Scalable Distributed Process Group Control and Inspection via the File System

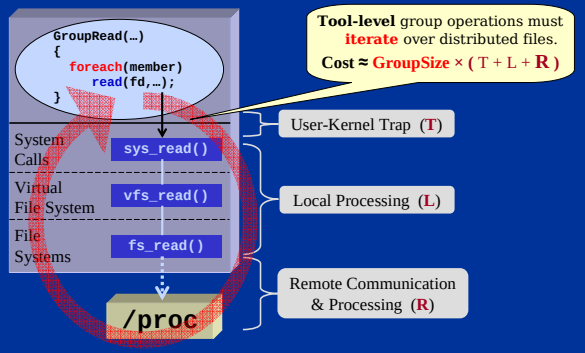
Michael J. Brim
Computer Sciences
Advisor: Barton P. Miller
University of Wisconsin-Madison

Research Overview

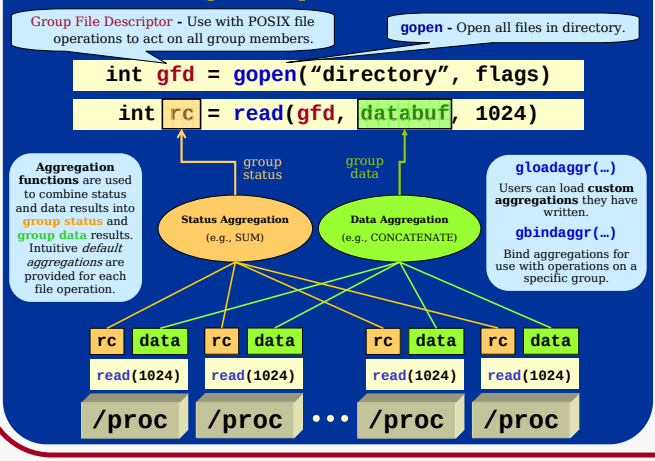
- Scalable Tools and Middleware are ***Crucial***
  - Run, analyze, debug, and improve applications & systems
  - Require group operations on distributed files and processes
- Problems at scale
  1. How to *access* files on thousands or millions of hosts?
  2. How to efficiently *operate* over large file groups?
  3. How to handle the *data explosion* for large file groups?
- Solutions
  1. *Access*: TBŌN-FS Distributed File System
     - Provides global namespace that combines remote file systems
     - Uses Tree-Based Overlay Network (TBŌN) for scalability
  2. *Operate*: Group File Operation Idiom
     - Eliminates explicit iteration imposed by file system interface
  3. *Data Explosion*: Data Aggregation
     - Explicit semantics for aggregation in group file operations
     - TBŌN-FS architecture for scalable distributed aggregation

Group File Operations

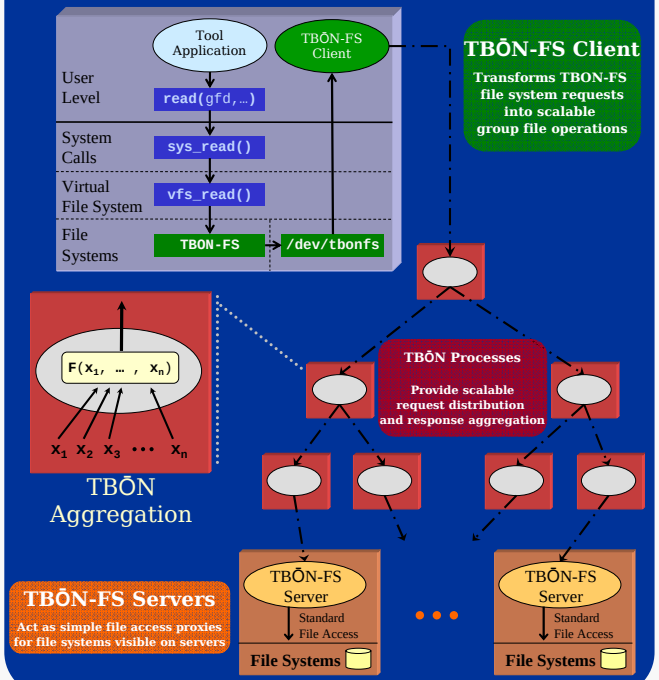
- Problem: Current File System API Forces Serial Access



- Solution: Group File Operation Idiom



TBŌN-FS Architecture



TBŌN-FS Namespace

- Client Namespace Aggregates Server Namespaces
  - Ideal Namespace: directories already contain desired file groups
  - FiNAL: File Namespace Aggregation Language
    - Goal: allow users to specify interesting groups with custom namespace
    - Operators on FS hierarchies: insert, remove, replace, move, merge
  - TBŌN provides scalable aggregation and name resolution
- Example: Global Linux `/proc`
  - Contains files providing system and process information
    - System Files: `cpuinfo`, `loadavg`, `meminfo`, `modules`, …
    - Processes: directory per unique process id
      - Files: `cmdline`, `environ`, `mem`, `stat`, `statm`, …

Simple

Directory per Server

```
/tbonfs/proc:
  serverA/
          /cpuinfo
          /1/…
          /4/…
  …
  serverZ/
          /cpuinfo
          /1/…
```

Merge

Common Files ⇒ Directory (per server files)
Common Directories (per server sub-directories)

```
/tbonfs/proc:
  cpuinfo/
          /serverA
          /…
          /serverZ
  1/
   /serverA/…
   /…
   /serverZ/…
  4/
   /serverA/…
```

Deep Merge

Start merge at leaves of common sub-hierarchy, rather than root

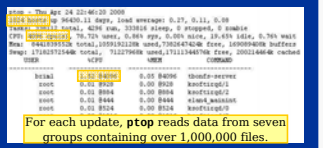
```
/tbonfs/proc:
  cpuinfo/
          /serverA
          /…
          /serverZ
  1/
   /cmdline/
            /serverA
            /…
            /serverZ
  4/…
```

All the World's a File

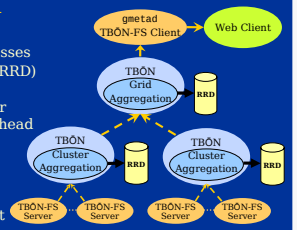
- Goal: Abstract ***user-defined functions*** (UDFs) as files
  - Enables use with group file operations
  - UDFs with file interface: Previous work exists
    1. Plan9 operating system: 9P file service protocol
    2. FUSE: user-level file systems
       Good: Arbitrary code for file system calls
       Bad: Too much file system specific code, requires kernel support
  - Our Approach: ***Synthetic Files***
    - Similar to FUSE: UDFs for file system calls
    - No kernel support required: UDFs execute in TBŌN-FS server
- Example: **Process Group Control and Inspection**
  - File access for all control and inspection operations
    - read / write process memory or registers
    - handle signals and exceptions (e.g., breakpoints)
    - stop, continue, step
    - thread control and inspection
  - Three Components
    1. Portable process control and inspection layer
    2. *Synthetic File* for each control / inspection operation
       - additional control file for process execution (a la *Xcpu*)
         - capture standard I/O streams as files too
    3. Custom FiNAL namespace
       - combine new synthetic files with existing files (i.e., `/proc`)

Demonstrations

- System Administration: Parallel Linux commands
  - **`ptop`** : host & process monitor
  - **`pgrep`** : parallel text search
    - identify configuration differences
  - **`ptail`** : parallel log monitoring
    - with --follow option
  - **`pcp`**, **`psync`** : file distribution
    - send whole file, or only changes


For each update, `ptop` reads data from seven groups containing over 1,000,000 files.

- System Monitoring: Ganglia
  - Original System
    - `gmetad` cluster / Grid aggregation processes
      - store data to round-robin database (RRD)
    - `gmond` monitoring daemons
      - multicast data updates within cluster
      - linear memory / CPU / network overhead
  - Ganglia-tbonfs
    - top-level `gmetad` TBŌN-FS client
    - scalable TBŌN-FS aggregation
    - `gmonds` replaced by TBŌN-FS servers
      - eliminates overhead due to multicast



- Example Code: Distributed Debugger

1. **Define groups**
2. **Stop all processes**
3. **Set a group breakpoint**
4. **Continue all processes**
5. **Wait for processes to hit breakpoint**
6. **Read program variable, generate equivalence classes for current values**

```
1. mkdir("grp_ctl_dir"); mkdir("grp_mem_dir");
   foreach( member in `/tbonfs/proc/[1-9]*` ) {
     symlink("member/signal", "grp_ctl_dir");
     symlink("member/mem", "grp_mem_dir");
   }
   ctl_gfd = gopen("grp_ctl_dir", O_WRONLY);
   mem_gfd = gopen("grp_mem_dir", O_RDWR);
2. write(ctl_gfd, SIGSTOP, 4);
3. lseek(mem_gfd, brkpt_addr, SEEK_SET);
   write(mem_gfd, brkpt_code_buf, code_sz);
4. write(ctl_gfd, SIGCONT, 4);
5. WaitForAll(ctl_gfd);
6. lseek(mem_gfd, var_addr, SEEK_SET);
   gbindaggr(mem_gfd, OP_READ, EQUIV_CLASS_AGGR, …);
   read(mem_gfd, grp_var_classes, var_sz);
```