the platform. Cluster administration will naturally require a description of the cluster configuration provided by the administrator.
 - b. Easy to deploy services written in supported languages. Deploying services in other languages will require action by the system administrators.
 - c. ?? Automatically update the framework
 - d. ?? Automatically update services written in byte-code languages
3. Robust:
 - a. Restart/Check point capability for the static computing nodes
 - b. Eliminate the possibility of infinite running of job in the computing nodes
4. Efficient:
 - a. Make full use of the hardware capacity
 - i. Make use of the file server, or shared file system if there is one in the cluster
 - ii. ?? Make use of the direct connection if each computing nodes can communicate with outside world
 - iii. Make use of cached data and use the local data as much as possible
 - iv. Make use of SMP
 - v. ?? Prevent memory swap
 - b. Make fully use of the software capacity
 - i. Share the process for those memory-intensive job and long startup job.
 - c. Automatic load balancing for the system
5. Security:
 - a. User authentication and authorization:
 - i. Does the user have the right to connect to the server?
 - ii. Is the user the one that he claims?
 - iii. Is there any intruder between the user and the server?

- iv. Does the user have the privilege to do the work that he has requested, such as to read/write file system, to run the local services, ...?
- b. Computing node authentication:
 - i. Is the computing node trustworthy?
 - ii. Is the computing node one that it claims?
 - iii. Is there any restriction to running services in that computing node?
- c. Service authentication
 - i. Authentication for the registered services
 - ii. Sand-box for user-provide services

Individual requirements

1. requirements needed by the final users
 - a. The user can query the request history, the available services, hardware capacity and usage
 - b. The user can submit the request, check the request status, and stop the running request
 - c. The user will be notified when the request service has received, started, finished, or failed.
 - d. The user can get the error message if the request fails
 - e. The user can get the service results if the request finished successfully
 - f. The user can get the active information about his collaborators
 - g. The user can (un) register to receive the information related to the current work.
 - h. The user can provide some requirements to run the request, such the max. cpu time, the number of cpus it will use, the priority, etc.
 - i. The user can change the requirements to run the request if the request is in the waiting list
2. requirements needed for the view developers
 - a. The developers can easily plug the new tools into the frame work.
 - b. Provides high-level plotting tools accepting scene descriptions for 2D and 3D plots, and for 3D geometries
 - c. Define a standard interface for the view toolkits
3. requirements needed for the model developers
 - a. Standard interface for the input/output
 - b. Standard interface for the logging and archive
 - c. Standard interface to store and extract job-related information
 - d. Standard interface to send the message to other nodes (parallel kernels)
[Why not MPI/PVM?]
4. requirements needed for the system administrators
 - a. The admin can manage the project and the user:
 - i. add/remove the project and the user
 - ii. set the user's identification
 - iii. associate the users and the projects,
 - iv. set the privilege for the user

- v. set the computing resource limitation, such as cpu time, size of total file, for the projects and the users
- b. The admin can manage the available computing nodes:
 - i. Set the configure
- c. The admin can manage the available services for each computing nodes:
 - i. The manager can install various modeling resources on compute nodes
- d. The admin can set the hints for the fine scheduling of the request so make it running more efficient:
 - i. Specify the hardware infrastructure of the cluster.