ソフトウェア界面

Kosha: A Peer-to-Peer Enhancement for the Network File System

Ali R. Butt
· Troy A. Johnson · Yili Zheng · Y. Charlie Hu

Journal of Grid Computing (2006) 4: 323–341

Robert Gravina

Agenda

- Introduction
- Enabling technologies
- File distribution across nodes
- Design of Kosha
- Implementation
- Evaluation
- Related Work

Introduction

- Systems used in grid computing typically use off-the-shell equipment
  - Of 500 instructional machines at Purdue, 84% have 40GB, 90% free space.
- Use of NFS is common in these environments
- Design goal: utilise this cheap storage to create a distributed file system with NFS semantics

Introduction

- Peer nodes may fail over time or have different disk capacities
- Design goal: Provide:
  - Location transparency
  - Mobility transparency
  - Load balancing
  - High availability (file replication and transparent fault handing)

Enabling Technologies

- Uses a DHT implementation called Pastry
  - Scalable, fault resilient, self-organising
  - Each node has a nodeId randomly assigned form a circular 128-bit identifier space
  - Given message and a key, Pastry reliably routes message to node numerically closest to key
- Each node maintains a routing table
  - Rows of other nodes nodeIds, which share increasingly longer prefixes with the current nodes nodeId
    - Node forwards message to a node that has the longest prefix in common with the key.

Enabling Technologies

- Each node maintains a leaf set
  - *l* nodes which are numerically closest to the present nodes nodeId (*l*/2 smaller and *l*/2 larger)
  - Leaf set ensures reliable message delivery and used to store replicas of application objects

Distribution of File System

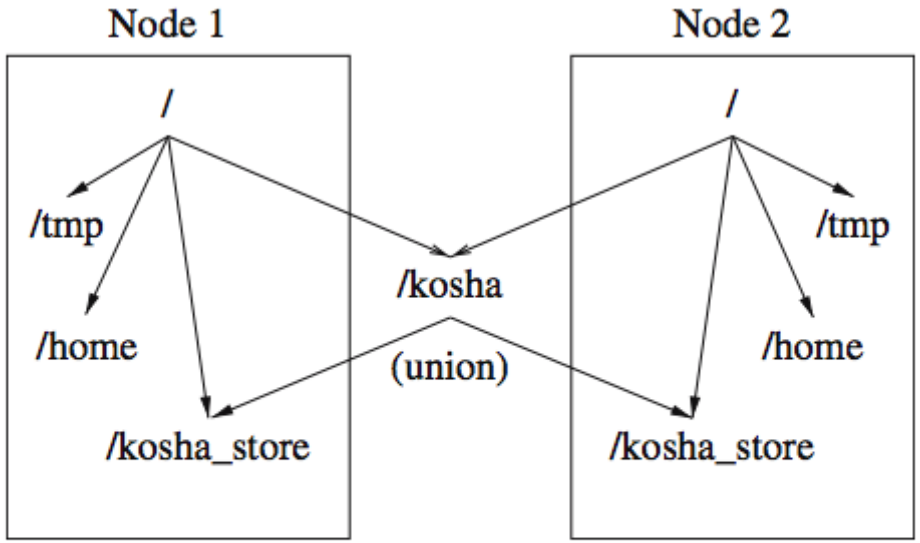


Figure 1 Virtual directory hierarchy: */kosha* is the virtual directory and is the union of */kosha_store* on all the nodes.

- Mounted as /kosha
- All files in a directory (but not subdirectories) reside on the same node
  - helps to reduce the costs of hashing/lookups for storage node while maintaining good load balance

Distribution of File System

- e.g. To locate a node for */dir/file1* Kosha performs the following mapping:
  - */kosha/dir1/file1*
    - *DHT(hash(dir1)) : kosha_store/dir1/file1*

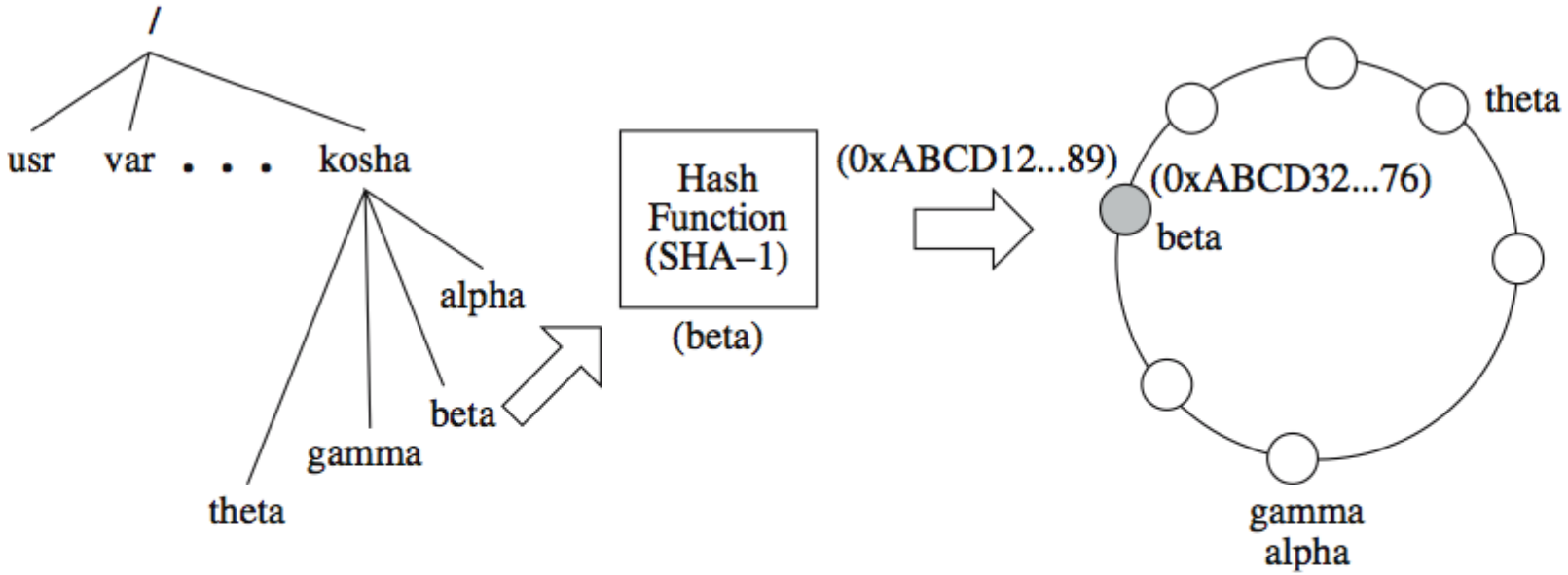


Figure 2 Example of file distribution to multiple nodes. The virtual mount point is */kosha*. The directory name is first hashed using a general hashing function such as SHA-1 to generate a unique key, which is then routed using Pastry to a node whose `nodeId` is numerically closest to the key. The selected Pastry node will provide the physical storage for the directory. The actual file operations, however, are performed via the NFS protocol (not shown).

Optimisation

- What happens when a node is full?
- On create file, redirect
  - Concatinate random salt to filename, rehash, find new node
  - Special soft link created in parent directory with same name as original file
    - Target is new filename on new node

Optimisation

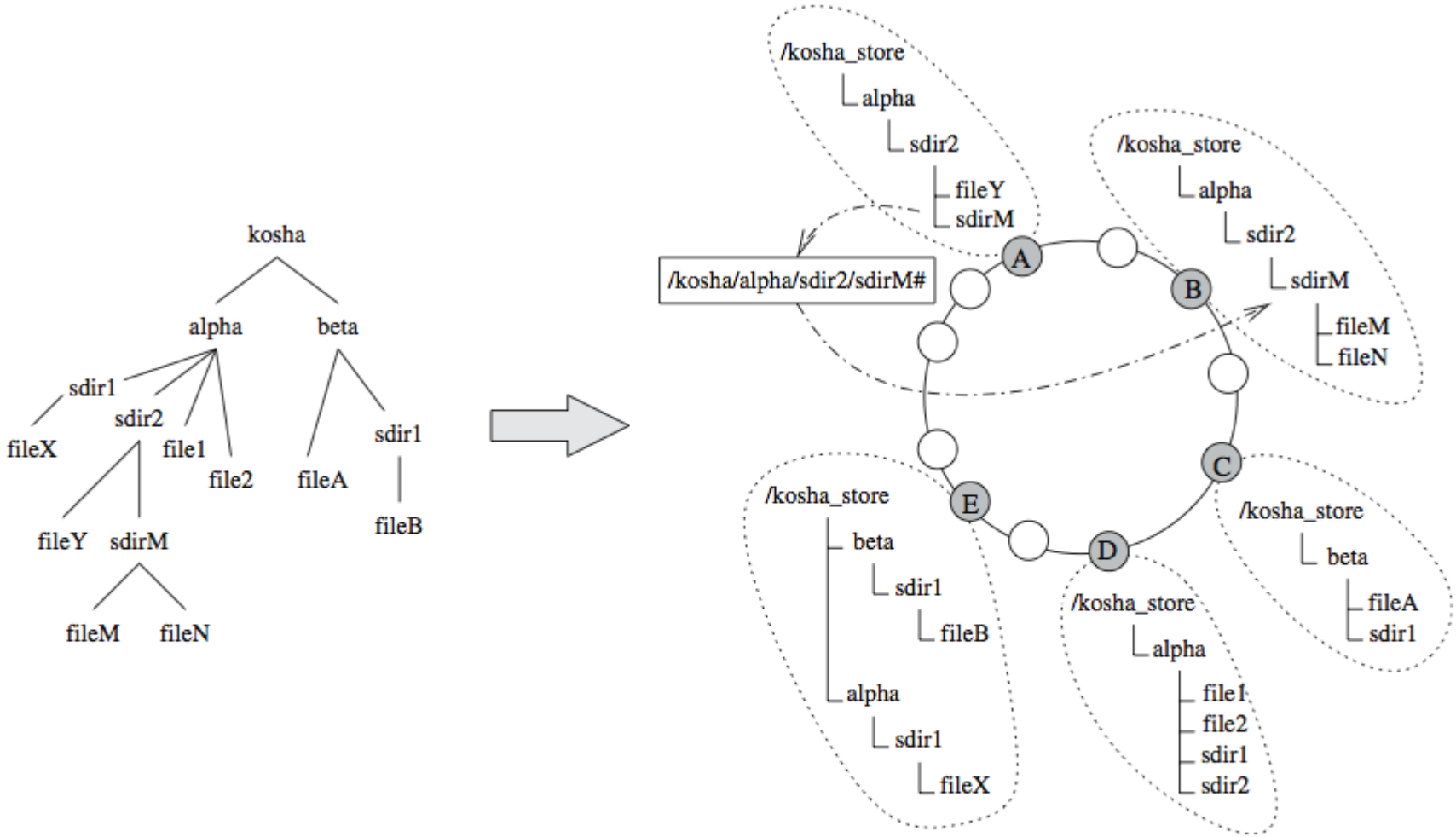


Figure 3 Example of subdirectory distribution to multiple nodes. The files in the same directory are stored on the same node as the parent directory. However, the subdirectories are distributed to remote nodes. The distribution level is set to 2, and *A* has limited capacity which causes */alpha/sdir2/sdirM* to be redirected to *B*. The example contents of the special link are shown in the *rectangle*. Node *B* is chosen for storing the redirected subdirectory because $DHT(hash(sdirM\#)) = B$.

Kosha design

- When an operation on /kosha occurs
  - Kosha determines node on which a file is stored
    - Hash, lookup, maybe redirection
  - NFS RPC is modified to occur on /kosha_store on selected node, rather than /kosha on client node
  - Selected node performs operation and returns results to Kosha
  - Kosha returns control to the client

NFS Operations Support

- All accesses to a file are guaranteed to be sent to the same storage node
  - i.e. Every user sees same instance of file
  - In case of failure, Kosha can still provide access to a file (unlike NFS)
- Virtual File Handles
  - A handle to a file in Kosha
  - Kosha keeps a mapping table client file handle -> virtual file handle
  - i.e. Location transparency

Managing Replicas

- Kosha maintains K replicas of a file on the neigbouring K nodes in the node identifier space (not proximity!)
  - Randomly assigned, so dispersed for godo fault tolerance
- For each file, there is a primary replica
  - i.e. *DHT(hash(dir1)) : kosha_store/dir1/file1 (from earlier)*
  - All accesses to the file are sent to the primary replica
    - It is responsible for mataining the K replicas of the file on K neighbouring nodes

Node Addition and Failure

- If a node fails, any files with primary replicas there will have a new primary replica chosen from one of the K replicas
- Pastry handles nodes leaving/joining and notifies a Kosha node, N, when any nodes in N's leaf set are affected.
  - Kosha then adjusts where it stores its files accordingly

Node Addition and Failure

- Pastry evenly divdes key space between adjacent nodes
  - Node with nodeId numerically closest to the file key will be responsible for the file
- When a node is added, only its immediate two neighbours need to be modified
  - Because file keys on the neighbous nodes may be numerically closer to the new node
  - Kosha examines files stored on N and N's leaf set to determine which files need to be moved
- This migration of files ensures a new node has the files for which it is the primary node

Node Addition and Failure

- Pastry informs Kosha when a replica node fails or a new node is added via a callback function
  - The local Kosha then creates a copy of files it is the primary replica and sends the copy to the newly added node which now serves as one of the K replicas.
- All Kosha data on a revived node is purged
  - Will have a new key, so old files no longer stored here

Transparent Fault Handling

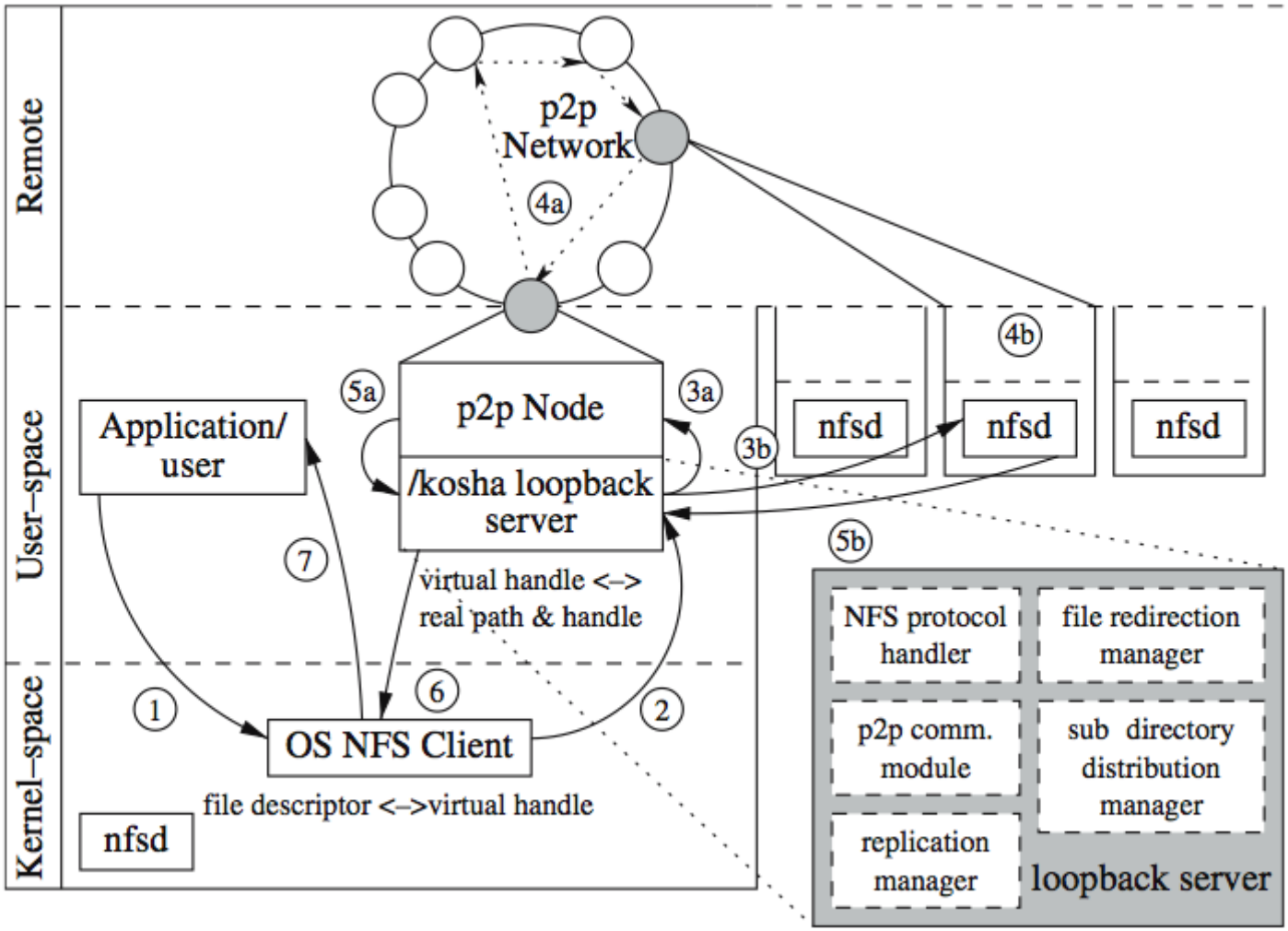
- When a primary node fails:
  - Kosha detects an RPC error
  - Removes mapping in virtual handle table
  - Does a lookup on the requested file thus finding a replica to become the primary node

Load Redistribution

- DHT provides good load balancing
  - but it is possible that a node will be close to full while neighbouring nodes store little data
- So, we perioidically redistribute files
  - Proactive load balancing
  - When a node is a pre-specified percent full
    - Contacts its left and right neighbours and if they have space, moves some files there
  - Only one node can transfer files to another at a time
    - To prevent multiple nodes dumping files on the same node

Software Architecture

Figure 4 Kosha architecture: **1** application makes an I/O system call, **2** kernel makes an RPC call, **3a** local port request to peer substrate or **b** handle substituted and RPC forwarded, **4a** overlay locates node storing file or **b** file I/O occurs, **5a** local port reply from peer substrate or **b** I/O result returned, **6** RPC returns with virtual handle or result, **7** system call returns control to application.



Evaluation

- Compared modified Andrew benchmark to NFSv3
- 16-nodes
  - 4-nodes of 2GHz P4, 512MB RAM, 40 GB
  - 12-nodes of 3GHz Xeon, 2GB RAM, 146GB
  - Distribution level 1 (no subdirs distrubted) to remove effect of subdir distribution
  - Replication level 1
  - 35 GB space alloced (i.e. More than enough so redirection doesn't occur)
- NFS tests used two nodes (client, server)

Evaluation

Performs better than NFS with one node because communications are local to the node

Under 6% overhead for 16 nodes. Only 4.5% additional overhead going form 2 to 16 nodes.

Table 1 Performance of a modified Andrew benchmark on Kosha with increasing number of nodes.

Benchmark	NFS exec. time	Kosha				
		1 Node	2 Node	4 Nodes	8 Nodes	16 Nodes
		exec. time (overhead)	exec. time (overhead)	exec. time (overhead)	exec. time (overhead)	exec. time (overhead)
mkdir	2.242	2.241 (1.000)	2.243 (1.000)	2.259 (1.008)	2.261 (1.008)	2.304 (1.028)
copy	17.503	16.496 (0.942)	17.401 (0.994)	17.601 (1.006)	17.791 (1.016)	18.207 (1.040)
stat	1.531	1.513 (0.988)	1.533 (1.001)	1.534 (1.002)	1.537 (1.004)	1.595 (1.042)
grep	3.709	3.235 (0.872)	4.026 (1.085)	4.030 (1.087)	4.092 (1.103)	4.181 (1.127)
compile	21.897	22.933 (1.047)	23.231 (1.061)	23.560 (1.076)	23.860 (1.090)	24.254 (1.108)
Total	46.882	46.418 (0.990)	48.434 (1.033)	48.984 (1.045)	49.541 (1.057)	50.541 (1.078)

The table shows average execution times for each phase and the respective overhead of Kosha compared to NFS. The distribution level for Kosha was fixed at 1 for these measurements. All times are in seconds.

Evaluation

Overhead in subdirectory distribution for levels relative to level 1:

- level 2 4.8%
- level 3 6.3%
- level 4 6.8%

i.e. Having a large distribution level is OK

Table 2 Performance of a modified Andrew benchmark on Kosha as the distribution level is increased. For these measurements, the number of nodes was fixed at 8. All times are in seconds.

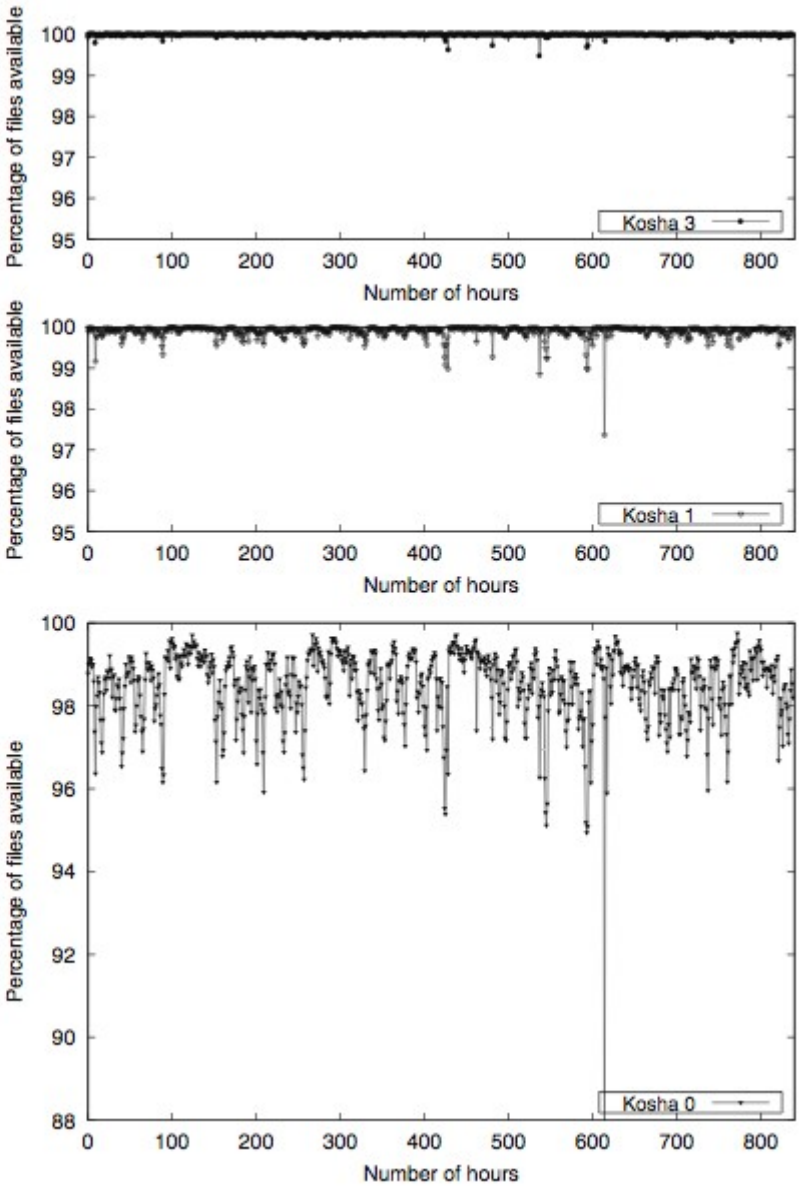
Benchmark	Dist-level 1	Dist-level 2	Dist-level 3	Dist-level 4
	exec. time	exec. time (overhead)	exec. time (overhead)	exec. time (overhead)
mkdir	2.261	2.483 (1.098)	2.623 (1.160)	2.692 (1.191)
copy	17.791	19.231 (1.081)	19.286 (1.084)	19.301 (1.085)
stat	1.537	1.752 (1.140)	1.786 (1.162)	1.801 (1.172)
grep	4.092	4.297 (1.050)	4.331 (1.058)	4.352 (1.064)
compile	23.860	24.147 (1.012)	24.623 (1.032)	24.770 (1.038)
Total	49.541	51.910 (1.048)	52.649 (1.063)	52.916 (1.068)

Evaluation

- File replication overhead
  - Used replica of NFS file setup from department server on a five-node Kosha server
  - Failed one node
    - Took 37 mins to copy 12.2 GB of data
    - Since mean time to failure should be days in expected usage, this is small

Fault Tolerance

Figure 7 Percentage of total files that are available over a period of 840 h. The distribution level was fixed at 3 for these results. The largest number of failures (4, 890) occurred at hour 615, where over 12% files became unavailable for Kosha-0 compared to only 0.16% for Kosha-3.



Used an availability trace (up or failed, recorded hourly) of 51,633 machines in a large corporation over a consecutive 35-day (840h) period.

Simulated Kosha for 51,663 machine cluster and used this data to introduce failures.

Kosha-0 (i.e. No replicas) shows system performance is affected when large number of failures occur.

Kosha-1 increases availibility significantly.

Kosha-3 gives 99.991% availibility! (since new replicas are created when old ones become unavilable)

Related Work

- Gnutella, Freenet, Kazaa
  - Basic P2P large-scale data sharing
- Oceanstore, CFS, PAST
  - Strong persistence and reliability
- Ivy, Farsite, Pangaea
  - Wide-area file systems with reliability
  - Kosha instead extends NFS (i.e. LAN)
- NFSv4 provides system replication (static only) and migration
  - Kosha utilises unused space on cluster nodes and desktops.
  - Kosha could make use of NFSv4 in a later version.