cloudera
Ask Bigger Questions

What's New in Apache HTrace

by Colin P. McCabe

About Me

- I work on HDFS and related storage technologies at Cloudera
- Committer on the Hadoop and HTrace projects.
- Previously worked on the Ceph distributed filesystem

Introducing Apache HTrace

- A new Apache project to do distributed tracing
- Owl-themed

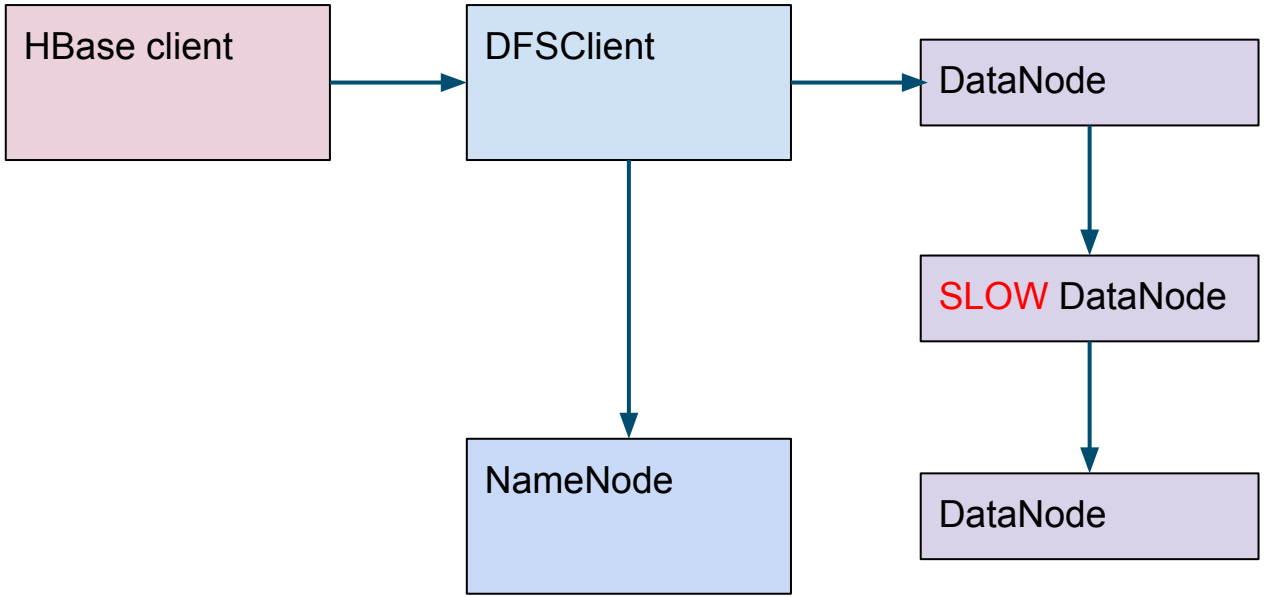
What is Distributed Tracing?

- Follow specific requests across the entire cluster
- Follow requests across network **and project** boundaries

Why do Distributed Tracing?

- Diagnosing distributed performance is hard
- Many timeouts and fallbacks
- Performance problems often not 100% repeatable

HBase + HDFS Performance Analysis



Real-World Scenarios

- The cluster is “running slower” lately... why?
- Is it worthwhile to spend time optimizing X?
- Why was the cluster slower over the weekend?
- Is the performance problem a MapReduce problem or an HDFS problem?
- Why so many “EOFException” logs?

Previous Approaches: log4j

- Use log4j to log “especially slow” disk I/O
  - What’s “especially slow”? Won’t logging make it slower?
  - There is no good way to map the log messages back to the requests that had problems
  - Too many DataNode log files to look at, usually no motivation to look
  - Similar problems with other log4j approaches

Previous Approaches: metrics

- Single node metrics through jmx, top, vmstat, etc.
  - Good for getting an overall view of throughput, bad for identifying latency problems.
  - Average bandwidth, CPU, disk I/O, etc. numbers often hide significant outliers
  - Hard to figure out **why**
    - Disk I/O stats are low… because of I/O errors? Bottlenecks elsewhere? Low load?

HTrace Approach

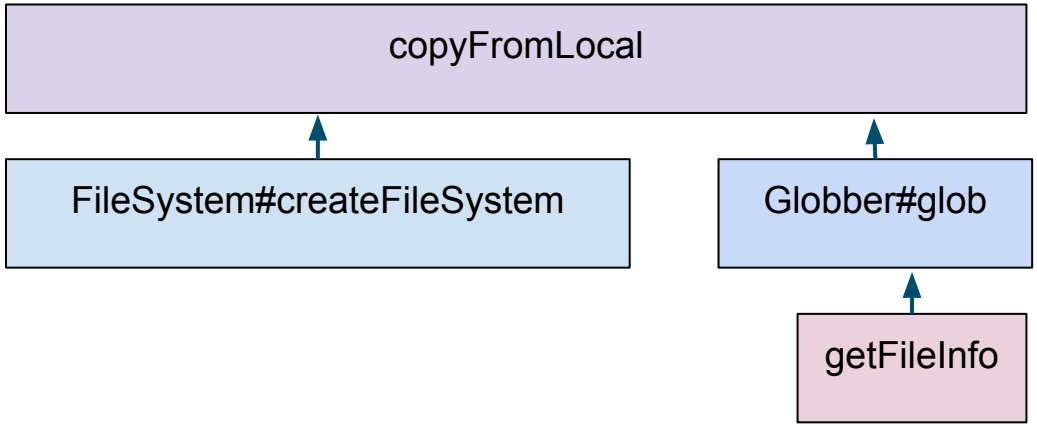
- Decompose requests into **trace spans**
- Each distributed system uses the HTrace client software to create trace spans while performing certain operations.

Trace Spans

- A trace span represents a length of time. They have:
  - A description
  - Start time in milliseconds
  - End time in milliseconds
  - Unique identifier
  - Tracer ID
  - Other metadata

Trace Span relationships

- Trace spans can have “parents.” Spans form a directed acyclic graph (DAG)



Correlating Spans

- Span IDs are passed over Hadoop and HBase RPCs
- Each process reports the spans that were generated, and the parents of each span.
- The processes that are being traced send their spans to a SpanReceiver
- The SpanReceiver stores all of the span objects for later examination.

Sampling

- Tracing all requests generates an enormous amount of data
- It's usually more useful to do sampling-- to trace only < 1% of requests
- Sampling rate and type is configurable
  - CountSampler -- samples at a fixed period
  - ProbabilitySampler -- samples with a uniformly random probability

Goals

- Language-agnostic
- Framework-agnostic
- RPC-agnostic
- Can trace both libraries and applications

Goals

- Support multiple storage backends
- Stable, well-supported client API
- (Near) Zero impact when not in use
- Can be used in production
- Integration with upstream big data and Hadoop projects, to allow end-users to enable tracing without writing code.

Modularity

- HTrace is language-agnostic
  - Supports Java, C, C++, …
- HTrace is RPC-agnostic
  - Hadoop RPC, HBase RPC, etc.
- Many different “span receivers” are available.

Modular Architecture

- Client library
  - htrace-core / libhtrace.so
- Span Receivers
  - htrace-hbase
  - htrace-accumulo
  - htrace-htraced
- Web UI

Client Library

- Applications use the client library to generate traces
  - For Java: *htrace-core4.jar*
  - For C/C++: *libhtrace.so*
- The client library is a shim that insulates the traced application from the rest of htrace.
  - The client doesn't need to care what SpanReceiver, Sampler, or GUI we are using.
  - Downstream projects depend *only* on htrace-core

Client Library Key Concepts

- Tracer
  - Represents a service or client being traced.
  - For example, the FsClient, DataNode, and NameNode each have their own Tracer objects.
  - Created by *Tracer#Builder*
  - A single process may have multiple Tracer objects.
  - Typical tracer ID: *NameNode/192.168.0.1*

Client Library Key Concepts

- Sampler
  - Samplers are stored inside Tracer objects.
  - Samplers implement sampling by deciding when trace spans should be created.
  - Samplers are created from the HTrace configuration passed into the *Tracer#Builder* object.

Client Library Key Concepts

- TraceScope
  - Trace scopes are created by *Tracer#newScope*.
  - The new TraceScope may or may not have a span associated with it. If it does, that span will be sent to the SpanReceiver when the TraceScope is closed.

Client Library Key Concepts

- *Tracer#newScope*
  - If there is already a Trace scope in the current thread, we use that existing scope's decision about whether to trace.
  - If there is not already a trace scope then the new scope is "top-level" and we ask the Tracer's Sampler whether we should create a span
  - We trace complete requests, not fragments

Client Library Key Concepts

- TraceScope
  - Creating a TraceScope creates
  - For example, the FsClient, DataNode, and NameNode each have their own Tracer objects.
  - A single process may have multiple Tracer objects.
  - Typical tracer ID: *NameNode/192.168.0.1*
  - Tracer objects are used to create trace scopes.

Example Code

```
Tracer tracer = Tracer.Builder(“MyApp”).build();
try {
  TraceScope scope = tracer.newScope(“runFoo”);
  try {
    runFoo(Arrays.copyOfRange(argv, 1, argv.length));
  } finally {
   scope.close();
  }
} finally {
  tracer.close();
}
```

SpanReciever

- A SpanReceiver is a place to send trace spans.
- To be useful, a SpanReceiver needs to provide easy access to those trace spans. It should store them in some indexed form to support lookups and analysis.
- SpanReceivers may use existing NoSQL datastores to scale (like HBaseSpanReciever) or implement their own infrastructure (like HTracedSpanReceiver)

HBaseSpanReceiver

- Stores HTrace spans in HBase
- See the htrace-hbase subproject in htrace.git
- Very effective for users who already have HBase deployed
- Very scalable

AccumuloSpanReceiver

- Stores HTrace spans in Accumulo
- Maintained by the Accumulo community

HTracedSpanReceiver

- Stores HTrace spans in a separate htraced daemon
- htrace uses LevelDB to store trace spans in an optimized and indexed format
- Easier to get started with than other options
- Better integration with GUI (for now…)

What's New in HTrace 4.0?

4.0

API Improvements in HTrace 4.0

- Better support for tracing library code
  - Previously we focued on tracing just server code
  - The 4.0 API avoids instantiating multiple SpanReceiver objects if tracing is configured for multiple services (example: HDFS client + HBase server)
  - Support multiple Tracer objects with appropriate names (i.e. FileSystem vs. RegionServer)

API Improvements in HTrace 4.0

- Simplified configuration
  - Can specify the span receiver to use only once by setting hadoop.htrace.span.receiver.classes, rather than once per type of Hadoop daemon.
  - Samplers are now configured through HTrace itself, rather than the applications which use HTrace.

API Improvements in HTrace 4.0

- Easier to use API for programmers
  - 3.x required a lot of boilerplate to add tracing to an application; 4.x takes care of most of this automatically.
  - Programmers don’t need to decide whether to pass in a Sampler object or not when creating a trace scope. The sampler will be used automatically for top level scopes.

API Improvements in HTrace 4.0

- Moved to using 128-bit randomly generated IDs for Spans
  - For all practical purposes, 128-bit IDs eliminate the possibility of collisions that existed with 64-bit IDs

API Improvements in HTrace 4.0

- Adopted backwards compatibility policy for htrace-core
  - We will not make backwards-incompatible changes to htrace-core during HTrace 4.x
  - Avoid CLASSPATH issues in downstream projects
- Smooth migration from 3.x to 4.x, since htrace-core from 3.x and 4.x can both be on the CLASSPATH. Different namespaces and jar names.

Build System Improvements in HTrace 4.0

- Fixed numerous build bugs and Maven warnings
- We established a postcommit build on Jenkins, using Docker
  - Using Docker also helps new developers get started quickly without hunting down dependencies
  - Reproducible build

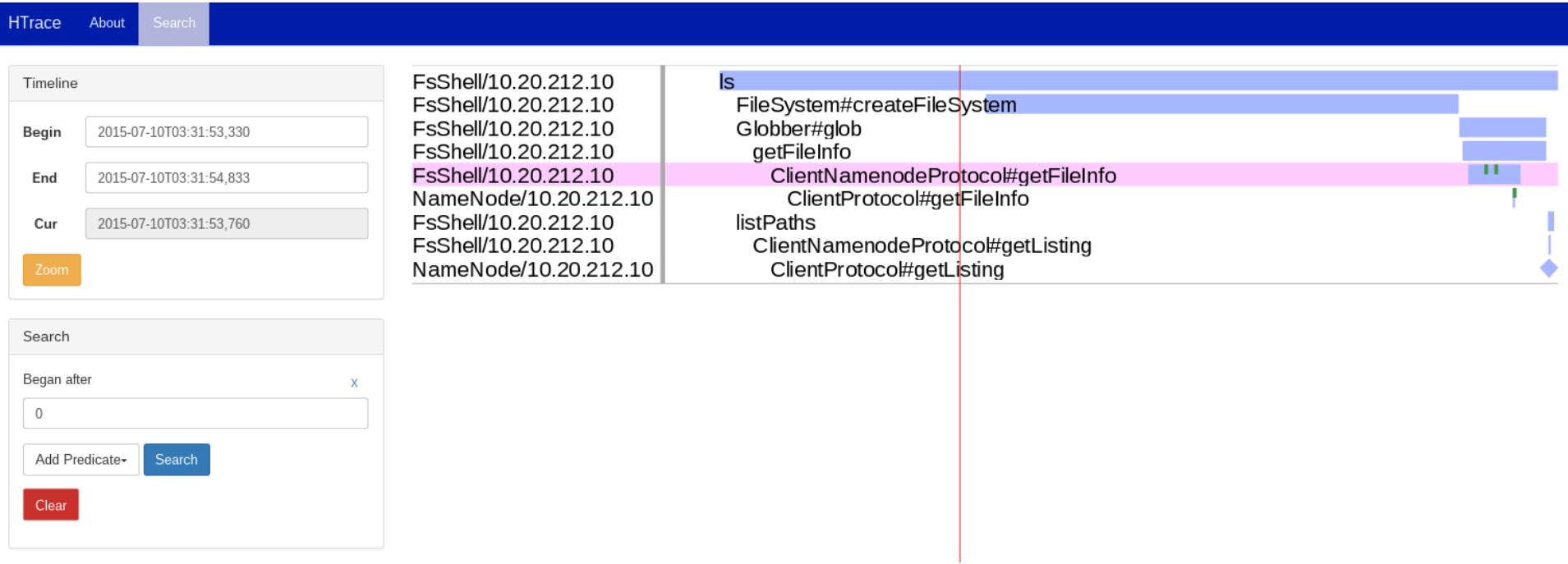
New Graphical User Interface in HTrace 4.0

- Supports visualizing requests from end-to-end.
- Can search for trace spans by multiple different criteria.
- Can compare two different requests on the same timeline.

New Graphical User interface in HTrace 4.0

- Implementation
  - Javascript / REST
  - Bootstrap
  - Backbone.js
- Goal: connect graphical user interface to many different SpanReceivers. Currently hooked up only to HTracedSpanReceiver.

HTrace Graphical Interface



HTrace 4.0 Summary

- Many improvements to the core client library and to our compatibility policies.
- Ready for production.
- New graphical user interface supplies the missing piece for understanding trace data.

HTrace Community

- Vibrant upstream community
  - Contributors from NTT Data, Cloudera, Hortonworks, Facebook, and others
  - Two releases in the last few months-- 4.0 and 4.0.1
  - Integration and sharing of ideas with Hadoop and related projects

Cool things to work on in HTrace

- Language support
  - Test C/C++ support
  - Add CPython or Rust clients wrapping libhtrace.so
  - Write a better Golang client
- Optimization
  - Optimize HTraced span receiver and other span receivers to minimize network traffic and performance impacts

Cool things to work on in HTrace

- Integration
  - Integrate HTrace into YARN and MR
  - Update Accumulo integration to 4.x
- GUI
  - Connect the GUI to HBaseSpanReceiver and other span receivers
  - Better aggregate views?

HTrace Q & A

Thanks for Listening!

http://www.cloudera.com/careers