MDSim - Load Simulator

Alexey Lyashkov <Alexey_Lyashkov@xyratex.com>
Agenda

• Project ideas
• Conceptual design
• Userland application structure
• Kernel module structure
  – MD extension
  – Data extension
Testing various FS aspects we frequently find it difficult for applications to provide a replicable workload. To solve this we need a tool to generate a more direct load.

Advantages of project

- Flexible
- Extensible
Conceptual design

LoadSim

Virtual Machine

Kernel API

Metadata API

Data API

Kernel part

Userland part

parser

Bytecode generator

Visualization results
Userland application structure

MDSIM

- lexical scanner
- Bison based parser
- “Compiler” (convert parsing result into assembler code” and resolve labels)
- Encode assembler code into binary form
- Kernel Interface
Kernel module structure

- Client Environment
  - VM
    - extensions
      - sys extension
        - OS interface (like delay, race emulation, change effective UID, GID)
      - Metadata extension
        - Metadata related commands (like make dir, file.. etc)
      - Data extension
        - Data related commands (like data IO, striping.. etc)
Virtual machines

Polish notation, Forth like

Stack machine

Data Stack

Return stack

Program code, data

Register machine

Memory access unit

Registers

Memory

Like real CPU
Kernel module: environment

- Client Environment
- Application specific data
  - MD Environment
  - Data Environment
  - Statistic

Like files, states
VM Extensions

Extensions

- System
- Metadata
- Data IO

OS api

Local
- Lustre
- MDD

Planed

- Local
- ost_echo
- osc_echo
MD (metadata) extensions – plugin API to enable simulator use with different filesystem types via native functions.

Plugins are responsible for executing any metadata modification operations atomically.
procedure test1
    make_work_dir 0777
    cd "test1" expected FAIL
    mkdir "test1" 0666 expected OK
    cd "test1" expected OK
    open "test-file1" O_CREAT|O_RDWR 0666 20 expected OK
    close 20 expected OK
    stat "test-file1" expected OK
    chmod "test-file1" 0666 expected OK
    chtime "test-file1" 222 expected OK
    chown "test-file1" 999:999 expected OK
    truncate "test-file1" 2000 expected OK
    softlink "test-file1" "soft-1" expected OK
    hardlink "test-file1" "hard-1" expected OK
    readlink "soft-1" expected OK
    rename "hard-1" "hard-2" expected OK
endproc

server MDS0 192.168.69.3@tcp "lustre"
client "CLI[0-800]" test1
procedure test1
  $R1 = 5
  while ($R1 != 0)
    $R0 = printf "file-%s-%d-%d" [ $$cli_name, $R1, $$pid ]
    open $R0 O_CREAT|O_RDWR 0666 20 expected OK
    close 20 expected OK
    $R1 = $R1 - 1
  endw
endproc

server local "/mnt/tmp"

client "CLI0" test1

That example creates a 5 files with names ‘file-CLI0-[5 .. 0]-$process_pid"
cli CLI0_0 : last op 279 total 279 rc -2

VM_MD_CALL_CD : 999/53997/27498
VM_MD_CALL_MKDIR : 2999/2999/2999
VM_MD_CALL_REaddir : 1000000000/0/0
VM_MD_CALL_UNLINK : 1000000000/0/0
VM_MD_CALL_OPEN : 999/999/999
VM_MD_CALL_CLOSE : 1000000000/0/0
VM_MD_CALL_STAT : 1000000000/0/0
VM_MD_CALL_CHMOD : 1000000000/0/0
VM_MD_CALL_CHOWN : 1000000000/0/0
VM_MD_CALL_CHTIME : 1000000000/0/0
VM_MD_CALL_TRUNCATE : 1000000000/0/0
VM_MD_CALL_SOFTLINK : 1000000000/0/0
VM_MD_CALL_HARDLINK : 1000000000/0/0
VM_MD_CALL_READLINK : 1000000000/0/0
VM_MD_CALL_RENAME : 1000000000/0/0

Client executed all op’s (total == last) last operation result -2 (ENOENT)

Time of executing for each supported operation (min, avg, max)
Data extensions

Data

Layouts

IO

Local

ost_echo

osc_echo
Layout module designed for testing different QoS disciplines and controls low level data object allocations and striping parameters.
data access extensions

- Responsible
- Object allocation
- Read/Write IO
DATA program example

targets
OSC1 .... OSC2 .... endtarget

layouts
layout 1 { LOV_RAID0, stripe size, ... }
layout 2 { PDCLUST, start target, ... }
endlayout

objects
object 1 layout1 endobj

procedure test1
create 1
seek 1 YYYYY
write 1 size YYYYY
seek 1 0
read 1 EOF
destroy 1
endproc

client ...
client ...

endproc
Conclusions

MDSim useful for testing because:
- May simulate many clients on single host
- Has flexible language to describe any type of workload
- Has plugin API to create connectors to any FS
- May be possible to create a tool to convert a Lustre log to test program
Thanks!

Alexey Lyashkov <Alexey_Lyashkov@xyratex.com>