Scaling HDFS with Consistent Reads from Standby Replicas



Konstantin V Shvachko
Sr. Staff Software Engineer
LinkedIn

February 24, 2020
Santa Clara, CA

VAULT '20

Agenda

- Background & Motivation. Design document
- Standard HDFS Architecture
- Consistent Reads Challenge: HDFS-12943
  - Stale read problem
  - Consistency model
  - Design and Implementation
- Performance and scalability
- Community support

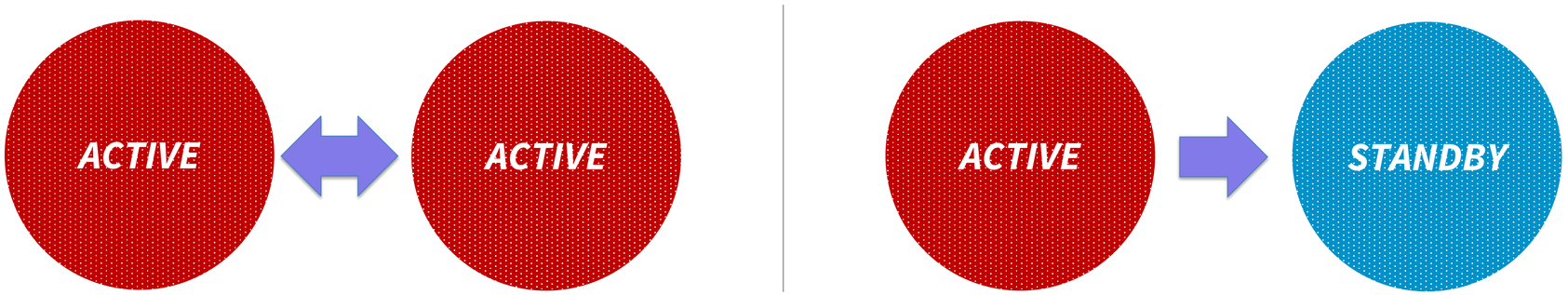
Background

High Availability

ACTIVE-PASSIVE ARCHITECTURE

- Active-active vs active-passive architecture tradeoffs
- ***High Availability* in distributed systems, databases**
  - The *active* (master, primary) service (server, node) serves client requests
  - *Passive* (slave, standby, backup) service(s) receive updates from the active
  - *Journal Tailing* to keep passive up to date with the active
  - *Failover* procedure transitions active role to a standby

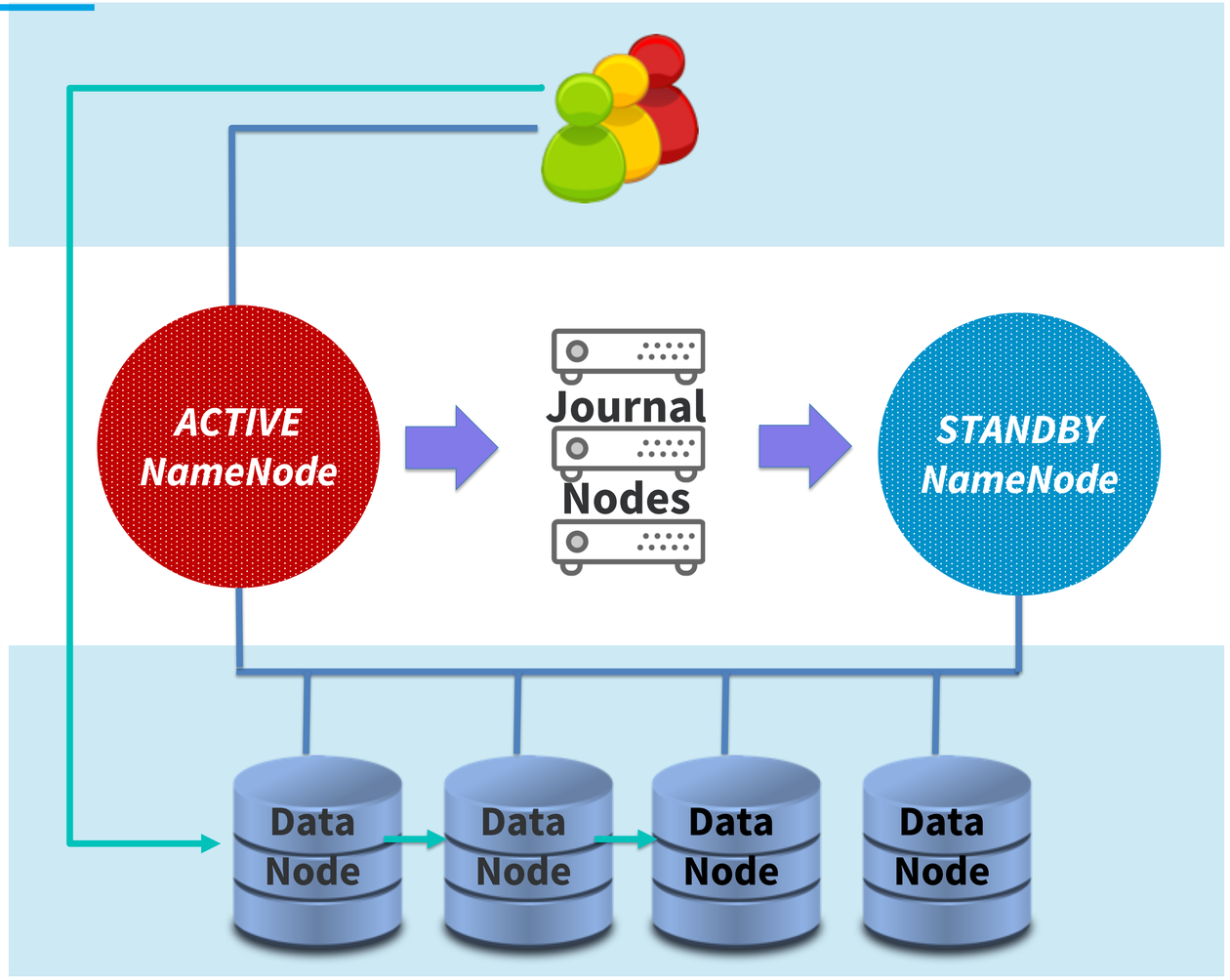
- ***Can standby be used to serve read requests?***



Hadoop Distributed File System

STANDARD HDFS ARCHITECTURE

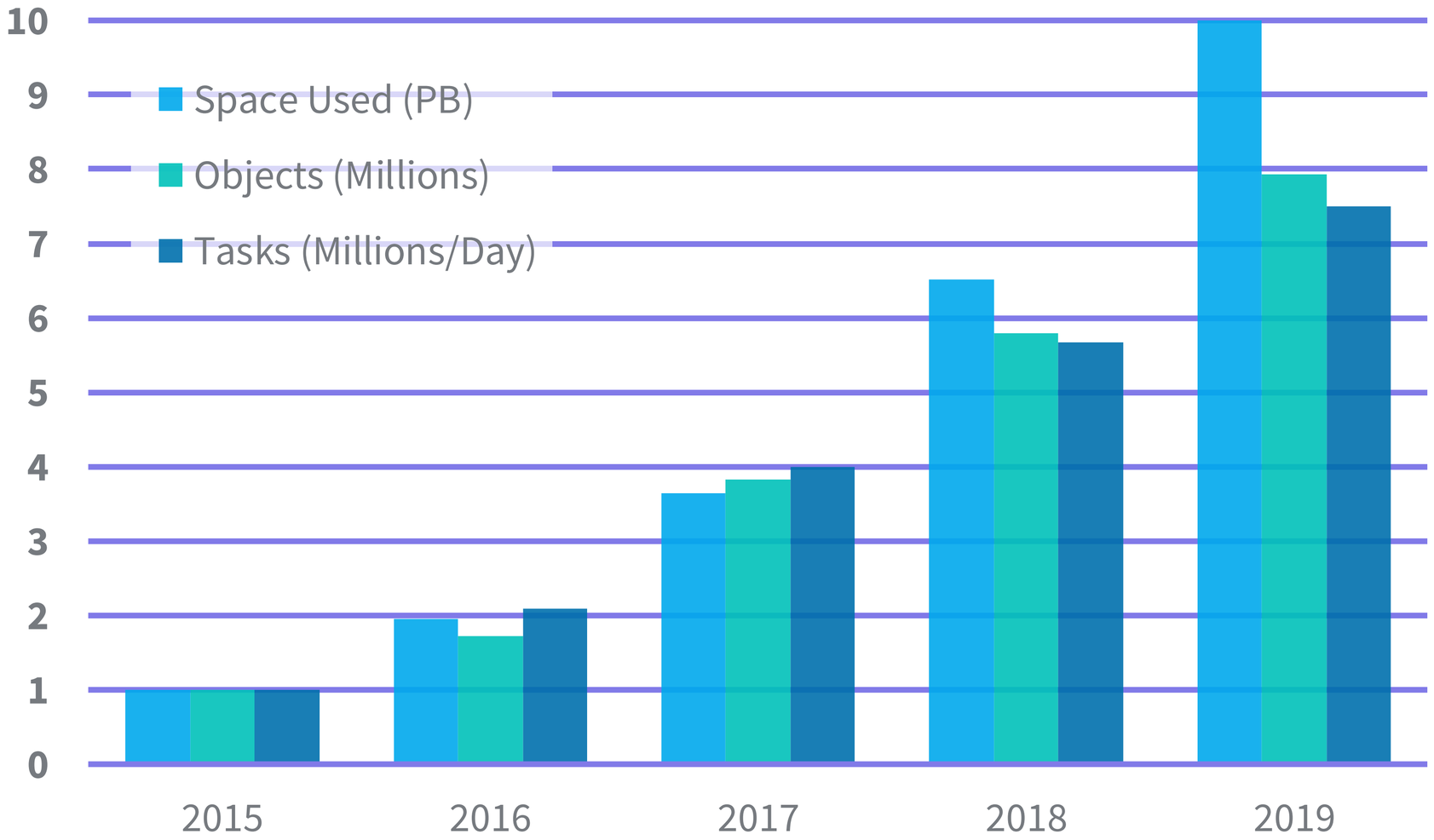
- HDFS metadata is decoupled from data
- **NameNode** keeps the directory tree in RAM
- **Persistent state**: checkpoint + journal log
- Thousands of **DataNodes** store data blocks
- **HDFS clients** request metadata from active NameNode and stream data to/from DataNodes



Motivation

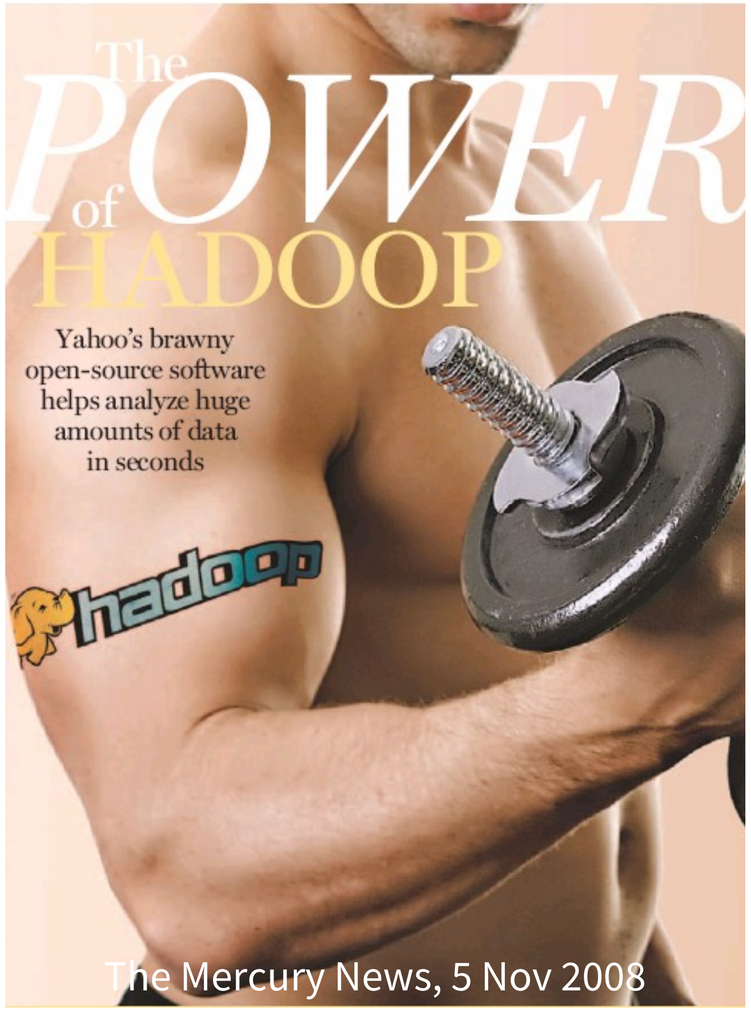
Cluster Growth 2015 - 2019

THE INFRASTRUCTURE IS UNDER CONSTANT GROWTH PRESSURE



POP QUIZ

What is the Scalability Limit of Hadoop clusters?



The Mercury News, 5 Nov 2008

Number of Objects = files + blocks

- Unlimited
- 300 million
- 700 million
- 1 billion

HDFS Infrastructure at LinkedIn

STORAGE

- **All Hadoop Clusters**
  - Number of files/blocks: 2.0 billion
  - Data storage: capacity 650PB, used 500PB
  - Compute: 600K cores with 2.0PB memory

- **Largest cluster**
  - 300PB capacity, 250PB (85%) used
  - 800M objects (files + blocks)
  - *70K metadata QPS on average*

Motivation

SCALE METADATA OPERATIONS

- **Exp Growth/Year In Workloads and Size**
  - Approaching active Name Node performance limits rapidly
  - Need a scalability solution for metadata operations

- **Key Insights:**
  - Reads comprise 95% of all metadata operations in our production environment
  - Another source of truth for read: Standby Nodes

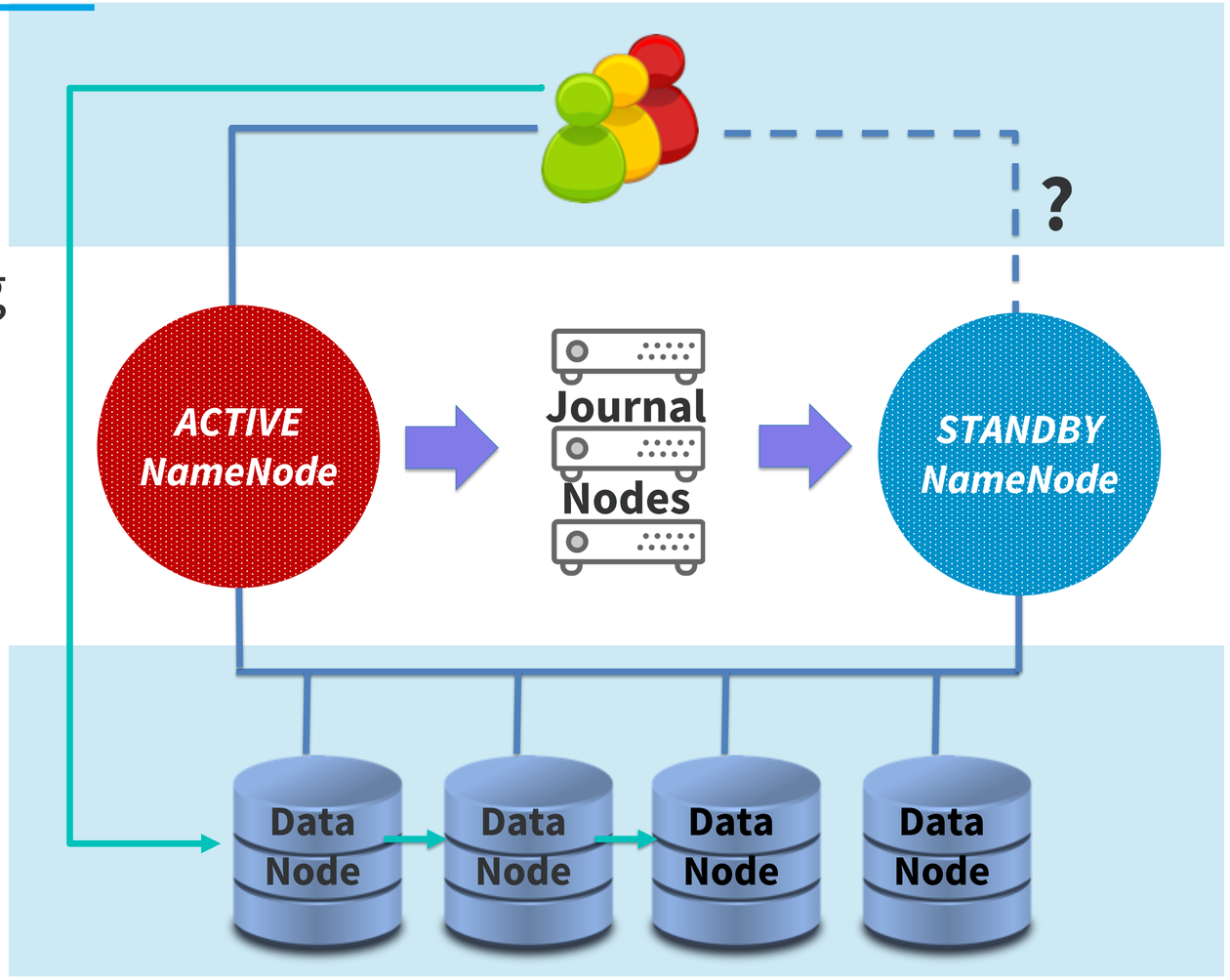
- **Standby Nodes Serving Read Requests**
  - Can substantially decrease active Name Node workload
  - **Allowing cluster to scale further!**

Consistent Reads From Standby Node

HDFS High Availability

ROLE OF STANDBY NODES

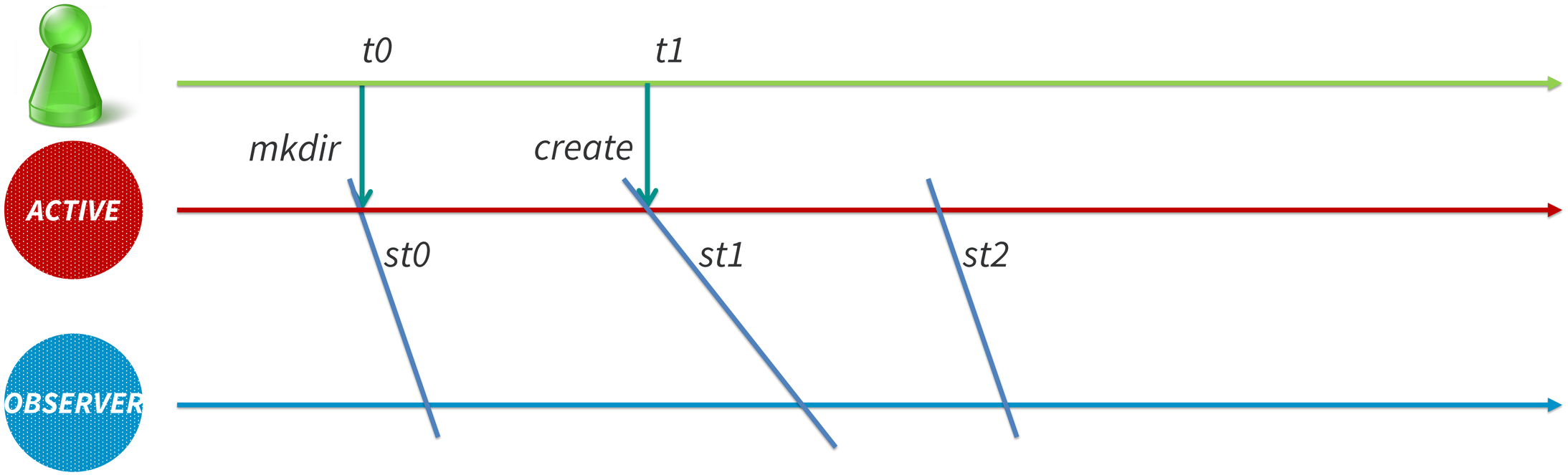
- Hot Standby nodes have the same copy of all metadata (with some delay)
  - Active NameNode publishes journal log (edits) to JournalNodes
  - StandbyNode syncs its state by tailing journal log from JournalNodes
- ***ObserverNode*** is a StandbyNode that serves read requests
  - All reads can go to ObserverNodes
  - Time critical applications can still choose to read from Active only



Consistency Model

FILE SYSTEMS REQUIREMENTS FOR STRONG CONSISTENCY

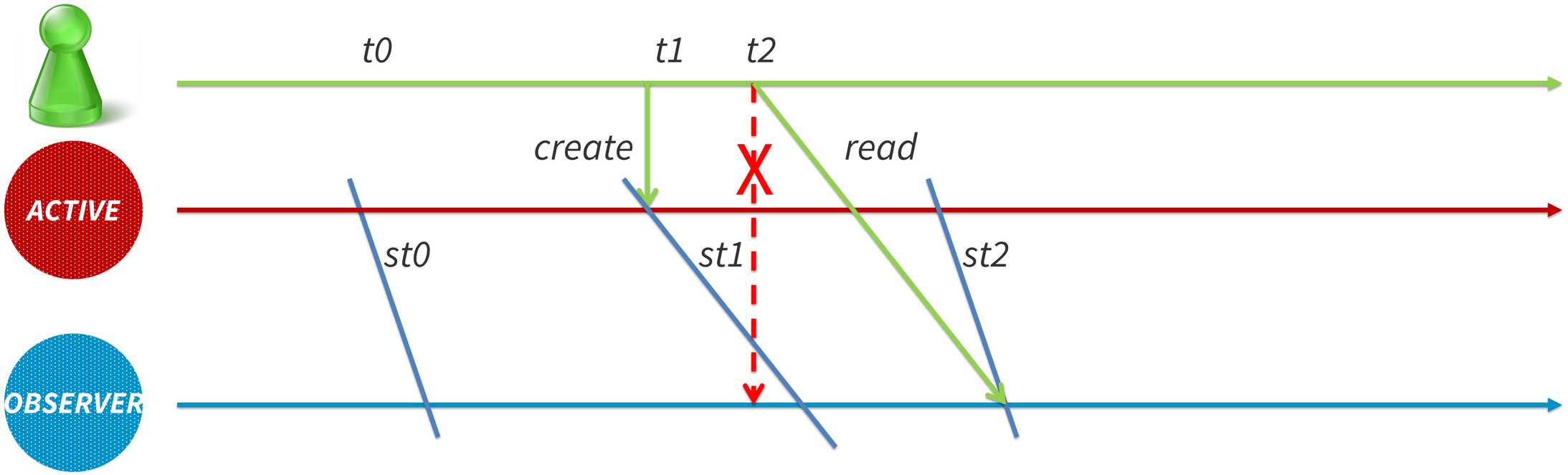
- **Namespace states id**
  - Monotonically increasing number with each modification
  - Corresponds to Active NameNode journal log transaction Id



Consistency Model

FILE SYSTEMS REQUIREMENTS FOR STRONG CONSISTENCY

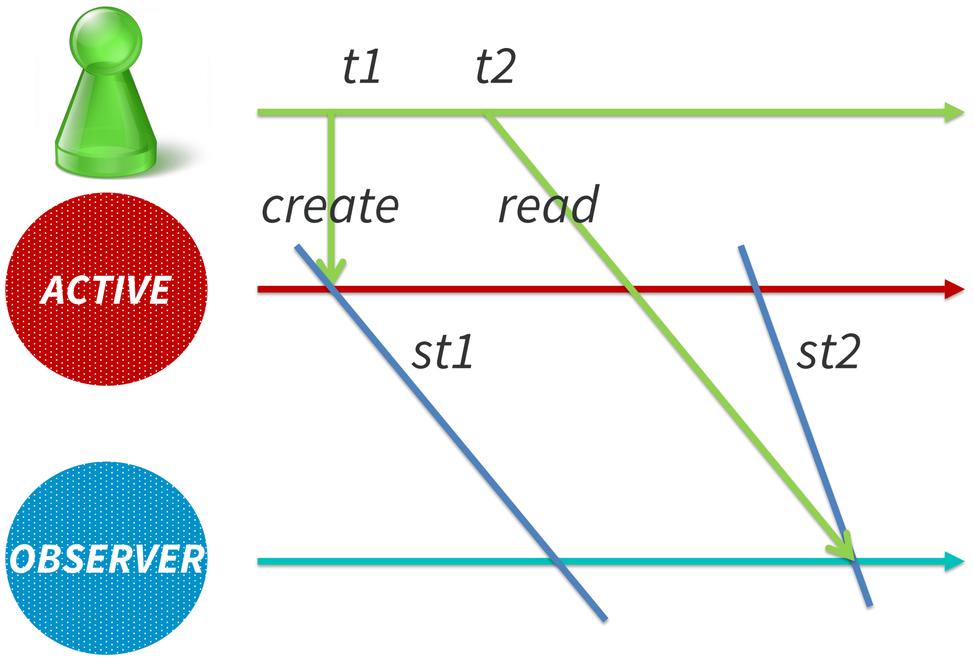
- **Consistency Principle:** ***clients should never see the past***
  - *If client **C** sees or modifies an object state at st1 at time t1, then in any future time t2 > t1, **C** will see the state of that object at st2 >= st1*



Type I

RYOW

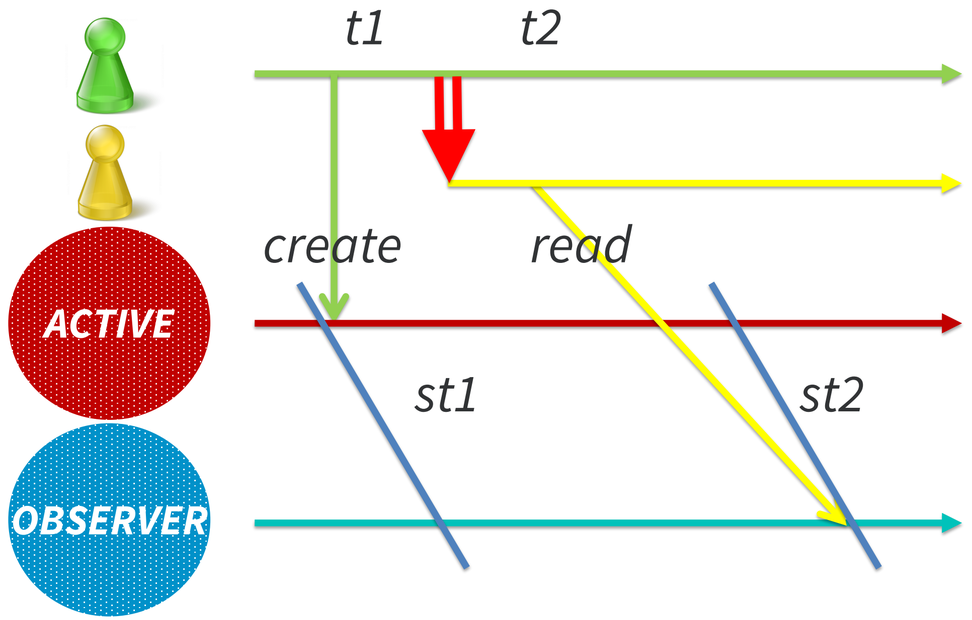
- **Read your own writes (RYOW)**
  - If a single HDFS client modifies the namespace on Active and switches over to Observer it should be able to see the same or a later state of the namespace, but not an earlier state.



Type II

3PC

- **Third-party communication (3PC)**
  - If one client modifies the namespace and passes that knowledge to other clients, the latter ones should be able to read from Observer the same or a later state of the namespace, but not an earlier state.



Examples

INCONSISTENT OPERATIONS

I. Client ***creates*** a file via Active NameNdoe but cannot read it himself from Observer

II. Client opens a file via Observer, which is concurrently ***deleted*** or modified on Active
   a. Deleted or modified by the same client (RYOW)
   b. Deleted or modified by somebody else (no problem)

III. MapReduce ***job submission*** consists of certain step (3PC)
   a. Job client creates job configuration and job jar files on HDFS
   b. Notifies ResourceManager to schedule the job
   c. Each job task reads job configuration and jar files during startup
      If the files are still not available on Standby due to the delay the job may fail

Implementation Details

Read Your Own Writes

THE SOLUTION

- **LastSeenStateID**
  - Monotonically increasing Id of Active namespace state id
  - Kept on client side, client's known most recent Active state
  - Sent to ObserverNode, which only replies after it has caught up to this state
  - Seamless. Passed in RPC header

- **Problem:** Long delay in journal tailing by StandbyNode
  - Roll edits file every 2 min by default
  - JournalNode serves finalized segments of edits from disk

Fast Journaling

MINIMIZE THE DELAY

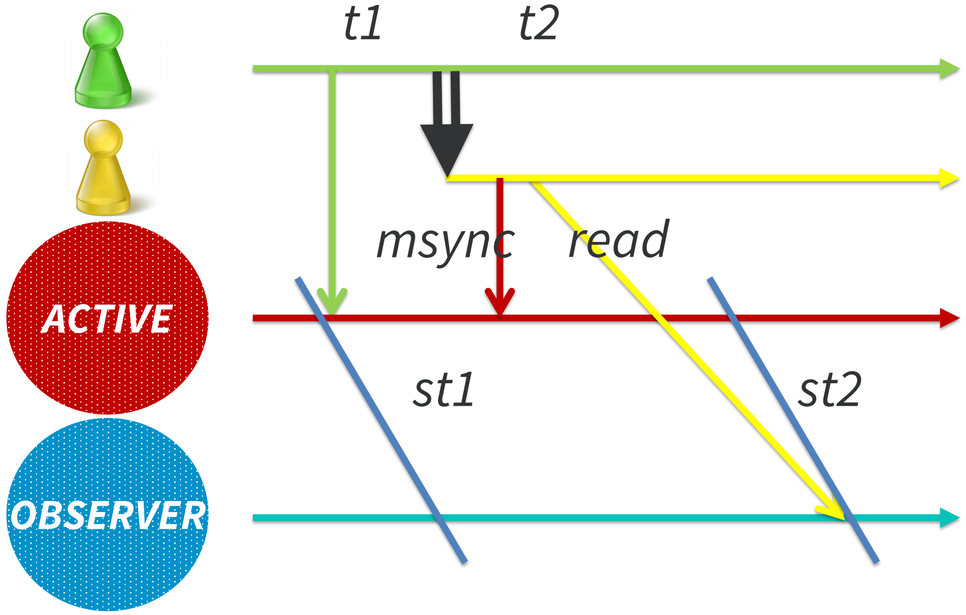
- **Fast Edit Tailing** HDFS-13150
  - Optimization on JournalNode and Standby/Observer nodes
  - JournalNodes cache recent edits in memory, only applied edits are served
  - StandbyNode requests only recent edits through RPC calls
  - Fall back to existing mechanism on error

- **ObserverNode delay is no more than a few msec in most cases**
  - Client-perceived lag of ObserverNode is < 6 msec on average
  - The average transaction processing time is 25-30 msec
    From the moment it is submitted on Active until it is applied on Observer

msync API

FOR THIRD PARTY COMMUNICATION

- **Dealing with Stale Reads: FileSystem.msync()**
- Sync between existing client instances
- Force the HDFS Client to sync up to the most recent state of Acfive NameNode
  - Client #2 calls msync first
  - msync updates *lastSeenStateId* from Active
  - Then the read from Observer is consistent
- **Problem:** Change application logic to add msync



Auto-msync

INTENDED TO AVOID CHANGING APPLICATIONS

- No-cost automatic client ***msync on startup***

- ***Periodic msync*** mode HDFS-14211
  - Configuration parameter: `auto-msync-period = 1 sec`
- ***Always-msync***: `auto-msync-period = 0 sec`

- ***Always-Active*** option: never talk to Observer
  - Use standard pluggable `FailoverProxiProvider`
  - `ObserverReadProxiProvider` implements Observer reads logic

- ***Long-running read-only clients***

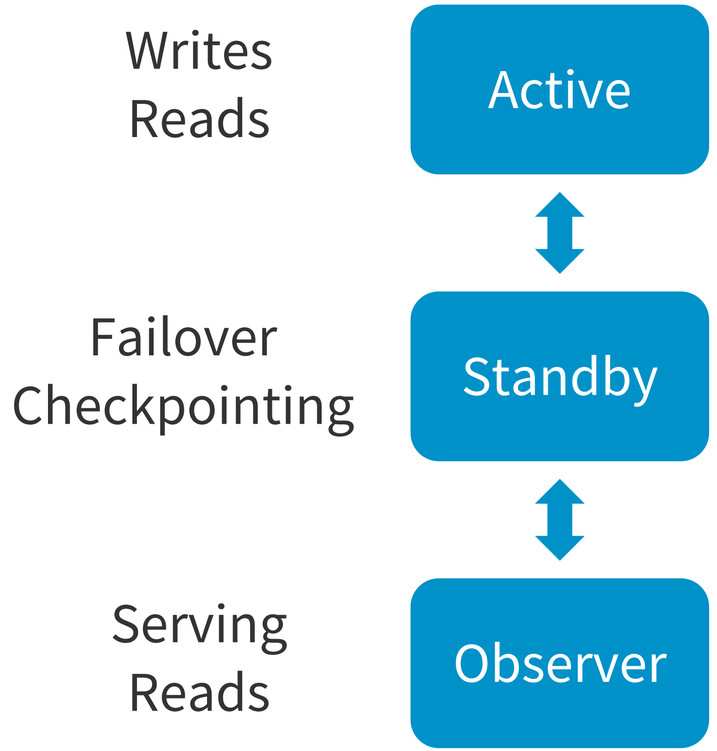
Observer Node Back-off

REDIRECT TO ACTIVE WHEN TOO FAR BEHIND

- ObserverNode state id is substantially smaller than client's lastSeenStateId HDFS-13873
  - Node is restarting, slow network or local disk IO
- Slow Observer rejects the request by throwing an exception
- Client may retry on another Observer or on Active NameNode

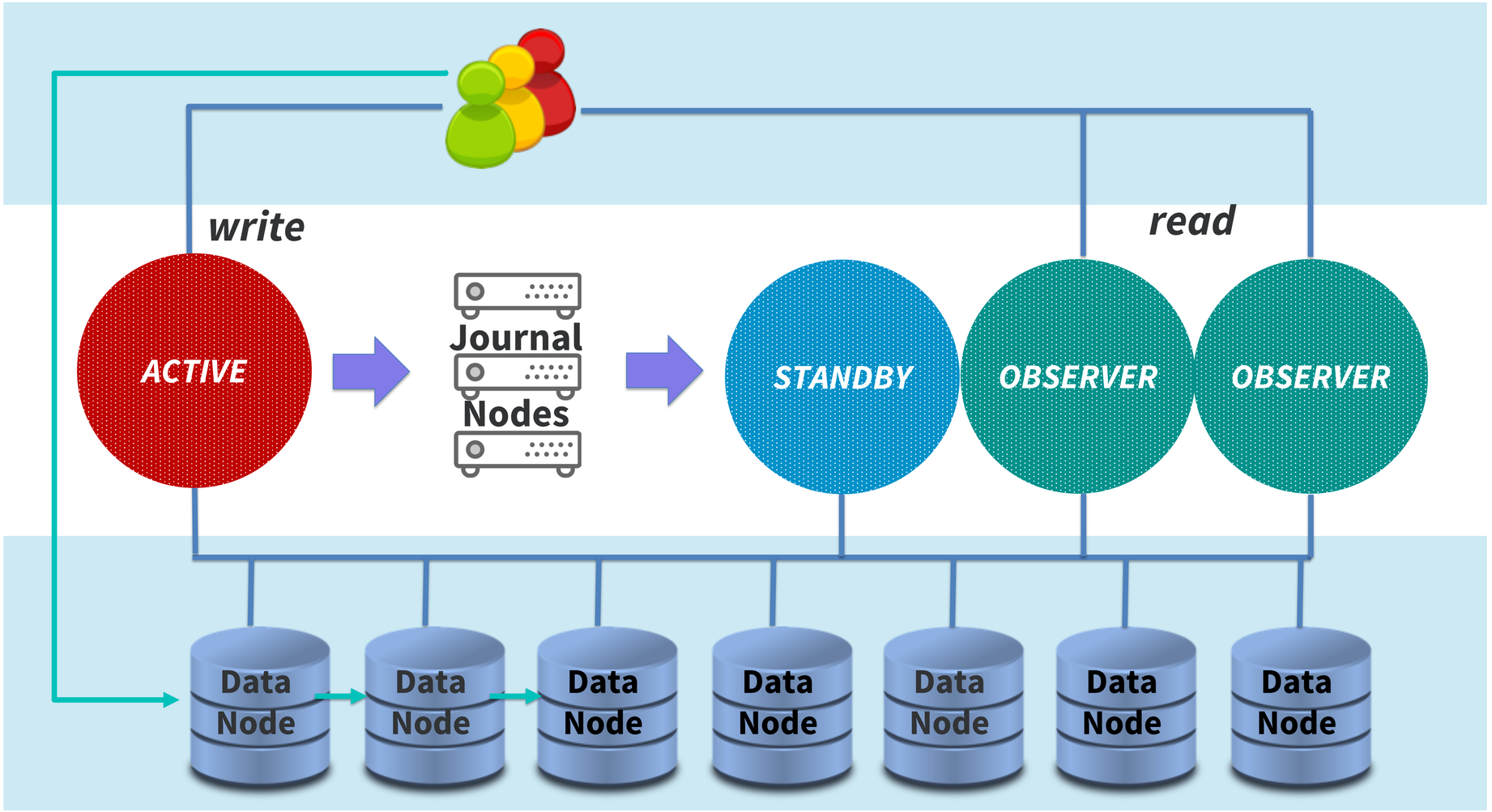
Configuration and Startup Process

- **Configuring NameNodes**
  - Use HA configuration for NameNodes
    - `hdfs-site.xml` lists all NameNodes as equal
  - All NameNodes start as Standby
  - `haadmin` command transitions Standby to Active or Observer
- **Configuring Clients**
  - Configure to use ObserverReadProxyProvider
    - If not, client still works but only talks to Active
  - ObserverReadProxy discovers states of all NameNodes



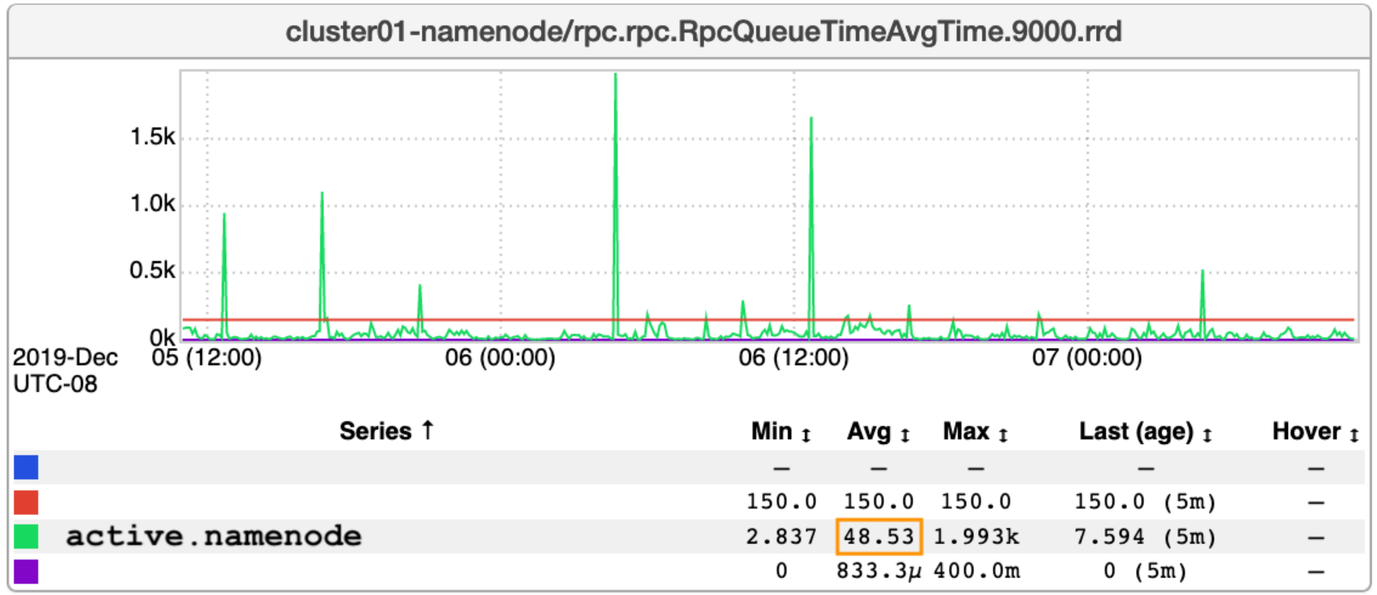
HDFS Cluster with Observers

ACTIVE / STANDBY PAIR FOR FAILOVER AND CHECKPOINTS + REDUNDANT OBSERVERS

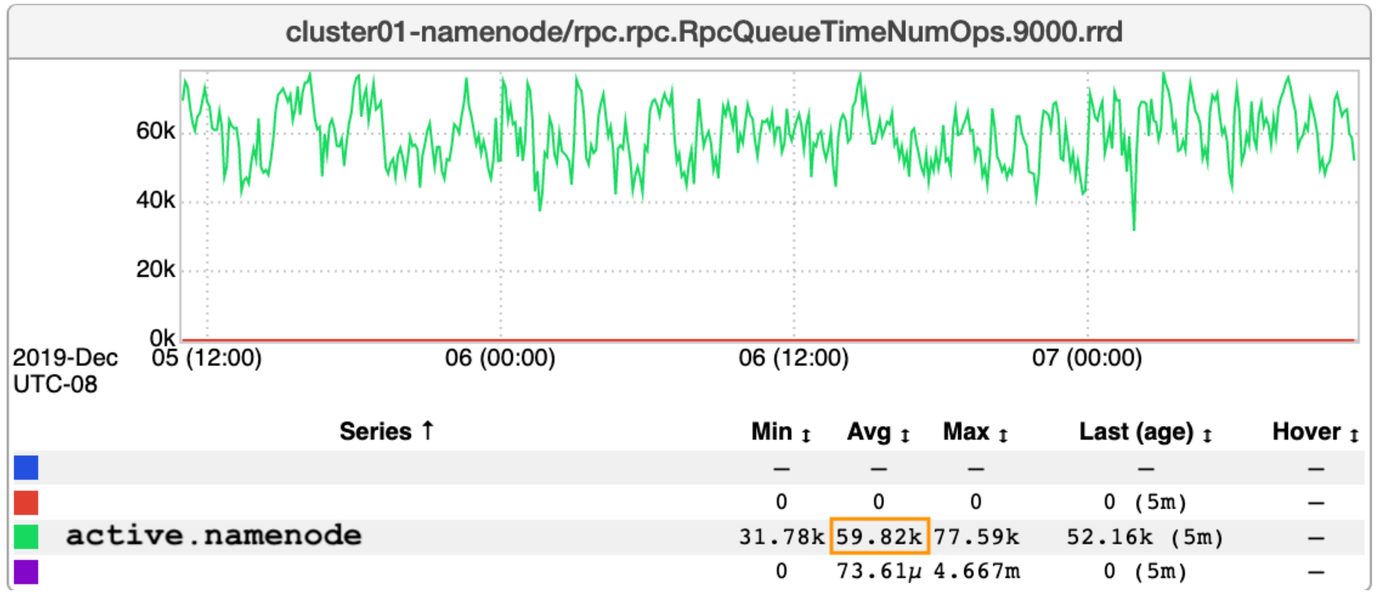


Metrics & Performance

RPC Queue Time: Latency & Throughput



Series ↑	Min ↕	Avg ↕	Max ↕	Last (age) ↕	Hover ↕
	–	–	–	–	–
	150.0	150.0	150.0	150.0 (5m)	–
active.namenode	2.837	48.53	1.993k	7.594 (5m)	–
	0	833.3μ	400.0m	0 (5m)	–



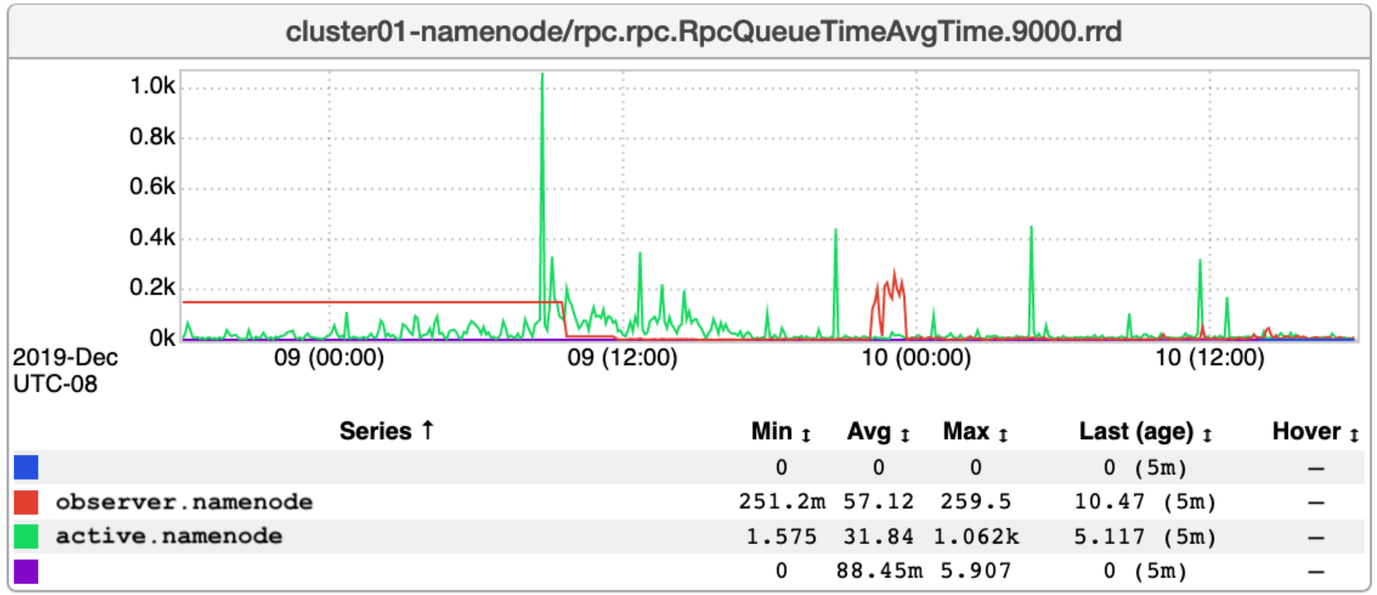
Series ↑	Min ↕	Avg ↕	Max ↕	Last (age) ↕	Hover ↕
	–	–	–	–	–
	0	0	0	0 (5m)	–
active.namenode	31.78k	59.82k	77.59k	52.16k (5m)	–
	0	73.61μ	4.667m	0 (5m)	–

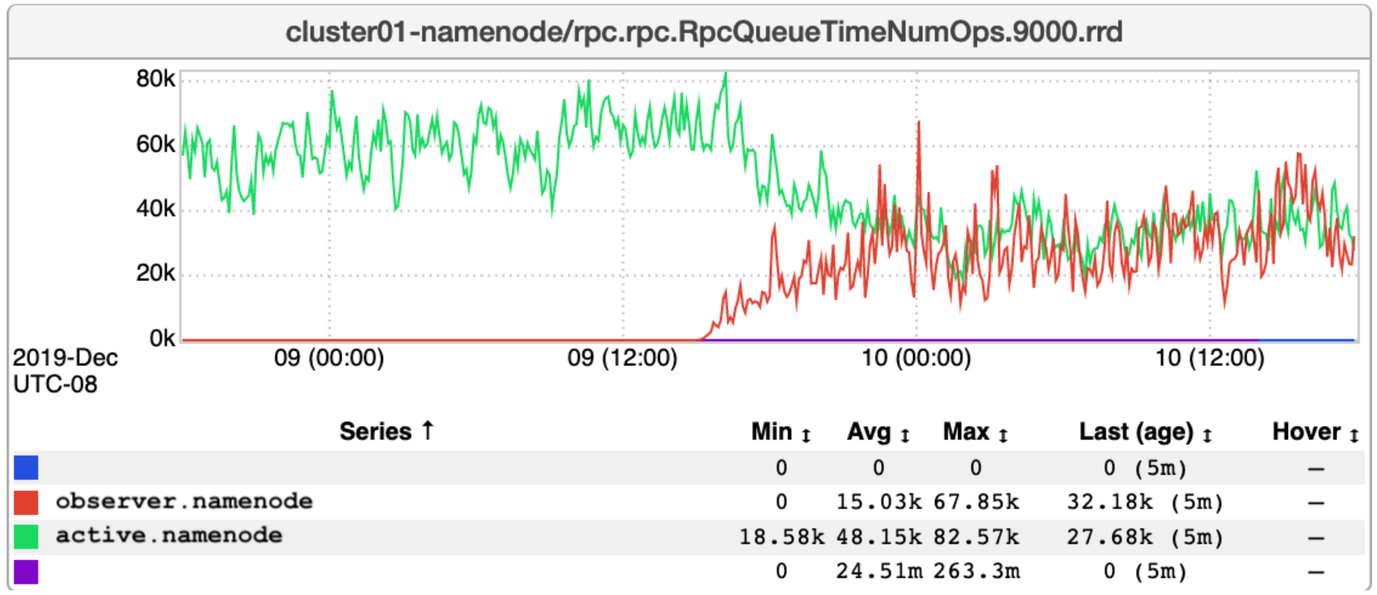
BEFORE

- ***RPCQueueTimeAvgTime***
  average RPC processing time
- *50 msec*

- ***RPCQueueTimeNumOps***
  number of RPC operations per second QPS
- *60K*

Rolling Upgrade: HDFS Clients



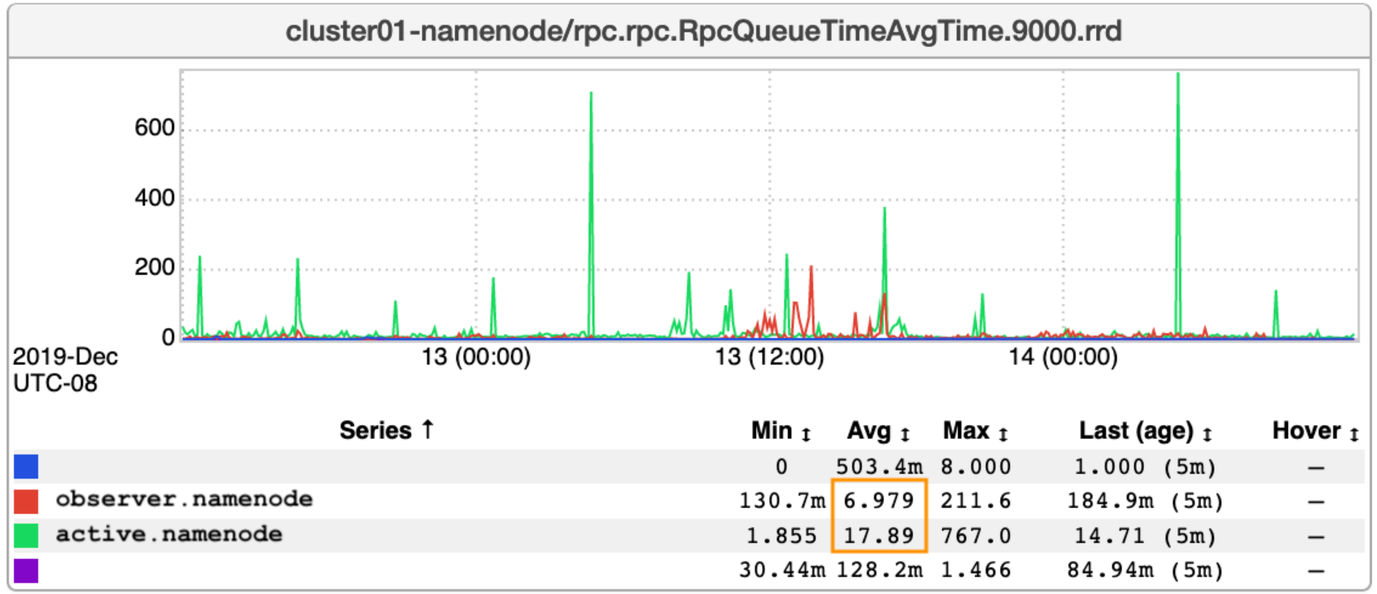


DURING

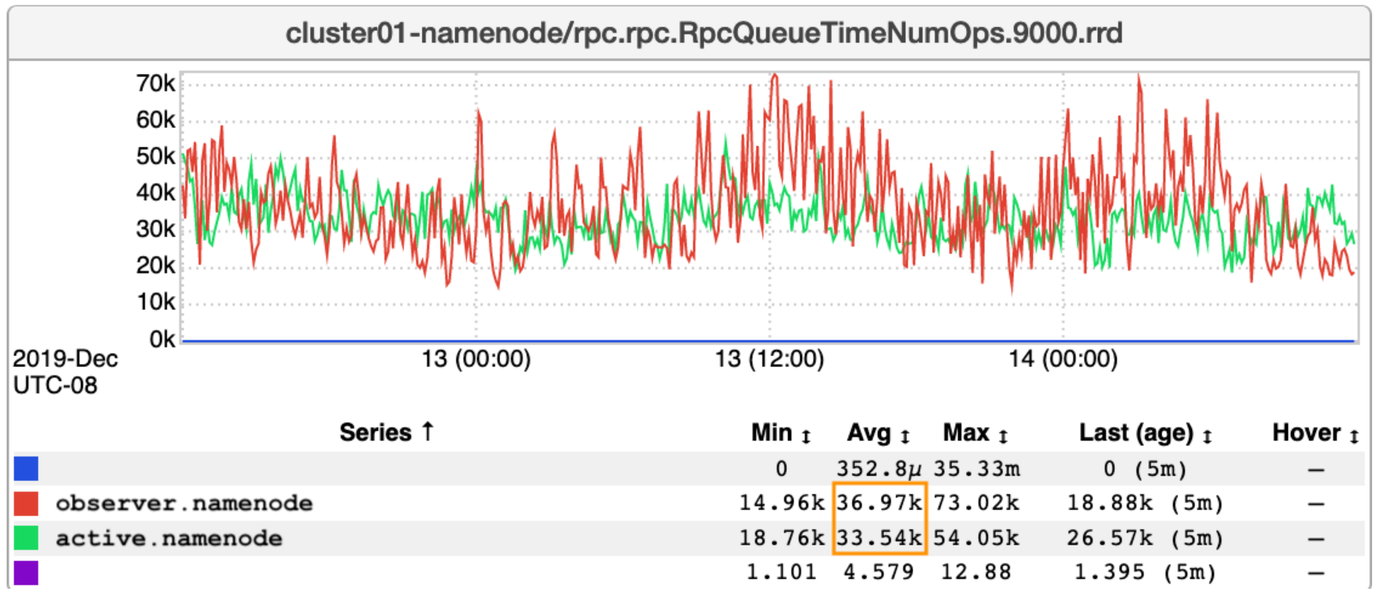
- ***RPCQueueTimeAvgTime***
  average RPC processing time

- ***RPCQueueTimeNumOps***
  number of RPC operations per second QPS

Post Upgrade: 50% Clients



Series	Min	Avg	Max	Last (age)	Hover
	0	503.4m	8.000	1.000 (5m)	–
observer.namenode	130.7m	6.979	211.6	184.9m (5m)	–
active.namenode	1.855	17.89	767.0	14.71 (5m)	–
	30.44m	128.2m	1.466	84.94m (5m)	–



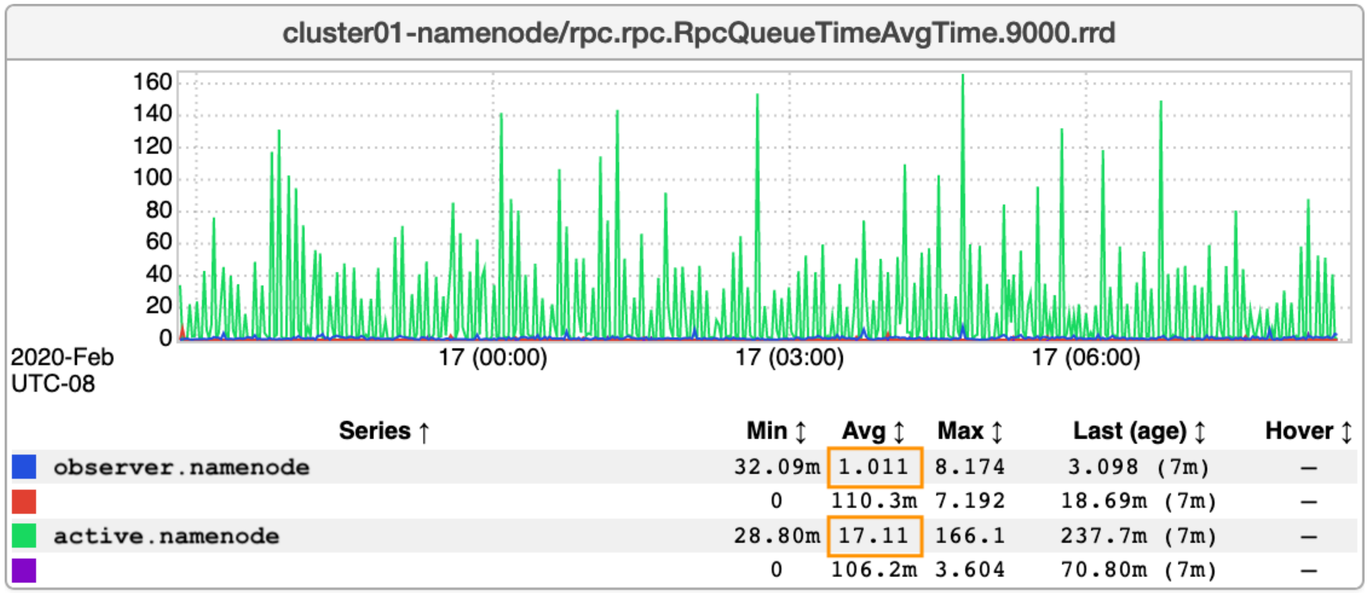
Series	Min	Avg	Max	Last (age)	Hover
	0	352.8μ	35.33m	0 (5m)	–
observer.namenode	14.96k	36.97k	73.02k	18.88k (5m)	–
active.namenode	18.76k	33.54k	54.05k	26.57k (5m)	–
	1.101	4.579	12.88	1.395 (5m)	–

AFTER

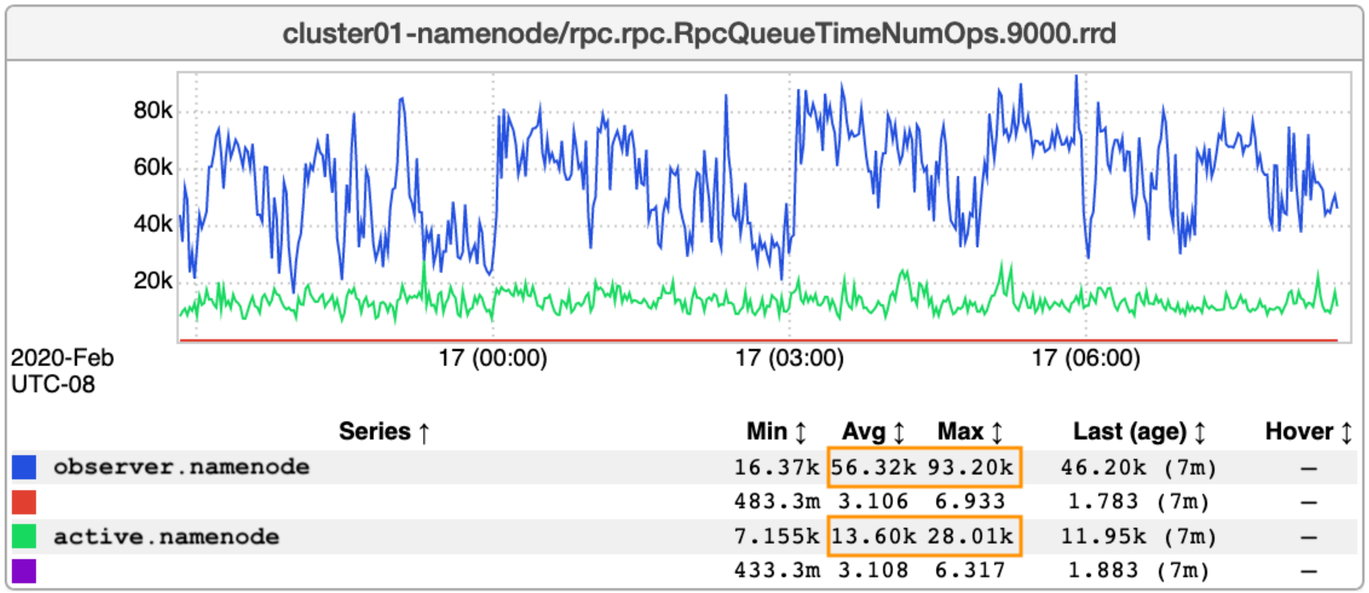
- ***RPCQueueTimeAvgTime***
  average RPC processing time
- *read 6 msec / write 17 msec*

- ***RPCQueueTimeNumOps***
  number of RPC operations per second QPS
- *read 37K / write 33K*

Post Upgrade: 100% Clients



Series ↑	Min ↕	Avg ↕	Max ↕	Last (age) ↕	Hover ↕
observer.namenode	32.09m	1.011	8.174	3.098 (7m)	–
	0	110.3m	7.192	18.69m (7m)	–
active.namenode	28.80m	17.11	166.1	237.7m (7m)	–
	0	106.2m	3.604	70.80m (7m)	–



Series ↑	Min ↕	Avg ↕	Max ↕	Last (age) ↕	Hover ↕
observer.namenode	16.37k	56.32k	93.20k	46.20k (7m)	–
	483.3m	3.106	6.933	1.783 (7m)	–
active.namenode	7.155k	13.60k	28.01k	11.95k (7m)	–
	433.3m	3.108	6.317	1.883 (7m)	–

FINAL

- ***RPCQueueTimeAvgTime***
  average RPC processing time
- *read 1 ms / write 17 ms*

- ***RPCQueueTimeNumOps***
  number of RPC operations per second QPS
- *read 56K / write 14K*

RPC Performance Comparison

SINGLE ACTIVE VS ACTIVE + OBSERVER

	Before	After
Read Time Avg (msec)	50	1
Write Time Avg (msec)	50	17
Read QPS Avg / Max		55K / 100K
Write QPS Avg / Max		15K / 30K
Total QPS Avg / Max	60K / 80K	70K / 130K

Community

Community Support

APACHE HADOOP PROJECT

- The core team:
  - Erik Krogen (LinkedIn)
  - Chen Liang (LinkedIn)
  - Chao Sun (Uber)
  - Plamen Jeliazkov (Paypal)

- Joint project of LinkedIn + Uber + Paypal
  - Contributions from Cloudera – backports to Hadoop 3.0
  - Chinese Internet companies

Thank You!

Scaling HDFS with Consistent Reads
from Standby Replicas



Konstantin V Shvachko

LinkedIn