# Demand Attach / Fast-Restart Fileserver

Tom Keiser
Sine Nomine Associates

# Introduction

- Project was commissioned by an SNA client
- Main requirement was to reduce fileserver restart time by $> 60\%$
- Development & testing performed over 7 months in 2005–2006 timeframe
- Resulted in $> 24,000$ line patch to OpenAFS HEAD
- Patch was committed to OpenAFS CVS in 03/2006
- Pulled up to development branch in time for 1.5.1 release

# Motivations for Redesign

- The on-disk format has a number of limitations
  - Metadata is stored in per-volume files
  - Metadata has poor spatial locality
  - No journalling
- Volume Package locking model doesn't scale
  - One global lock
  - No notion of state for concurrently accessed objects
  - Lock is held across high-latency operations
- Breaking CallBacks takes a lot of time
- Keeping the server offline throughout the salvage process is unnecessary (with certain caveats)

# Proposed Changes

- Introduce notion of a volume finite-state automata
- Only attach volumes as required (demand attachment)
- Only hold volume lock when absolutely necessary
- Perform *all* I/O outside of the global lock
- Automatically salvage volumes as required
- Parallelize fileserver shutdown process
- Stop breaking CallBacks during shutdown
- Save/Restore of host/callback state
- Volume garbage collector to offline infrequently used volumes

# Implementation

# Demand Attach Architecture

- Demand Attach consists of five components
  - demand attachment
  - `salvageserver`
  - host/callback state save and restore
  - changes to `bos`/`bosserver`
  - ancillary debug and introspection tools
- Components have substantial interdependencies
- It would be difficult to make them separable

# Fileserver Startup Sequence

*Old Fileserver*

- vice partition list is built
- all volumes are fully attached

*Demand Attach Fileserver*

- host/callback state is restored
- host/callback state consistency is verified
- vice partition list is built
- list of volumes on each partition is scanned
- all volumes are placed into “pre-attached” state

# Fileserver Shutdown Sequence

*Old Fileserver*

- break callbacks
- shut down all volumes

*Demand Attach Fileserver*

- quiesce all host and callback state
- shut down all online volumes
- verify consistency of host/callback state
- save host/callback state

# Feature Comparison

| Feature | 1.2.x | 1.4.x | DAFS |
|---|---|---|---|
| Parallel Startup | no | yes | yes |
| Parallel Shutdown | no | no | yes |
| Tunable data structure sizes | no | no | yes |
| lock-less I/O | no | no | yes |
| automatic salvaging | no | no | yes |
| lock-less data structure traversals | no | no | yes |
| volume state automata | no | no | yes |

# Salvageserver

- Receives salvage requests from the fileserver or bos salvage
- Maintains a salvage priority queue
- Forks off new salvager worker children as needed
- Handles log file combining
- Maintains N concurrent worker children whose task orders are load balanced across all vice partitions

# Volume Garbage Collector

- Underlying assumption is many volumes are accessed so infrequently that it is inefficient to keep them attached
- The inefficiency has to do with optimizing fileserver shutdown time
- The garbage collector is modeled after the generational GC paradigm
  - Frequently accessed volumes must be dormant for a longer period to be eligible for offlining
  - Infrequently accessed volumes are scanned more frequently to determine their offlining eligibility
  - A background worker thread occasionally offlines a batch of eligible volumes

Performance

# Startup Performance

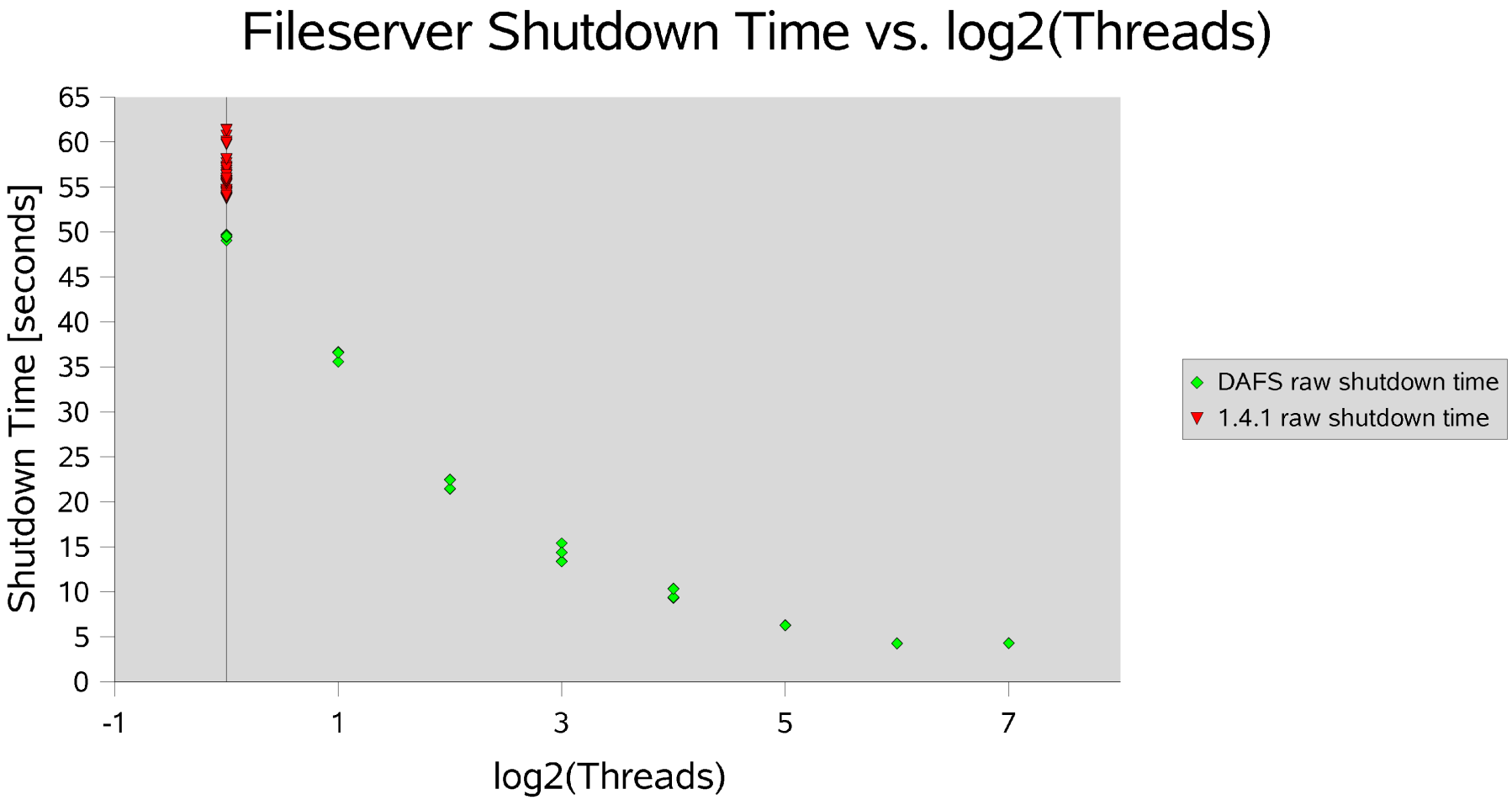
- Startup time is now heavily limited by underlying filesystems directory lookup performance
- Even for 60,000 volumes across 3 partitions, startup times were still under 5 seconds running namei on Solaris UFS
- Host/CallBack state restore can take several seconds if the tables were nearly full
  - This is a cpu-bound problem, so faster machines mostly obviate this
  - Actual restore is fast; verification is slow
  - Verification can optionally be turned off with the `-fs-state-verify` argument

# Shutdown Performance

Benchmark Environment

- Sun E450 (4x 480MHz 8MB cache, 4GB RAM)
- 4Gb aggregate multipathed FC fabric attachment
- Solaris 10 zone
- 56 vice partitions spread across LUNs on various IBM FAStT and Sun arrays
- 44 physical spindles behind the vice LUNs
  - mixture of 10k and 15k FC-AL disks
- 128 volumes per partition (to keep test iteration time low)

# Shutdown Performance



# Shutdown Performance

- It turns out the test system was sufficiently slow to turn the shutdown algorithm into a CPU-bound problem much beyond 32 threads
- Kernel statistics collected point to very high context switch overhead
- As a result, we don't know the true lower bound on shutdown time for the underlying disk subsystem
- Aggregate microstate CPU time was 85% kernelspace
- Taking N=32 as the last good value, we get a parallel speedup of $\frac{49.49}{6.28} \approx 7.88$ for 32 threads
- This yields a parallel efficiency of $\frac{7.88}{32} \approx 0.25$

# Administration

# New Bos Create Syntax

The `bos create` syntax has been changed to deal with the online salvage server:

```
bos create <hostname> dafs dafs \
  /usr/afs/bin/fileserver \
  /usr/afs/bin/volserver \
  /usr/afs/bin/salvageserver \
  /usr/afs/bin/salvager
```

# Bos Salvage

- `bos salvage` must change because implementation is intimately tied to bnode name
- bnode name had to change to resolve ambiguity between MR-AFS and Demand Attach Fileserver
- Cannot manually salvage volumes on demand attach fileserver with older `bos` client
- New `bos` client maintains identical salvage arguments, except `-forceDAFS` must be passed to perform a manual salvage on a demand attach fileserver

# Salvageserver

- Automated salvaging is not without its pitfalls
- Infinite salvage,attach,salvage,... loops are possible
- To halt this progression, a hard limit on salvages for each volume is imposed
- This counter is reset when the filserver restarts, or when the volume is nuked
- The `-Parallel` argument controls the number of concurrent salvage workers
- Occasional full-server salvages (requiring an outage) are still a good idea

# Introspection and Debugging Utilities

- `salvsync-debug` provides low-level introspection and control of the salvage scheduling process
- `fssync-debug` provides low-level introspection and conrol of the fileserver volume package
- `state_analyzer` provides an interactive command line utility to explore and query the fileserver state database
- All three of these utilites have context-sensitive help commands
- `fssync-debug` and `salvsync-debug` commands should be understood before use; misuse could cause Bad Things (tm) to occur

## Concluding Thoughts

# When not to use Demand Attach

- environments where host tables overflow
  - e.g. lots of mobile clients, NATs, etc.
  - multi_Rx needs to be replaced before we can overcome this limitation
- environments where fileserver crashes happen frequently
  - demand salvages take longer to run than full-partition salvages
  - future work should solve this problem

# Potential Future Development Directions

- Fine-grained locking of volume package
- Replace current daemon_com mechanism with unix domain sockets or Rx RPC endpoints
- Continually store list of online volumes into fileserver state database
- Seed salvageserver with list of previously attached volumes following a crash
- Provide fine-grained volume state from vos commands
- Dynamic vice partition attachment/detachment without fileserver restart

# Demand Attach Fileserver Clustering

- Automated failover is very fast due to demand attachment
- Fewer volumes to salvage following a crash
- Salvage time is not a component of failover time
- There is potential to build on the Demand Attach infrastructure to provide robust fileserver clustering
  - Treat volume/host/client/callback state as a memory-mapped on-disk database during runtime, as well as during startup/shutdown
  - If/When Partition UUID support happens, dynamic rebalancing of partitions across an N-node fileserver cluster becomes feasible

# Conclusions

- `Fileserver` restart time has been drastically reduced despite poor scaling properties of the on-disk format
- Further performance gains are going to be increasingly difficult
- New on-disk format could be very fruitful
  - Improve spatial locality
  - Optimize for DNLC performance characteristics on namei
  - Balance disk I/O subsystem performance characteristics with cache hierarchy performance characteristics using something like self-similar B-trees

# Conclusions

- Diminishing returns associated with context switch overhead are a serious limit to how far synchronous I/O algorithms can scale
- Using asynchronous I/O (on certain platforms) could yield significant further performance gains
- Demand Attach architecture opens up a new realm of possibilities in the space of High Availability and Fileserver Clustering

# Demand Attach Fileserver

Questions?