Tech Corner

Minor technical updates

StringView

Introduction

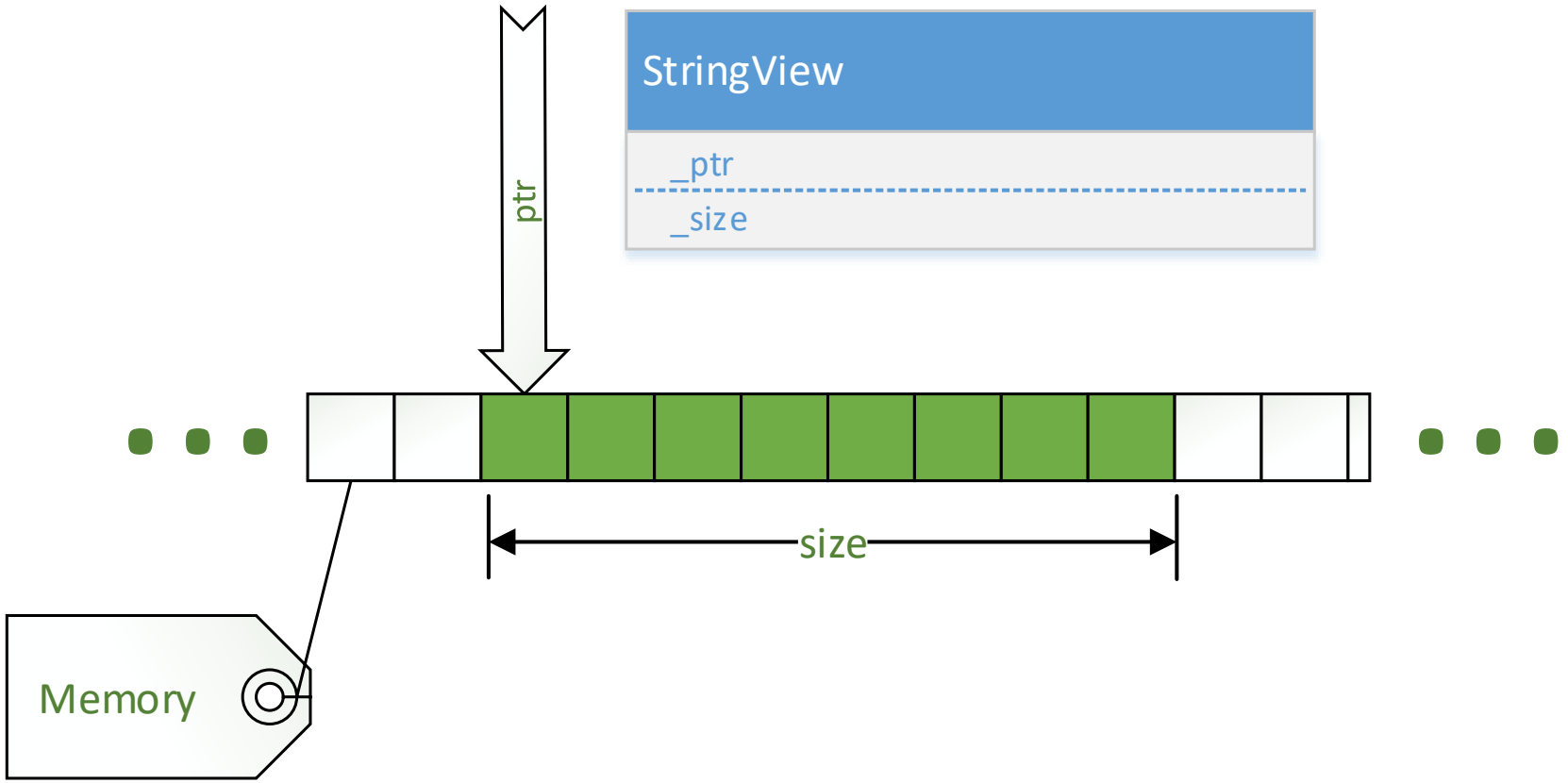
- StringView is a class defined in lib/ts/MemView.h.
- Contains a **read only** view of memory as character data.
- StringView never allocates.
- StringView instances depend on some other object owning the memory.
  - Must be careful of data lifetime issues.
- Useful where
  - A substring is needed only while the original still exists.
  - Instead of explicitly passing a pointer and length for a string.

STL Compliance

- C++17 adds “std::string_view” class.
- Very similar in function to StringView but done independently.
  - Boost.StringRef was considered as a model but it lacked essential features.
- Different method naming
  - For 8.0 StringView will have its method names and naming style changed to conform to std::string_view for ease of use.
- StringView is a superset of std::string_view functionality.
- StringView will be made to interoperate with std::string_view.

Basics

- A StringView instance contains a base pointer to the start of the memory view and a length for the size of the view.
- Acts like a pointer as much as possible.
  - Equality operator compares only pointer and length.
  - Compare contents with methods and free functions.
- StringView provides methods to make string parsing easier.
  - Find characters / character types.
  - Split StringView (as with strtok but without modifying the string).
- Overloaded strcmp family of functions.


StringView
_ptr
_size
ptr
size
Memory

Benefits

- StringView instances are cheap to create and copy.
- Inexpensive way to reference temporary substrings.
- More convenient than passing an explicit pointer and length and just as fast.
- Strong support for string parsing.
  - Easily replaces strtok and the internal “tokenizer” library.

Example: VIA header

- PR 1520 – Change VIA header to include full protocol stack.
- Leif bikeshedded about all the strlen use.
- PR 1534 is a StringView based version.
  - No memcpy in the API.
  - Only memcpy to the output VIA string without using strlen.
  - Enhanced StringView to avoid allocation even on initialization.
  - Source strings are static – no lifetime issues.

Parsing support

- Can copy or split a prefix from the StringView.
  - Easy to do the equivalent of strtok without allocation or string modification.
  - Trivial to do on substrings because of the lack of modification. E.g. split on white space first, then on another separator such as ‘=‘.
- Can trim characters from front and back.
- Can increment like a pointer.
- Lifetime can be managed by reading the entire file and keeping that buffer around. Instance of StringView are views of that buffer.

Parsing

- Style is layered.
  - Outer loop splits out lines.
  - Next loop splits out tokens.
  - Tokens can be further split as needed (e.g. key value pairs split on ‘=‘).
- CacheTool uses StringView to parse the storage.config and volume.config files with much less library support.

```
Errata
Cache::loadSpanConfig(FilePath const &path)
{
  static const ts::StringView TAG_ID("id");
  static const ts::StringView TAG_VOL("volume");

  Errata zret;

  ts::BulkFile cfile(path);
  if (0 == cfile.load()) {
    ts::StringView content = cfile.content();
    while (content) {
      ts::StringView line = content.extractPrefix('\n');
      line.ltrim(&isspace);
      if (!line || '#' == *line)
        continue;
```

```
ts::StringView path = line.splitPrefix(&isspace);
if (path) {
  // After this the line is [size] [id=string] [volume=#]
  while (line) {
    ts::StringView value(line.extractPrefix(&isspace));
    if (value) {
      ts::StringView tag(value.splitPrefix('='));
      if (!tag) { // must be the size
      } else if (0 == strcasecmp(tag, TAG_ID)) {
      } else if (0 == strcasecmp(tag, TAG_VOL)) {
        ts::StringView text;
        auto n = ts::svtoi(value, &text);
        if (text == value && 0 < n && n < 256) {
        } else {
          zret.push(0, 0, "Invalid volume index '", value, "'");
        }
      }
    }
  }
```

```
        zret = this->loadSpan(FilePath(path));
      }
    }
  } else {
    zret = Errata::Message(0, EBADF, "Unable to load ", path);
  }
  return zret;
}
```

CPP API

- CPP API should be converted to use StringView in most places std::string is used now.
  - This is particularly true for the transform logic. Copying the data to be processed by the override method seems unnecessarily expensive.
  - However passing a raw pointer and length was too ugly.
  - StringView provides the best of both.

Notes on VIA changes

- Internal strings for ALPN were changed to be StringView instances
- A special constructor was added to cause the StringView to be a view of the literal string.
  - Only one copy of the string.
  - Not even strlen is used in the constructor.
  - No lifetime issues.
- Many of the strings were moved to ink_inet.cc because that seemed a more appropriate home than the HTTP proxy string parsing.

Notes

- Would like to make StringView available to plugins.

MemView

Read only memory view

Basics

- MemView is very similar to StringView.
- Originally the same class but differences became too large to stay unified.
- MemView treats its contents as a slab of bytes.
- Allows searching for a value or condition in its view as if it were an array of arbitrary types.
  - Rarely useful for classes.
  - Very handy for multi-byte integral values.
- Allows array access for arbitrary types. Similar utility as for searching.

Notes

- Not as useful as StringView but still handy in places where currently a void * and length are passed. MemView is more convenient.
- Very tempted to make a writeable version.
  - I've encountered a number of situations where having a memory window that is writeable would be quite nice. Not yet convinced it's a good idea.

Cache Tool

traffic_cache_tool

Cache Tool

- New command line tool for Traffic Server.
- Operates on Traffic Server cache
- Currently provides some analysis on cache structure.
- Can pre-allocate cache volumes.
  - Being used internally to fix the problem reported by Yahoo! and Comcast where a disk fails but when replaced is still not used by the cache.

History

- Originally done via the “stripe inspector” using the --check option to traffic_server.
- This was in practice not usable because of the memory requirements, which in turned required stopping Traffic Server.
- Assigned to fix problem with replaced cache disks not being used.
- Built traffic_cache_tool from scratch.
  - Reduce run time footprint.
  - Create understandable code.
  - Experiment with new cache data structures.
  - Add additional features

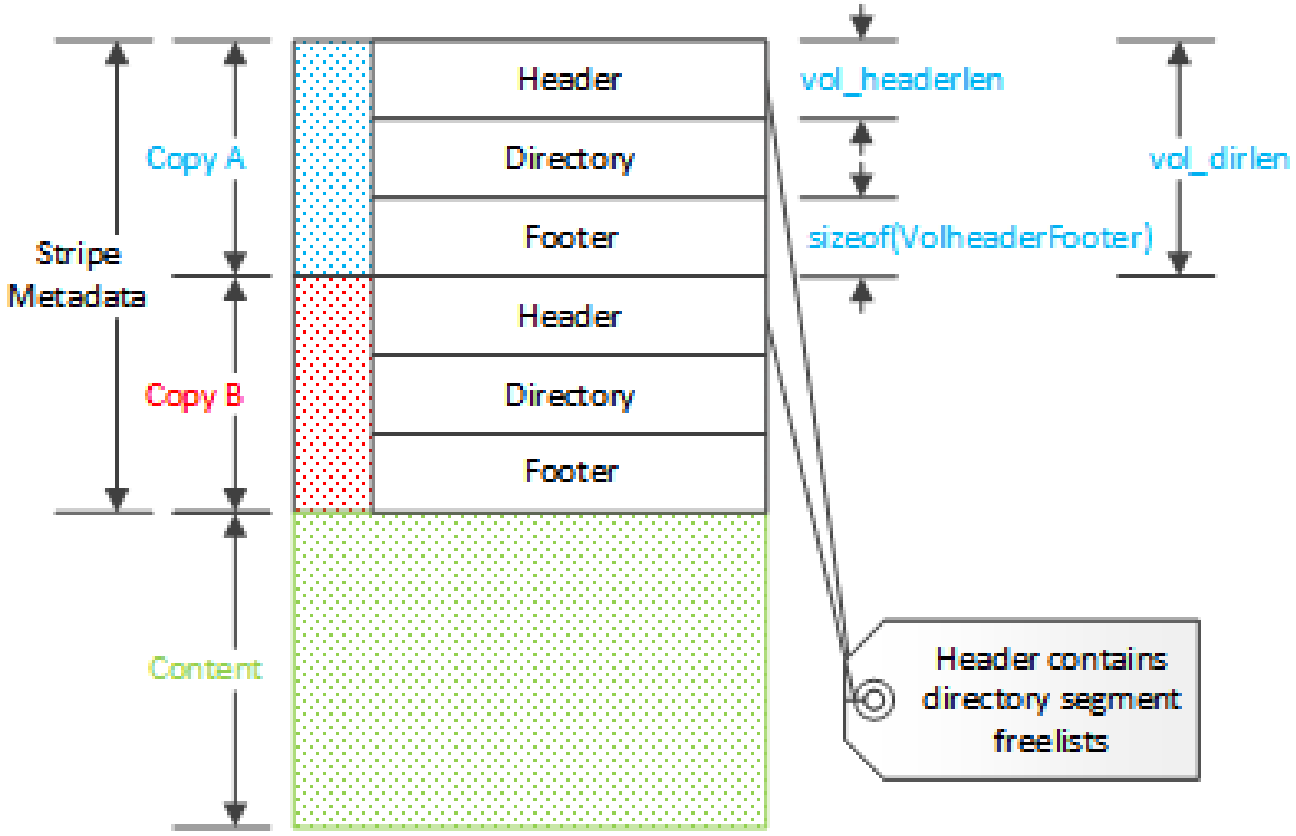
Scratch Built

- Reduce run time footprint.
- Create understandable code.
- Experiment with new cache data structures.
- Experiment with new layout algorithms.
- Additional features that only make sense for a separate tool.

The Challenge

- Critical bits of information are missing from the storage.
- Traffic Server recomputes all the cache structure every process start.
  - Resets cache if computations don’t agree with cache state.
  - Critically the average object size controls data structure sizes.

Cache Stripe Layout



Techniques

- Need to read the content directory until the footer is found.
- Can validate against the B copies because the sizes are determined if the footer had been found.

Improvements to base logic

- Much faster and more even allocation of disk space to volumes.
- Better organized data structures.
- Accept layout in preference to clearing cache.

Future work

- Directory analysis
  - Still not feature complete with the stripe inspector.
- Robustness in the face of live cache
  - Double check the A and B copies to validate a consistent read.

Changes

- Change stripe meta data to have
  - Number of segments
  - Number of buckets per segments
  - So little data, so much win…
- Problem with backward/forward compatibility
  - A problem if a newer version is rolled back with a different stripe structure.
  - Use two structures. One with extra data tucked behind existing data and another that is the entirely new structure.
  - 8.0 adds the extra without changing the current data to existing caches, uses the new structure for new caches.
  - Cleared caches aren't forward compatible but that's OK.

Goals

- With additional data in the stripe meta data those values can be read instead of computed and therefore changed.
- Then structure of an existing cache can be changed without destroying the cache.
- The effective average object size could then be adjusted with minimal loss of data. This is simply not currently possible.
- Eventually will be possible to disable cache initialization in Traffic Server and allow on the CacheTool to put disks into and out of cache.

Scalar

Quantized and scaled integer values.

Scaled values

- Scalar is a template library that allows the creation of scaled / quantized values.
- A scalar instance has a scale and a value.
  - Scale is the multiplier.
  - The value is the unitary value of the scalar.
  - A scalar with a scale of 100 has possible values of 0, 100, 200, 300…
- The scale is a compiler constant, an instance is only the size of an integer.

Operations

- Scalars silently convert from large scales to smaller ones because information is not lost.
- Conversion from smaller to larger scale requires a cast to indicate rounding up or down.
- Construction is very cheap.
- Increment / decrement is by scale.
- Binary operators require homogenous types or scale casting.
  - With mixed types, rules for automatic resolution become incomprehensible.
  - Unfortunately result is “x += 4;” is valid but “x = x + 4;” is not.
  - I consider this the lesser evil.

Use

- Driven by the needs of the CacheTool and all the crazy sizes inside the cache.
- Makes it much harder to pass the wrong scaled value.
- Still need to be careful about rounding.
- Still a bit of a work in progress, testing out in the CacheTool implementation.

Possible changes

- Add function operator that converts a raw integral value into a scaled value of the same scale.

```
Scalar<100> x(4); // x value is 400
Scalar<10> y(8); // y value is 80
x += 4; // x value is now 800
x = round_up(y) + x(6); // x value becomes 700
```

Partial Object Caching

The millstone of my life

Current Status

- I’m working on it.