A User-Transparent Recoverable File System for Distributed Computing Environment

Hyeong S. Kim
Heon Y. Yeom

Seoul National Univ.
Seoul, Korea

Presented by Hyeong S. Kim

Outline

- Introduction
- Previous works
  - MPICH-GF
- ReFS
  - Architecture
  - Implementation
- Evaluation
- Conclusions and future works

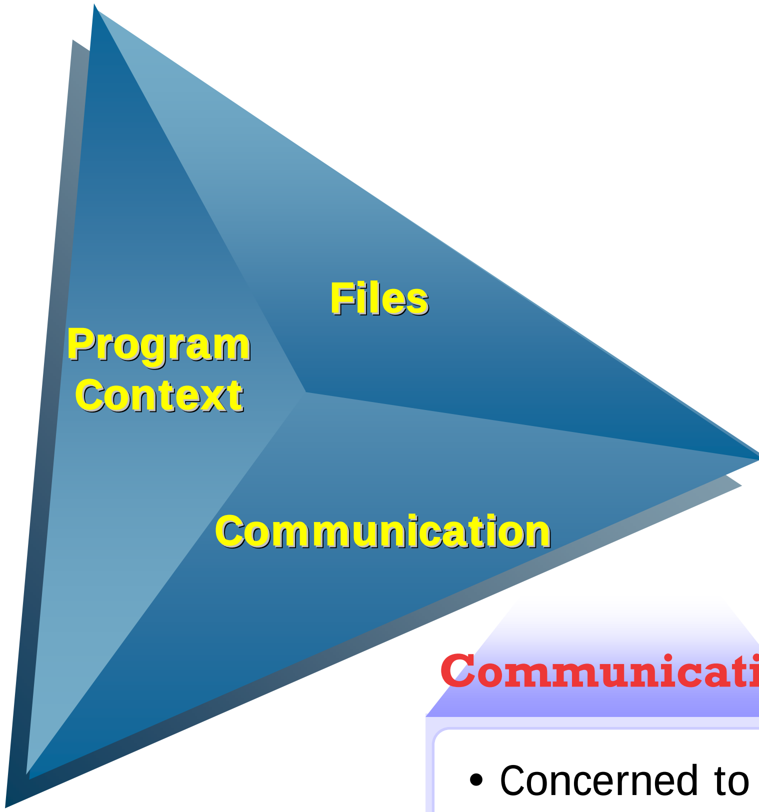
Challenges

- The nature of the distributed computing environment including the Grid
  - Require that the system endure sudden failures ➔ Fault-Tolerance
- Our goal
  - Construct the practical fault-tolerance system for message-passing applications on the Grid

Fault-Tolerance – Things to Cover

Program Context

- Concerned to the address space of the process
- Keep the full image and context of the process in a reliable storage
➔ Checkpointing mechanism



Files

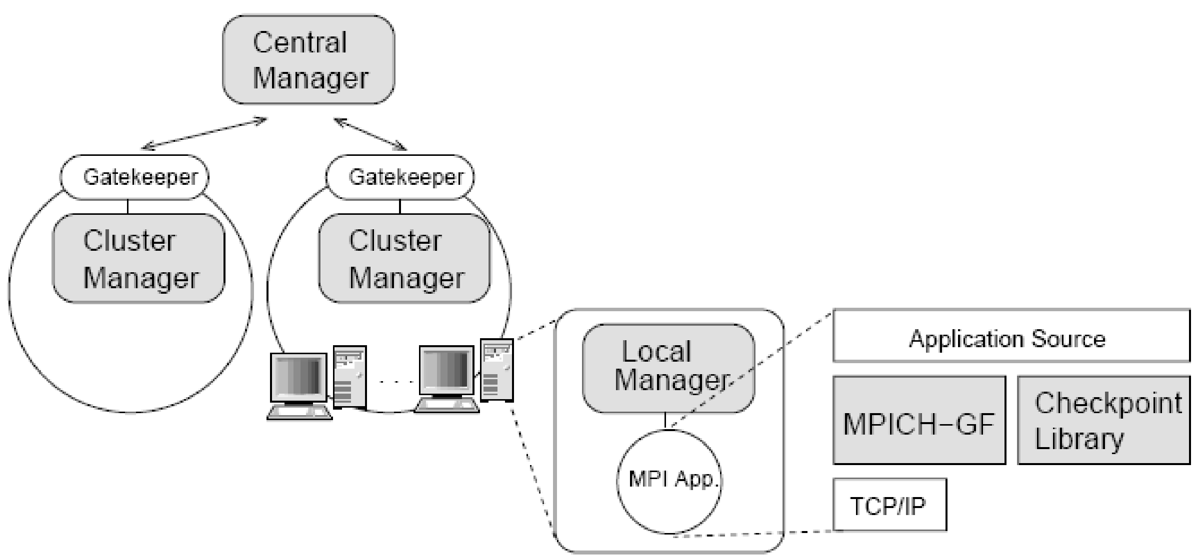
- Concerned to the files
- The files should be remained unchanged when the process rollbacks to the previous checkpoint state

Communication

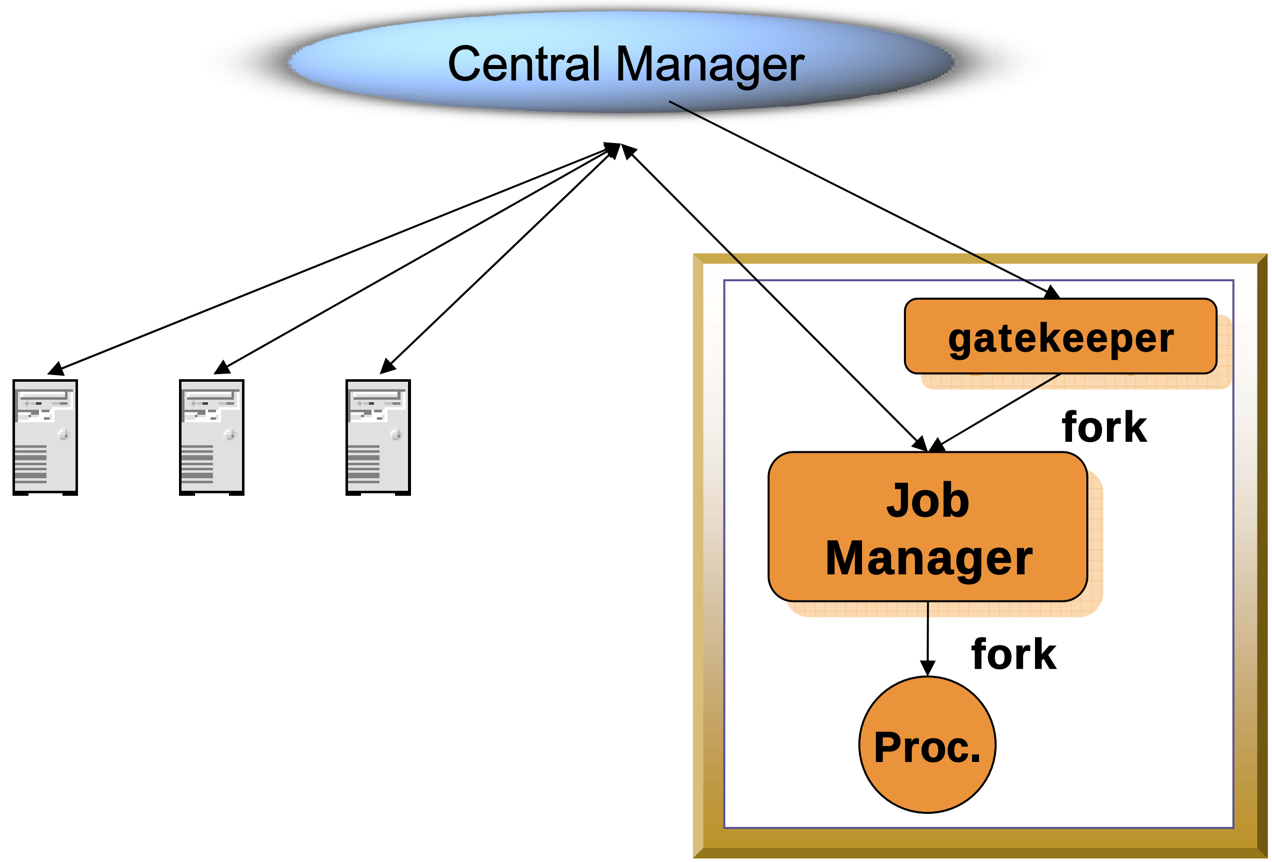
- Concerned to the messages
- Any in-transit message during checkpointing is never lost

MPICH-GF

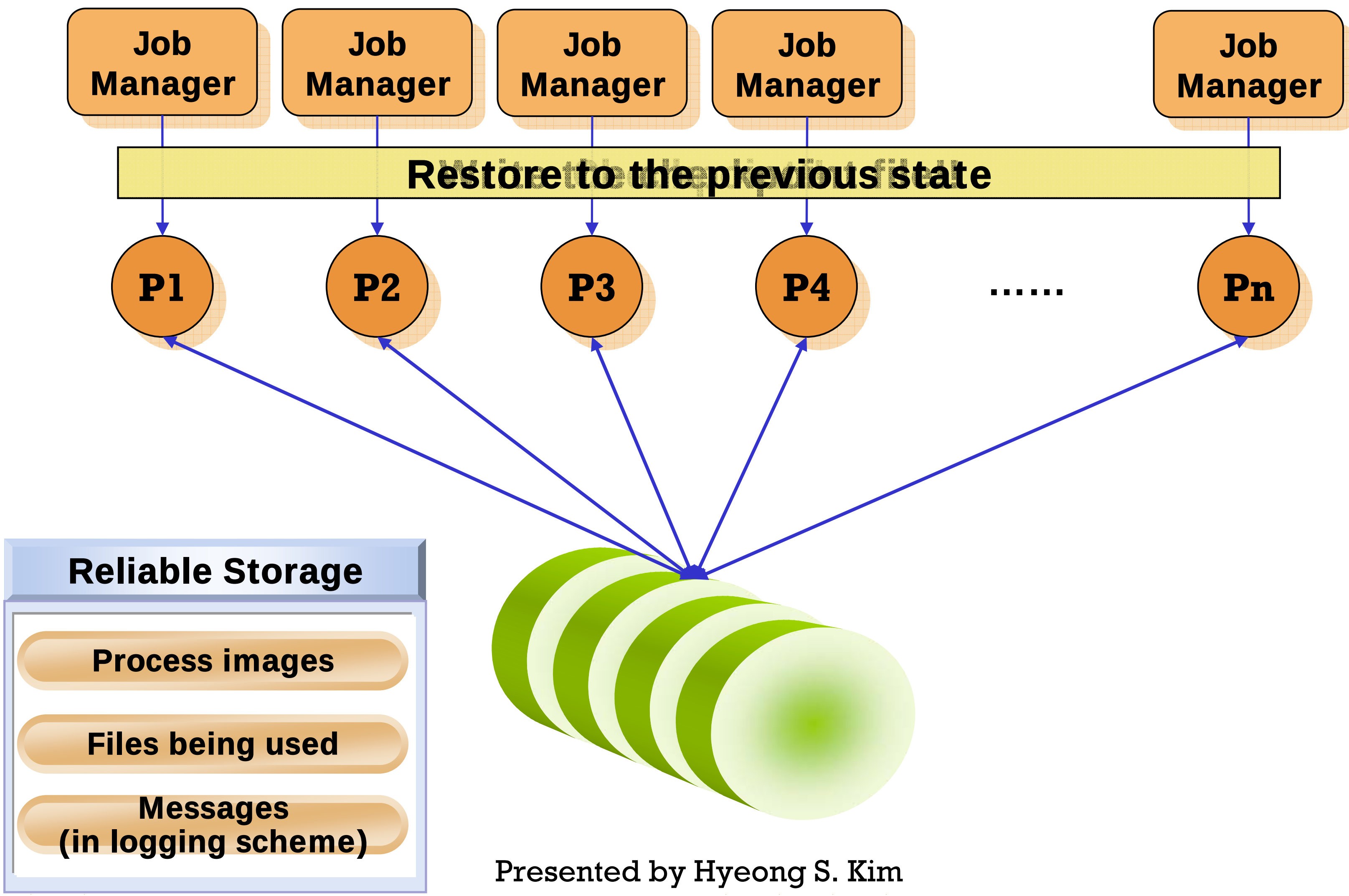
- Based on MPICH-G2 (MPICH for Globus 2)
- Equipped with task migration, dynamic process management for MPI, and message queue management
- Integrated with checkpoint library
- Recovers the communication channel as well as the process itself
- User transparent



Job Submission in MPICH Globus



Scenario



Job Manager
Job Manager
Job Manager
Job Manager
Job Manager
Restore to the previous state
P1
P2
P3
P4
……
Pn
Reliable Storage
Process images
Files being used
Messages
(in logging scheme)

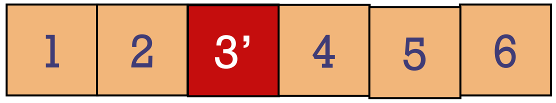
ReFS Sketch

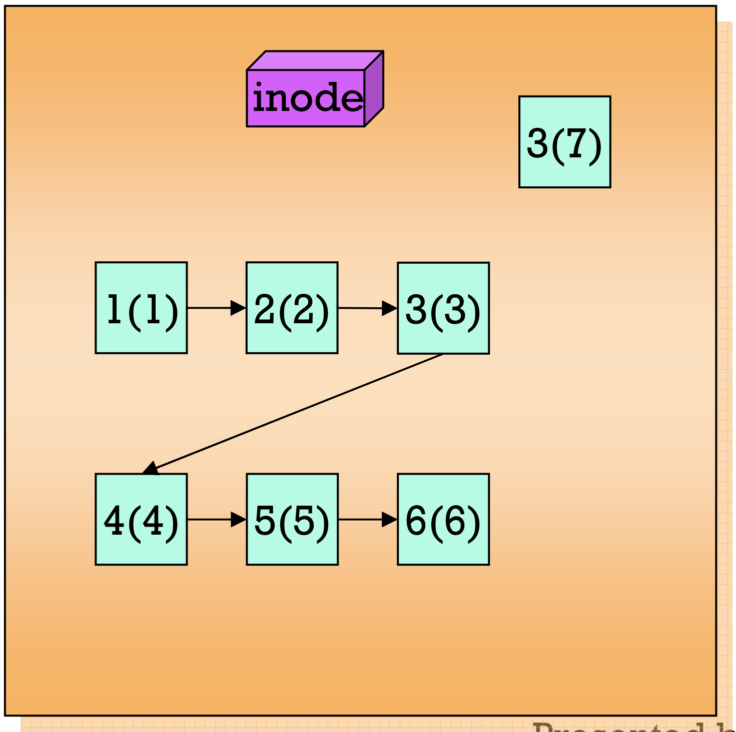
- Idea
  - It is not our concern whether a file is modified or not between the checkpointings
  - Rather, we focus on the property that the file should be unchanged after the most recent checkpoint has been done
- A kind of Versioning File Systems
  - Retain earlier version of modified files allowing recovery from user mistakes or system corruption
- An easy integration method with MPICH-GF or other fault-tolerant systems → simplified system calls
- No overwrite on existing data block
  - A concept of Log-Structured File System

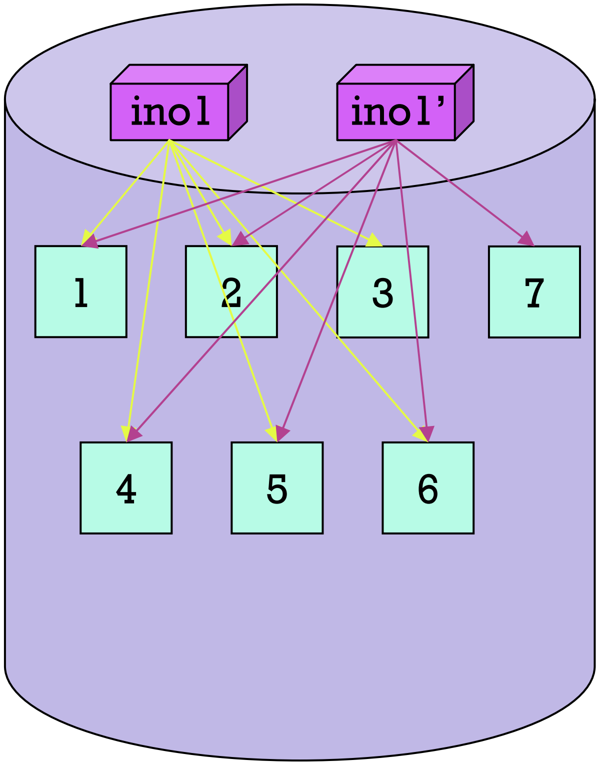
Example

Move to block 3;

write(A, "sample text");



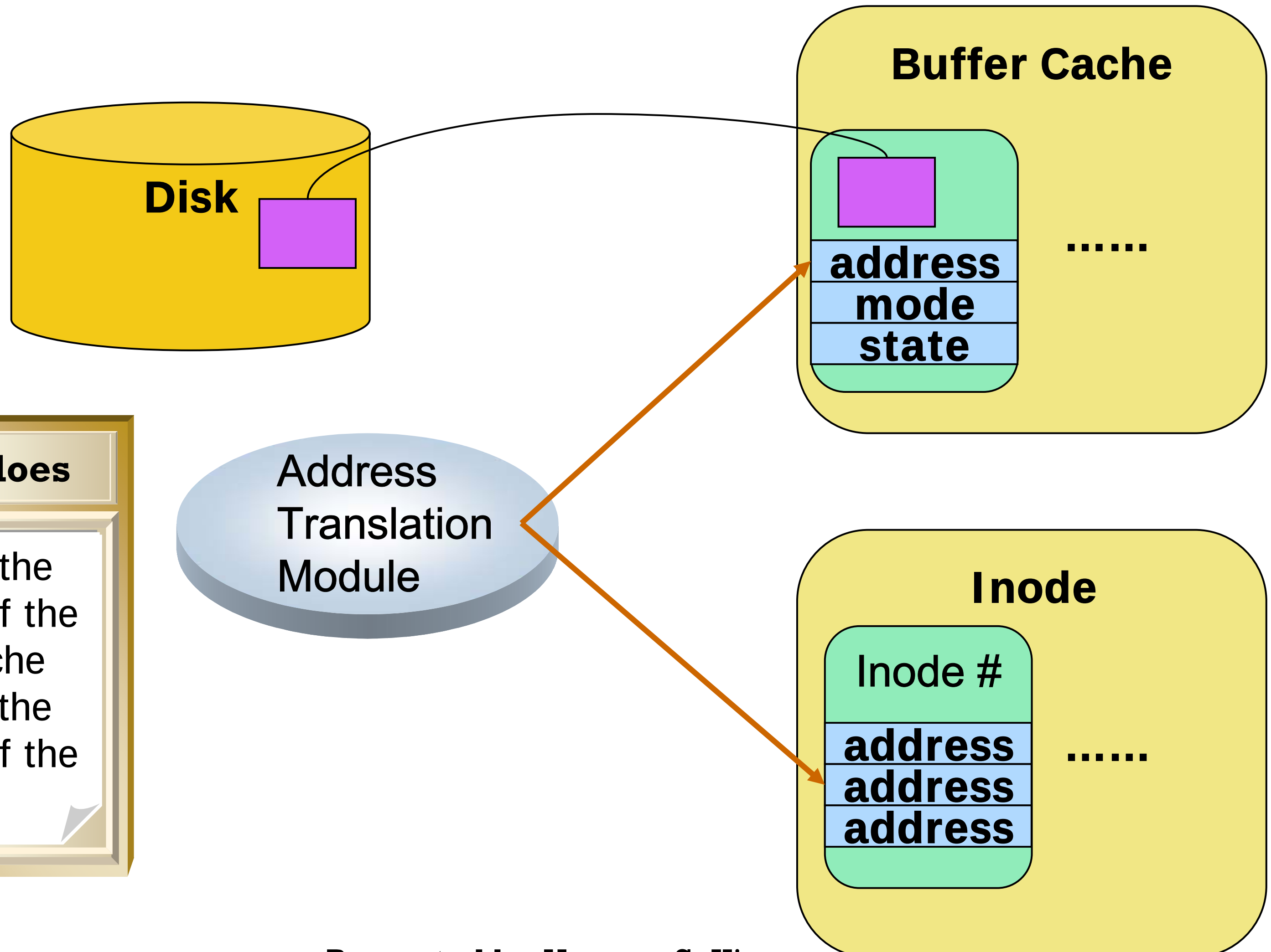




ReFS

- VFS
  - Upper layer of the file system
  - Common layer among the target file systems
- Target file system
  - Lower layer of the file system such as ext2, ext3, …
- Address Translation Layer
  - Responsible for modifying the address of the block
  - A layer that prevents the write from overwriting the existing blocks

Address Translation Layer



What he does

- Modifies the address of the buffer cache
- Updates the address of the inode

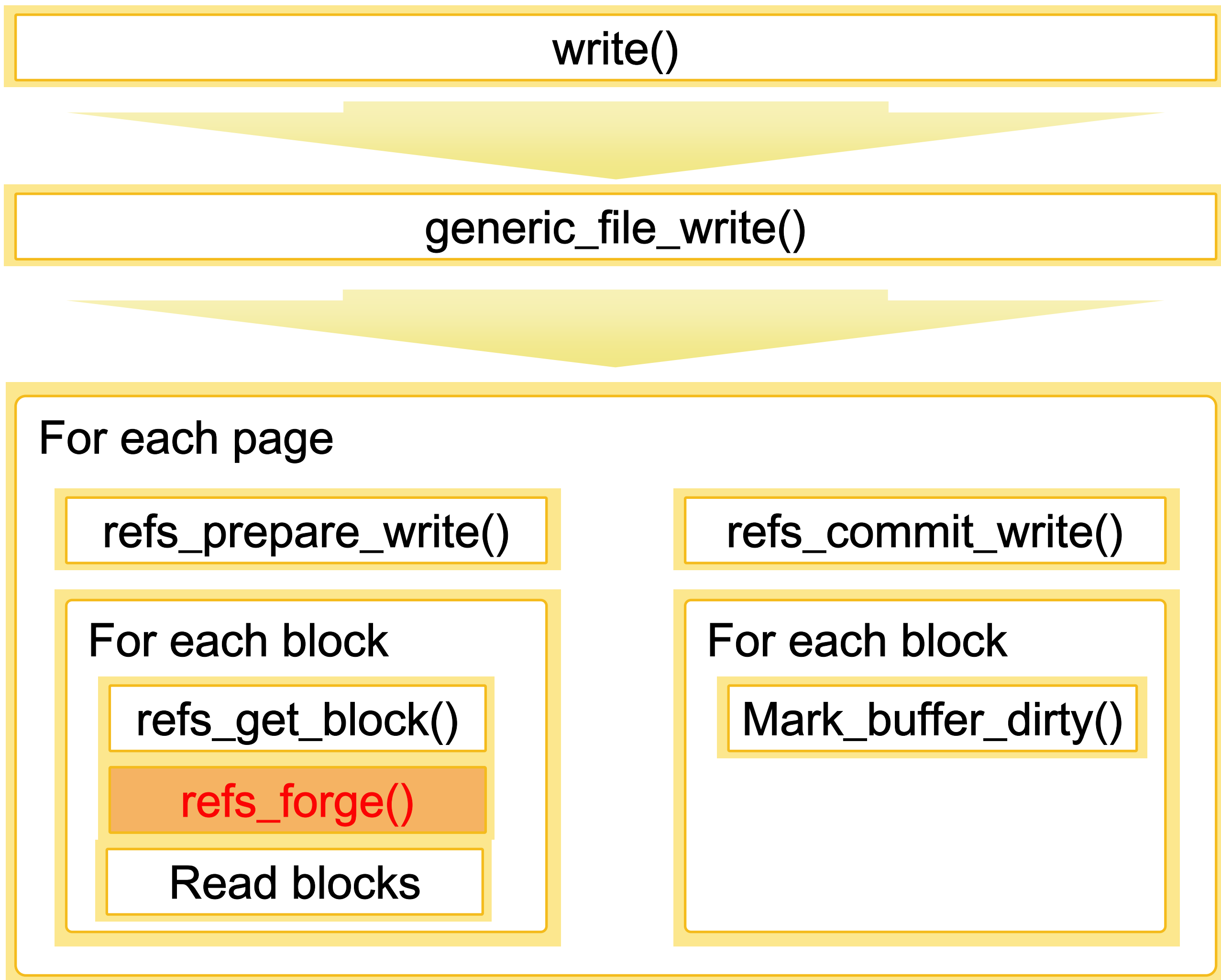
Checkpoint and Restore

- Checkpoint
  - Enumerate the files being used and repeat the following
    - Copy the related metadata and keep it in a designated dir.
    - Get the file offset and store it in a safe area
- Restore
  - Reopen the file
  - Install the fd into the original position
  - Update the offset of the file

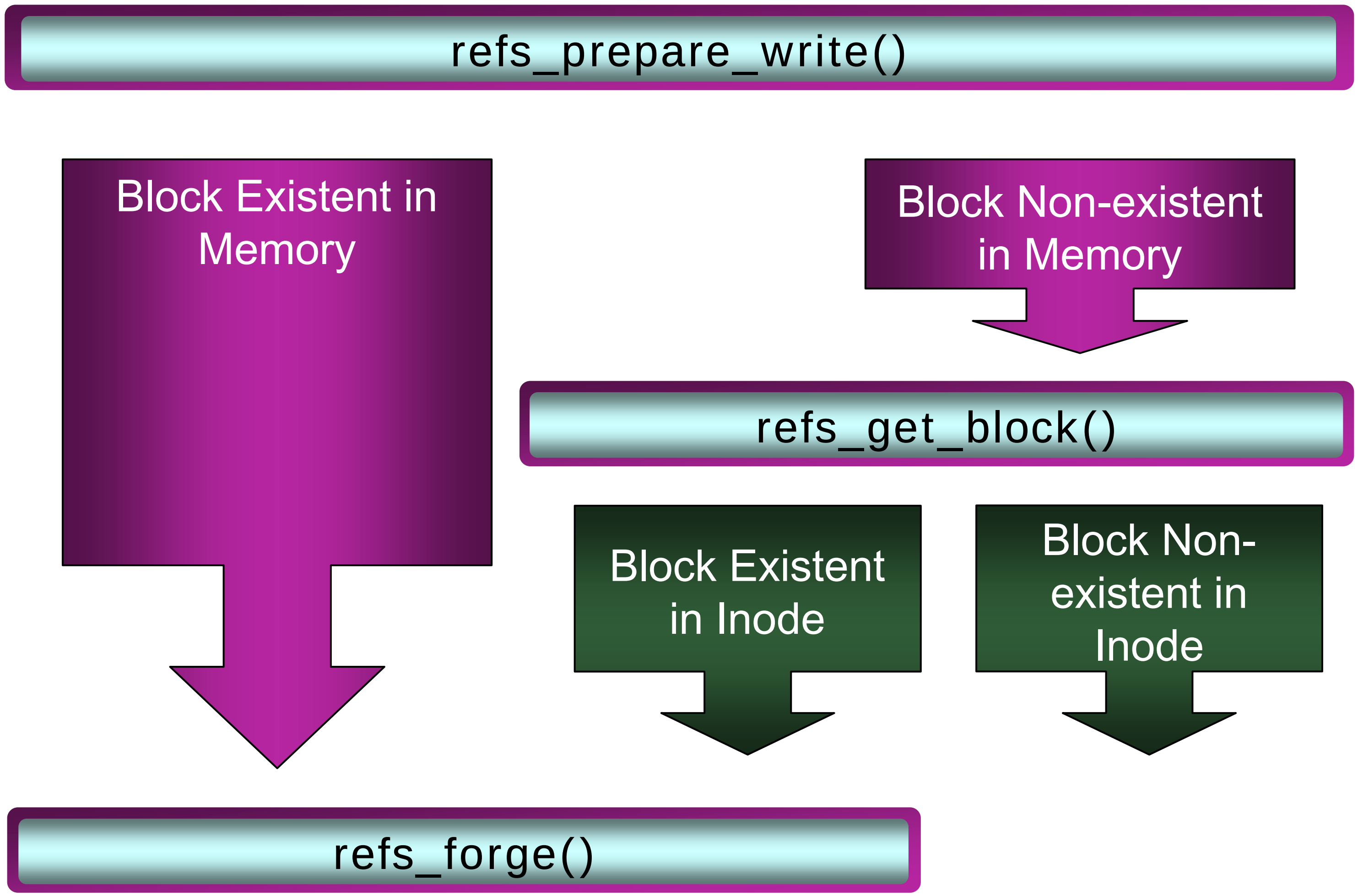
Implementation

- Prototype in the Linux kernel 2.6.3
- Based on ext2
- System call interfaces
  - refs_handler()

Write Operation



No Overwrite – in Detail



Refs_forge()

- Implements address translation module
  - Forges the address of the block into another empty address
- Called block-by-block
  - We can operate on each block to write

- Sequence
  1. Allocate a block in disk
  2. Modify the inode
  3. Map the block as the block number of a new block

Checkpointing and Recovery

- Checkpointing
  - Save the data structure into the designated kernel memory called "Core" which indicates the checkpointed files
- Recovery
  - Recover the file table of the process using the data in the Core

Experimental Environment

- A Linux Box
- In order to get rid of the caching effect we have repeated the following step for each experiment
  - Mount the device
  - Run the application
  - Unmount the device
- Two experiments
  - Sequential write
    - A new file is created and sequential write is submitted according to the given size
  - Partial overwrite
    - A write request whose data range is within the size of the target file ➔ generate an overwrite situation

Evaluation

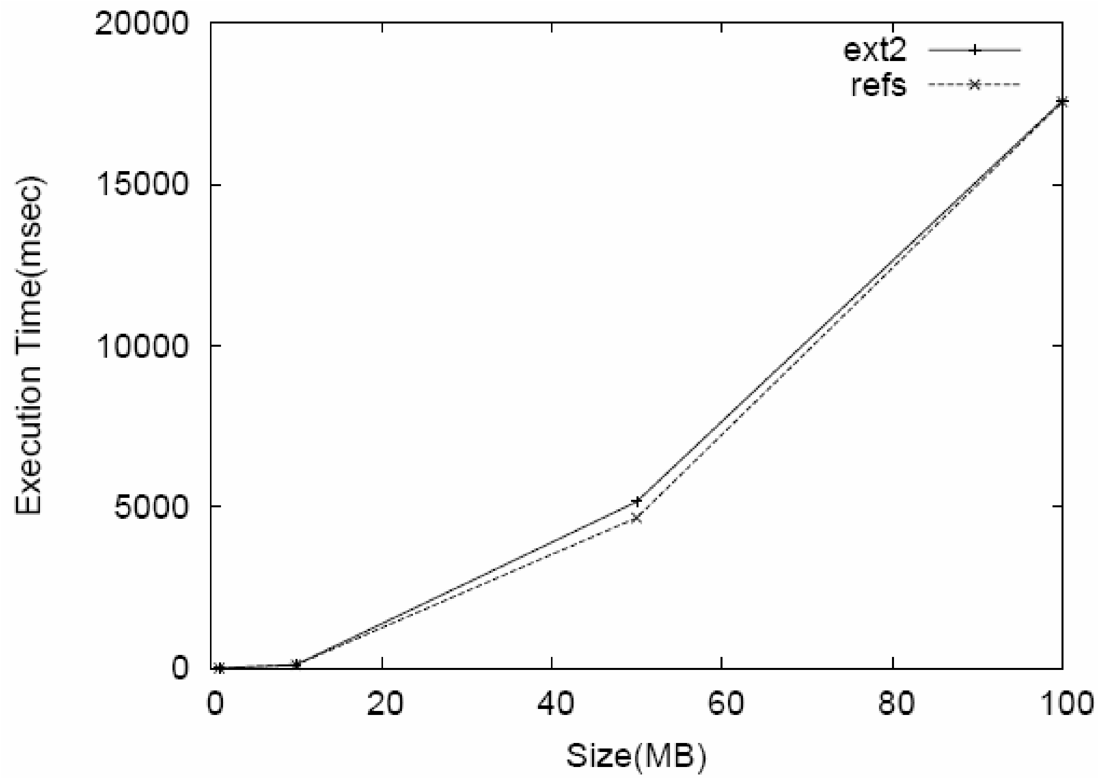


Figure 5. Performance Comparison of ReFS and ext2: Sequential write

Evaluation

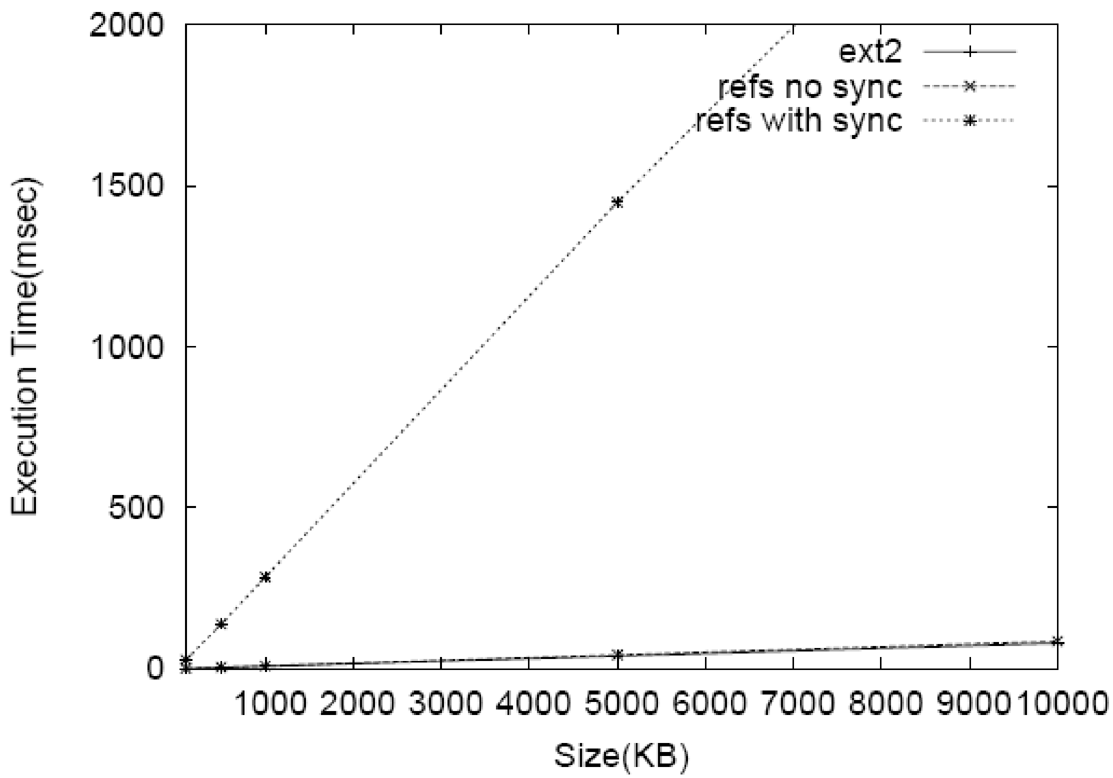


Figure 6. Performance Comparison of ReFS and ext2: Partial write

Conclusions and Future Work

- We have developed a recoverable file system for MPICH-GF
  - User-transparent
  - Can be integrated with other fault-tolerant systems

- We have also developed a simple user-level mechanism to provide fault-tolerance for MPICH-GF
  - Included in the current deployment
- We are currently developing a user-level file system that can reduce the burden the kernel has
  - Deploying a kernel module is not as easy as deploying applications