About Me

● I work on HDFS and related storage technologies at Cloudera
● Committer on the HDFS and Hadoop 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 an Impala 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
  - Process ID and IP address
  - Other metadata
Trace Span relationships

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

Diagram:
- copyFromLocal
- FileSystem#createFileSystem
- Globber#glob
- getFileInfo
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
Brief History of the HTrace Project

- HTrace 1.x: proof-of-concept. Developer tool only
- HTrace 2.x: integration with HBase. Still very “beta”
- HTrace 3.2: Many, many improvements
- HTrace 4.0: cleaned up the API to make it work better with library code. Make stuff work in production!
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 jars
- Span Receivers
  - htrace-hbase
  - htrace-accumulo
  - htrace-htraced
- Web UI
- Very different than many other tracing tools
htrace-hbase

- Stores HTrace spans in HBase
- Very effective for customers who already have HBase deployed
- Very scalable
htrace-accumulo

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

- 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…)

©2015 Cloudera, Inc. All rights reserved.
HTrace Graphical Interface

- Graphical Javascript interface
- Allows searching for trace spans by multiple different criteria
Recent Progress

- 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
Planned

● Fix some issues in client API
  ○ 128-bit trace span IDs to avoid collisions
  ○ Remove some globals that are causing problems
  ○ Reduce client-side “boilerplate”
  ○ Remove deprecated functions
● View aggregate span data in the GUI
● Integrate GUI with htrace-hbase and htrace-accumulo
Example Code

```java
TraceScope scope = Trace.startSpan(instance.getCommandName(), traceSampler);
try {
    exitCode = instance.run(
        Arrays.copyOfRange(argv, 1, argv.length));
} finally {
    scope.close();
}
```
HTrace Community

- Vibrant upstream community
  - Contributors from NTT Data, Cloudera, Hortonworks, Facebook, and others
  - Two releases in the last few months-- 3.1.0 and 3.2.0
  - Integration and sharing of ideas with Hadoop and related projects
Targets for HTrace 4.x

● End-to-end tracing for all of Hadoop
● Spot quality and performance issues early
● Accurately diagnose which component is having a problem
● Deal with hardware failures and slowdowns effectively
● Improve and test C/C++ support
HTrace in CDH5.5

- Will be available as a Cloudera Labs “beta”
- Integrated into HDFS and HBase
- RPMs and debs will be available for htraced
- Will be installable in the QuickStart VM
- Documentation provided
- Must use Cloudera Manager “Safety value” to configure HDFS and HBase with HTrace
Similar Projects

- Twitter Zipkin
- Google Dapper
- XTrace
HTrace Q & A