OpenAFS and Large Volumes

Mark Vitale <mvitale@sinenomine.net>

AFS and Kerberos Best Practices Workshop
19 August 2015

objective / overview

- We will explore some of the current challenges and solutions for using OpenAFS with very large volumes.

How big is a piece of string?

case study

A large university stores research project data on a single OpenAFS volume:

- millions of files and directories, almost 1 terabyte total
- content-addressable directory hierarchy
  - e.g. one directory per person “query_subject_nnnnn”
  - AFS files-per-directory limits forced awkward workarounds
- original volume okay, but unable to make a usable copy
  - ‘vos move/copy/dump/restore’ would run for many hours then fail
  - triggered salvage which would run for many more hours, then either fail or produce damaged data
  - workaround: tar/untar “WAY out of band”

outline / agenda

- big data considerations
- internals
- addressing the issues
- futures
- questions and answers

big data considerations

- architectural limits
- implementation limits
- server limits (disk, memory)
- operational limits (time, SLA)
- configuration changes
- performance issues (scalability)
- bugs – increased exposure of infrequently exercised code paths

architectural limits

SINE NOMINE ASSOCIATES

- OpenAFS must maintain interoperability with other AFS servers and clients as described in existing architecture, specification, and standards documents.
- Much of this is “wire format” for rx, RPCs, and objects.
- Two that are relevant to large volumes are the FID and the directory object.

the FID (File Identifier)

SINE NOMINE ASSOCIATES

- uniquely identifies a file on the wire

```
struct AFSFid {
afs_uint32 Volume;
afs_uint32 Vnode;
afs_uint32 Unique;
}
```

- architectural (wire) limit 2^32 (4Gi)
  - Volumes per cell
  - Vnodes per volume (dirs, files, symlinks)

the directory object

SINE NOMINE ASSOCIATES

- architectural (wire) limits:
  - no more than 64436 (not 65536) objects (files, dirs, symlinks) per directory
    - each name no longer than 15 chars
    - longer object names reduce the runtime limit for a given directory by consuming more (contiguous) blocks
    - fragmentation may also reduce the runtime limit
  - object name 256 bytes or less
- details in Mike Meffie's EAKC 2012 presentation:
  http://conferences.inf.ed.ac.uk/eakc2012/slides/eakc-2012-dir-defrag.pdf

the Volume Location database (VLDB)

- architectural (wire) limit in current RPCs of 2 GiB maximum size for an ubik database
- this is enough to hold approximately 14.5 million volumes per cell in the current VLDB format 4
  - far less than the 4 Gi limit the FID imposes

implementation limits

- 255 fileservers per cell
- 255 vice partitions per fileserver
- fileserver vice partition formats:
  - old “inode”
  - Windows “namei”
  - Unix “namei”

vice partition format – Unix namei

SINE NOMINE ASSOCIATES

- **/vicepXX** partition
  - **Vnnnnnnnnn.vol** files (VolumeDiskHeader)
  - **AFSIDat** directory
    - 1st level volume directories (hashed lower bits of RW volid)
      - volume directories (hashed RW volid)
        - 1st level inode directories (hashed high bits of vnode number)
          - 2nd level inode directories (hashed next 9 bits)
            - inode files (more hashing)
        - **special** directory
          - volume info – 1 per volume in the VG (RW, RO, BK, clones)
          - large vnode index – 1 per volume in the VG
          - small vnode index – 1 per volume in the VG
          - link index – 1 per VG
- deliberate incorporation of metadata redundancies

SINE NOMINE
ASSOCIATES

I'm not really sure what (the) intended purpose is. To me, this is one of those "historical reasons" where "historical reasons" means "it doesn't make any sense but it was like this when we got it".

-- Andrew Deason

namei implementation limits

- NAMEI_VNODEMASK = 0x03FFFFFF
  - 26-bit (not 32) vnode number
  - 67,108,863 max vnodes per volume
    - half "small" – files, symlinks (including AFS mountpoints)
    - half "large" – directories (and associated ACLs)
- max 20 ACLs per directory
- max 7 links to a shared (cloned) vnode (RW, RO, BK, 4 clones)

“Working As Implemented”

bugs

- WAD – "Working As Designed"
- WAI – "Working As Implemented"
- *unintentional* implementation limits
- large volumes + infrequently exercised code paths = undiscovered bugs
- examples:
  - salvage
  - vos operations: move, copy, dump, restore

Can AFS make a volume so large that it cannot move it?

typeless arithmetic

SINE NOMINE ASSOCIATES

- the heart of many of these issues is that the vnode number is used in unsafe calculations:
  - bit offset into an internal volume memory map
  - byte offset into the vnode index special files

vos move

- bugs are rarely exposed for very large volumes because no one wants to move them - it takes too long
- performance may mask bugs, as well as being a kind of bug in its own right.

salvager

- looks for errors and inconsistencies in AFS volume data and metadata, and makes a best effort to repair them.
- one of those programs (like restore) that is not used very often, but when you do you really hope it works!
- what if the salvager breaks, or takes too long?

SINE NOMINE
ASSOCIATES

Fix all the bugs you want…
we'll make more!

(with apologies to Lay's potato chips)

addressing the issues

SINE NOMINE ASSOCIATES

- files-per-directory limit
  - workaround: implement site-specific nesting of user directories to skirt the AFS limits
  - workaround: use shorter file and directory names when possible
  - workaround: use defrag tool (ask Mike)
  - standardization: extended directories

addressing the issues

SINE NOMINE ASSOCIATES

- errors and corruption during volume operations (copy/move/dump/restore)
  - **root cause:** incorrect (signed) type for a vnode number led to a 31/32 bit overflow when calculating an offset into a vnode index.
  - patch submitted to fix the specific issue (gerrit 10447)
    - reviewers called for code audit and a general fix
  - new patches in test:
    - 31/32 bit (signed/unsigned) type errors
    - type-dirty macros (index offsets) – convert to inline static functions
    - error recovery bugs exposed along the way

addressing the issues

SINE NOMINE ASSOCIATES

- salvager assertion fail while salvaging a large volume
  - exposed by vos move bugs (previous slide)
  - root cause: SalvageIndex had an incorrect (signed) type for a vnode number; this caused the vnode index offset logic to overflow at 2^31 and pass a negative file offset to IH_WRITE
  - patch in gerrit

addressing the issues

SINE NOMINE ASSOCIATES

- mkdir may crash dafileserver
  - bug 1: allocating a new vnode overflows the index logic at 2^31 due to incorrect type; issues “addled bitmap” and goes to rarely-traversed error recovery code
    - patch in gerrit
  - bug 2: error recovery mishandles volume reference counting and crashes on a failed assert
    - patch in gerrit

addressing the issues

SINE NOMINE ASSOCIATES

- salvager resource requirements
  - uses “invisible” (unlinked) temp work files
    - e.g. salvage.inodes – contains info on each inode
    - default location is on the partition itself
    - may grow quite large for large volumes, leading to unexpected disk space exhaustion during salvage
    - recommend using –tmpdir parm to put these elsewhere
  - uses large memory-resident tables
    - “vnode essence” tables that “distill” the vnode index metadata
    - may result in a visit from the Linux OOM-killer

addressing the issues

SINE NOMINE ASSOCIATES

- configuration changes
  - just creating the test volumes exposed some problems in my cell config:
    - cache manger: memcache/disk cache; separate partition; separate device; adequate stat caches (to prevent vcache invalidation and VLRU thrashing)
    - fileserver: adequate callbacks (to prevent callback GSS)
  - first accesses exposed more perf considerations
    - cold CM caches
    - cold fileserver caches
    - ls command coloring causes extra FetchStatus calls
    - CM prefetch activity

addressing the issues

SINE NOMINE ASSOCIATES

- performance & scalability
  - vos move/copy/dump/forward profiling
  - salvager dir buffer optimizations
  - salvage index scanning bottlenecks
  - fileserver housekeeping optimizations

"Depend upon it, sir, when a man knows he is to be hanged in a fortnight, it concentrates his mind wonderfully.“

-- Samuel Johnson

current status

- under gerrit review:
  - **10447** vol: VnodeId type consistency for vnode numbers
  - **11983** DAFS: dafileserver crash after "addled bitmap“
  - **11984** DAFS: dafileserver failed assertion (vp->nUsers >= 0)
  - **11985** vol: DEBUG_BITMAP for vnode allocation debug, test
  - **11986** vol: vnodeIndexOffset not typesafe

futures

- Finish current audit/bug hunt/cleanup project
- Death to salvager?
  - Why do we even have a salvager?
    - to repair (after the fact) damage caused by:
      - hardware software failures
      - AFS design shortcomings (non-logging, etc?)
      - AFS bugs
  - DAFS improved some pain points in the salvager:
    - greatly reduced salvage related outages
    - salvage on demand, with no outage for other volumes
    - salvage in parallel
- Extended directory objects?

Questions?