A Storage Algorithm for Grid Systems

S. Subha
SITE
VIT, Vellore, India

ABSTRACT

This paper proposes some methods to allocate file systems to requesting clients in a grid system. The clients can be processes, jobs, tasks, or users. A global book keeping algorithm is suggested to keep track of the available and used space in a grid. Sharing of file system at file level is suggested. Remote file mounts, duplicate mount points for files are suggested improvements to the existing systems which improve the transparency of the file systems to the clients.

General Terms

Grid computing, Storage Allocation

Keywords

Allocation policy, Grid Systems

1. INTRODUCTION

Grid systems are used in many areas. The resources in grid environment are shared among the hosts in the grid environment. Resource sharing is a widely studied topic in this aspect. One of the common applications of using grid systems is to have disk space for clients to proceed with their computations. A client could be a task, process, job, and user. Any of them need temp space to proceed with their computations. Some of them need to access some parts of the files available on the grid to complete their execution. The problem is how to allocate the space to these clients so as to enable them to have access to the files that they are in need of and achieve good performance. Some of the clients need to access files in the local host while others may need to access files in remote hosts. This can be achieved usually by fetching a local copy of the file and mounting on the current host. Care should be taken for data consistency between the remote and local copy.

This paper proposes some techniques to allocate storage to clients in grid environment. Methods and OS support required to allocate space on a local host to clients are proposed. Method to request a remote host to mount a file system to its local mount point and have remote client access to that file system with required permissions is suggested in this paper. An improvement to the storage allocation that minimizes fragmentation to the method suggested in [3] is proposed in this paper. OS support to have multiple mount points and data consistency for the same file is proposed in this paper. The proposed methods were simulated on Linux system.

The rest of the paper is organized as follows. Section 2 elucidates allocation policies and compares it with other proposed techniques, Section 3 proposes the mounting of remote file systems for local access with multiple mount points for the same file, Section 4 concludes and Section 5 lists the references.

2. ALLOCATION POLICIES

This section lists some allocation policies. It describes the process of allocation and contrasts it with the policy proposed in [3]. The authors in [1] discuss the concept of storage resource managers that complement the grid functions of computation where tasks are distributed in a network using management of network sharing using bandwidth reservation techniques. The authors in [2] propose the benefit of sharing jobs among independent sites of grid along with parallel multi-site job execution. The grid as indirect NFS is described in [4]. An overview of research and development challenges for managing data in Grid environments is provided in [5]. This paper relates issues in data management at various levels of abstraction, from resources storing data sets, to metadata required to manage access to such data sets. A common set of services is defined by the authors as a possible way to manage the diverse set of data resources that could be part of a grid environment. The authors in [6] describe the design and implementation of a fast and configurable remote data access system called eavivdata. The proposed system supports configurable remote operations. The authors in [7] present extensible and open grid architecture to address the problem of flexible, secure and coordinated resource sharing. The authors in [8] describe a grid-enabled implementation of the Message Passing Interface (MPI) that allows a user to run MPI programs across multiple computers at different sites using the same commands that would be used on a parallel computer. The authors in [9] propose a flexible software-only storage appliance designed to meet the storage needs of the grid. The proposed model is described next. Clients request free space during their life time in the grid environment. For the local files that need to be accessed during the process's execution, a temporary file system on top of the file system supported by the OS can be constructed. This is analogous to the file system proposed in [3]. However, each node in the file system has only one attribute that indicates the amount of space the node and its descendants occupy. This is indicated by the *used* field. At the root level there is an additional field called *size* which indicates the total available space for the file system. During the construction of the file system, initially the size is set to the disk's size and used is set to the size occupied by the root of the file system. When requests come the following actions are taken based on the type of the request.

1. Request for allocation arrives. The *size* field is checked if it can accommodate the request. The system makes a call to mallocate(path, req size). The mallocate is a function that has to be supported by the OS. The system allots req_size bytes to the path created. The path is created by the mkdir(path) command. The *size* attribute is decremented by req_size bytes. All the above actions take place only if the user/client has permissions to execute the system

commands involved.

2. A file/directory gets deleted. The request is invoked with deallocate(path). The *used* field is reclaimed for the specified sub file system to the size attribute. The path is deleted using rmdir(path) or rm filename.
3. Inquire if a path exists. This is achieved by executing inquire(path) primitive. The path is checked in the file system and a boolean answer is returned based on its existence.

Figure 1 gives the allocation scheme. The root node has 24GB of space available. The nodes of the tree have used amount of space. Each node of the tree has a field called *used* that gives the total space occupied by the node. If a new node arrives, it requests the root for the space and gets allocated if the space is available. The used field of root node and the new node are adjusted accordingly. If a node is deleted, the used space of the deleted node is reclaimed by the root. The commands listed above can be used to allocate and reclaim the space from interior and leaf nodes of the tree.

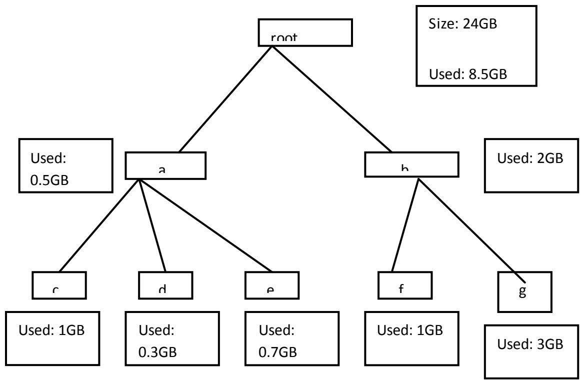

Figure1 Allocation Scheme

All the above commands can be executed atomically and are idempotent in nature. The suggested method of having one global size field minimizes the fragmentation which occurs in [3] with the presence of size and used fields at each node level. For any allocation, a request is made to the root level to allocate space from the *size* available. On every deletion, the *used* field is reclaimed at the global size field. Consider this system in contrast with one proposed in [3]. Here each node has *size* and *used* fields. The (*size-used)* space is lying vacant at any point of time. This fragmentation is avoided in the proposed model.

3. PARTITIONS IN A SYSTEM

This section proposes some changes to the mount system and partitioning of the data so as to have better performance.

In order to have more number of file systems accessible to the client, have a partitioning system which names each partition as say /dev/<integer>. Have extensible devices i.e. the system can support any number of devices by suitable hardware manipulations. For example, the system can have a scheme wherein it supports ten devices. The tenth device can be connected to another I/O subsystem which supports say twenty devices. This is analogous to indirection in the case of i-nodes. Mount the files to the mount points and make them transparent to the client. The physical limitation is the bound for the number of partitions.

In order to make the files accessible, have a scheme which allows a client to request a remote client to mount a file system in its local host. Let mountremote(path) be the command to support this. Here path gives the path to be mounted. This file system is accessible to the remote client. For book keeping purposes, have a central database which lists the mounted files on various hosts and the access rights to them. Each mount point is listed as <host #><mountpoint> and has <file system name> associated with it. The <host#><mountpoint> uniquely identifies a mount point. This arrangement will also enable request for allocation from remote disks. This is because the centralized pool contains the space in all the disks. The following system commands are to be used to make use of this feature.

1. conn = connect(mount point)

This command will establish a connection to the remote mount point via conn. The connection is one of the ways to transfer data from local to remote host.

2. read(conn, filename, buf1)

This command will read into buf1 from the filename in conn mount point.

3. write(conn, filename, buf1)

This command will write from buf1 to filename on conn.

4. open(conn, filename)

This command will open filename in conn mountpoint.

5. close(conn, filename)

This command will close filename in conn mount point. Other commands like mv, cp, are also supported to move or copy the file from conn mount point to any other mount point or same mount point. The mv(pt1, filename_source, pt2, filename_destination) moves filename_source from point pt1 to filename_destination in point pt2. This command should be used sparingly. The move operation involves transfer of the file contents from pt1 to pt2. This involves creation of the i-node for the file in the pt2 and removal of the i-node in point pt1. This command can be used only if one user is accessing the file. The cp command copies the file from pt1 to pt2. Its syntax is cp(pt1, filename, pt2). Both the copies can work independently afterwards. Concurrency can be achieved by making use of suitable semaphores and other operations between the two files as on a needed basis. One possible semaphore usage is to have all the clients have read permissions. In order to write, the client locks the file to write and releases the lock after writing. Counting semaphores can be used to have count of the number of requests.

6. unmount(mount point).

This command unmounts the file system on the mount point.
Any future access to the file system is an error.

A client can request for some space. This will be given from one of the mount points where the client has access rights. This way the available free space is increased if the mounted file system has space. To decrease the communication costs, provision to mount same file in multiple places is given. Semaphores are used to maintain concurrency of data among the mount points that mount the same file. The following gives the trace of mounting, unmounting file systems in a grid environment with the above specifications and model. Assume there are four hosts participating in a grid. Let file system A, B, C, D be on hosts 1, 2, 3, and 4 respectively. Suppose a client on host 1 wants file system A. He requests for the space to allocate it to him and mounts it. Suppose he wants file system D. He communicates with host number 4 and requests him to mount it. The mounted system will have mount point id as <4, /dev/tty00> say for example. The individual files are accessed by executing the following commands.

```
f1=connect(“4,/dev/tty00”)
```

```
Read(f1, file1, buf1)
```

Suppose that the file has to be copied in the local place then the following command is used.

```
cp(4,/dev/tt00, file1, 1,  /dev/tty01)
```

which copies file1 from D to A.Here the copy command is modified to have the syntax cp(source host, mount point,destination host, mount point)

In case the file accessed is an interior node, the path of the file is given . For unmounting, the file system the following is used.

```
close(f1, file1)
unmount(f1)
```

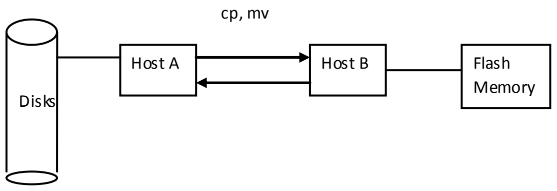

Figure 2 Schematic Representation of copy and move between mount points

Figure 2 gives a schematic representation of copy and move commands. The files are copied or moved on the communication channel from one host to another as on a needed basis. The commands involve changing the file directory entries. The communication channel in this case is assumed to be present. The performance improvement based on this method to the one that transfers the subtask to the remote host to be executed can be calculated as follows. Suppose a file F of size s bytes is to be used by the client. If F is remotely located the subtask is sent to the remote host and commands are executed on it. Suppose it takes t_1 t_2 t_3 units of time to do this subtask where t_1 is the time to transmit the request of the subtask, t_2 is the time of computation, t_3 is the time to receive the result. Suppose there is n number of such subtasks with varying time requirements. Consider copying the file to local host. It takes t_4 units to transfer the file F from remote host to local host. Performance improvement is achieved if the time of computations locally along with the copy time is less than executing the subtasks remotely. Similarly, if move were to be used, an analogous argument follows.

4. CONCLUSION

Some methods to allocate space in grid environment have been proposed in this paper. Methods to allocate space on local host in contrast with one proposed in [3] has been proposed. The proposed model avoids fragmentation problem. Methods to enhance mounting of file systems among various hosts have been proposed. This ensures transparency among the processes/clients executing tasks. The proposed model has been simulated on LINUX environment and is verified to be feasible.

5. REFERENCES

[1] A. Shoshani, A.Sim, J.Gu, Storage resource managers:Middleware components for grid storage, 19th IEEE Symposium on Mass Storage Systems, 2002

[2] Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Ramin Yahyapour, Achim Streit, On Advantagesof Grid Computingfor ParallelJob Scheduling, Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid, 2002

[3] Douglas Thain, Operating System Support for Space Allocation in Grid Storage Systems, Proceedings of 7th IEEE/ACM International Conference on Grid Computing, 2006

[4] DouglasThain, Jim Basney,Se-Chang Son,and Miron Livny, The Kangaro Approach to Data Movement on the Grid , Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing, 2001

[5] Heinz Stockinger, Omer F. Rana, Reagan Moore, Andre Merzky Data Management for Grid Environments, Proceedings of Nineth International Conference on High-Performance Computing and Networking, 2001

[6] Hutanu, A.; Allen, G.; Kosar, T.; High-performance remote data access for remote visualization, Proceedings of GRID, 2010

[7] Ian Foster, Carl Kesselman, Steven Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organization, The International Journal of High Performance Computing Applications, 15(3), (2001) 200-222.

[8] Ian Foster, Nicholas T. Karonis, A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems, Proceedings of SC'98

[9] J.Bent , V.Venkataramani, N.LeRoy, A.Roy, J.Stanley, A.Arpaci Dusseau M.Livny, Flexibility, Manageability, and performance in a grid storage appliance, 11th IEEE Symposium on High Performance Distributed Computing, 2002