Toward An Integrated Cluster File System

Adrien Lebre

February 1st, 2008

XtreemOS IP project is funded by the European Commission under contract IST-FP6-033576

Outline

Context

- Kerrighed and root file system
- Parallel file system vs Symmetric file system

kDFS, kernel Distributed File System

- Building a distributed FS upon kddm mechanisms
- Architecture overview
- Performance

kDFS, integration with other cluster services

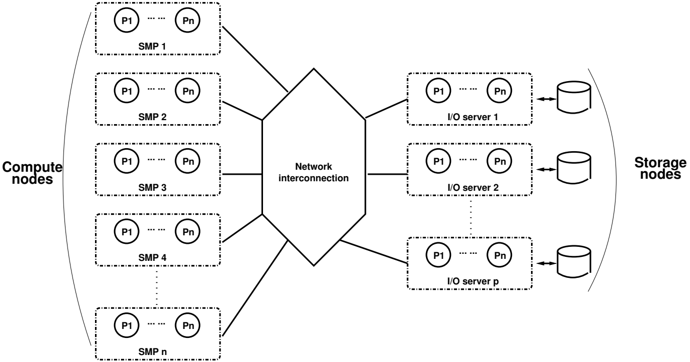
- Scheduling policies, checkpoint/restart, hotplug, ...

kDFS, conclusion, future work

Kerrighed and The Root File System

Background

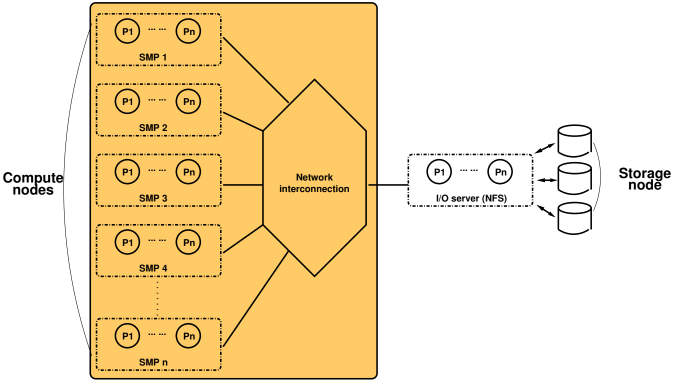
- A cluster, generally based on the historical model : compute *vs* storage nodes
- Lot of works have been done (Parallel FS, NAS, SAN, . . .)
- Inefficient use of disk capabilities (space and throughput)



Kerrighed and The Root File System

Background

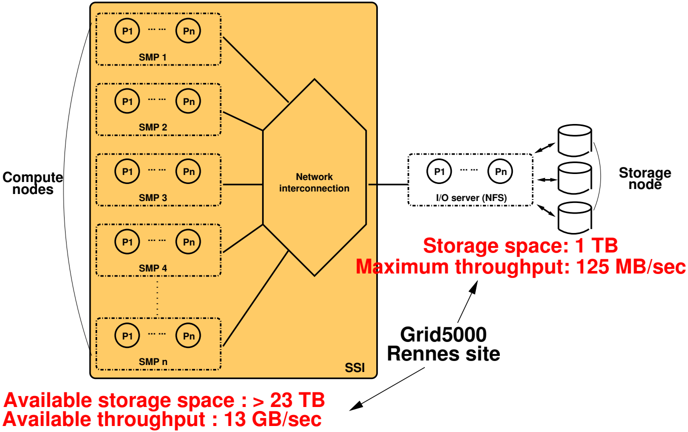
- A cluster, generally based on the historical model : compute *vs* storage nodes
- Lot of works have been done (Parallel FS, NAS, SAN, ...)
- Inefficient use of disk capabilities (space and throughput)



Kerrighed and The Root File System

Background

- A cluster, generally based on the historical model : compute *vs* storage nodes
- Lot of works have been done (Parallel FS, NAS, SAN, . . .)
- Inefficient use of disk capabilities (space and throughput)



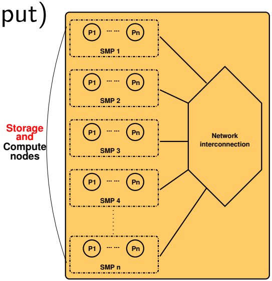
Storage space: 1 TB
Maximum throughput: 125 MB/sec

Available storage space : > 23 TB
Available throughput : 13 GB/sec

Kerrighed and The Root File System

Background

- A cluster, generally based on the historical model : compute *vs* storage nodes
- Lot of works have been done (Parallel FS, NAS, SAN, . . .)
- Inefficient use of disk capabilities (space and throughput)



Objectives

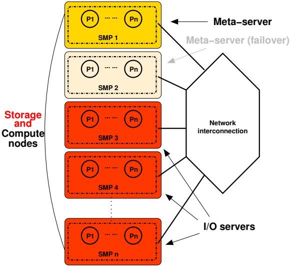
- Federate available hard drives :
  aggregate storage spaces
- Fine and efficient use of disk throughput :
  data striping, distributed I/O scheduling, redundancy
- Transparency from both application and resource usage point of views

Parallel File System vs Symmetric File System

Parallel File System

- One meta-server and several I/O servers
- Single Point Of Failure ⇒ failover server
- Scalability issue

Performance and reliability
⇒ FS services present on each nodes !



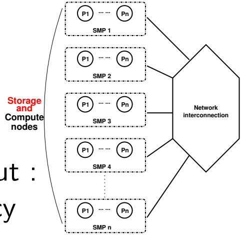
Symmetric FS

- No *SPOF*, better load-balancing
- Design and implementation much more complex (consistency) : Several proposals but no real implementation (xFS, serverless FS)

How to take into account application requirements ?
(CPU / memory / ...)

Toward an Integrated Cluster File System for *HPC*

First objective : a symmetric file system



- Federate available hard drives : aggregate storage spaces
- Fine, transparent and efficient use of disk throughput : data striping, distributed I/O scheduling, redundancy

Performance / transparency / reliability ⇒ Symmetric Kernel FS !

Second objective : integration (Kerrighed philosophy ;))

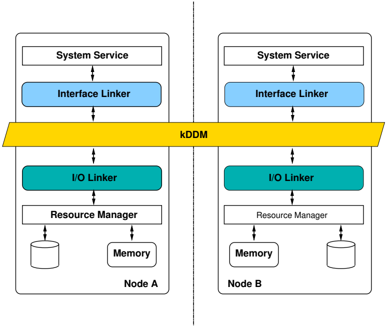
- Integrate the cluster file system with other cluster services : scheduler, checkpointing, hotplug, . . .
- Take advantage of services complementarity

Provide fine mechanisms to continue to improve cluster usage

kDFS and Kerrighed (1/2)

Kerrighed, one of the most complete *SSI*

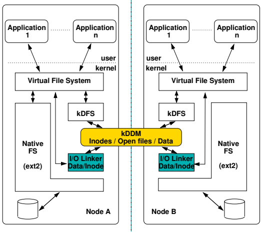
- Developed during 6 years in the PARIS project-team
- Since 2006, developed as an open source project (Kerlabs, XtreemOS, ...)
- Lot of features : C/R, live migration, load-balancing, hotplug, ...
- All of them based on the Kernel Distributed Data Manager
  Clusterwide data-sharing at kernel level [Lottiaux01]



kDFS and Kerrighed (2/2)

Building a DFS upon kddm mechanisms (and only !)

- kDFS, Kernel/Kerrighed Distribued File System : Clusterwide file system at kernel level
- Based on cooperative caching mechanisms



From kerFS (2004-2005)

- More a "proof of concept"
- Nodes have to participate in the physical structure of the file system
- Meta-data replicated on all nodes : overhead to maintain consistency

To kDFS (2006-2010)

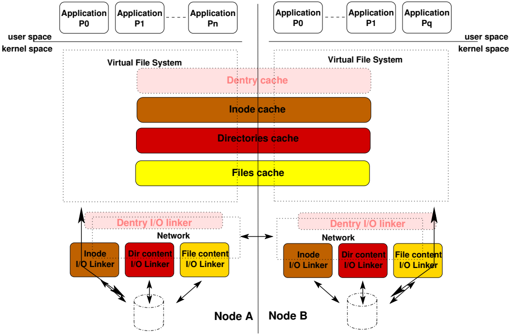
- 4 years to work on an integrated FS
- Nodes can access to kDFS files without providing storage spaces
- Meta-data fully distributed : performance, reliability (later)

kDFS keeps several kerFS proposals but developed from scratch !

kDFS, Architecture Overview

4 kinds of kddm-set to provide a cooperative cache

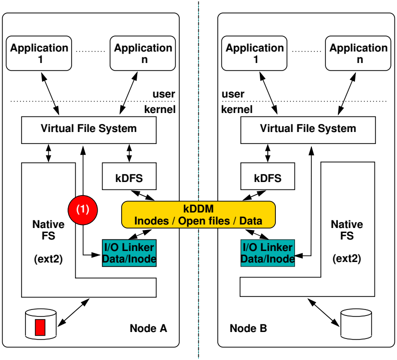
- Inode management,
- Content Management (directories and files),
- Dentry management



kDFS, Performance (1/2)

"Caching" mechanisms

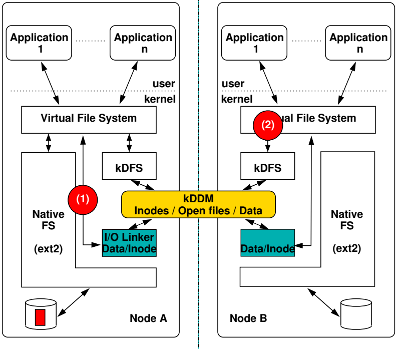
- Exploit local Linux mechanisms at kDFS low level (1) (read-ahead and write-back)



kDFS, Performance (1/2)

"Caching" mechanisms

- Exploit local Linux mechanisms at kDFS low level (1) (read-ahead and write-back)
- Exploit local Linux mechanisms at kDFS high level ? ? (2)
  - read-ahead : usefull / useless ?
  - write-back : reduce network traffic but from tolerance point of view ? write-through : impact of keeping data synchronized



kDFS, Performance (2/2)

"Striping", two modes

- Transparent (implicit) : data are written locally
  "Parallel programs are the best to discover suited striping parameters"
- User (explicit) : users provide parameters on a directory/file basis

"Redundancy"

- Users should notify (RAID 1)
- Reduce impact on "non-tolerant" applications

"User mode" requires to extend POSIX calls

kDFS, Integration with Other SSI Services (1/2)

kDFS and SSI scheduler

- Interaction between kDFS and SSI scheduler to improve data locality (processes are launched where required files are stored)
- Exploit I/O probes to improve :
  Scheduling decision (load-balancing, reducing network traffic, ...)
  File distribution (which are the 'best' nodes for storing data)

kDFS, SSI scheduler and migration mechanisms

- IOR benchmark from LLNL (MPI Parallel I/O application), two phases :

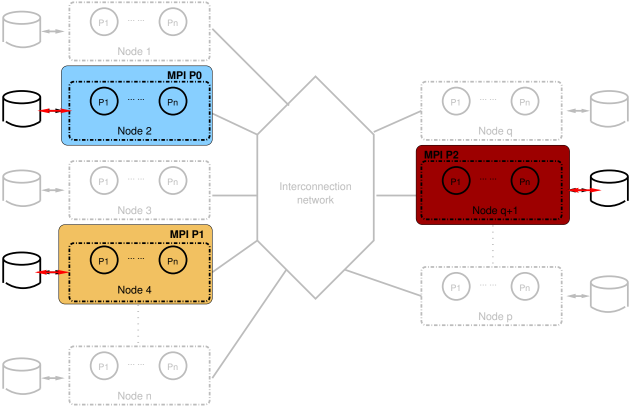
  1./ For each process :
  Reads particular data from one common file (according to its MPI rank),
  Processes them,
  Writes results in a second common file

  2./ Permutation between processes is made
  Restart step 1./ from last result file.

- Instead of making remote access, migrate processes to the 'right' nodes.

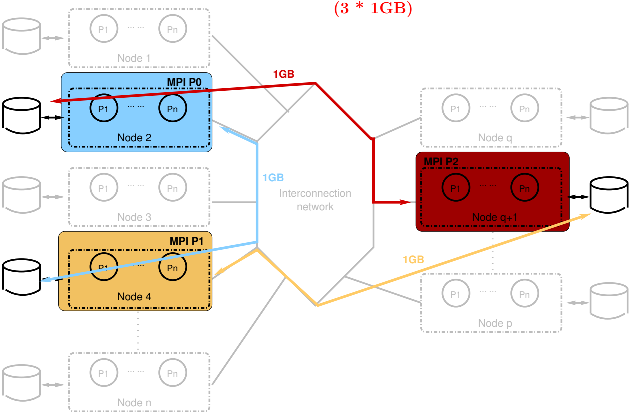
kDFS, Integration with Other SSI Services (1/2)

1./ Each mpi process reads and then writes (let's say 1GB)



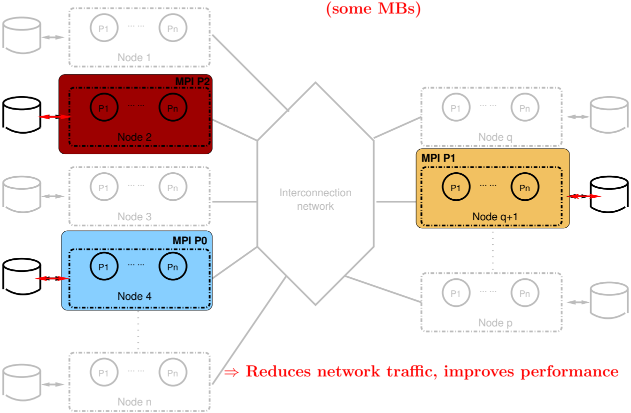
kDFS, Integration with Other SSI Services (1/2)

2./ After permutation, each mpi process accesses remote data (3 * 1GB)



kDFS, Integration with Other SSI Services (1/2)

2./ Detect remote accesses and exploit migration mechanisms (some MBs)



⇒ Reduces network traffic, improves performance

"Proof of concept" almost done with NFS client cache mechanisms

kDFS, Integration with Other SSI Services (2/2)

kDFS and hotplug

- Manage 'human' nodes addition/removals
- Transfert meta-data and content files (size issue)
- Exploit a particular mode where some files are not reachable
  (Notifiy SSI scheduler to 'sleep' impacted processes)

kDFS and checkpoint mechanisms

- Extend the VFS to provide incremental snapshot for a specified file.
- Define the 'best' node to save the checkpoint
  (reduce 'system noise')

Conclusion

kDFS, towards an integrated cluster file system for HPC

- Build a symmetric kernel file system :
  Based on cooperative caching strategies
  Without applying "intrusive kernel patch"
- Focus as soon as possible on the integration with other services
- Presentation based on my current work (XtreemOS/Kerrighed framework)

kDFS, roadmap for next months

- kDFS, an alpha version available since the 15th october ("proof of concept")
- Fix page cache management issue ("out-of-memory" in the alpha version)
- Me, "efficiency" features (mainly striping and scheduling)
- 2 master students :
  - Pierre Riteau - File checkpoint mechanisms
  - Marko Novak - I/O probes and scheduling coordination

kDFS, 3000 LOC, it takes times : look for some volunteers :)

Toward An Integrated Cluster File System

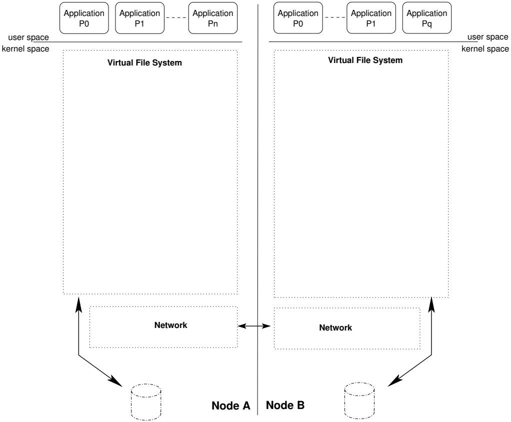
Adrien Lebre

February 1st, 2008

kDFS, Implementation Overview

4 kinds of kddm-set to provide a cooperative cache

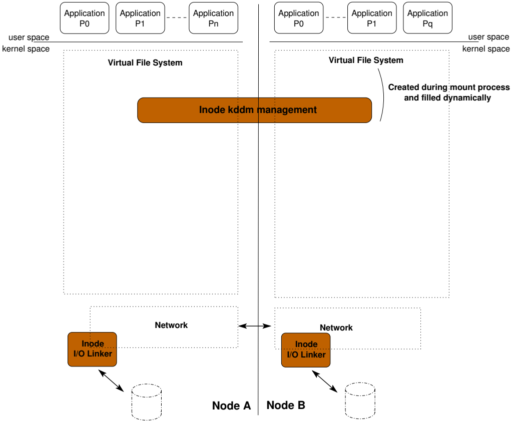
- Inode management (INODE_LINKER)
- Content management : meta-data (DIR_LINKER) and file content (FILE_LINKER)
- Dentry management, (DENTRY_LINKER)



kDFS, Implementation Overview

4 kinds of kddm-set to provide a cooperative cache

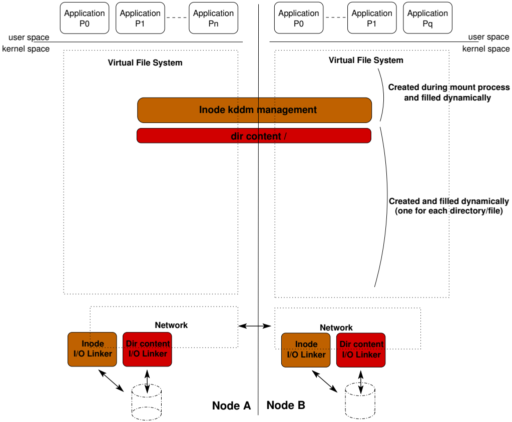
- Inode management (INODE_LINKER)
- Content management : meta-data (DIR_LINKER) and file content (FILE_LINKER)
- Dentry management, (DENTRY_LINKER)



kDFS, Implementation Overview

4 kinds of kddm-set to provide a cooperative cache

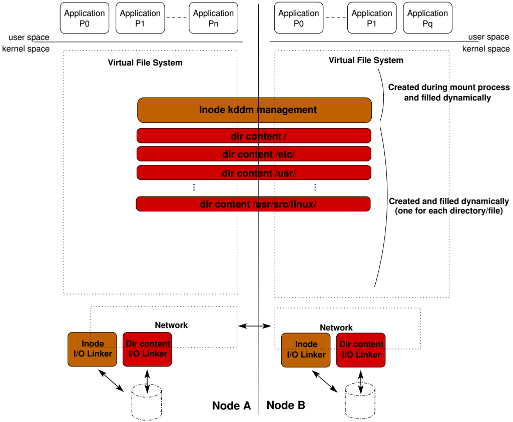
- Inode management (INODE_LINKER)
- Content management : meta-data (DIR_LINKER) and file content (FILE_LINKER)
- Dentry management, (DENTRY_LINKER)



kDFS, Implementation Overview

4 kinds of kddm-set to provide a cooperative cache

- Inode management (INODE_LINKER)
- Content management : meta-data (DIR_LINKER) and file content (FILE_LINKER)
- Dentry management, (DENTRY_LINKER)

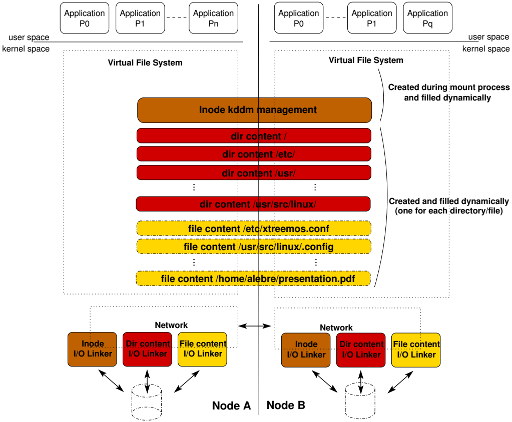


questions

kDFS, Implementation Overview

4 kinds of kddm-set to provide a cooperative cache

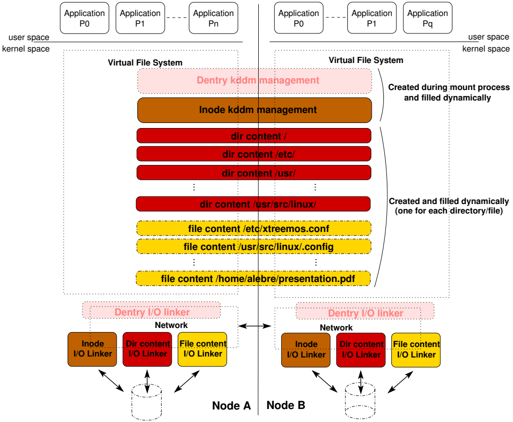
- Inode management (INODE_LINKER)
- Content management : meta-data (DIR_LINKER) and file content (FILE_LINKER)
- Dentry management, (DENTRY_LINKER)



kDFS, Implementation Overview

4 kinds of kddm-set to provide a cooperative cache

- Inode management (INODE_LINKER)
- Content management : meta-data (DIR_LINKER) and file content (FILE_LINKER)
- Dentry management, (DENTRY_LINKER)



kDFS, Formating and mounting a kDFS Structure

Formatting a kDFS "partition"

- `mkfs.kdfs DIRECTORY_PATHNAME ROOT_NODEID`
- Create the kDFS "superbloc" file for the node
  (kdfs bitmap for inode id allocation, reference to ROOT_NODEID)
- if NODEID equals ROOT_NODEID, create root meta-file

Mounting/Accessing a kDFS system

- `mount -t kdfs ALLOCATED_DIRECTORY|none MOUNT_POINT`
- Current limitations :
  Only one kDFS MOUNT_POINT per node,
  'none' mode has to be finalized (few days)
- Advanced version :
  Import local and network file systems inside kDFS
  Add some QoS parameters (such as storage space size)

kDFS, Meta-data Management

Structure of a kDFS "partition"

- Independent from native file system
- `KDFS_DIR/...` ⇒ "superbloc file"
- `KDFS_DIR/0-99/`, `KDFS_DIR/100-199/`, ..., "meta-data" files, "content" files
- on ROOT_NODEID `KDFS_DIR/0-99/1` correspond to the kDFS '/'
- Meta-files are stored in a binary mode

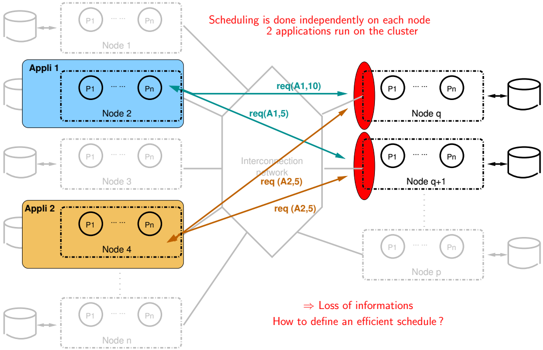
kDFS inode id and meta-files

- 32 bits, 8 for node id, 24 for local inode id (now a scalability limitation ☹)
- If possible creation is done locally :
  get an inode id (nodeid + free id from local bitmap)
  Create corresponding meta-file
  `mkdir /foo` ⇒ `./0-99/2`
- Two kinds of meta files :
  "directory" meta-file stores directory structure (directory entries)
  "file" meta-file stores file distribution (based on an object approach)

kDFS, Performance

"I/O scheduling"

- Block I/O schedulers are not sufficient (on a block basis, no global view)
- Manage I/O requests incoming for all applications



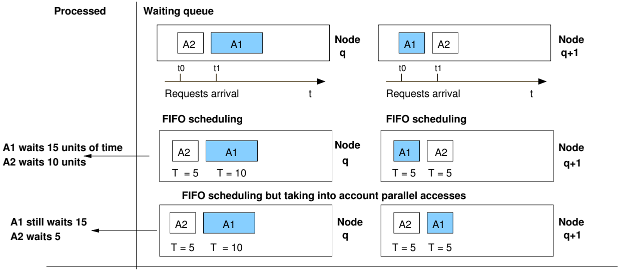
Scheduling is done independently on each node
2 applications run on the cluster

⇒ Loss of informations
How to define an efficient schedule ?

kDFS, Performance

"I/O scheduling"

- Block I/O schedulers are not sufficient (on a block basis, no global view)
- Manage I/O requests incoming for all applications



Processed

Waiting queue

A2 A1 Node q

A1 A2 Node q+1

t0 t1 Requests arrival t

t0 t1 Requests arrival t

FIFO scheduling

FIFO scheduling

A1 waits 15 units of time
A2 waits 10 units

A2 A1 T = 5 T = 10 Node q

A1 A2 T = 5 T = 5 Node q+1

FIFO scheduling but taking into account parallel accesses

A1 still waits 15
A2 waits 5

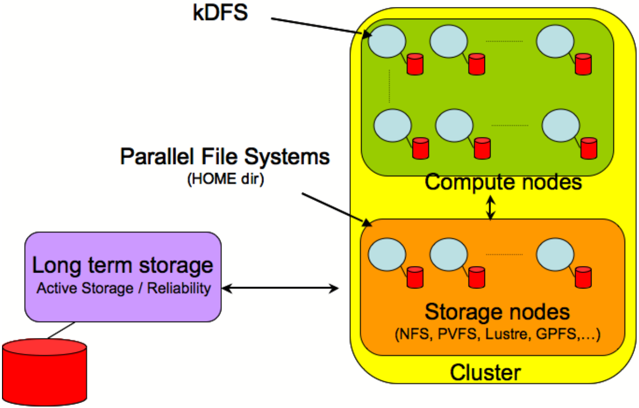
A2 A1 T = 5 T = 10 Node q

A2 A1 T = 5 T = 5 Node q+1

Looking one step ahead (1/2)

Hierarchical Storage

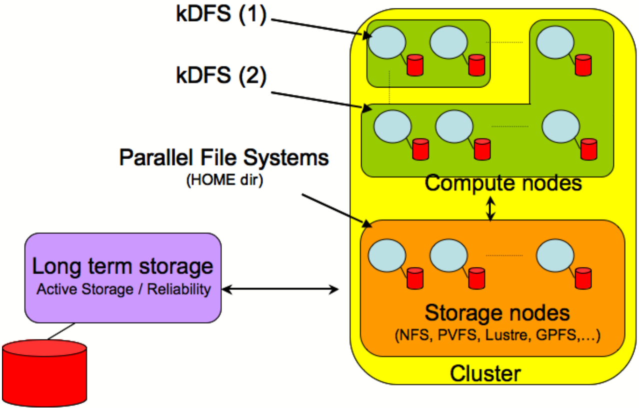
- kDFS during the execution of application
  (Distributed cache, cooperation with cluster services, ...)
- Concurrent applications ⇒ Several kDFS



Looking one step ahead (1/2)

Hierarchical Storage

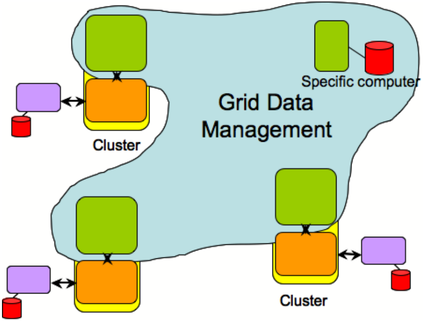
- kDFS during the execution of application
  (Distributed cache, cooperation with cluster services, ...)
- Concurrent applications ⇒ Several kDFS



Looking one step ahead (1/2)

kDFS as a Grid FS building block

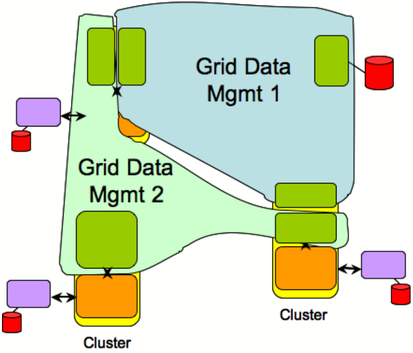
- Grid Data Mgmt system is built on kDFS
- Coordination between Grid Data Mgmt and other Grid services (Grid scheduler, Network probes, ...)
- Concurrent applications ⇒ Several Grid Data Mgmt System



Looking one step ahead (1/2)

kDFS as a Grid FS building block

- Grid Data Mgmt system is built on kDFS
- Coordination between Grid Data Mgmt and other Grid services (Grid scheduler, Network probes, ...)
- Concurrent applications ⇒ Several Grid Data Mgmt System


Grid Data Mgmt 1
Grid Data Mgmt 2
Cluster
Cluster