Optimistic Concurrency Control in a Distributed NameNode Architecture for Hadoop Distributed File System

Qi Qi

Instituto Superior Técnico - IST (Portugal)
Royal Institute of Technology - KTH (Sweden)
Swedish Institute of Computer Science - SICS (Sweden)

qiq@kth.se

19 September 2014

Motivation

Industrial Standard in Big Data Era

Apache Hadoop Ecosystem

Limits to growth in HDFS

Number of Files	**Memory Requirement**	**Physical Storage**
1 million	0.6 GB	0.6 PB
100 million	**60 GB**	**60 PB**
1 billion	600 GB	600 PB

Hops-HDFS and Its Limitation

Distributed NameNode Architecture

Maintain HDFS Strong Consistency Semantics

Concurrency Restricted

Problem Statement

HDFS

System-level Lock: Single Writer

MySQL Cluster

Read Committed Isolation Level: Anomalies

Hops-HDFS v1

System-level Lock: Single Writer + Network Latency

Hops-HDFS v2 (Pessimistic Concurrency Control - PCC)

Row-level Lock: Implicit Locking ->Single Writer + Network Latency

Contribution

Architecture and Namespace Concurrency Control Accessment

GFS, HDFS, Hops-HDFS

Performance Accessment and Limitation Analysis

HDFS v.s. Hops-HDFS v2 (PCC version)

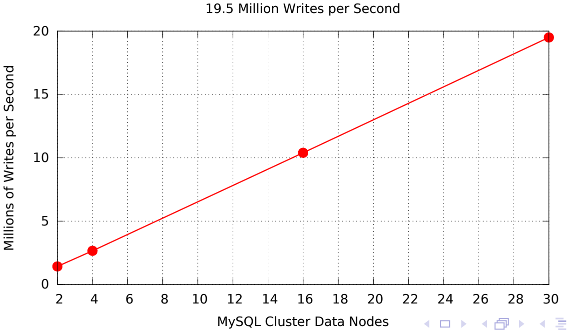
Solution for Hops-HDFS

- Optimistic Concurrency Control with Snapshot Isolation on Semantic Related Group
- Performance Increase Up to 70 %
- Correctness ensured by Passing 300+ Apache HDFS Unit Tests: maintain HDFS semantics

MySQL Cluster

Distributed, In-memory, Replicated Database

- Scalable
- Fault-tolerance
- High throughput
- **BUT**: Supports only **Read Committed** Isolation Level



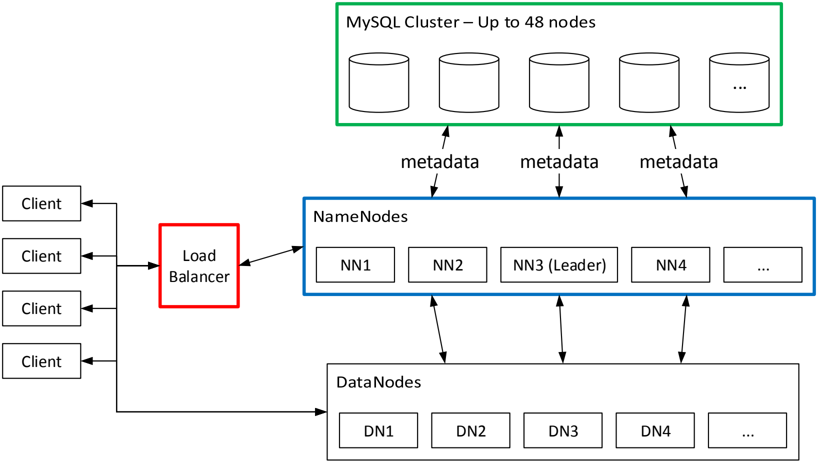
Hops [1]-HDFS

Overcome Limitations in HDFS NameNode

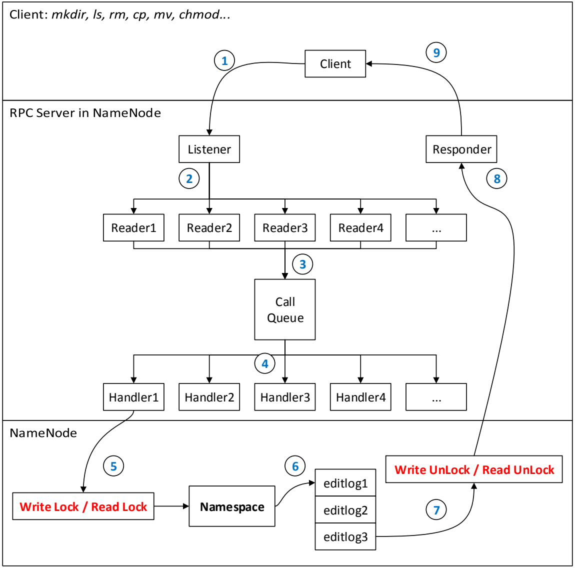
- Scalability of the Namespace
- Throughput Problem
- Failure Recovery

[1] Hadoop Open Platform-as-a-Service

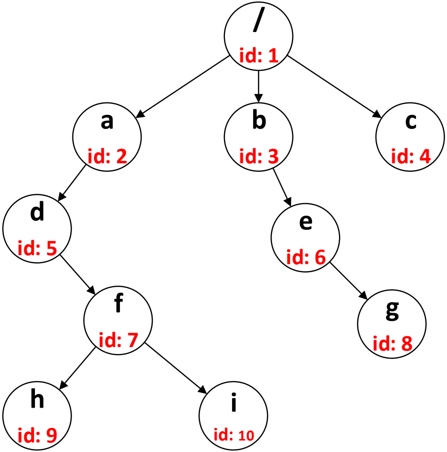
Hops-HDFS Architecture



HDFS Namespace Concurrency Control Assessment



Hops-HDFS Namespace Structure



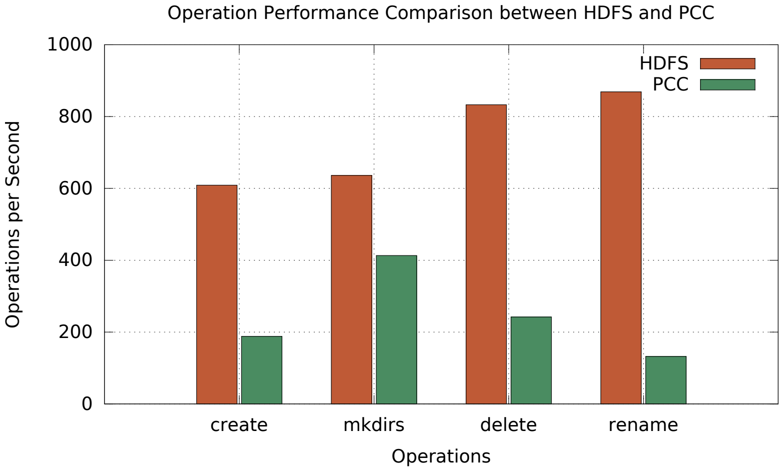
id	parent_id	name
1	0	/
2	1	a
3	1	b
4	1	c
5	2	d
6	3	e
7	5	f
8	6	g
9	7	h
10	7	i

Limitations in Hops-HDFS Namespace Concurrency Control (PCC)

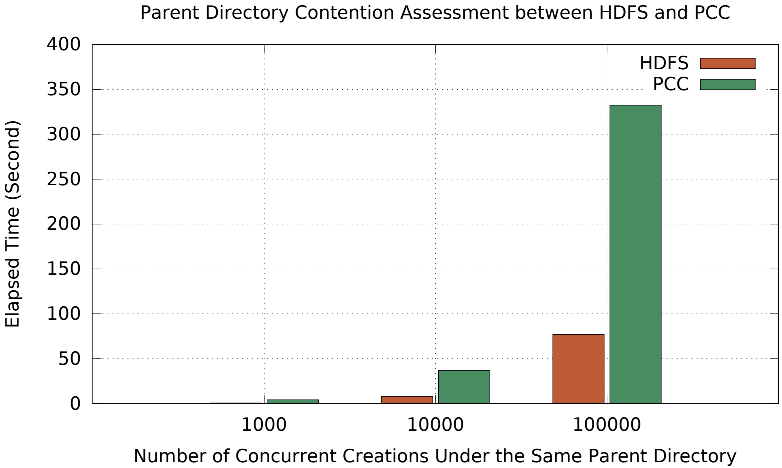
- Duplicated Round Trips
- Implicit Parent Locks:

id	parent_id	name	Locks by Tx1	Locks by Tx2
1	0	/	R	R
2	1	a	R	R
3	1	b		
4	1	c		
5	2	d	R	R
6	3	e		
7	5	f	W	W **(Block)**
8	6	g		
9	7	h (Tx1)	W (Implicit)	W (Implicit) **(Block)**
10	7	i (Tx2)	W (Implicit)	W (Implicit) **(Block)**

NameNode Throughput Benchmark - HDFS v.s. PCC



Parent Directory Contention Assessment - HDFS v.s. PCC



Resolving the Semantic Related Group

Path: /a/d/f/h

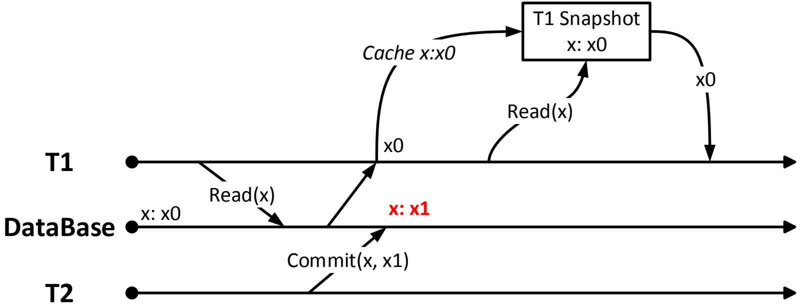
h: {/->a->d->f}

	id	parent_id	name	other parameters...
Related *	1	0	/	...
Related *	2	1	a	...
	3	1	b	...
	4	1	c	...
Related *	5	2	d	...
	6	3	e	...
Related *	7	5	f	...
	8	6	g	...
Selected ✓	9	7	h	...
	10	7	i	...

Per-Transaction Snapshot Isolation

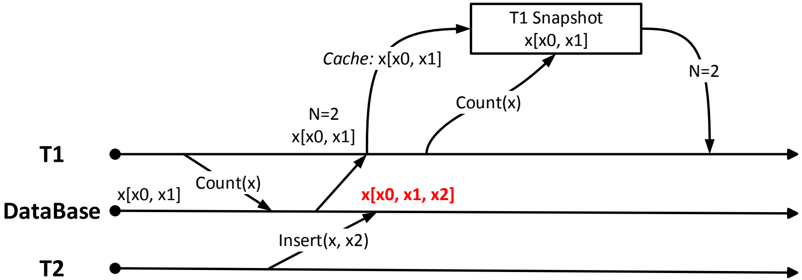
- Snapshot the whole Semantic Related Group
- Transaction performs on its own snapshot
- Preclude: **Fuzzy Read** & **Phantom Read**

Snapshot Isolation Precludes **Fuzzy Read** [2]



[2] Fuzzy Read: A transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data.

Snapshot Isolation with Semantic Related Group Precludes Phantom Read [3]



[3] Phantom Read: A transaction re-executes a query returning a set of rows that satisfies a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition.

Lock Mode in MySQL Cluster

Read_Committed [4] Lock Mode: Consistent nonlocking reads (based on MVCC)

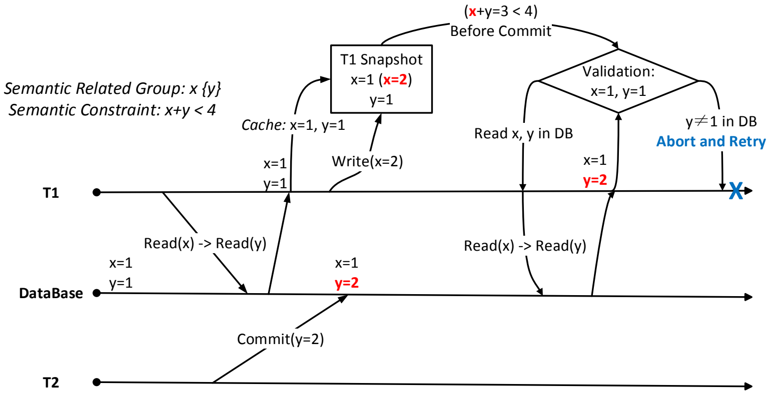
Lock Mode	**Shared**	**Exclusive**	**Read_Committed**
Shared	✓	Block	✓
Exclusive	Block	Block	✓
Read_Committed	✓	✓	✓

[4] Read_Committed here is the name of *lock mode* used in MySQL Cluster, not referring to *Isolation Level*

Optimistic Concurrency Control

- Read Phase: **Read_Committed** lock on snapshot
- Validation Phase: **Shared** lock on related Rows, **Exclusive** lock on modified rows ->Compare versions with snapshot ->Abort & Retry or Update
- Preclude: **Write Skew**

OCC with Snapshot Isolation on Semantic Related Group Precludes **Write Skew** [5]



[5] Write Skew: Two concurrent transactions read the same data, but update different data that are related and the combination of updates leads to an inconsistency.

Total Order Update, Abort & Retry, and Version Increase

- Total order update by *ids* ->preclude lock cycles
- Abort and retry transactions if "new" rows already exist
- Increase versions for successful *update phase*

Four Phases in Algorithm

- **Read Phase**: resolve semantic related group & cache it as transactions snapshot copy
- **Execution Phase**: transactions operate on its own snapshot
- **Validation Phase**: validate snapshot versions with values in database ->abort & retry or go to update
- **Update Phase**: total order update, abort & retry and version increase

Experimental Testbed

MySQL Cluster

6 data nodes, 1 Gbps LAN, Intel Xeon X5660 CPU @ 2.80GHz, 6*6=36 GB RAM, 2 data replicas

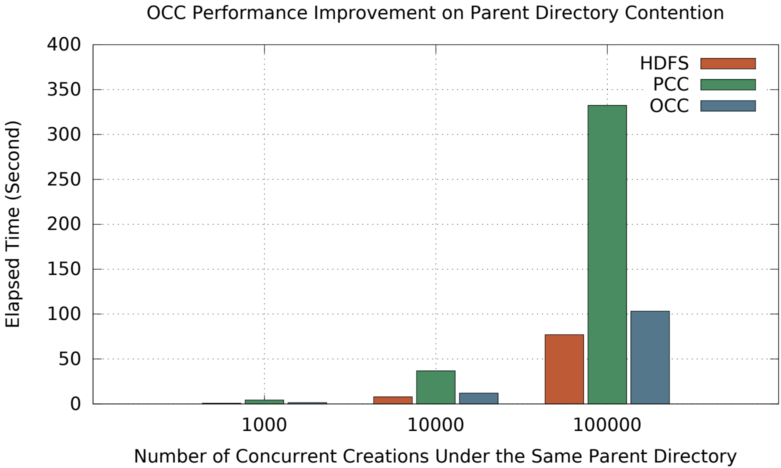
NameNode and Clients

Intel i7-4770T CPU @ 2.50GHz and 16 GB RAM

MySQL Cluster and NameNode

100 Mbps LAN

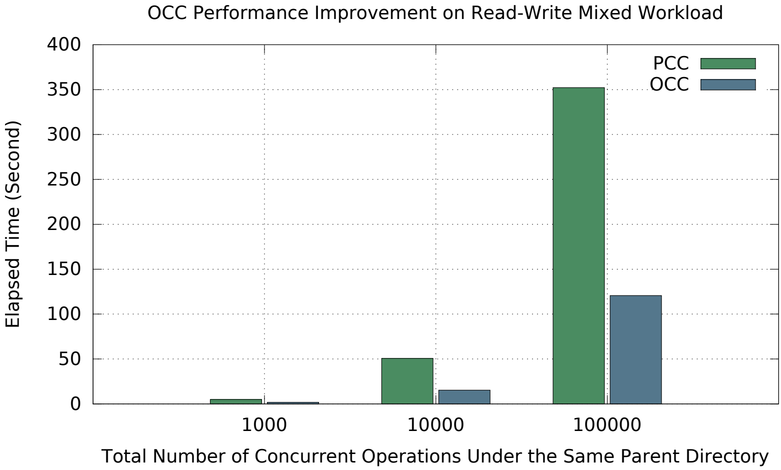
Parent Directory Contention Assessment (1/2)



Parent Directory Contention Assessment (2/2)

Num. of Concurrent Creation	**1000**	**10000**	**100000**
HDFS	0.82s	7.83s	77.13s
PCC	4.35s	36.74s	332.36s
OCC	1.36s	12.01s	103.23s
PCC / HDFS	530.5%	469.2%	430.9%
OCC / HDFS	165.9%	153.4%	133.8%
OCC Improvement: (PCC-OCC) / PCC	**68.7%**	**67.3%**	**68.9%**

Read-Write Mixed Workload (1/2)



Read-Write Mixed Workload (2/2)

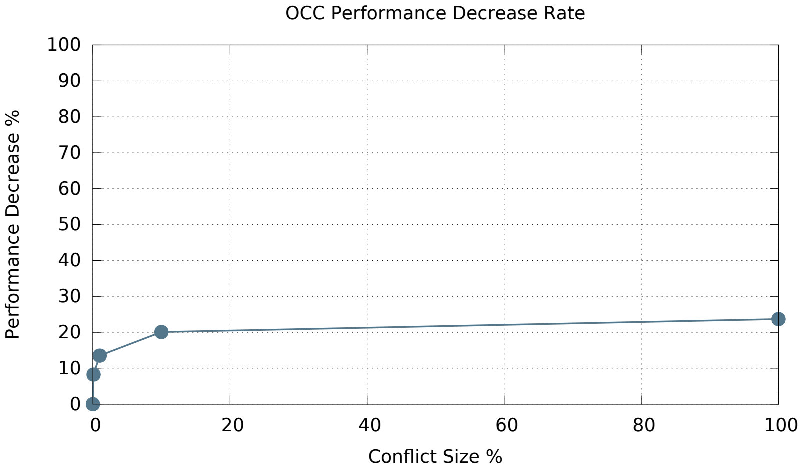
Concurrent Read+Creation	**1000**	**10000**	**100000**
PCC	4.92s	50.69s	352.25s
OCC	1.78s	15.31s	120.64s
OCC Improvement: (PCC-OCC) / PCC	**63.8%**	**69.8%**	**65.8%**

OCC Performance with Different Size of Conflicts (1/2)

Performance Decrease compares to *0% conflict size*

Creations for 10000 Operations	Conflict Size	Elapsed Time (Second)	Performance Decrease
1	100%	14.53	23.7%
10	10%	14.11	20.1%
100	1%	13.51	15.0%
1000	0.1%	12.72	8.23%
10000	0%	11.75	0%

OCC Performance Decrease Rate (2/2)



Implementation Correctness Assessment

Ensured by passing 300+ Apache HDFS Unit Tests

Conclusion & Future Work

Conclusion

- Increase Performance up to 70 %
- Bounded Performance Degradation for OCC conflict
- Maintain HDFS Strong Consistency Semantics

Future Work

- OCC implementation on other operations
- OCC evaluation on Hops-HDFS with multiple NameNodes: prove that it outperforms HDFS with single NameNode

Thank you!