

# Group File Operations: A New Idiom for Scalable Tools

Michael J. Brim  
Paradyn Project

Paradyn / Condor Week  
Madison, Wisconsin  
April 30 - May 3, 2007

# Talk Overview

- Group Process Control and Inspection
- A New Group File Idiom
- TBÖN-FS: Scalable Group File Operations

# Research Domain

- HPC Tools & Middleware
  - Middleware: run applications and manage system
  - Tools: diagnose and correct problems
- Large scale systems
  - Tools and middleware are **CRUCIAL**
  - More resources to manage
  - Many problems appear as scale increases

Tools/middleware that can be used on the largest current systems are scarce

# Example Tools & Middleware

- Parallel Application Runtime Environments
  - MPI, PVM, BProc, IBM POE, Sun CRE, Cplant yod
- Parallel Application Monitoring and Steering
  - Paradyne, Open|SpeedShop, MATE
- Distributed Application Debuggers
  - TotalView, DDT, Eclipse PTP, mpigdb
- Resource Monitoring and Management
  - SLURM, PBS, LoadLeveler, LSF, Ganglia

# Group Process Control and Inspection

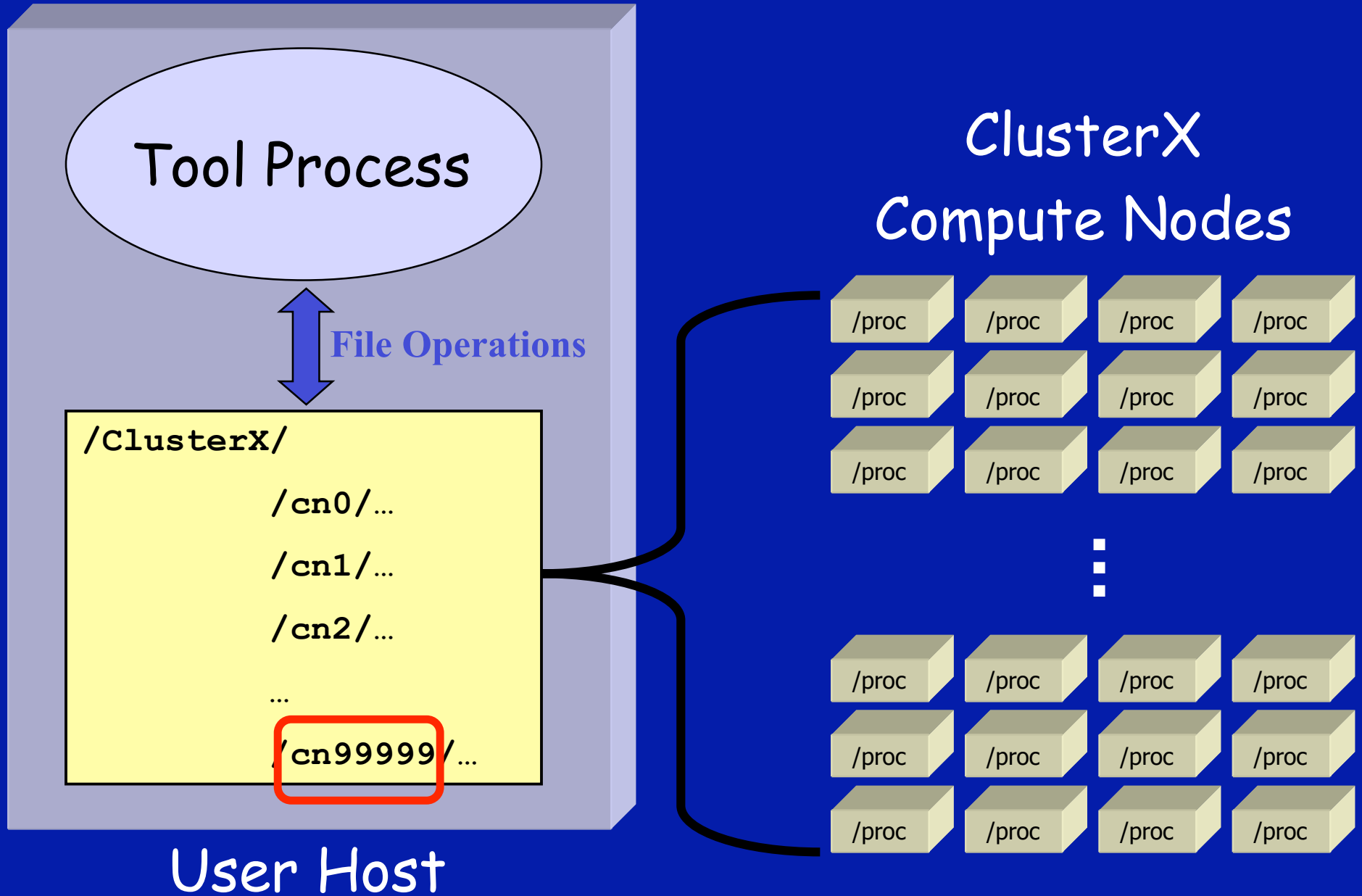
- Modify or examine process state
  - Launch processes and manage stdin/out/err
  - Send job control signals (e.g. STOP, CONT, KILL)
  - Read and write memory, registers
  - Collect asynchronous events (e.g. breakpoints and signals)
  - Read process information files (i.e. Linux /proc)
- For groups of 10,000 - 100,000 processes

**And More!!!**

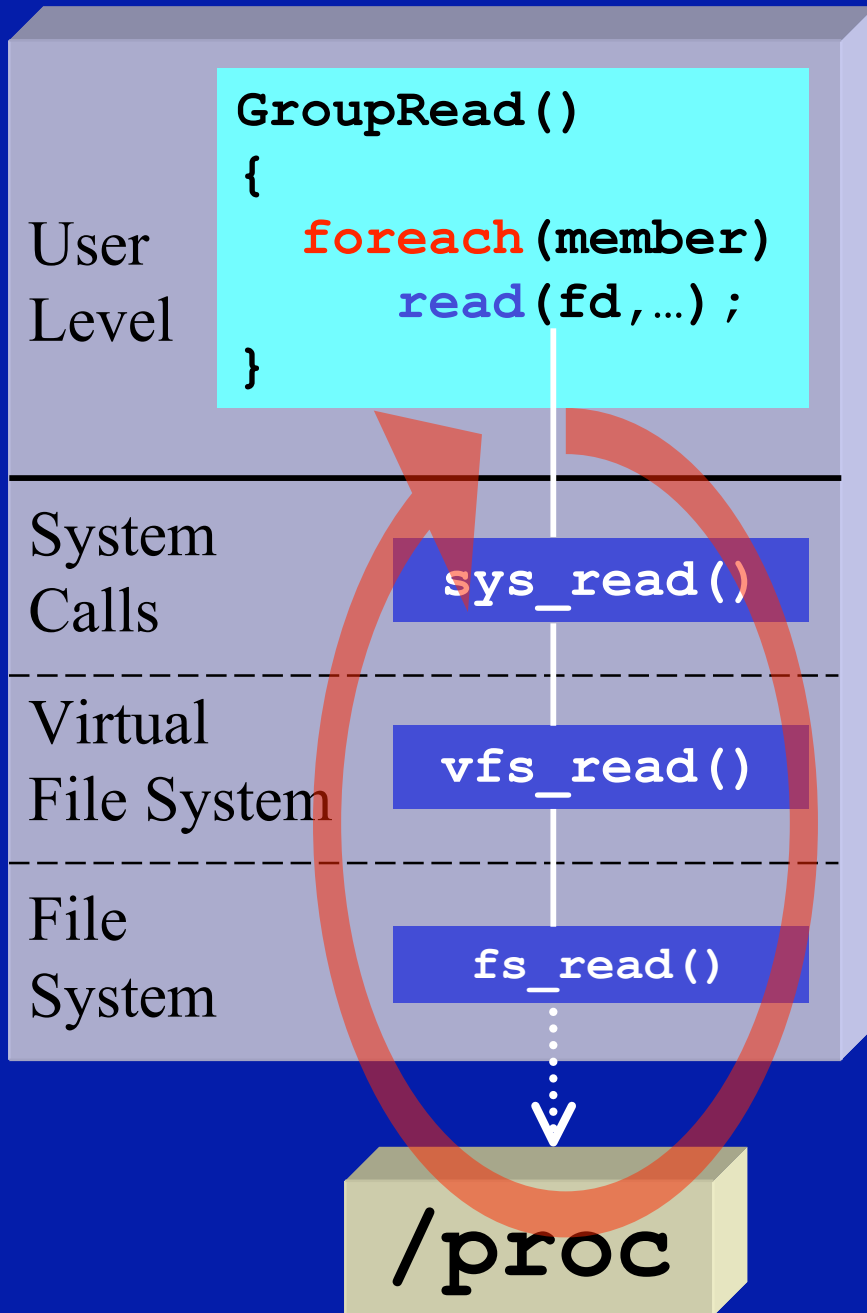
# New Idiom: Group File Operations

- Abstract all operations as file access
  - Natural, Intuitive, Portable
  - /proc
    - 8<sup>th</sup> edition UNIX (1985)
    - Plan9 (1992) → 4.4BSD (1994), Solaris 2.6 (1997)
    - Linux
- Global mount of remote files
  - Distributed OS: LOCUS (1983), ..., BProc (2002)
  - Remote mount: UNIX United (1987), ..., Xcpu (2006)
- Operate on groups of files (processes)
  - How to do so in a scalable manner?

# Global Mount



# Group Operations: Current Technology



User-level group operations **iterate**.

$$\text{Cost} \approx \mathbf{G} \times (\mathbf{T} + \mathbf{L} + \mathbf{R})$$

User-Kernel Trap  
(**T**)

Local Processing (**L**)

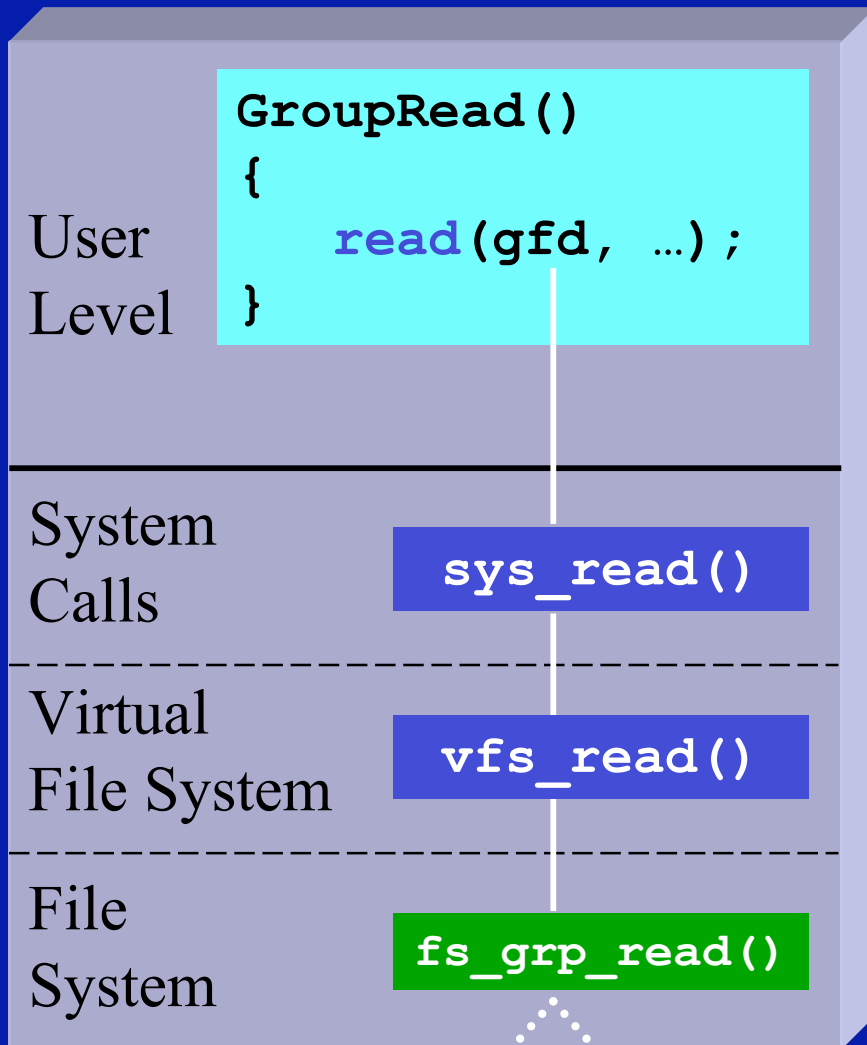
Remote Communication &  
Processing (**R**)



# Scalable Group File Operations

- How to avoid iteration over files?
  - Explicit groups: **gopen()**
  - One OS interaction for each group operation
- How to provide scalable group operations?
  - Group-Aware File System: **TBÖN-FS**

# Group Operations: Scalable Approach



With **OS** and **File System** support, group operations can **use scalable techniques**.

$$\text{Cost} \approx T + L + (\log(G) \times R)$$

# Group File Operations

- Forming Groups

- Directory = a natural file system group abstraction

<code>mkdir/rmdir</code>	: create/delete group
<code>mv, cp, ln</code>	: add members
<code>rm</code>	: delete members

- Accessing Groups

```
gfd = gopen(char* gdir, int flags)
```

- Operating on Groups

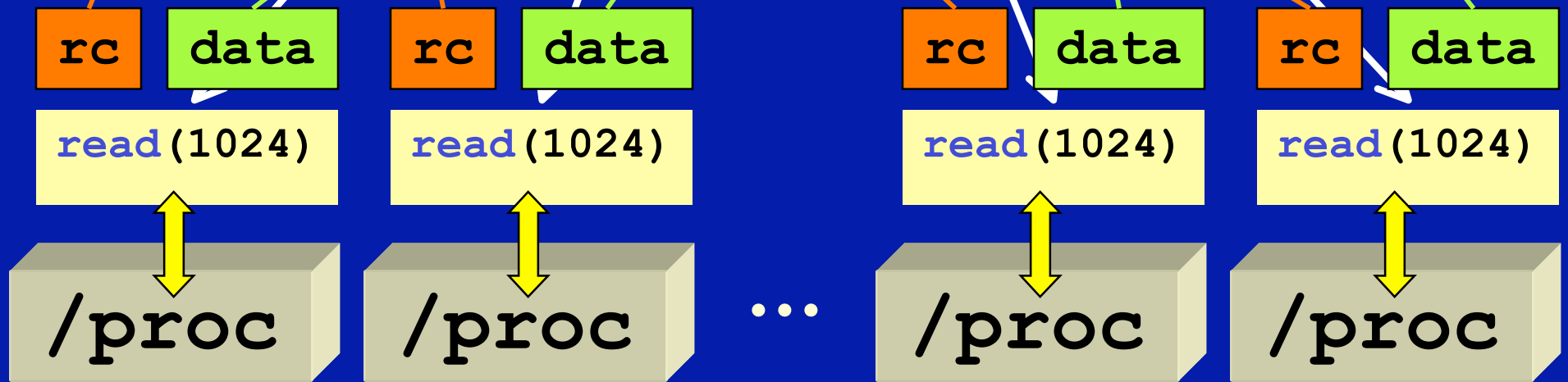
- Pass group file descriptor to file operations
  - e.g., `read`, `write`, `lseek`, `chmod`
- Semantics - operation applied to each group member

# Group File Operations

```
int rc = read(gfd, databuf, 1024)
```

Return Code  
(status/error)

Data Output  
Buffer



# Data Aggregation

- Definition: construct a whole from parts
- Provides various levels of data resolution

## SUMMARY

- min
- max
- average
- sum

## PARTIAL

- $x > 0.9$
- $y \in \{\dots\}$
- TopN(z)

## COMPLETE

- concatenate
- equiv. class

# Aggregating Group Results

- Fit existing interfaces
  - Status → summary
    - Need to choose appropriate default for each op
  - Data → concatenate
- New operations for controlling results
  - Retrieve individual status **gstatus (...)**
  - Load custom aggregations **gloadaggr (...)**
  - Bind aggregations to operations **gbindaggr (...)**

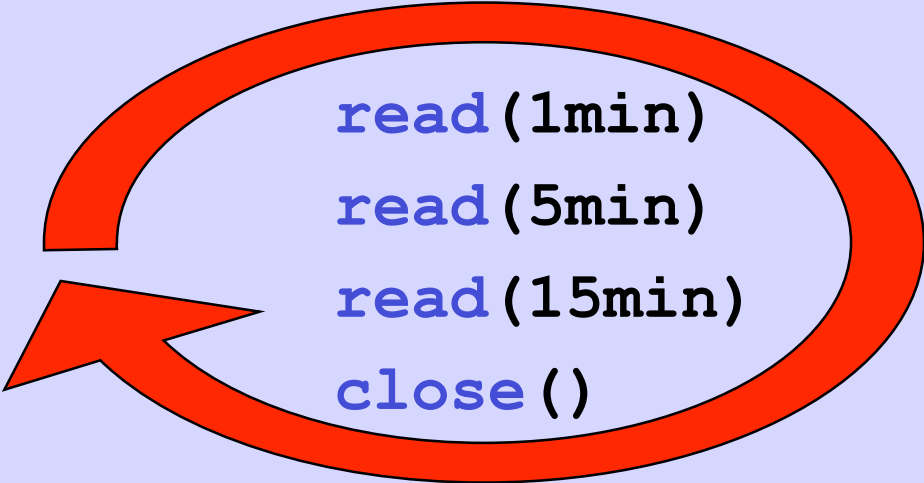
# Example: System Resource Monitor

- Collects 1-, 5-, 15-minute load averages
  - Reads `/proc/loadavg` from each node
- Calculates (for each granularity)
  - Minimum load across all nodes
  - Maximum load across all nodes
  - Average load across all nodes

## BEFORE



`open()`



`read(1min)`  
`read(5min)`  
`read(15min)`  
`close()`



`ComputeMMA(...)`

## AFTER

`gdefine()`

`gfd = gopen("grp_dir")`

`// Bind read to aggr`  
`gbindaggr(gfd, OP_READ,`  
`AGGR_MIN_MAX_AVG)`

`// Read & Compute`  
`read(gfd, 1min)`  
`read(gfd, 5min)`  
`read(gfd, 15min)`

`close(gfd)`



# Group File Operations: Other Uses?

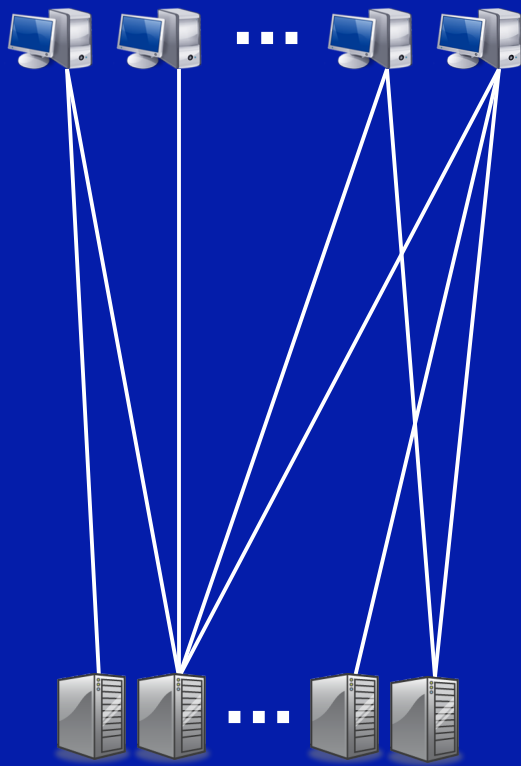
- Distributed System Administration
  - Disk-full clusters
    - System file patching
    - Software installation
  - System log monitoring
- Utility programs that operate on file groups
  - *e.g.*, `ps`, `top`, `grep`, `chmod/chown`
- Internet Applications
  - Peer2Peer - file retrieval a la BitTorrent
  - Search/Crawl - websites are really just files

# TBÖN-FS

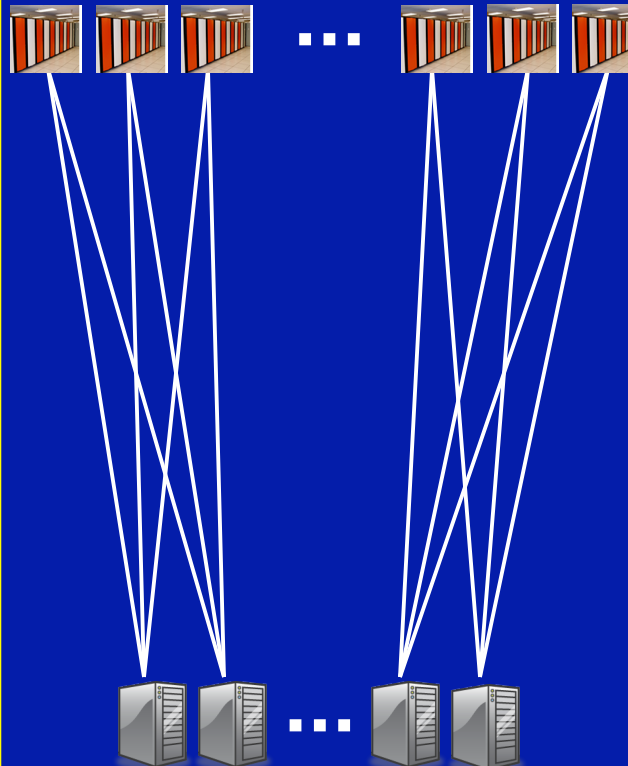
- New distributed file system
  - Scalable group file operations
  - Efficient single file operations
  - Tens to hundreds of thousands of servers
    - Single mount point
- Integrates Tree-Based Overlay Network
  - One-to-many multicast & gather communication
  - Distributed data aggregation

# Scalable Group File Operations

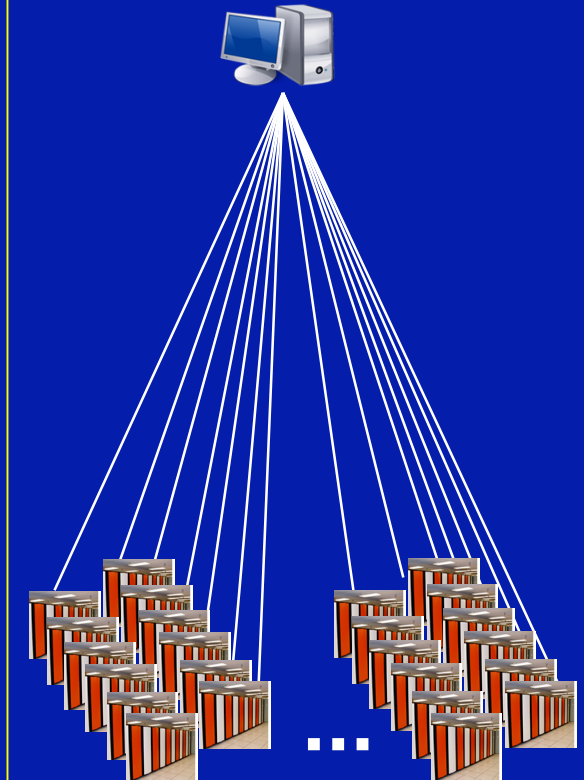
- Why not use a distributed/parallel FS?



**Distributed File System**

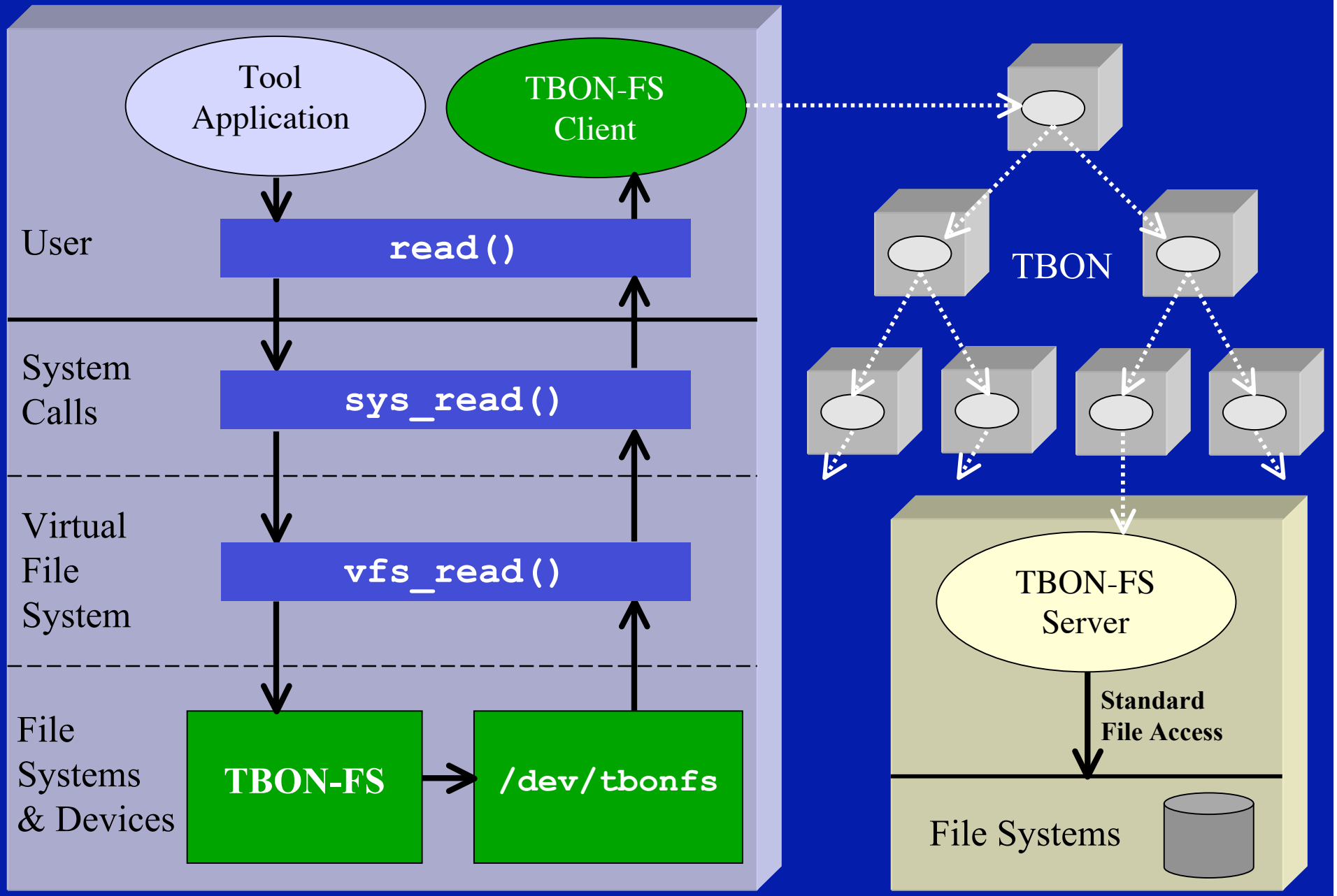


**Parallel File System**

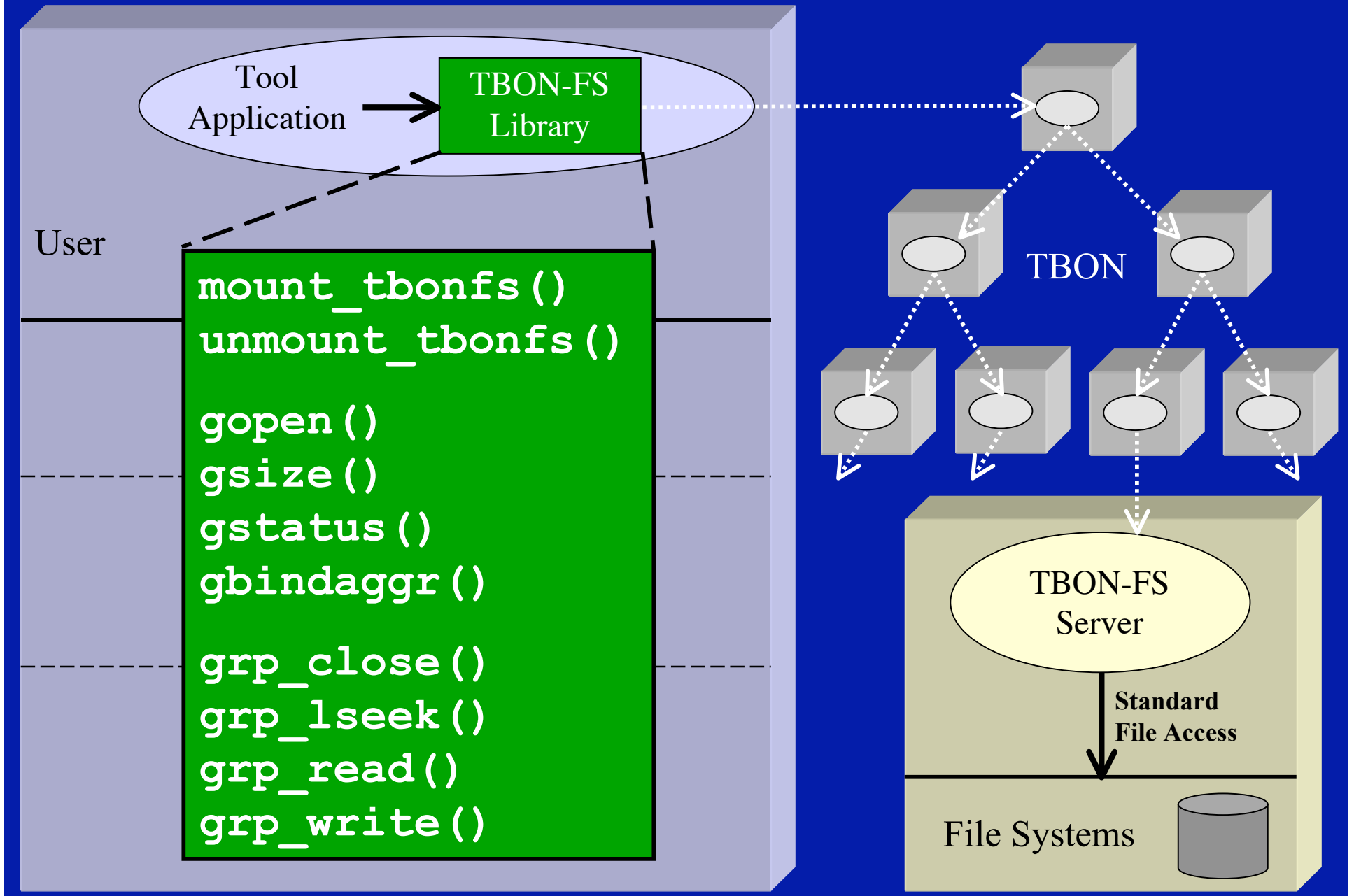


**TBON-FS**

# TBON-FS: Proposed Architecture



# TBON-FS: Current Prototype



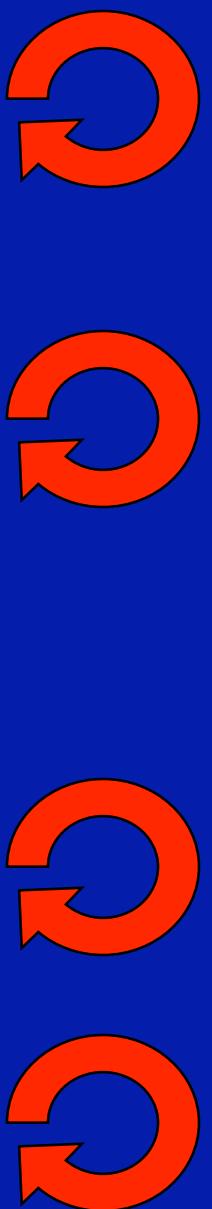
# Current & Future Research

- Group file operations
  - OS support
  - More file types & operations (e.g., sockets and pipes)
- Tool integrations
  - Ganglia wide-area system monitor (in progress)
  - TotalView debugger
- TBON Model Extensions
  - Topology-aware filters
  - Persistent host state
  - Multi-organization TBON

# Summary

- “Iteration is the bane of scalability.”
- Group File Operations
  - Are natural, intuitive, and portable
  - Eliminate iteration
  - Allow for custom data aggregation
- TBON-FS: scalable group file operations

# Distributed Debugger (BEFORE)



```
// Open all /proc/<pid>/mem
foreach file ( `ClusterX/cn*/[1-9]*/mem' )
  fds[i] = open(file, flags);
  grp_size++;

// Set breakpoint & wait
for i=0 to grp_size
  lseek(fds[i], brkpt_addr, SEEK_SET);
  write(fds[i], brkpt_code_buf, code_sz);
WaitForAll();

// Read variable & compute equivalence classes
for i=0 to grp_size
  lseek(fds[i], var_addr, SEEK_SET);
  var_buf = grp_var_buf[i];
  read(fds[i], var_buf, var_sz);
  close(fds[i]);

ComputeEquivClasses(grp_var_buf, var_classes_buf);
```



# Distributed Debugger (AFTER)




```
// Open all /proc/<pid>/mem
foreach file ( `ClusterX/cn*/[0-9]*/mem' )
    // add link to file in group directory
    symlink(file, "grp_dir");

gfd = gopen("grp_dir", flags);
grp_size = gsize(gfd);

// Set breakpoint & wait
lseek(gfd, brkpt_addr, SEEK_SET);
write(gfd, brkpt_code_buf, code_sz);
WaitForAll();

// Read variable & compute equivalence classes
lseek(gfd, var_addr, SEEK_SET);
gbindaggr(gfd, OP_READ, AGGR_EQUIV_CLASS, var_sz);
read(gfd, var_classes_buf, var_sz);
close(gfd);
```



# System Monitor (BEFORE)



```
// Open all /proc/loadavg
foreach file ( `ClusterX/cn*/loadavg` )
  fds[i] = open(file, flags);
  grp_size++;

// Read 1-minute, 5-minute, 15-minute loads
for i=0 to grp_size
  read(fds[i], 1min_buf[i], load_sz);
  read(fds[i], 5min_buf[i], load_sz);
  read(fds[i], 15min_buf[i], load_sz);
  close(fds[i]);

// Compute min/max/avg for each granularity
ComputeMinMaxAvg(1min_buf, 5min_buf, 15min_buf);
```



# System Monitor (AFTER)



```
// Open all /proc/loadavg
foreach member_file ( `ClusterX/cn*/loadavg` )
    // add link to member in group directory
    symlink(member_file, "grp_dir");

gfd = gopen("grp_dir", flags);

// Read 1-minute, 5-minute, 15-minute loads
// and calculate min/max/avg
gbindaggr(gfd, OP_READ, AGGR_MIN_MAX_AVG, load_sz);
read(gfd, 1min_buf, load_sz);
read(gfd, 5min_buf, load_sz);
read(gfd, 15min_buf, load_sz);
close(gfd);
```

# Related Work

- Xcpu
  - File system interface for distributed process management
  - Uses Plan9 9P protocol and recent Linux support (V9FS)
- HEC POSIX I/O Extensions
  - Explicit sharing of files by process groups
  - `openg` and `sutoc`