Data Consistency and Fault Tolerance

Jose R. Ortiz Ubarri
University of Puerto Rico @ Mayaguez
CISE PhD Seminar
cheo@hpcf.upr.edu
Agenda

• Introduccion
• Fault Tolerance
• Data Consistency
• Redundancy
• Examples (Gfarm, PVFS)
• My Two Cents
• Conclusion
Introduction

Grid Computing and cluster computing are very popular in solving problems that requires high performance computing. The main difference between both is that Grid focus on large-scale resource sharing among dynamic collections of individuals, institutions, and resources. In the dynamic nature of Grids, resources may enter and leave the grid at any time without the applications control. Fault tolerance and Data consistency are very important in Grid Computing because the dependability of individual grid resources may not be able to be guaranteed. The grid should be able to detect, correct, remove faults and be able to continue delivering acceptable services.
A fault tolerance definition

Avizienis defined fault tolerance in his paper The N Version Approach to Fault Tolerant Software as: “... to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service.”
Data Consistency

- Data consistency is an important issue regarding fault tolerance. Data may become inconsistent, corrupted and even unavailable with the failure or malicious use of a node in the Grid.

- Distributed data storage is one of the most popular examples where fault tolerance is needed to ensure data consistency.

- There is a lot of research in tools and protocols for the sharing of data storage through distributed computers.

- Examples of the protocols are the Parallel Virtual File System (PVFS) used in computer clusters and Grid Datafarm (Gfarm) for Grids.
Traditional Distributed System Fault Model

- **Network Errors**
  - Packet Loss
  - Packet Corruption
- **Time Faults**
  - Early faults
  - Late faults
- **Response Faults**
  - Value faults
  - State transition faults
- **Physical faults**
  - Faulty CPUs, Memory Storage, Etc
- **Life-cycle faults**
  - Versioning faults
Traditional Distributed System Fault Model

• Omission faults – Messages can be delayed or lost

• Interaction faults
  - Protocol incompatibilities
  - Security incompatibilities
  - Policy problems
  - Timing Overhead
Additional Grids Faults

The traditional Distributed System fault model is not enough for Grids. To those fault problems we have to add Service Expiry to the Life-cycle fault that happens when a node in a grid has assigned a fixed amount of use time to the services. Also we have to add Metered Access which is used to meter the resource use time of a process.
Replication

- In order to tolerate faults and provide data consistency the most popular and traditional solution is replication.
- Replication is used in most of the tools and protocols implemented to provide data consistency and fault tolerance, including Gfarm, PVFS with fault tolerance support and Raid 1, Raid 10, etc.
- This technique improves read performance in data access, while decrease a little of the write performance.
Replication in Consistency Protocols

- Consistency protocols use data replication for performance issues, to allow multiple nodes to read the same data in parallel via local accesses.

- When a node modifies a data copy, the consistency protocol is activated. Then the other copies must be updated or invalidated, to prevent subsequent read operations from returning invalid data.
Replication is also used by fault tolerance mechanisms to enhance availability in an environment with failures. When a node hosting a data copy crashes, other copies can be made available by other nodes. Other situation can arise if the data is corrupted or lost, then it can be rebuild by other copies.
Replication for Consistency and Fault Tolerance

In distributed systems where both consistency and fault-tolerance need to be handled, replication can be used with a double goal. There are two architectural designs that handle data consistency and fault-tolerance with replication.
Integrated Design

The integrated design approach consists in addressing consistency and fault-tolerance at the same time, relying on the same set of data replicas. What is done is that data copies created by the consistency protocols to enhance data locality can serve as backup if crashes occur. On the other hand, backup replicas created for fault tolerance can be used by the consistency protocol.
Decoupled Design

This approach consists in designing the consistency protocol and the fault-tolerance mechanism separately. With this approach the design of consistency protocols is simplified and then it is possible to use existing consistency protocols. Also consistency protocols can be developed independently. Various combinations of consistency and fault-tolerance algorithms can be implemented and tested at the same time.
Hierarchical consistency protocol example

- For a cluster of computers a home-based protocol is used, in which a home node is associated to each data.
- The node is responsible for maintaining a reference copy for that data and a lock associated to it.
- When a process enters a critical section protected by such a lock, the associated shared data is updated on the node hosting that process (if necessary).
- If there were any local modifications then they are transmitted to the local home node.
A Hierarchical Consistency Protocol Example
Hierarchical Consistency Protocol Example (cont)

- For a group of clusters a two-level hierarchy of home nodes is used. On each cluster a local home will server access from the local cluster, whereas a global home will serve data access to the clusters local homes.

- When a client needs access to some data it will require the associated lock to its local home. If this home owns the corresponding access rights to the data, it can satisfy the access. Otherwise, it will request the lock from the global home, with an updated copy of the data.
A Fault-tolerant Hierarchical Protocol

The local homes and the global home cannot be assumed to be failure-free nodes. A fault-tolerant approach is to replace each local home by a set of nodes called Local Data Group. And the Global home is replaced by a Global Data Group, whose members are the LDGs.
LDGs and the GDG properties

1) All messages sent to such a group are received by all members of the group, in the same order (atomic multicast).

2) The groups are self-organized: they maintain some use-specified replication degree by dynamically adding new members when necessary.
A Fault-tolerant Scheme

The replication scheme consists of:

- A UDDI repository which is a service capable of receiving jobs, executing them, performing checksum operations on them and sending the jobs back.

- Two co-ordination services which are used to detect point of failures
How the scheme works?

• The coordination services wait for a client to send them a job via multicast.

• The primary coordination service determines which nodes to send replicas of the job.

• The nodes process the jobs until they complete or fail.
How the scheme works?(cont)

• Each node generates a checksum based upon the results it has generated and then broadcasts the checksum to the co-ordination services.

• The co-ordination service wait until a given number of node have completed, and then vote on the returned checksums.

• The selected result will be the one with the majority of checksums received.
The Parallel Virtual File System (PVFS)

PVFS is a RAID-0 style system that does only striping in its current implementation. PVFS is developed for using in clusters.

Its components are:

- Metadata is information which describes a file, such as its name, its place in the directory hierarchy, its owner, and how it is distributed across nodes in the system.

- The I/O server handles storing and retrieving file data stored on local disks connected to the node. These servers actually create files on an existing file system on the local node.
The PVFS (cont)

- PVFS native API provides user-space access to the PVFS servers. This library handles the scatter/gather operations necessary to move data between user buffers and PVFS servers.

- PVFS Linux kernel support provides the functionality necessary to mount PVFS file systems on Linux nodes. This allows existing programs to access PVFS files without any modification.
CEFT-PVFS is a RAID-10 style parallel file system. The CEFT-PVFS has two logical groups of storage nodes. The first group is the primary storage group and the second is a backup storage group. CEFT-PVFS combines striping with mirroring by first striping among the primary group of storage nodes and then duplicating all the data in the primary group to its backup group to provide fault tolerance.
The Grid Datafarm (Gfarm) is an architecture for petascale data-intensive computing on the Grid. Gfarm takes advantage of access locality to achieve a scalable I/O bandwidth using an enhanced parallel file system integrated with process scheduling and file distribution. Gfarm schedules programs on nodes where the corresponding segment of data are stored to utilize local I/O scalability.
Gfarm File System

The Gfarm file system consists of the Gfarm files, File replicas, FS Metadata and the unified I/O and compute nodes.

- The Gfarm file is a large-scale file that is divided into fragments and distributed across the disks of the Gfarm filesystem.

- Gfarm may be replicated on an individual fragment basis by Gfarm commands and Gfarm I/O APIs manually or automatically. The file replicas are managed by the Gfarm filesystem metadata. The consistency management in Gfarm is not necessary since every gfarm file is write-once.
Gfarm File System (Cont)

- Metadata database of Gfarm consists of a mapping from a logical Gfarm filename to physically distributed fragments filenames, a replica catalog, platform information, and file status.
- Metadata also contains a file checksum used to check data consistency when replicating, and a command history used to re-compute the data when an node or a disk fails.
Gfarm File Recovery and Regeneration

- When there are no replicas of gfarm files fragments, Gfarm files are dynamically recovered by re-computation. Gfarm re-computes lost files by using the same program and arguments as were used for the original file generation. (Using the command history log).

- Gfarm can supports the GriPhyN virtual data concept which allows data to be generated automatically, and existing data retrieved, through an application-specific high level query. Gfarm generates this query using the command history log and the metadata.
My Two Cents

As argumented in this presentation, the most famous and best method to handle fault-tolerance is the replication. The problem with this method is the overhead in the case of replicating jobs and the space utilization when dealing with replicating data. In my opinion based on data replication. To decrease the disk space waste of replication there should be research in handling data fault tolerance using a RAID-5 type striping with parity bit algorithms for the regeneration of lost files because of faulted nodes or disks.
The problem with this idea is that this algorithms will increase computing overhead in the nodes, and it will be more difficult for developers to implement. Also with an exact RAID-5 type implementation cannot be possible since you may not guarantee that no more than one node is going to fail at the same time.
Conclusion

- Because of the dynamism of Grid Computing the regular algorithms for fault tolerance and data consistency can be used but are not enough for the grids.

- For a good Grid computing experience it is necessary to be able to tolerate faults and provide data consistency to the users since faults and unconsistency are very possible in a grid environment.

- Redundancy using algorithms like raid-(0|10) are the most used and implemented for data consistency and disks fault tolerance.
Questions?