Reworking Observability in Ceph
An Interesting Case Of Distributed Tracing

Deepika Upadhyay
Agenda

- Introduction to Ceph and It’s Complex distributed processes
- Background about the Problem
- Our Approach to solve it
- Why Jaeger?
- Jaegertracing in Ceph
- Learning & Future Scope
Background about the problem
Ceph RADOS Architecture

- **CLIENT**
  - MDS, RGW, librbd, etc
  - perform IO against one or more RADOS pools

- **OSD**: Object Storage Daemon
  - Responsible for IO, replication, and recovery

- **MON**: Monitor
  - Provides a consistent sequence of cluster maps
How a read/write request propagates...

1. Client app issues read request, RADOS sends request to primary OSD
2. Primary OSD reads data from local disk and completes read request

Read

Write

1. Client app writes data, RADOS sends data to primary OSD
2. Primary OSD identifies replica OSDs and sends data, writes data to local disk
3. Replica OSDs write data to local disk, signal completion to primary
4. Primary OSD signals completion to client app
Why might an IO be slow?

- Read might have to block waiting for a write to complete
- The set of target osds might die due to failure (or recovery!)
- Object might be in the process of recovery
- Some particular OSD might be overloaded
- ...or faulty?
- Really hard to figure out!
Monolithic vs Microservice Architecture

Illustration by Lev Polyakov, http://levpolyakov.com
Why Logs are insufficient?

- Sufficiently verbose debugging to find such a problem is **slow**
- Given a cluster of hundreds of OSDs, it can be **very hard to identify where debugging should be enabled**
- **performance overhead**
- Discrete and distributed requests and events, often fail because of **multiple triggers across different distributed components**.
- The logs require a pretty unreasonable amount of **knowledge of ceph internals**
How are we going to solve it?
Distributed Context Propagation!!

- Assign Unique ID
- Storing Context
- Metadata propagation (across process boundaries)
- Capture event and timing
- Reassemble call tree
- Visualization
Distributed Context Propagation!!!

- Each **Span Contains**:
  - An operation **name**
  - Start and finish **timestamp**
  - **Span Tags**: A set of zero or more key:value. The keys must be strings. The values may be strings, bools, or numeric types.
  - **Span Logs**: each of which is itself a key:value map paired with a timestamp. The keys must be strings, though the values may be of any type.
  - **SpanContext**
  - **Baggage Items**: key:value pairs that cross process boundaries
Jaegertracing in Ceph
Starting Tracer

```cpp
static void setUpTracer(const char* serviceToTrace) {
    static auto configYAML = YAML::LoadFile("../jaegertracing/config.yml");
    static auto config = jaegertracing::Config::parse(configYAML);
    static auto tracer = jaegertracing::Tracer::make(
        serviceToTrace, config,
        jaegertracing::logging::consoleLogger());
    opentracing::Tracer::InitGlobal(
        std::static_pointer_cast<opentracing::Tracer>(tracer));
}
```
Instrumentation Code

● Adding a span

OpRequest::marker_span = opentracing::Tracer::Global() -> StartSpan(s, {opentracing::v2::ChildOf(&(OpRequest::osd_parent_span) -> context())});
OpRequest::marker_span->Log({"hit_flag_points", hit_flag_points},
);

● Finishing off a span

OpRequest::marker_span->Finish();
Implemented Ceph Spans

- PGOpItem_span
- do_request_span
- do_pg_op_span
- mark_flag_point_string
Example of a simple trace in OSD i/o path

submit_transaction_begins

20 Traces

Compare traces by selecting result items

osd_tracing: submit_transaction_begins 8213dte 0.57ms

Today 9:56:37 pm
a few seconds ago
Example of a simple trace in OSD i/o path

submit_transaction_begins tracepoint

20 Traces

Compare traces by selecting result items

osd_tracing: submit_transaction_begins 8213ms

Min Duration
e.g. 1.2s, 100ms, 500us
Log Metrics and Metadata Propagation
Trace Timeline:

- **osd_tracer: op-request-created 1866322**
  - Trace Start: December 12, 2019 1:15 PM
  - Duration: 0.66ms
  - Services: 1
  - Depth: 2
  - Total Spans: 4

### Service & Operation

- **osd_tracer op-request-created**
  - Queued for pg
  - Queued span duration: 0.04ms

- **osd_tracer reached_pg**
  - Reached span duration: 0.04ms

- **osd_tracer started**
  - Started span duration: 0.03ms
Why Jaeger?
Why Jaeger?

- Adaptive sampling and smart sampling strategies that are configurable
Why Jaeger?

- Adaptive sampling and smart sampling strategies that are configurable
- Low performance overhead
Why Jaeger?

- Adaptive sampling and smart sampling strategies that are configurable
- Low performance overhead
- Always ON Tracing (For debugging in Production Environment)
Why Jaeger?

- Adaptive sampling and smart sampling strategies that are configurable
- Low performance overhead
- Always ON Tracing (For debugging in Production Environment)
- Standardization
Why Jaeger?

- Low performance overhead
- Always ON Tracing (For debugging in Production Environment)
- Standardization
- Adaptive sampling and smart sampling strategies that are configurable
- Backwards compatibility with Zipkin
Status

- OSD IO path annotations complete
- Current work in PR pending completion of jaeger dependencies being added to ceph’s cmake
- Future Work
  - Testing on use cases for OSDs:
    - detecting performance bottlenecks
    - service dependency analysis
    - root cause analysis
  - Instrumenting other parts of Ceph (RGW, BlueStore, etc)
Thank you.