Lustre Performance Analysis with SystemTap

Jason Rappleye

NASA Advanced Supercomputing Division

April 24, 2012

Introduction

- Need actionable performance and troubleshooting data at interactive speeds.
- /proc and lctl dk are useful, but:
  - Performance issues at scale
  - Don't want to pollute logs
  - Want more information

Systemtap

- SystemTap consists of a scripting language, translator and runtime.
- Provides system-wide tracing capabilities.
  - Kernel and userspace.
- strace traces a process tree; SystemTap provides visibility across the entire system.

Using SystemTap with Lustre

- Where to probe?
  - This is the hard part - need to have some understanding of how Lustre works.
- Extract data from functions as they are called/return.
  - Output as you go, or
  - Aggregate and periodically display
- Timing function calls.
  - Lustre service threads handle RPCs from start to finish, one at a time
  - Makes it easy to store timing, other information based on the thread handling the request.

Example: Timing ldiskfs block allocations

```
global start
global times
probe
 module("ldiskfs").function("ldiskfs_mb_new_blocks") {
   start[tid()] = gettimeofday_ms();
}
probe
 module("ldiskfs").function("ldiskfs_mb_new_blocks").return
   if ([tid()] in start) {
     times <<< gettimeofday_ms() - start[tid()];
   }
}
probe end {
  print(@hist_log(times));
}
```

Output

```
value  |--------------- count
    0  |@@@@@@@@@@@@@@@  27083
    1  |@                  675
    2  |                     6
    4  |                     2
    8  |                   158
   16  |                    74
   32  |                    33
   64  |                    10
  128  |                     6
  256  |                     2
  512  |                     1
       ~
32768  |                     1   <---- That's between 32
                                       and 64 seconds!
```

Examples

- Poor choice of stripe count.
- Fragmentation.
- High OSS load average.

Poor choice of stripe count

- The default stripe count on our filesystems is one.
- Average file size is relatively small, so this is OK.
- Except. . .
  - Large tar files can take up a significant portion of an OST.
  - Many ranks writing to a file on one OST can perform poorly.

big-object

- A SystemTap script intercepts calls in the write path on the OSS to gather the following information:
  - NID
  - OST name
  - object ID
  - FID
  - UID
  - object size
- When there's a a write to an object over a predetermined size, print it.
- A Python wrapper gathers additional information about the object and writing process, including the path.

big-object

```
service162 ~ # big-object
Fri Mar 23 10:21:09 2012 service61-ib1 ost:nbp2-OST0021
stripes:1 pid:4320 command:tar size:506123MB
name:/nobackupp2/.../something.tar
```

Fragmentation

- On-disk
  - Block allocator can't find a large enough chunk of contiguous free space.
  - Slows down writes; fragmented allocation will cause more I/Os for both reads and writes.
- Memory
  - The IB SRP driver can only handle scatter-gather descriptors up to length 255.

Showing I/O fragmentation in real-time

- Use SystemTap to hook into Lustre I/O path.
- A good I/O - 1MB or more in a single write:

```
nid:10.151.18.95@o2ib0 ost:nbp2-OST0010 uid:0
mdt_inode:0 sizes:256
```

- Memory fragmentation causing SRP to issue multiple I/Os:

```
nid:10.151.14.211@o2ib0 ost:nbp2-OST0008 uid:0 mdt_inode:0
sizes:255(255) 1
```

On-disk fragmentation

```
nid:10.151.32.127@o2ib0 ost:nbp2-OST0068 uid:12137
mdt_inode:200443860 sizes:6(6) 1(1) 1(1) 1(1) 1(1)
1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1)
1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1)
1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1)
1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1)
1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1) 1(1)
1(1)
```

High OSS load average

- High OSS load average is often due to long disk queues.
- A typical cause is many hosts performing I/O to a file on a small number of OSTs.
- You could mine the data in /proc.
  - On a large system, this takes time.
  - I want to know which file is being accessed.
    - Currently possible for writes.
    - May require modifications to Lustre for reads.

oststat

OST	r/s	w/s	aveq	rwat	wwat	%u	job/host	Ops
nbp2-OST06	80	42	4	48	0	72	23925.pbspl3	8
nbp2-OST0e	37	2	0	15	0	19	66274.pbspl1	6
nbp2-OST16	50	0	1	18	3	27	66348.pbspl1	6
nbp2-OST1e	24	0	1	55	1	37	66273.pbspl1	8
nbp2-OST26	44	18	3	82	0	59	66283.pbspl1	6
nbp2-OST2e	79	9	1	21	0	39	66428.pbspl1	6
nbp2-OST36	41	185	135	150	57	100	68894.pbspl1	126
nbp2-OST3e	80	2	3	34	0	42	68386.pbspl1	12
nbp2-OST46	63	2	3	48	1	54	66345.pbspl1	6
nbp2-OST4e	73	2	3	49	0	74	66336.pbspl1	6
nbp2-OST56	43	13	0	17	0	23	66443.pbspl1	6
nbp2-OST5e	35	1	0	16	6	18	66267.pbspl1	6
nbp2-OST66	67	1	2	28	1	39	66433.pbspl1	6
nbp2-OST6e	104	8	3	32	0	49	66278.pbspl1	6

Future work

- Working through the NASA open-source process. Distribution will include:
  - Lustre tapset library
  - big-object and oststat
  - Mechanism for mapping hosts to your site's batch system
- More tools!
  - Send me your ideas.
  - Better yet, patches :-)
- Visualization

References

- SystemTap
  - http://sourceware.org/systemtap/
- Understanding Lustre Filesystem Internals

Questions?

jason.rappleye@nasa.gov