Overview of tools, techniques and tips - Scaling Ozone performance to max out CPU, Network and Disk

Duong (duong@apache.org)

Ritesh (ritesh@apache.org)

Tanvi (tanvi.penumudy@cloudera.com)

Agenda

Surface area of projects

- What is Ozone?
- Cover the wide surface areas of projects to improve performance and scale
- Discuss lessons learned
- Tips and tricks

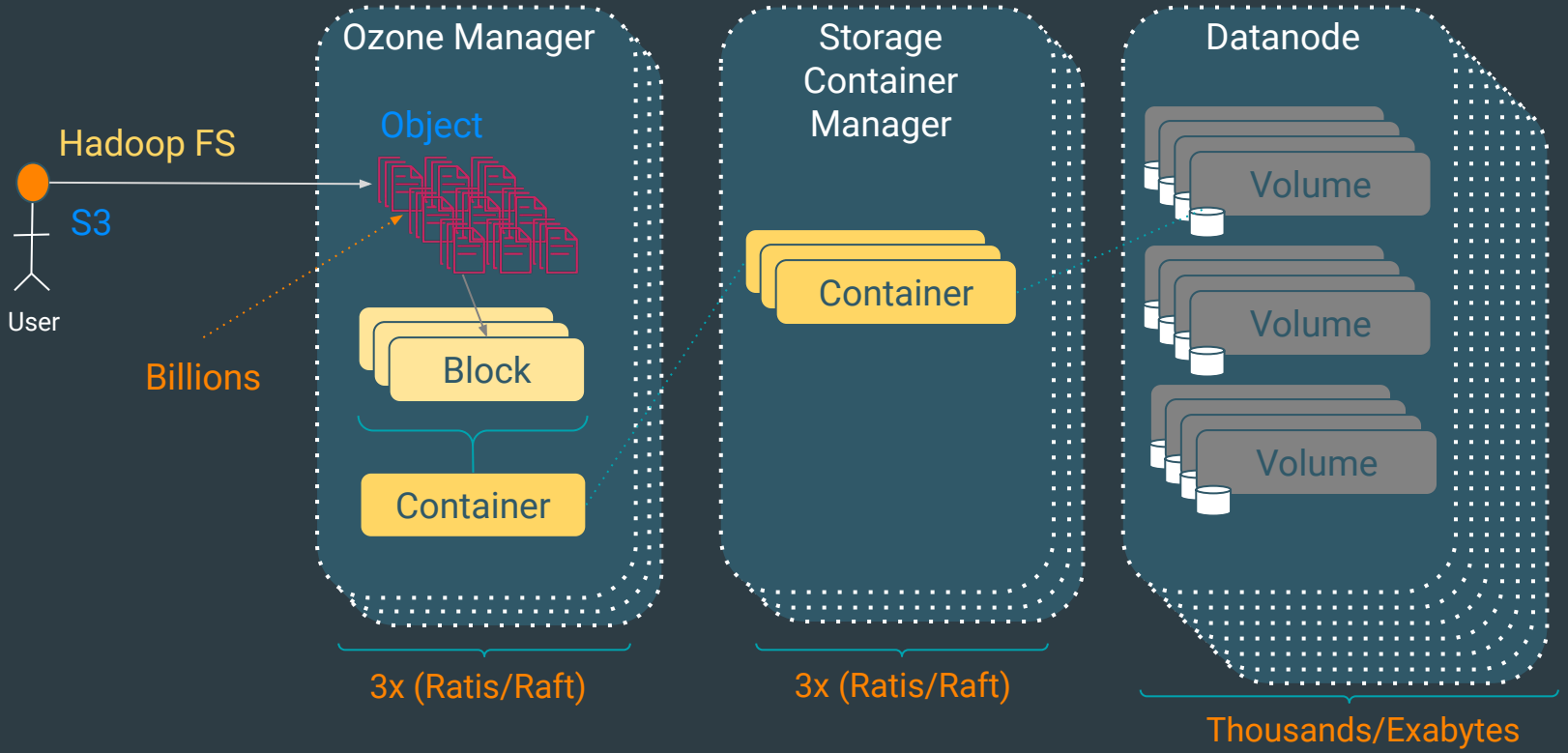
INTRODUCTION

What is Ozone?

- Apache Ozone is a top level Apache project: github.com/apache/ozone
- Object store created within the big data ecosystem
- Implements Hadoop Filesystem Interface and S3 API
- Designed to
  - Scale
  - Be strongly consistent
  - Optimize big data workloads
  - Efficiently cater to object store workloads
  - Provide atomic filesystem operations and object semantics

OVERVIEW OF OZONE

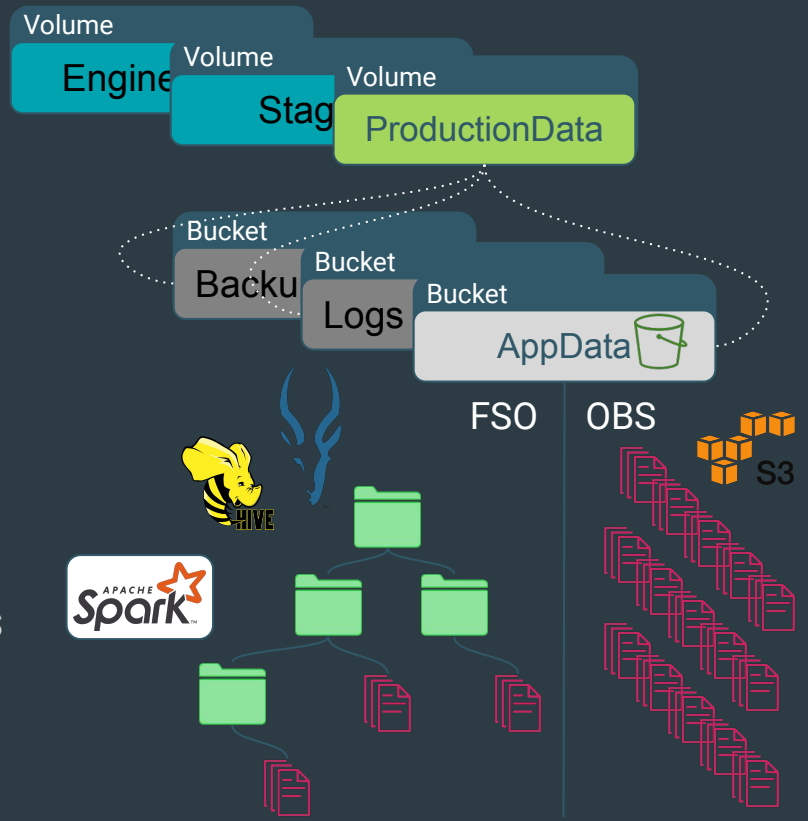
Major services and their scale



Ozone Namespace

/volume/bucket/key

- **Volume**
  - Top level namespace grouping
  - Can only contain buckets
  - Tenancy
- **Bucket**
  - Contains key
  - Types
    - OBS: S3 compatible keyspace
    - FSO: Filesystem with atomic renames
- **Key**
  - A dir/file stored in the system



Frugality over flexibility

List keys performance with protocol improvements: HDDS-9079

- Redundant information included per entry in listing keys to client.
  - Heavy penalty when serializing and deserializing
  - Total number of keys that can be listed in aggregate is directly proportional to the size of each object due to the cost of (deserailization and serialization)*2 + network transfer.
- Most common client side invocations needs only part of the payload
  - Optimize for common case
  - If client needs detailed object layout pay the price for multiple round trips.
- Performance difference for multiple clients issuing list keys in parallel.
  - 20% faster for 1kb objects (845 obj/sec)
  - 20k % faster for 10 GB objects (1,799,500 obj/sec)

Proto for ListKeys before

TMI

```
message ListStatusResponse {
    repeated OzoneFileStatusProto statuses = 1;
}
```

```
message OzoneFileStatusProto {
    optional KeyInfo keyInfo = 2;
    optional uint64 blockSize = 3;
    optional bool isDirectory = 4;

}
```

```
message KeyInfo {
    required string volumeName = 1;
    required string bucketName = 2;
    required string keyName = 3;
    required uint64 dataSize = 4;
    required hadoop.hdds.ReplicationType type = 5;
    optional hadoop.hdds.ReplicationFactor factor = 6;
    repeated KeyLocationList keyLocationList = 7;
    required uint64 creationTime = 8;
    required uint64 modificationTime = 9;
    optional uint64 latestVersion = 10;
    repeated hadoop.hdds.KeyValue metadata = 11;
    optional FileEncryptionInfoProto fileEncryptionInfo = 12;
    repeated OzoneAclInfo acls = 13;
    optional uint64 objectID = 14;
    optional uint64 updateID = 15;
    optional uint64 parentID = 16;
    optional hadoop.hdds.ECReplicationConfig ecReplicationConfig = 17;
    optional FileChecksumProto fileChecksum = 18;
    optional bool isFile = 19;
    optional string ownerName = 20;
    repeated hadoop.hdds.KeyValue tags = 21;
  // expectedDataGeneration, when used in key creation indicates that a
  // key with the same keyName should exist with the given generation.
  // For a key commit to succeed, the original key should still be present with the
  // generation unchanged.
  // This allows a key to be created an committed atomically if the original has not
  // been modified.
    optional uint64 expectedDataGeneration = 22;
}
```

```
message KeyLocationList {
    optional uint64 version = 1;
    repeated KeyLocation keyLocations = 2;
    optional FileEncryptionInfoProto fileEncryptionInfo = 3;
    optional bool isMultipartKey = 4 [default = false];
}
```

```
message KeyLocation {
    required hadoop.hdds.BlockID blockID = 1;
    required uint64 offset = 3;
    required uint64 length = 4;
    // indicated at which version this block gets created.
    optional uint64 createVersion = 5;
    optional hadoop.common.TokenProto token = 6;
    // Walk around to include pipeline info for client read/write
    // without talking to scm.
    // NOTE: the pipeline info may change after pipeline close.
    // So eventually, we will have to change back to call scm to
    // get the up to date pipeline information. This will need o3fs
    // provide not only a OM delegation token but also a SCM delegatic
    optional hadoop.hdds.Pipeline pipeline = 7;

    optional int32 partNumber = 9 [default = -1];
}
```

```
message Pipeline {
    repeated DatanodeDetailsProto members = 1;
    // TODO: remove the state and leaderID from this class
    optional PipelineState state = 2 [default = PIPELINE_ALLOCATED];
    optional ReplicationType type = 3 [default = STAND_ALONE];
    optional ReplicationFactor factor = 4 [default = ONE];
    required PipelineID id = 5;
    optional string leaderID = 6;
    repeated uint32 memberOrders = 7;
    optional uint64 creationTimeStamp = 8;
    optional UUID suggestedLeaderID = 9;
    repeated uint32 memberReplicaIndexes = 10;
    optional ECReplicationConfig ecReplicationConfig = 11;
    // TODO(runzhiwang): when leaderID is gone, specify 6 as the index of leaderID12
    optional UUID leaderID128 = 100;
}
```

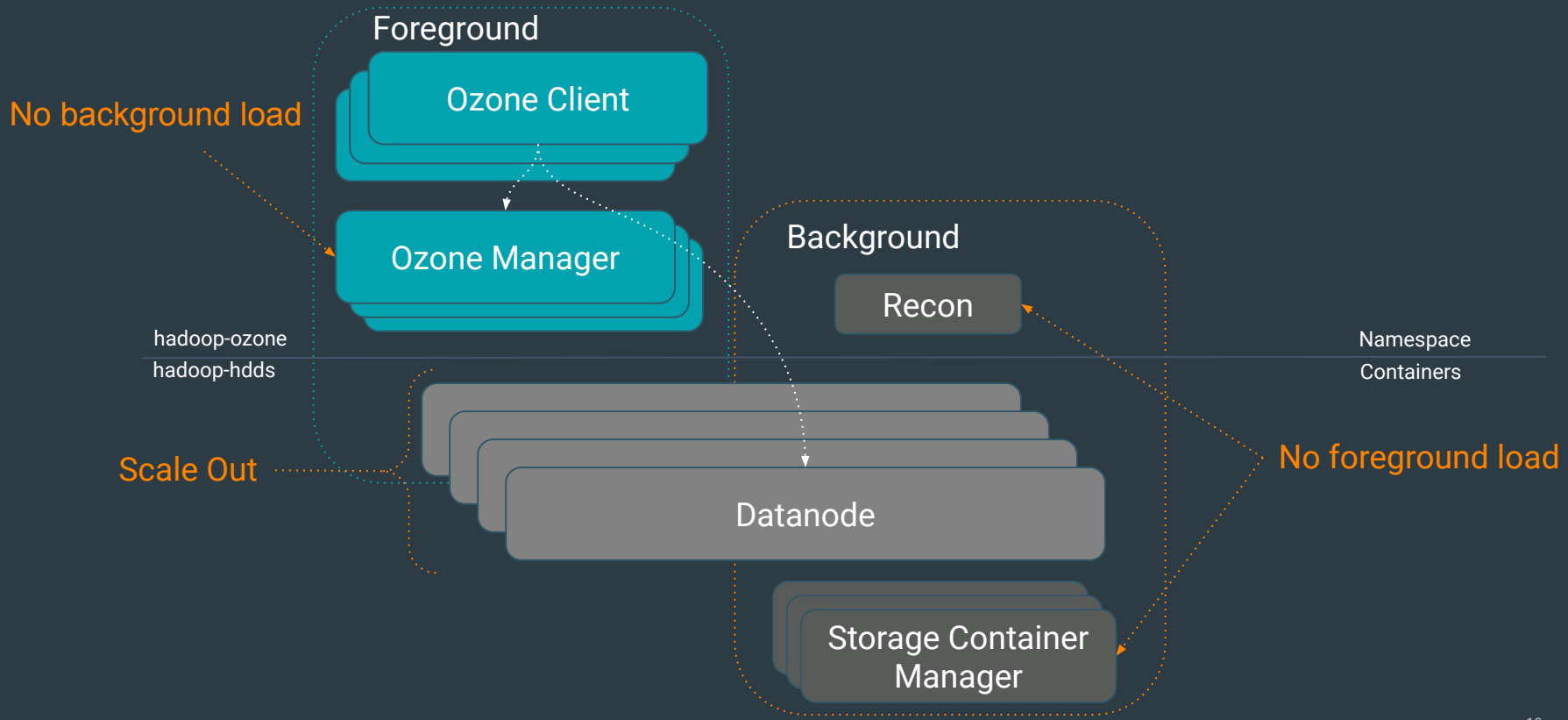
Proto for LightWeight ListKeys later

Serialization and deserialization load matters

```
message ListStatusLightResponse {
    repeated OzoneFileStatusProtoLight statuses = 1;
}
```

```
message BasicKeyInfo {
    optional string keyName = 1;
    optional uint64 dataSize = 2;
    optional uint64 creationTime = 3;
    optional uint64 modificationTime = 4;
    optional hadoop.hdds.ReplicationType type = 5;
    optional hadoop.hdds.ReplicationFactor factor = 6;
    optional hadoop.hdds.ECReplicationConfig ecReplicationConfig = 7;
    optional string eTag = 8;
    optional string ownerName = 9;
}
```

ARCHITECTURAL CONCURRENCY: SEPARATION OF CONCERNS



Separation of concerns vs. latency concerns

When foreground cares about background

- SCM knows the current status of containers and how they are mapped to Datanodes (50% impact of checking with SCM per read)
- SCM knows how to sort Datanodes based on data center topology. (Drop of 75% sorting with SCM per read)
- Solution:
  - Delegate to OM the capability to address IO path needs with exception handling that defers back to SCM.
  - Move ability of populating container to Datanode mapping for keys into OM HDDS-7223
  - Move ability of OM to sort Datanodes (move topology code support to OM). HDDS-9272
- Outcome:
  - OM can sustain 175-200K reads per second

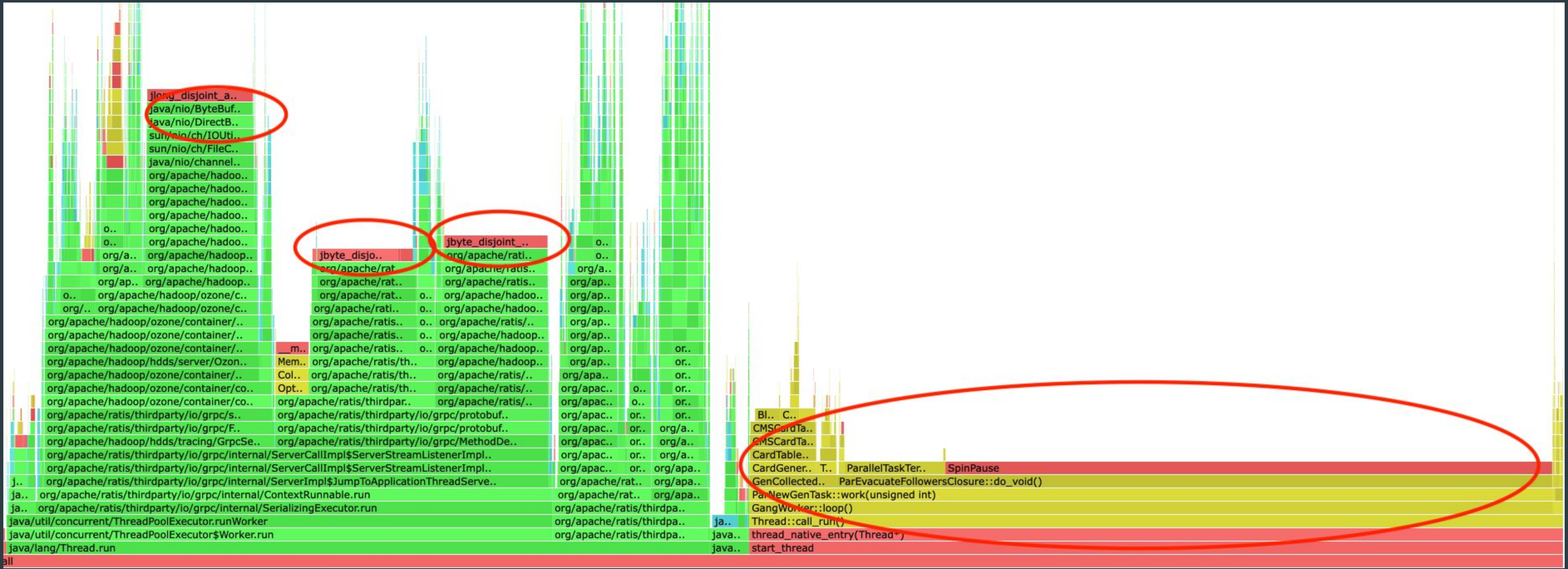
Zero Copy

Why zero copy matters for storage

- CPU burn for moving data around adds up in storage systems.
- Garbage Collection load adds up in Java
- Java NIO added zero copy abilities explicitly to make Java better for data processing systems.
- Ozone uses GRPC for reading data from Datanodes. GRPC in Java for longest time did not support zero copy buffers.

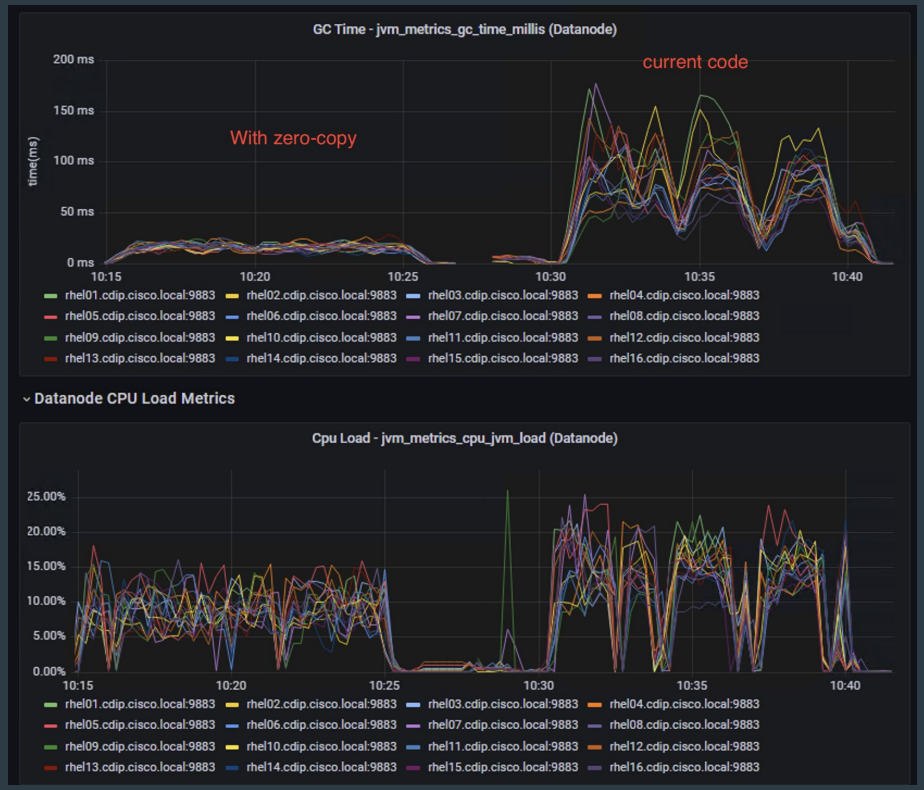
ZERO COPY

GRPC + Zero copy not a ideal match in Java



ZERO COPY

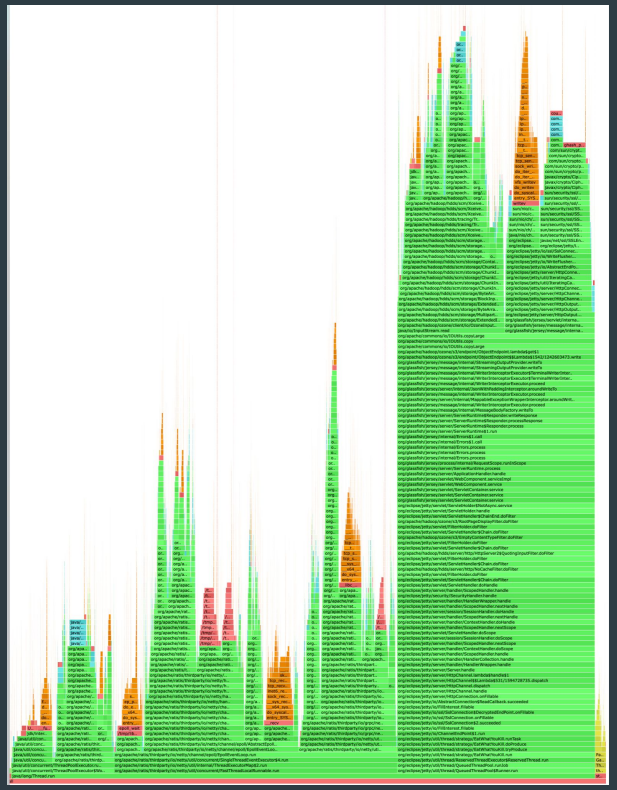
GRPC+zerocopy https://github.com/GoogleCloudPlatform/grpc-gcp-java/pull/77 => ~20% improvement + Better GC



Famegraphs vs metrics

When flamegraphs leave you in the dark

- S3 API performance down 5x for large files
- Flamegraph for S3 Gateway looks normal
- Flamegraph for Datanode looks normal
- Metrics to the rescue
  - Latency per call looks normal!
  - Control the variables, use fixed object size for predictable runtime (single round trip to Datanode expected per object read).
  - Metrics show rate of reads from Datanode 4x the expected rate based on object read rate.
- Change in how checksum boundaries lead to increased reads to Datanodes.



Tips for designing good dashboards

When flame graphs leave you in the dark

- Even Gemini and ChatGPT will give good and comprehensive advice.. With the risk of duplication here goes..
  - A dashboard should answer a question..
    - What is happening with read objects in the entire cluster?
    - The application sees X ms per object read, what is the breakdown of time spent?
    - Do the numbers add up?
      - One small read from application equates to how many RPC calls.
  - Redundancy in dashboards is given.
    - Per component dashboards vs cross cutting dashboards will have redundancy.
- Learn to use Grafana to generate good dashboards quickly.
  - LLMs are great at generating promQL or advice on how to use Grafana

Order of tooling

When the phone rings..

- Which operation is slow?
- Use or generate a dashboard for the relevant metrics (know your metrics!)
- Overlay the hardware usage metrics on top of the RPC metrics
  - Even utilization across drives, nodes, racks..
- If possible use load generation tools to pinpoint if this a hardware issue, scale issue or something new
  - RPC echo tools are amazing
    - Noop request and response that can progressively probe performance for each layer in the service. Ex: Round trip for rpc, followed by round trip including a consensus round for a replicated service, followed by request and response with varying payloads.
- Logs are useful but slow to debug.
- If nothing else works, bring out the flamegraph generators.

Organic premature optimization

Challenges of scaling open source projects

- Ozone to scale to cluster sizes that are hard to get hands on while developing
- Good engineering => think of ways to speed code up!
  - Reality: Assumptions that did not span out
  - Outcome: Code complexity and unexpected challenges to scale and performance.
- Incremental but significant changes on a continuous basis to max out performance based on new data that we encounter.

Q&A

Thanks for Attending!