Apache Traffic Server Cache Toolkit API

Speaker

- Alan M. Carroll, Apache Member, PMC
  - Started working on Traffic Server in summer 2010.
  - Implemented
    - Transparency, IPv6, other stuff (like cache!)
  - Works for Network Geographics
    - Provides ATS and other development services

Outline

- Basics of the current cache and API
- The Cache Toolkit API
- In practice – using the toolkit to build solutions to current user issues.

What do we have now?

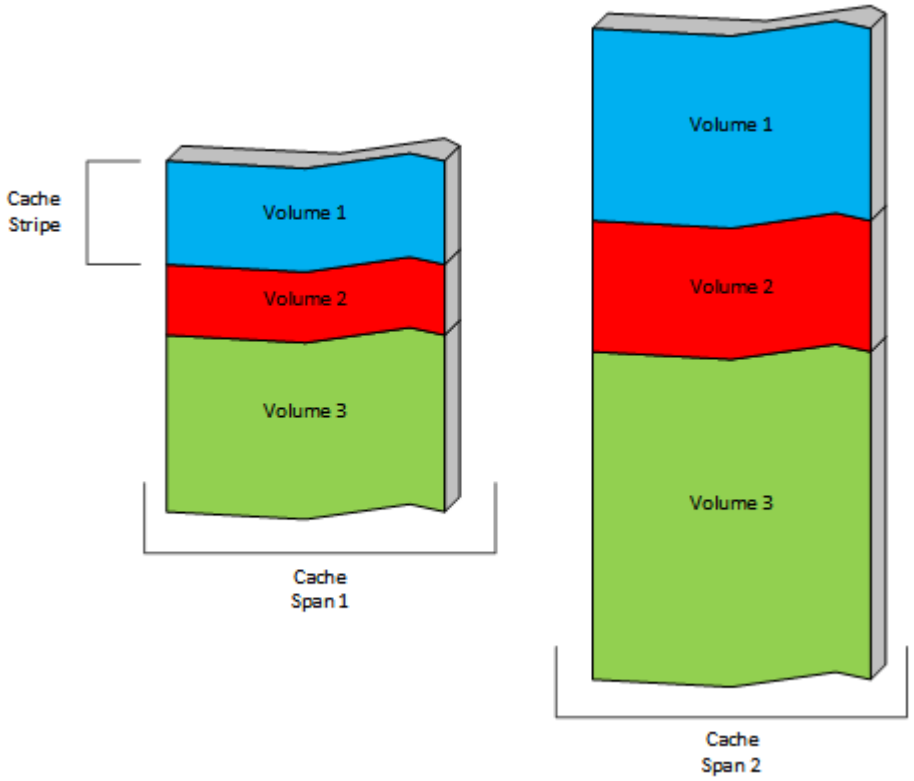
CURRENT CACHE AND API

Cache Structure

- Cache span – storage on a specific physical device or file.
- Cache volume – administrative unit of cache storage
- Cache stripe – intersection of cache span and cache volume

Spans, Volumes, Stripes

The default is to split each physical span in to a stripe for each volume. The volumes can be differently sized leading to differently sized stripes. The stripes are not currently visible from outside the ATS core, administrative control is of volumes.



Stripes

- Cache stripes are the internal base storage.
- Each stripe has its own directory of objects.
- Object storage is a circular buffer.
- Each object is stored entirely in a single stripe.

Stripe Assignment

- Storing an object requires selecting the stripe. This is called *stripe assignment*.
- Reading an object requires remembering where it was assigned for writing.

Keys and IDs

- Each object has a *cache key* which is a string.
  - Default is the URL for the object.
- Cache keys are hashed to create a *cache id*.
- The cache id is used as a locator in a stripe directory.
- Stripe assignment is **independent** of cache keys and cache ids.

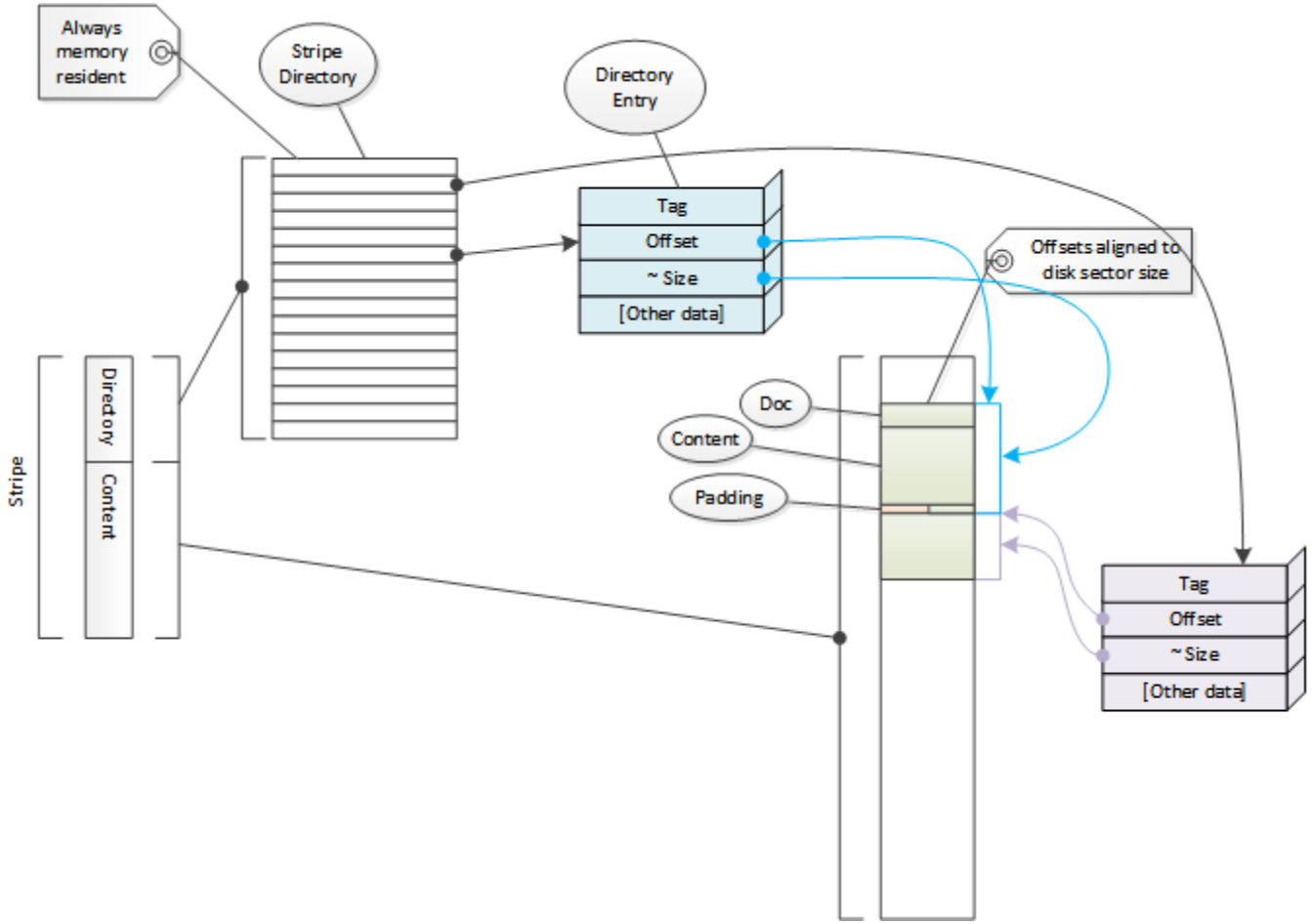
Notes

- Cache misses are usually fast, no I/O required.
- Objects can be evicted from cache due to overwrite via the circular buffer.
  - But, no fragmentation or GC issues.
  - No concept of “full” or “empty”.
- Data is never updated, only written anew.

Fragments

- Serialized object data is stored in a *fragment*.
- Fragments are grouped for write if small but read individually.
- Each fragment has a *Doc* which is the header.
- All metadata for an object is stored in the first fragment.
- Every stored fragment has an entry in the stripe directory.

Stripe Directory

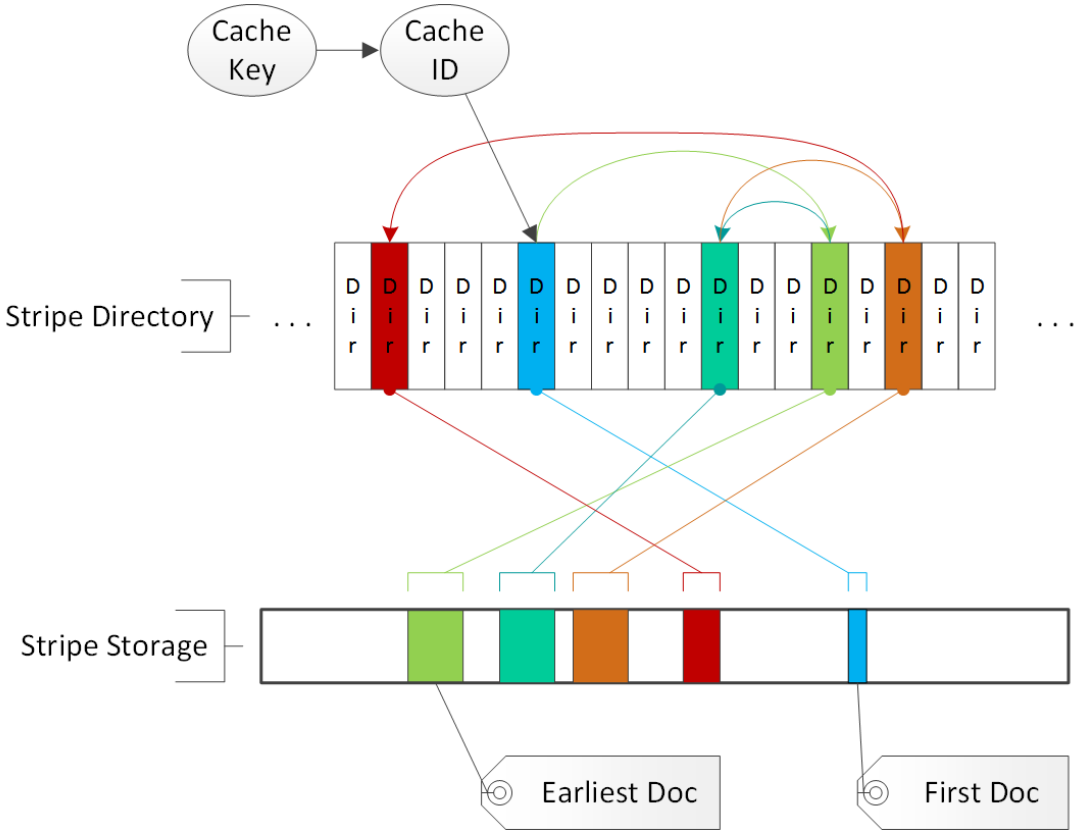


Large Objects

- Large objects are stored in multiple chained fragments in a single stripe.
  - There is a target maximum fragment size
- Because an object is always in a single stripe, object size is limited by stripe size, not volume size.
  - Therefore also by cache span size.
  - Therefore adding disks doesn't let you store *larger* objects, only *more* objects.

Multi-Fragment Object

The cache key is converted to a cache ID which is then used to locate a directory entry (“Dir”). This in turn is used to locate the first fragment with the metadata (“First Doc”) for the object. “Earliest Doc” is the first fragment with content. Each alternate has a different Earliest Doc.



Alternates

- An object can have multiple stored versions.
- Each version is an *alternate*.
- Alternates are based on the Vary field.
- Alternates have the same first fragment but different earliest fragments.

Current API

- Cacheability.
- Cache key.
- Alternate selection.
- Limited ability to scan objects.
  - Metadata (HttpInfo) API appears unimplemented.
- No access to cache data structures.

API Problems

- Can’t interact with stripes.
- Very limited inspection of cache state.
- Unstable, partially implemented, and undocumented.

What to build

CACHE TOOLKIT API

Motivations

- Inadequacies of current API
- Traffic Server increasingly used in more specialized / vertical ways
  - Leads to desire for very custom extensions
- Always good to get increased code cleanliness and modularity
  - Mechanisms used for the API can also be used by the core for better modularity.

Demotivations

- Users don’t want
  - Extra testing and design for the general case.
  - Constantly porting changes to new versions.
  - Publishing code
  - Legacy maintenance
  - Hiring expensive expertise like me
- We don’t want
  - Code complexity used in just one deployment
  - Similarly for configuration complexity

Back Story

- History
  - Started as tiered storage design effort.
    - Initial concept at previous ATS Denver Summit.
  - Presented at Yahoo! ATS Summit.
    - Everyone wanted different tiered features.
    - Then they wanted different cache features.
- Since I made the mistake of learning a little bit of how the cache worked, I was assigned to do this.

A Brilliant Idea

- Avoid these issues and get the good stuff with a general (“toolkit”) API for the cache that can be used with plugins.
    - “Brilliant!” – Leif
    - “Supremely clever!” – Brian
    - “Inspired!” – James
    - “Is it done yet?” - users

Goals

- Make cache state, data, operations visible.
- Override cache actions dynamically.
- Initiate cache actions.

- Provides the tools for users to build the cache feature they want.

Visibility

making what was unseen visible

- Expose internal structures
  - Directories, directory entries.
  - Cache volumes, disks, stripes.
- Opaque pointer with accessor functions.
- Iteration functions for search.
- Useful itself but also required by rest of API.

Hooks

Getting in to the action

- Cache events
  - Wrap, disk fail, life cycle, etc.
- Object events
  - Directory miss, disk miss, evacuation, delete (?)
- Special case support for pure statistic hook?
- Stripe assignment.

Stripe Assignment

- Stripe assignment controls accessibility and storage for objects.
- Parallel write.
- Parallel search on multiple stripes for read.
  - Callback per read complete, can accept or wait.
- Retry option for read
  - Sequential stripe searching.
  - Alternate key searches.

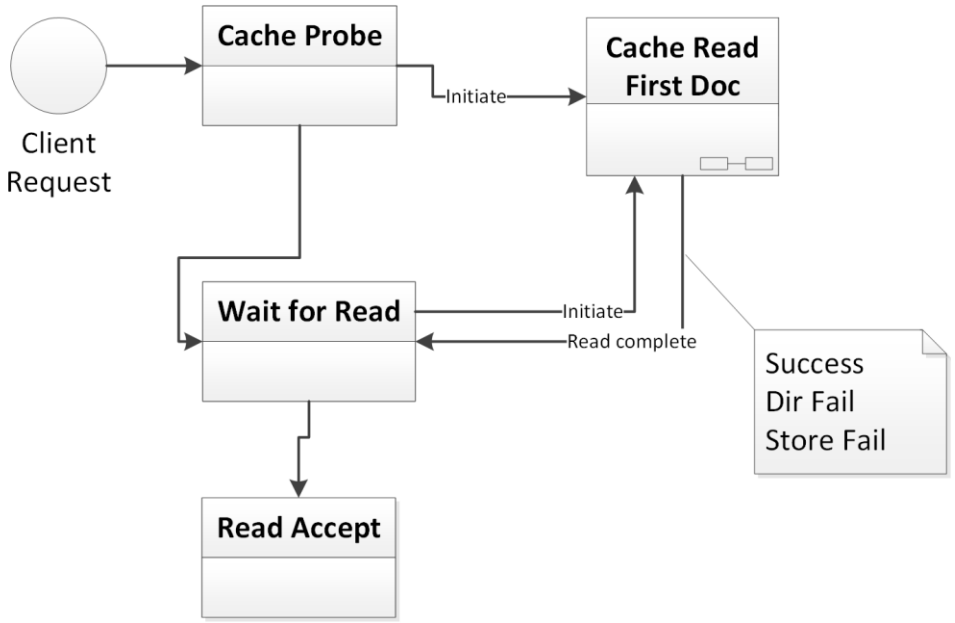
Cache Operations

- Copy objects from stripe A to stripe B
  - More efficient to parallel store if possible
- Probe, delete directory entries.
- Evacuation marking.
- Read, write cache objects.
- Alternate control.

Reading from Cache

The plugin is called once the client request has been parsed. It should then initiate cache reads (or fail and mark the request as a cache miss).

When a read completes the plugin can accept that as that source for a cache hit, discard the read, and/or initiate additional read operations. The plugin can reject any completed read as a miss.



Miscellaneous

- Allow storage and volume definitions to have key/value pairs that are visible to the API.
- Cache key / ID control
  - General string input.
  - Alternative hashing.
- Integration of RAM cache
  - Should be “just another stripe”.

Out in the fields

USE CASES

Awesome Yet Practical

- Not just a cool API, but useful for actual deployment problems!
- Design was use case driven.
- So let’s look at some of them.

Monitoring

- Problem: Can't optimize cache use if we have no knowledge of what's happening
- How much of my content is still valid?
- How many directory misses with disk I/O are occurring?
- What is the average chain length in the directory?
- How are objects distributed across stripes?

Monitoring

- Solution: Use API to explore / monitor cache state.
  - Hooks to
    - Watch for interesting events.
    - Generate statistics.
    - Generate logging data for post processing.
  - API to examine raw cache state
    - One off plugins for debugging.
    - Build external tools for examination.
      (product idea, don't steal!)

Annotation

- Problem: want better information about cache contents.
- Monitoring can be used to accumulate more extensive data about cache contents.
  - Live information.
    - TS data.
    - Custom tags/tables.
  - Offline log processing.

Tiered Storage

- Problem:
  - Two or more types of storage of significantly different access speeds.
  - Want to control content location
- Solution:
  - Create cache volumes based on storage type
  - Assign objects to stripes based on type of object and stripe storage.

Tiered Storage 2

- Can use multiple write to store on multiple tiers.
  - Demoting means just delete in directory (no I/O)
- Can still copy if needed.
- Can search tiers in parallel or serial

Persistence Control

- Problem: some content should be kept for long periods, longer than a stripe wrap.
- Solution: segregate content by persistent vs. transitory across stripes.
  - Large stable objects only evicted by other large stable objects, not small temporary objects.
  - Can read failover from long term to short term
  - Potentially do generational GC

Cache Key Flexibility

- Problem: Cache lookup commits to a single stripe.
  - Variant keys can causes misses
  - Any change in stripe assignment hides objects
- Solution: Use parallel read or retry
  - Read from a different stripe
  - Use a different key

Large Hot Objects

- Problem: A small number of large objects that are frequently accessed, causing excessive disk load on a stripe.
- Solution: Write the objects to multiple stripes and use stripe assignment to rotate access across the stripes.

Export

- If you can examine the cache via an API, you can export it.
- Cache state is becoming valuable to many users, export can preserve or copy it.
- Import can be done without changes, if with difficulty
  - But should try to make it easier

Cache Cleaning

- Problem: clean non-stale objects
- Solutions:
  - List of URLs (or URL regex) to purge.
  - Force cache miss on read if URL match.
  - Background scan to pre-emptively delete.

Note: This can be done in the current API. It will just become much easier.

Scripts and Resources

APPENDIX

State of Development

- Some work has been done on prototyping some of the features.
  - Learned much about the cache implementation
- Design is mostly finalized.
- Positive response so far from the community.
- Project is funded.
- Development should start late spring this year.

Versioning / Custom Metadata

- Speculative feature!
- Fragment type is not really used currently
  - Overload it to allow custom parsing of metadata
  - Use it to version objects/stripes.
    - Need _flen too, but that's OK.
- This would be quite complex
  - Need API to unpack alternates.

Resources

- ATS has online documentation, a wiki, mailing lists, bug tracker, and IRC channel. Access these via
  - http://trafficserver.apache.org
- Active community – become involved!
- NG Consulting services
  - http://network-geographics.com