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

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

The need for sharing resources

- Scientific applications
  - Complex computations
  - Large data sets
- Dedicated resources
  - Expensive
- Modern workstations
  - Powerful resource
  - Available in large numbers
  - Underutilized (CPU cycle, storage)

Disk space survey

- 500 instructional machines
  - 90% local disk space unused
  - 75%+ space used on central NFS servers
  - Expensive maintenance

    (quotas, regular addition, explicit backups)

Network File System (NFS)

- Widely-used in academic and corporate setups
- Provides remote access to shared files
- Supports local file system abstraction

- Server explicitly *exports* directories
- Client explicitly *mounts* directories

Our contribution: Kosha NFS enhancement via peer-to-peer

- Aggregates unused disk space on nodes to provide a single shared file system abstraction
  - Fault tolerance
  - Load balancing
- Implemented as an NFS enhancement
  - Preserves NFS semantics
  - Entails no changes to the OS
  - **More likely to see actual deployment**
- Achieves acceptable performance by distributing directories

Agenda

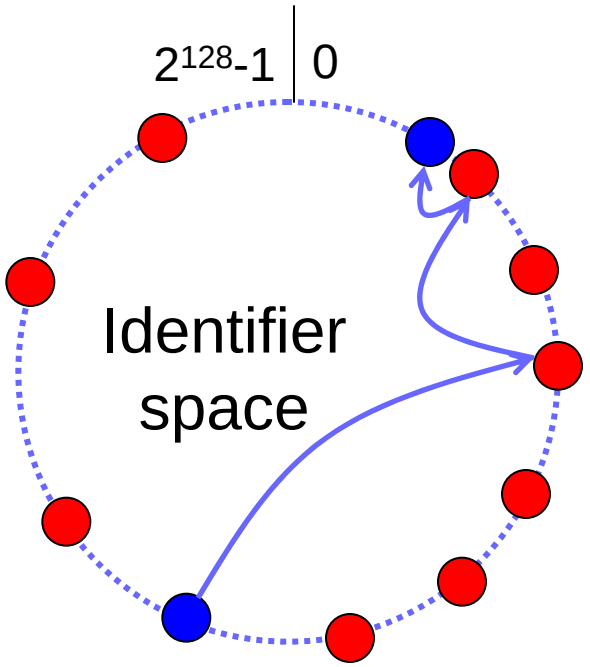
- **Background: DHTs**
- Proposed scheme
- Implementation and evaluation
- Conclusions

Distributed Hash Table (DHT)

- Peer-to-peer overlays with imposed structure
  - Each node has a unique random `nodeId`
  - Each message has a key
  - The `nodeId` and key reside in the same name space
- DHT: routes a message with a key to a unique node
- DHT abstraction is preserved in the presence of node failure/departure

Pastry

- 128-bit circular identifier space
- DHT: A message is routed to a node with `nodeId` numerically closest to the key
  - O(*log N*) routing state per node
  - $Log_{16}N$ overlay hops
- Each node maintains information about K neighbors



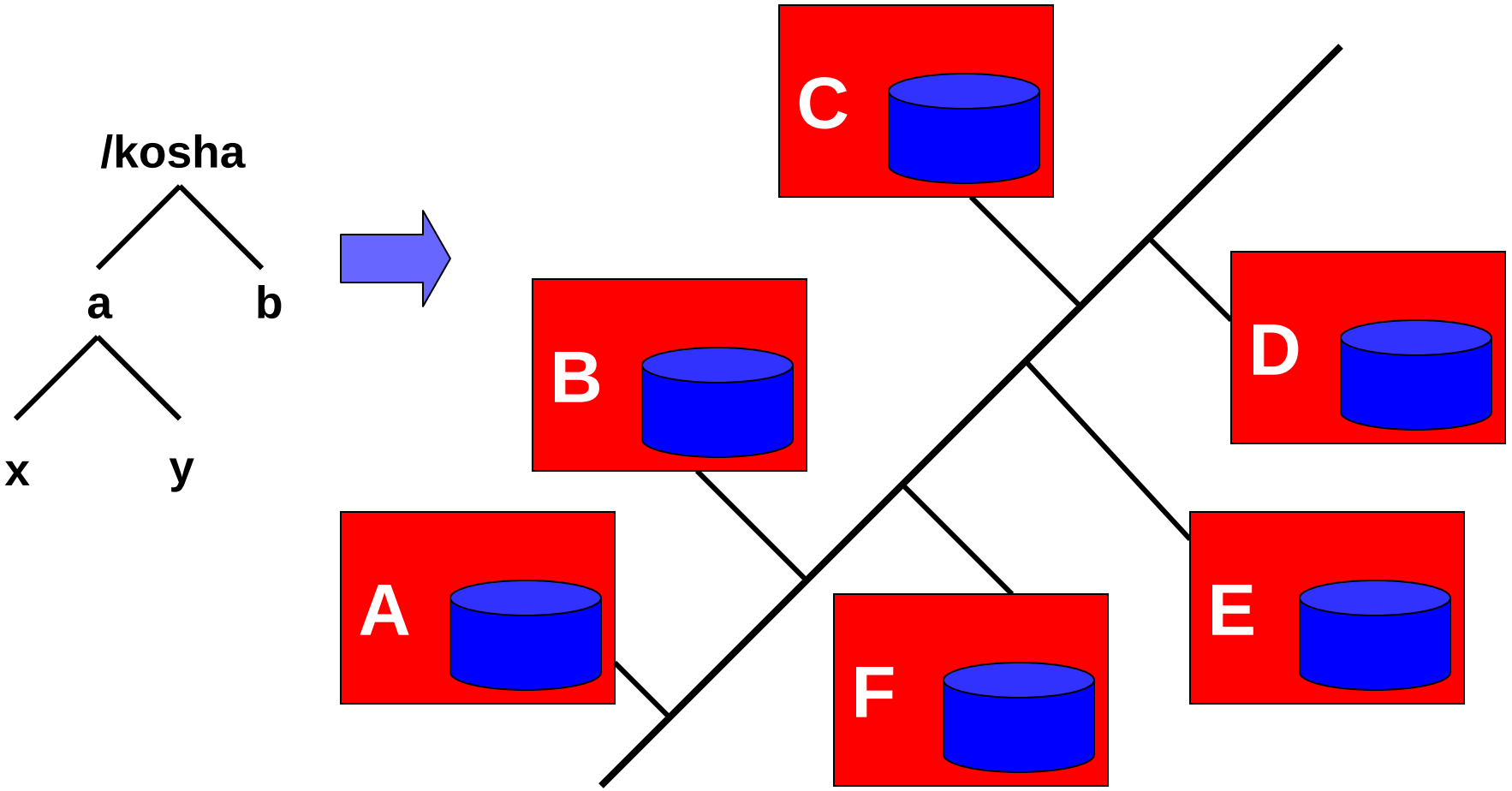
Agenda

- Background: DHTs
- **Proposed scheme**
- Implementation and evaluation
- Conclusions

Kosha tasks

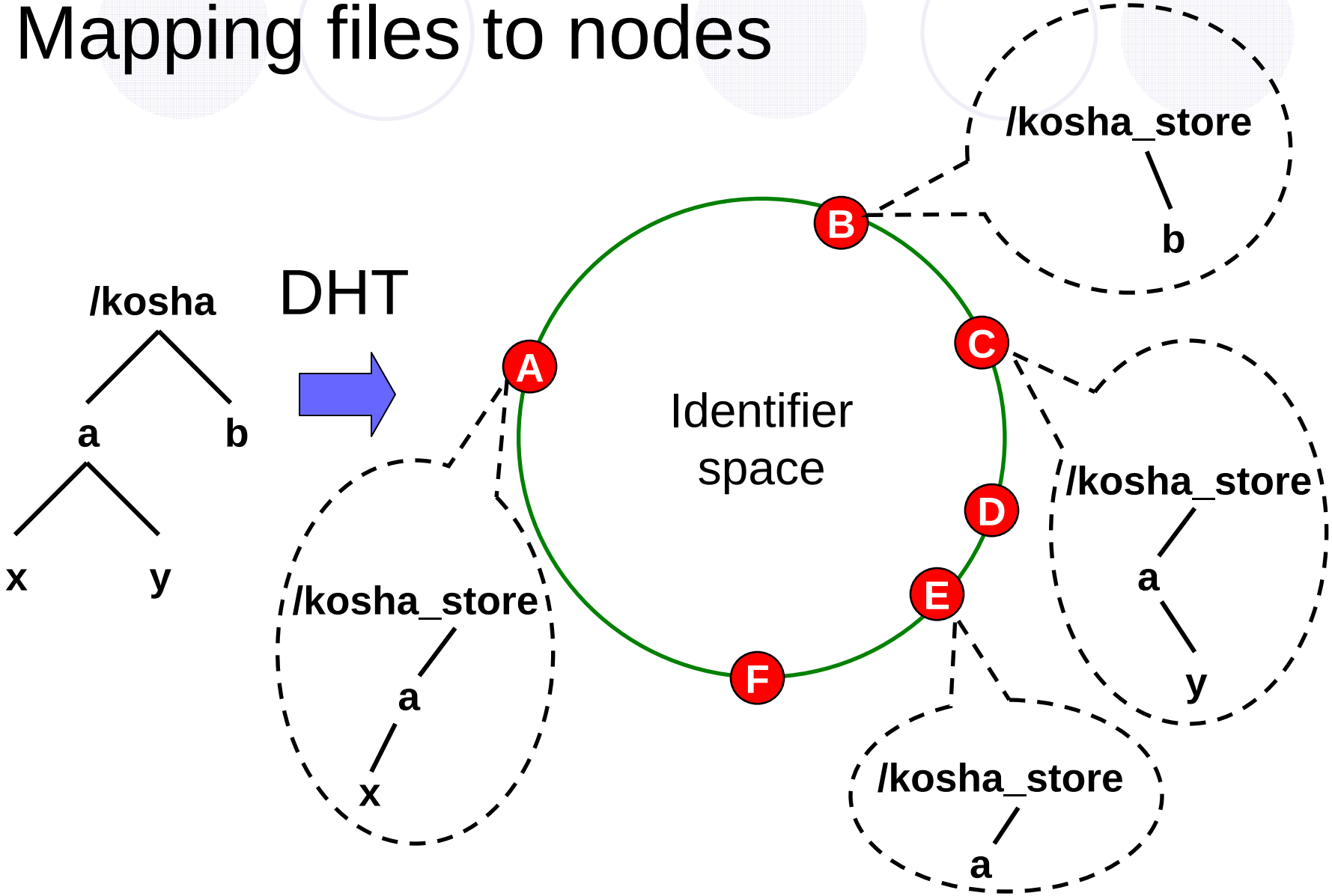
- **Distribution of files to nodes**
- Leveraging NFS
- Granularity control of distribution

File distribution



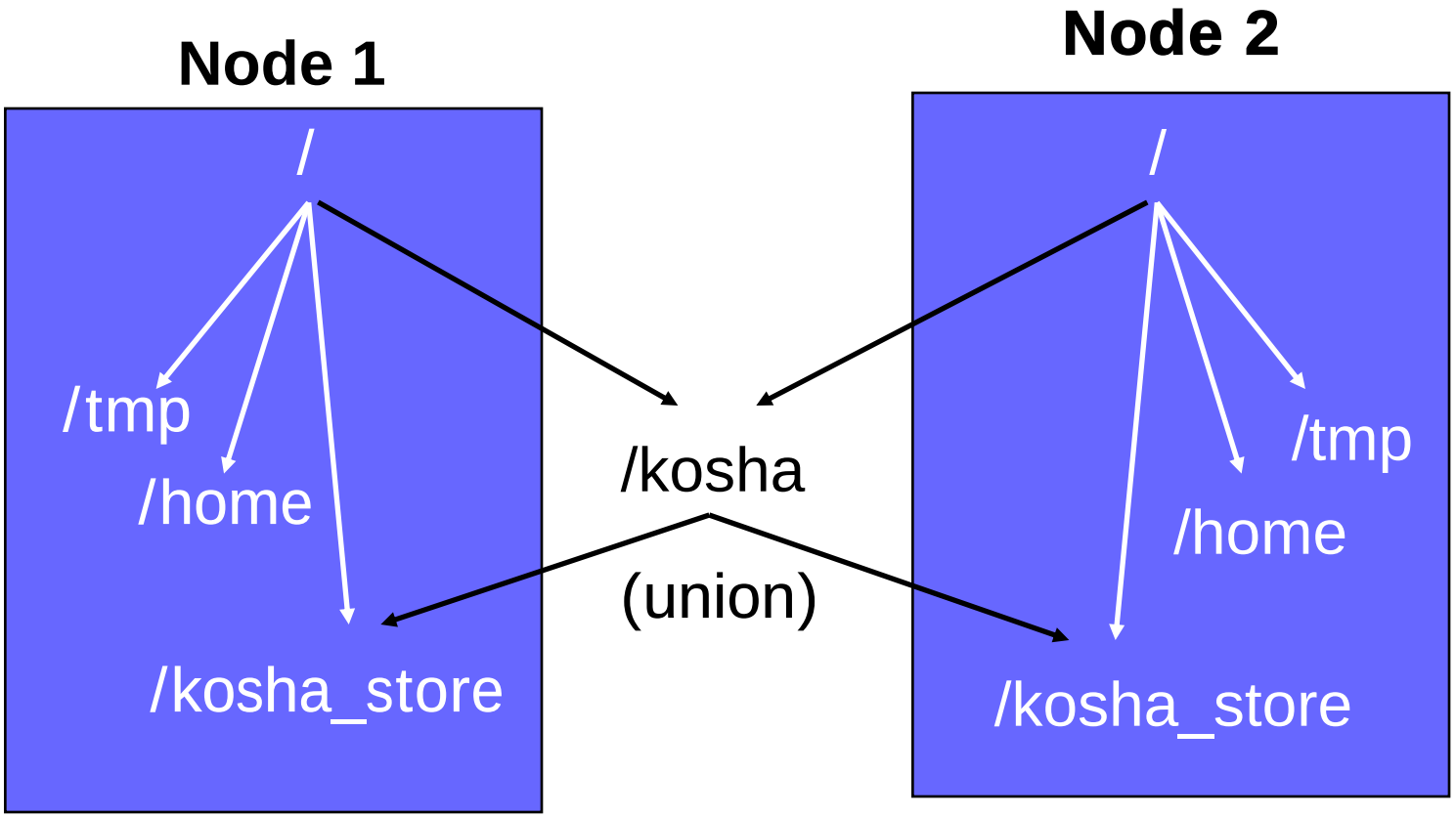
How to uniquely map files to nodes in a decentralized manner?

Mapping files to nodes



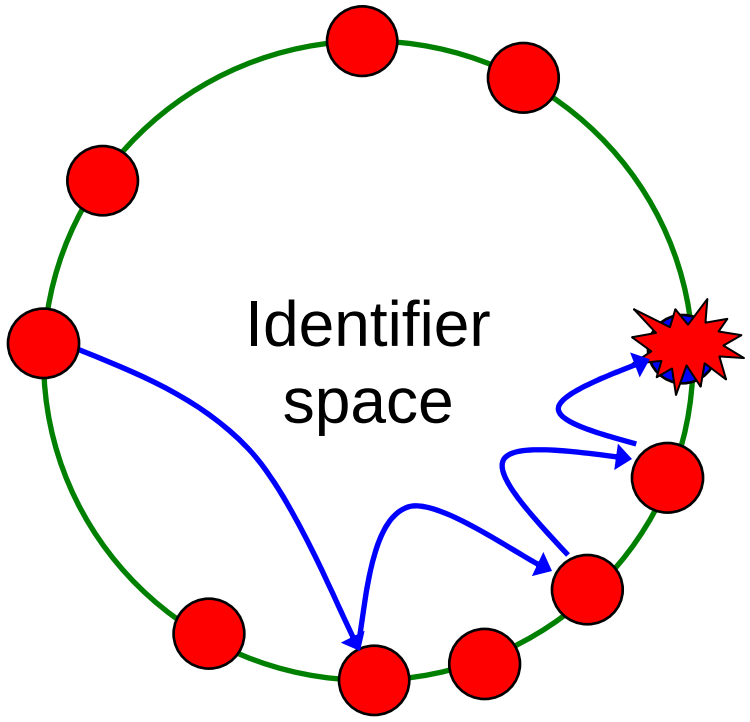
*/kosha/**a*** ➔ DHT(*hash*(***a***)):*/kosha_store/**a***

Virtual directory abstraction



Fault tolerance

- Employ lazy replication
  - Periodical update using rcp
- Data since the last replication is lost



Kosha tasks

- Distribution of files to nodes
- **Leveraging NFS**
- Granularity control of distribution

Leveraging NFS

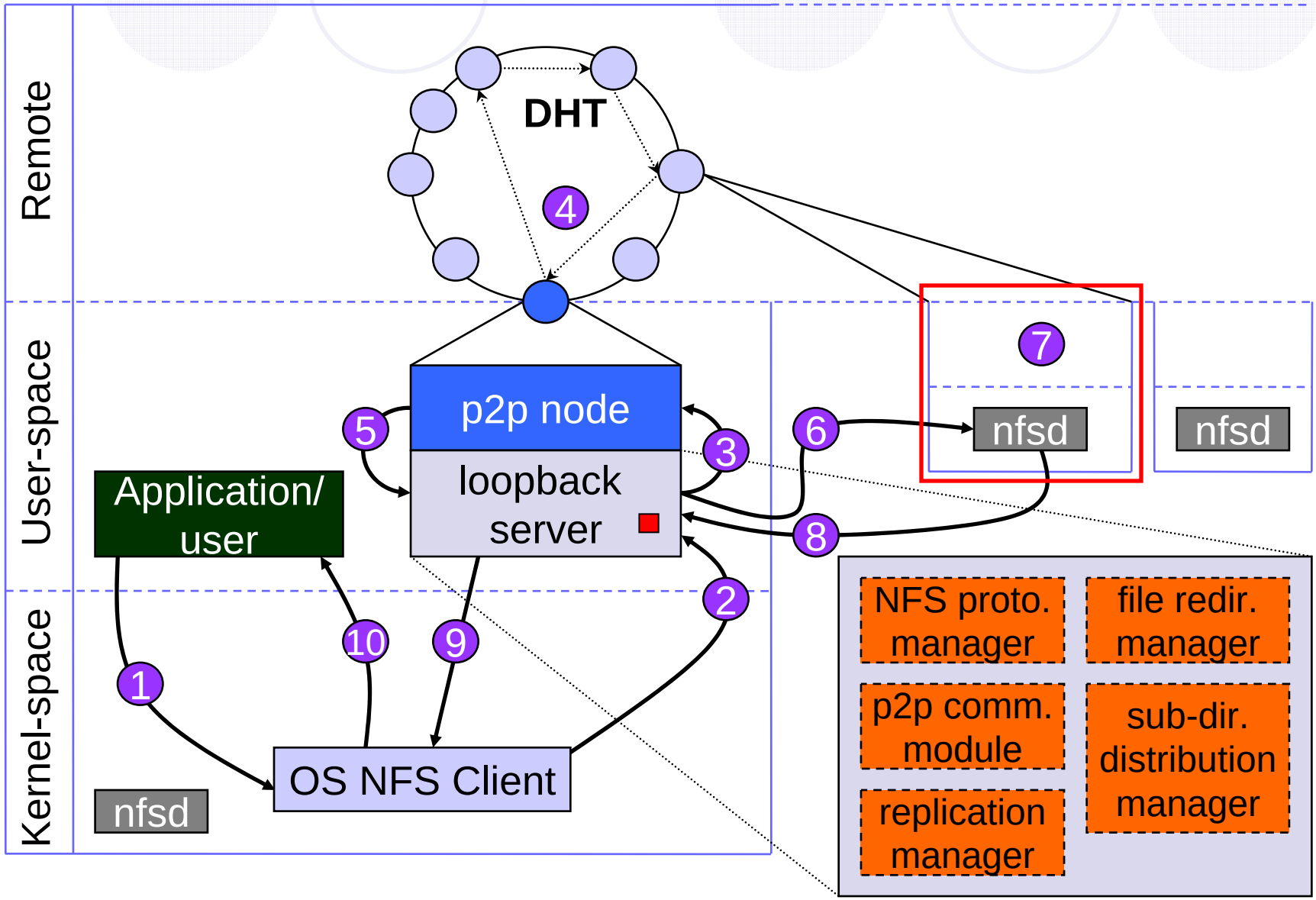
- Participating nodes run standard NFS servers
- Nodes also run Kosha server
- I/O to */kosha* sent to the local Kosha server
  - Leverage loopback server
- Kosha maps requests to appropriate remote nodes

Loopback server

- A modified NFS server that executes on the same node as the client
- Essentially send I/O requests made to */kosha* to the local Kosha server
  - Conceptually equivalent to:

    `mount localhost:koshaPort /kosha`
- Leverages opaque file handles
  - Unique identifier per handle can be substituted for server-issued handles

Kosha operation



Kosha tasks

- Distribution of files to nodes
- Leveraging NFS
- **Granularity control of distribution**

Drawbacks of distributing files

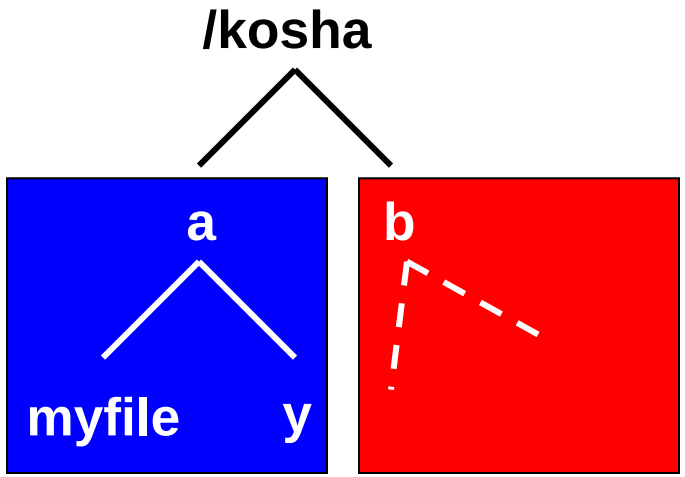
- Distribution of individual file can be costly
  - Requires DHT operation on access to each file

- Observation: users typically access many files in the same sub-directory together

Solution: Distribute sub-directories

- All files in a directory are stored on the same node

*/kosha/**a**/myfile* ➔ DHT(*hash*(***a***)):*/kosha_store/**a**/myfile*



Load balance

- Disk spaces contributed by nodes are not uniform
- Sizes of sub-directories vary

➔ Capacity limit can prevent a file to be stored on the node determined by DHT

- Solution: Redirect such files to different nodes

Agenda

- Background: DHTs
- Proposed scheme
- **Implementation and evaluation**
- Conclusions

Software

- Implemented as a daemon: *koshaD*
  - Leverages FreePastry 1.3 API
  - Runs on participating nodes
  - Manages self-organization of nodes
- Leverages Secure File System (SFS) toolkit
- Runs on FreeBSD 4.6

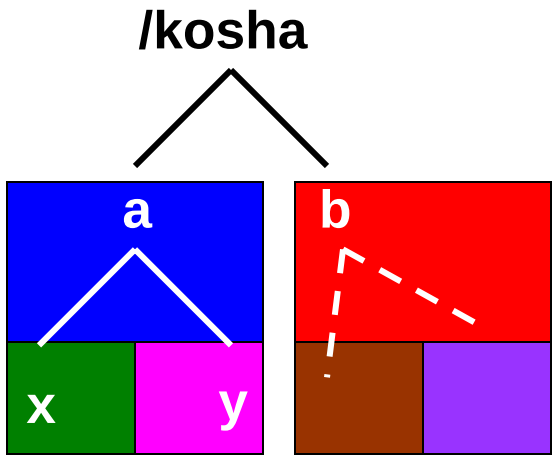
Evaluation: Setup

- Eight nodes
  - 2.0 GHz P4, 512MB RAM
  - 40GB 7200 RPM Barracuda Seagate hard disk
  - Connected via 100Mb/s Ethernet

Distribution level

- Number of levels of sub-directories distributed individually

Distribution level: 1 2



Measured results:
Running time for a modified Andrew Benchmark

- For 8 nodes, distribution level 1, only 5.6% overhead added to unmodified NFS
- For 8 nodes, distribution level 4, additional overhead less than 10%
- Overhead scales logarithmically with network size

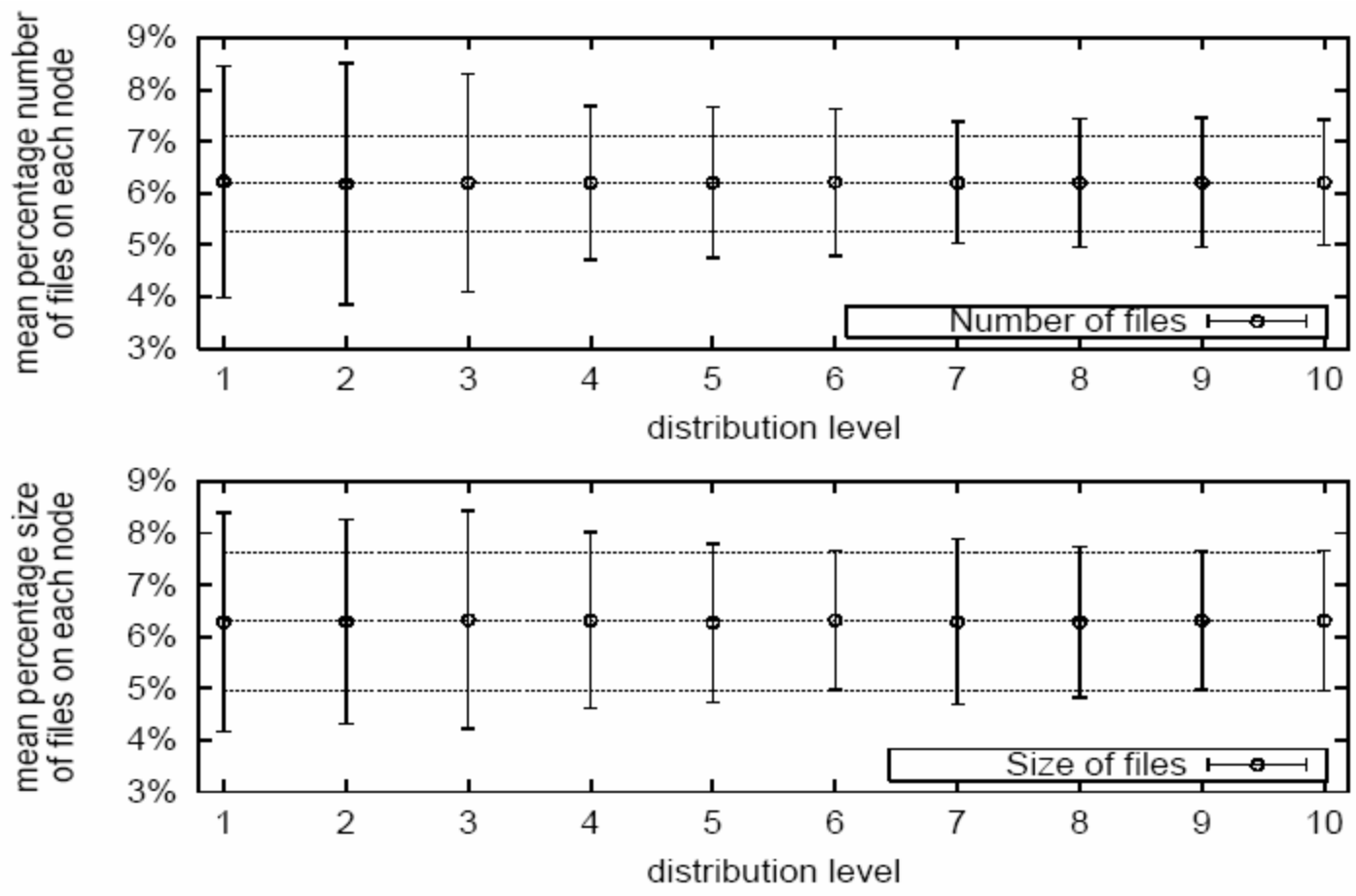
$$\text{Overhead} = I + (H * hc) * \frac{(N-1)}{N}$$

N	number of nodes in the network
I	interposition overhead
H	number of hops a message travels in the overlay (*log(N)*)
hc	average latency of each hop

Simulation

- File system trace from Purdue ECE servers
  - 221k Files of 130 users
  - 17.9 GB data
- Machine availability trace from Microsoft Corp.
  - 51k+ nodes
  - 35 days
  - Status of machines recorded hourly

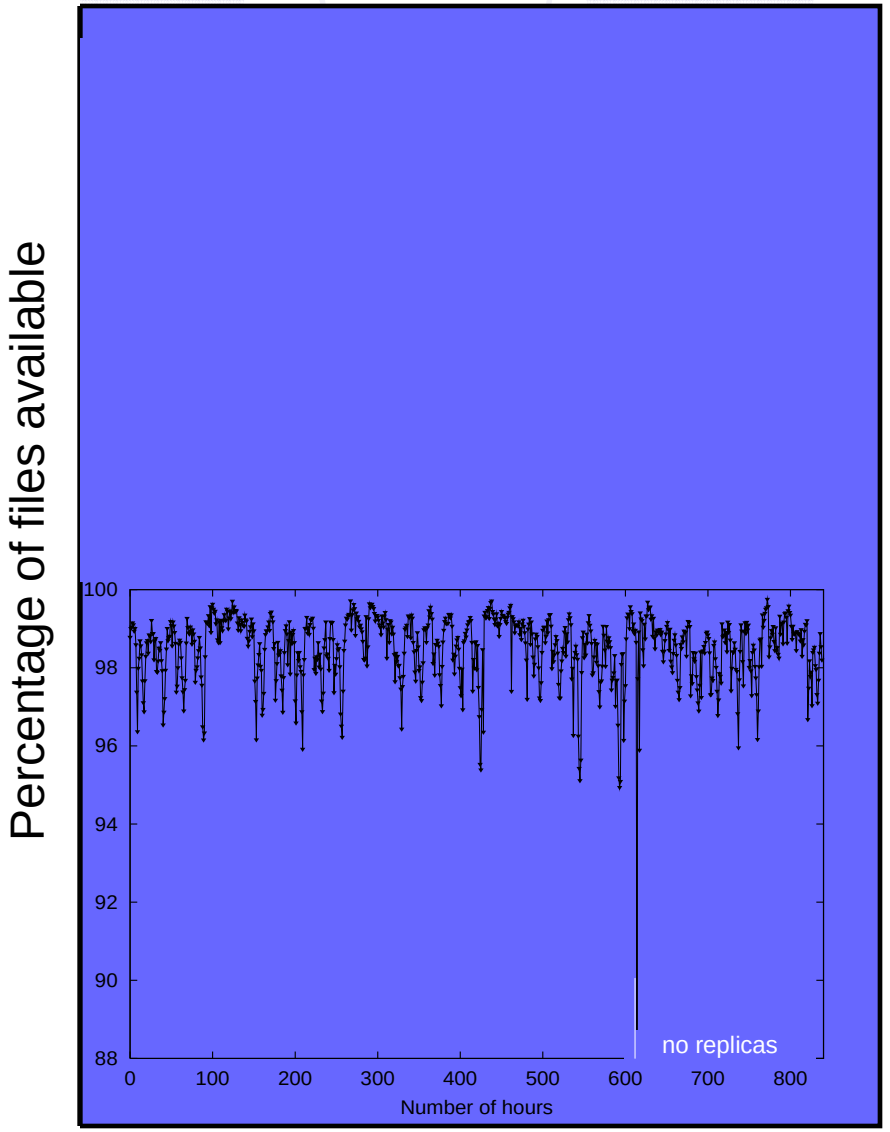
Load balance



Distribution level 4 or better achieves acceptable load balance

Fault tolerance

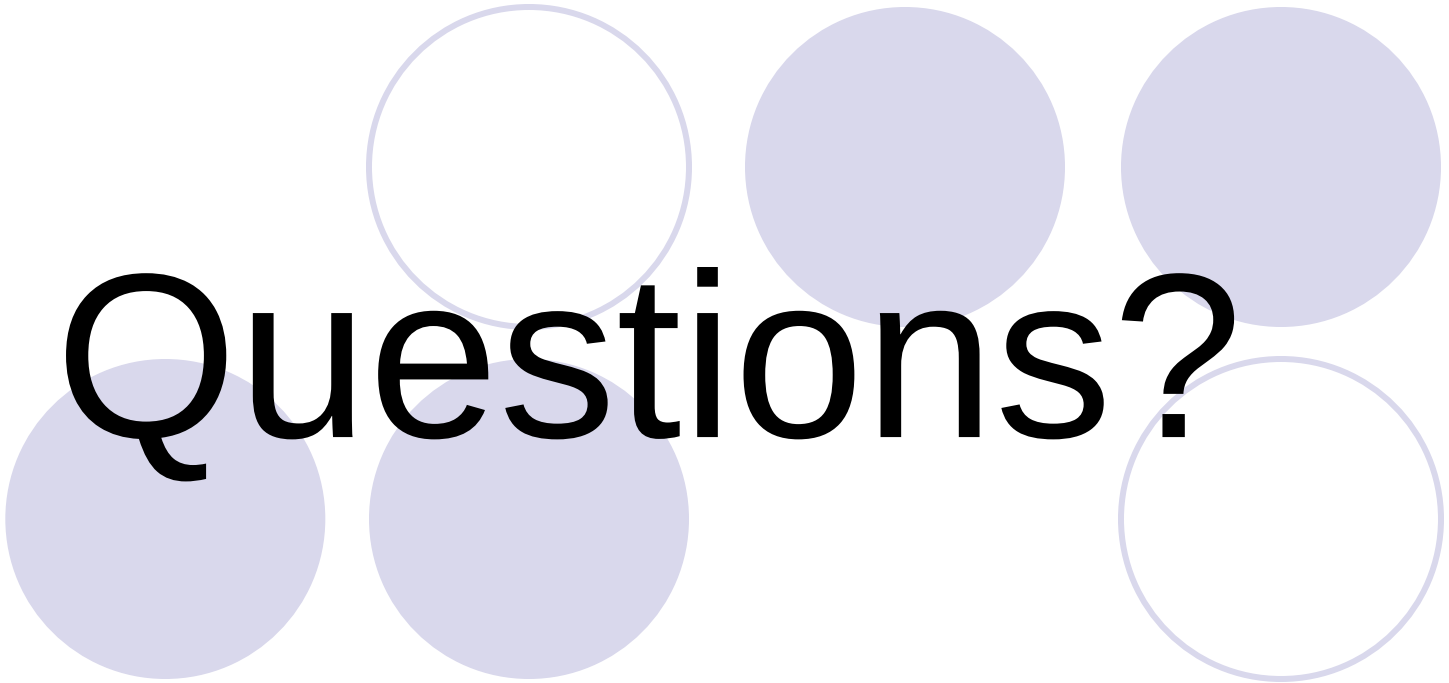
- 99.99% or better file availability



Available files over time

Conclusions

- Kosha blends strengths of NFS and DHTs
  - Aggregates unused disk space and exports a single shared file system abstraction
  - Implemented as an NFS enhancement
  - Imposes low overhead by distributing directories
- Kosha is more likely to see actual use than schemes that require changing NFS


Questions?