Introducing Apache HTrace
by Colin P. McCabe
About Me

- I work on the Hadoop Distributed Filesystem and related big data technologies at Cloudera.
- Previously, I worked on the Ceph distributed filesystem.
Overview

- Motivations for HTrace
- HTrace Architecture overview
- Using HTrace
- The HTrace community
- Demo
- Q&A
Big Data in 2016

- Volume of data continues to grow: petabytes to exabytes
- New open source projects
  - Apache Spark
  - RecordService
  - Kudu
Big Data Challenges

- Larger clusters (thousands of nodes)
- More disks (density)
- Lower latency targets
- Manageability
- Monitoring
- Heterogeneous clusters
- Complex stacks
Example Big Data Stacks

HDFS
HBase
HDFS
Linux
Impala
Hive
Spark
RecordService
HDFS
Linux
Diagnosing Distributed Systems is Hard

- Many timeouts and fallbacks
- Performance problems often not repeatable
- Difficult to follow requests across project boundaries and network boundaries
Diagnosing Distributed Systems is Hard
Diagnosing Distributed Systems is Hard
Metrics

- Many different metrics available
  - JMX
  - top
  - vmstat
  - iostat
- Aggregated
- Downsamplesd over time
Metrics

- 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?
Log Files

- Daemons all generate log files
  - HDFS audit log
  - log4j files
  - Client log files
- Usually stored on the nodes that generated them
- Kept for some length of time, then deleted
Log Files

- Good for getting detailed information about a particular operation or point in time
- Bad for getting a holistic view of a single request. Difficult to correlate what is going on on different systems via logs
- Tradeoff between performance and logging
- Split into many different files
  - Per-host, per-project, per-faculty
HTrace’s Approach

- Distributed Tracing
  - Follow specific requests across the entire cluster
  - Follow requests across network and project boundaries
  - End-to-end tracing on a sampled subset of requests
End-to-End Tracing

- Multiple cluster nodes
- Multiple projects
  - Follow a request from HBase to HDFS
- Multiple languages (app vs. lib)
  - Java, C, C++ language bindings
- Use available storage and compute stack
HTrace Goals

- Support multiple storage and compute backends
  - Not tied to any one RPC, language, framework
- Stable, well-supported client API
- Approximately zero impact when not in use
- Can be used on production clusters
- Integration with upstream big data and Hadoop projects, to allow end-users to enable tracing without writing code.
Trace Spans

- Annotations decompose requests into **trace spans**
- Trace spans can be nested (parent/child relationship)
- Parent node can be a different node or system

```
copyFromLocal

FileSystem#createFileSystem    Globber#glob

getFileInfo
```
Trace Spans

- A trace span represents a length of time
  - Description
  - Start time
  - End time
  - Parents
  - Unique Identifier
  - Process ID and IP address
  - Time Annotations
  - Key/Value Annotations
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 sampler is configurable
- Sampling is currently done at the level of the entire request
Pluggable Architecture

- htrace4-core is the library for creating spans
- SpanReceivers process spans created by htrace4-core
- htrace-web queries SpanReceiver data
LocalFileSpanReceiver

- Stores spans in files on the local filesystem
- Can post-process files later with MapReduce, Spark, etc.
Span JSON

```json
{
    "a": "f8e9e09c72e388f3fef51b32115beba5",
    "b": 1448220893721,
    "e": 1448220893788,
    "d": "ClientNamenodeProtocol#create",
    "p": ["f8e9e09c72e388f3dc6778916cf3a5ac"],
    "r": "FSClient/10.20.190.31"
}
```
HTracedSpanReceiver

- Easy-to-use SpanReceiver that stores spans in a central daemon
- Indexing, web UI, aggregation in one place
htraced

- Written in Go
- rpc
  - Serializes spans via msgpack
  - Exposes REST + JSON API for webapp and command-line tools
  - Java and C clients
  - Handles overload gracefully
htraced

- storage
  - Optimized for high write throughput
  - Uses multiple leveldb instances to store span data
  - begin time, end time, duration, and span ID are indexed so that range queries are fast
  - leveldb persists data to disk
Using HTrace

- Adding HTrace support to applications
- Configuring HTrace
- Using the HTrace web interface
Adding HTrace Support to Code

- Link against htrace4-core (java) or libhtrace.so (C/C++)
- Allow HTrace to access the application or library configuration
- Add trace spans to measure important events
- Add annotations to trace spans
Adding HTrace Support to Code

- Some applications and libraries will need to pass parent trace IDs over the network
htrace-core API

- Tracer
  - Creates trace scopes
  - Each tracer has its own sampling configuration
  - Use is thread-safe

Tracer tracer = new Tracer.Builder("FsShell").
    conf(TraceUtils.wrapHadoopConf(
        SHELL_HTRACE_PREFIX, getConf())).build();
htrace-core API

- **TraceScope**
  - Manages the trace span for this thread (nests)

```java
TraceScope piScope = tracer.newScope("calculatePi");
try {
    calculatePi();
} finally {
    piScope.close();
}
```
● Span
  ○ The Trace span itself

  Span piSpan = piScope.getSpan();
  if (piSpan != null) {
    piSpan.addKVAnnotation("piDigits",
      Integer.toString(numPiDigits));
  }
htrace-core API Wrapper Classes

- Wrappers automatically create spans for work items
  - TraceRunnable
  - TracerCallable
  - TraceExecutorService

```java
Runnable myRunnable = tracer.wrap(myPiRunnable, "calculatePi");
```
htrace-core API Internal Classes

- **Sampler**
  - Determines which requests to trace
- **TracerID**
  - Represents the span ID of a trace
- **TracerPool**
  - Used to manage a group of Tracers
  - Usually the default TracerPool is fine
Configuring HTrace

- Determine which SpanReceiver to use
- Set up configuration
- Run htraced or other daemons if needed
Configuring HTrace in Hadoop

- Add htrace-htraced.jar to CLASSPATH (or whichever SpanReceiver is being used)
- Set up `hadoop.htrace.span.span.receiver.classes` and other HTrace configuration keys
- Set up htraced
- More instructions at
HTrace Community

- Vibrant upstream community
  - HTrace is an Apache open source Project
  - Contributors from NTT Data, Cloudera, Hortonworks, Facebook, and others
  - Two releases in the last few months-- 4.0 and 4.0.1
HTrace Community

- Sharing ideas with other big data projects
  - Hadoop
  - HBase
  - OpenTracing
  - XTrace
  - Twitter Zipkin
Recent Work in HTrace

- More effective error checking in the htrace client
- Optimized RPC format for sending spans to htraced
- Better integration with HDFS
- New web GUI for visualizing spans
- Trace spans are now tagged with IP address or hostname
- Span IDs extended to 128 bits to avoid collisions
HTrace in Cloudera’s Distribution of Hadoop

- Available as a Cloudera Labs “beta” for CDH5.5 and later
- HDFS tracing is supported
- RPMs and debs are available for htraced
Planned

- Improve the HTrace integration in HBase
- Add more annotations to Hadoop span data to get more insight
- Support more SpanReceivers
- Better integration with cluster management systems
- Improve and test C and C++ support
- Create an aggregate view for spans
HTrace Q & A