

Snapshotting in HDFS for HOPS

Master of Science Thesis Defence -Pushparaj

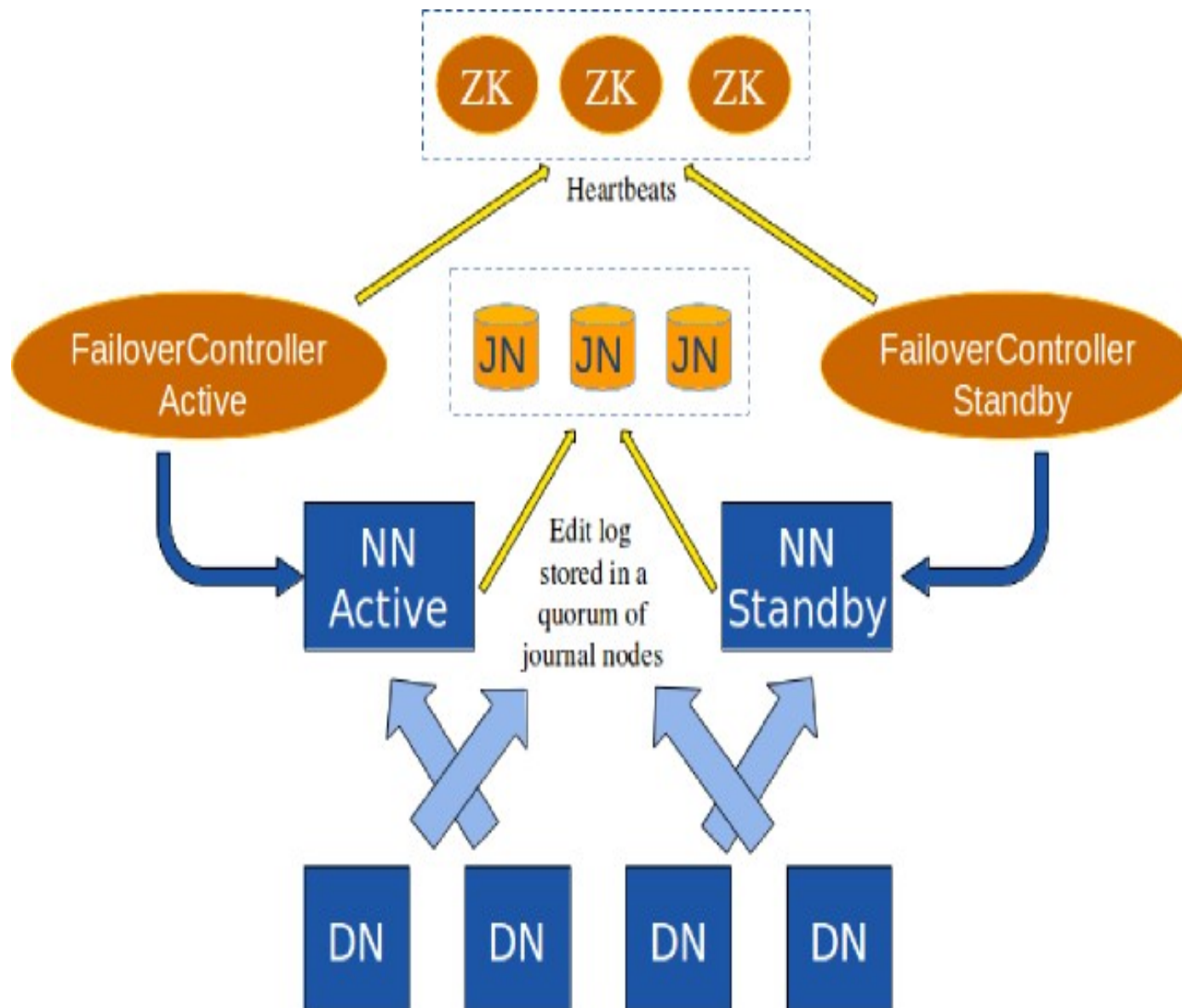
Introduction

- Definition of Snapshot
- Big Data Systems & Enterprise-Grade Storage Systems.

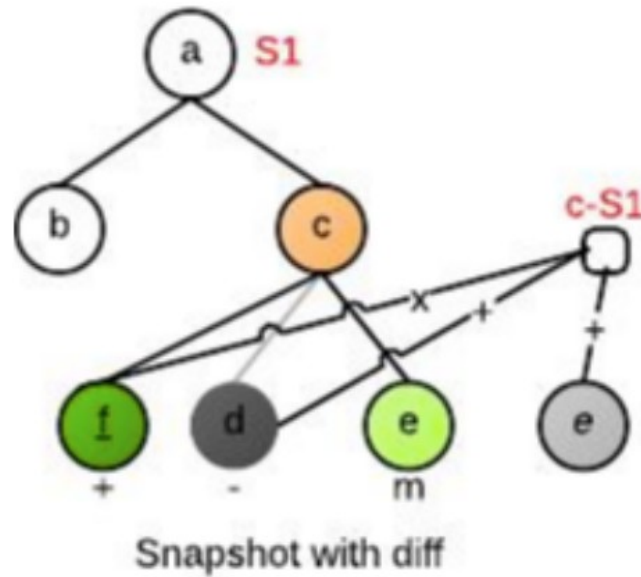
Use-Case Scenarios






- Software Upgrades
- Rollback from Errors
- Hot Backups
- Managing Real-time Data Analysis

HDFS v2 NameNode Primary/Secondary Replication Model



Related Work: Apache Vesrion 2

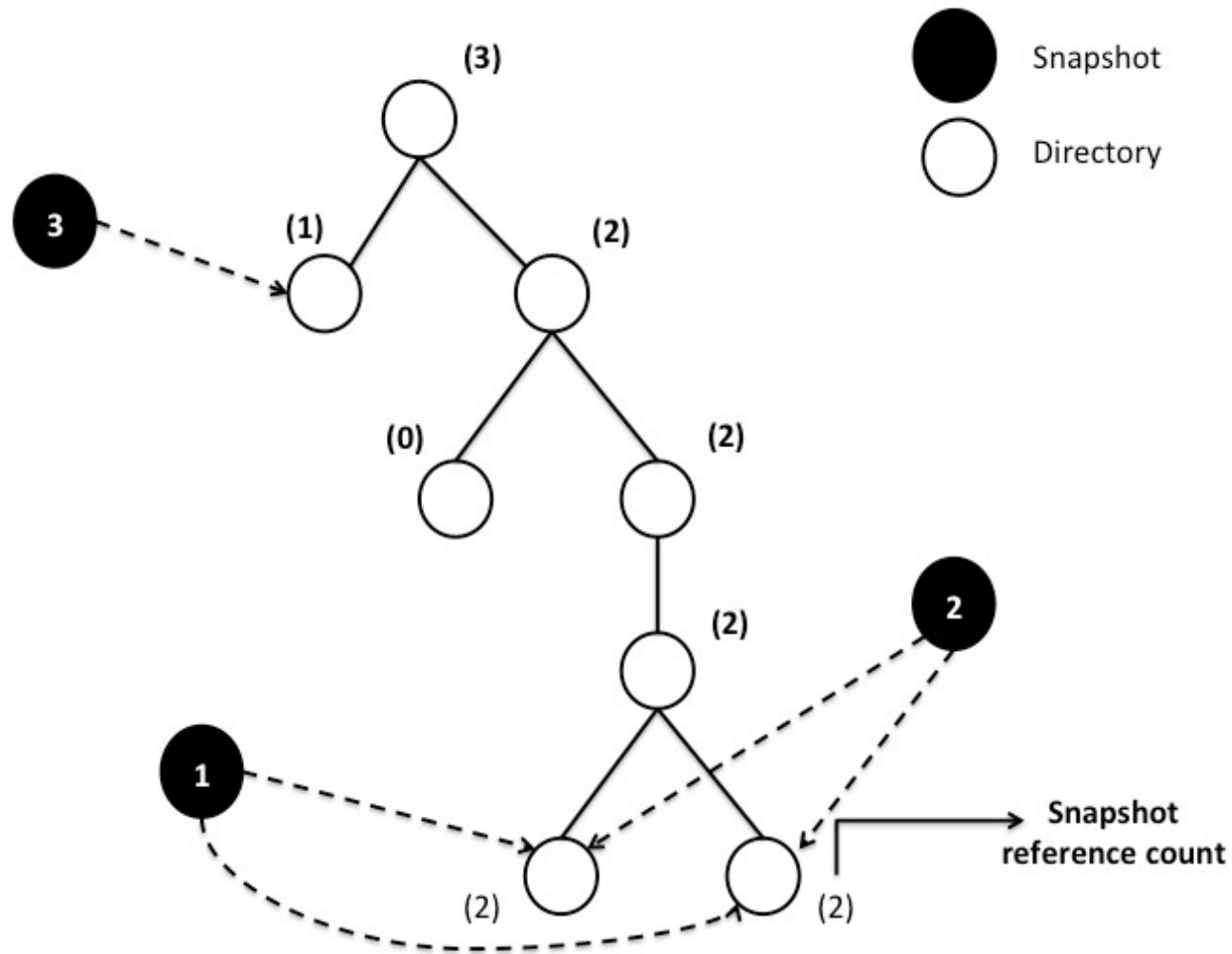


 Deleted Inode  Added Inode  Modified Inode  Snapshot Inode  InodeDirectoryWithSnapshot

Related Work: Apache Vesrion 2

- Operational costs
 - Modification of a file/directory is $O(\log(\text{diff size}))$
 - Access time for current files and directories is unchanged
 - Access time for snapshot files and directories has additional cost of $O(\log(\text{diff size}))$
- Storage costs - Proportional to the unique modifications since the snapshot is taken - $O(M)$

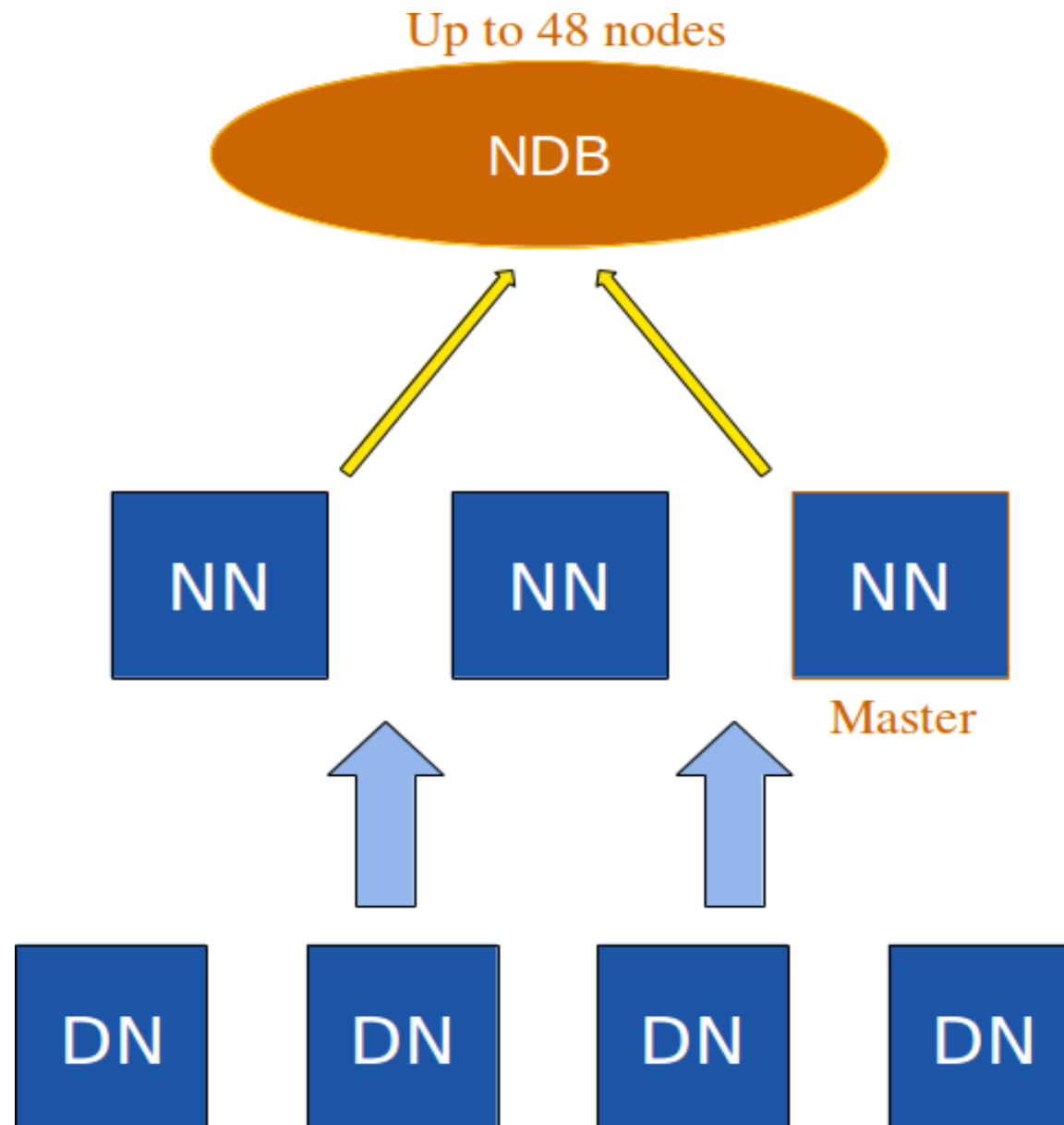
Related Work: Facebook



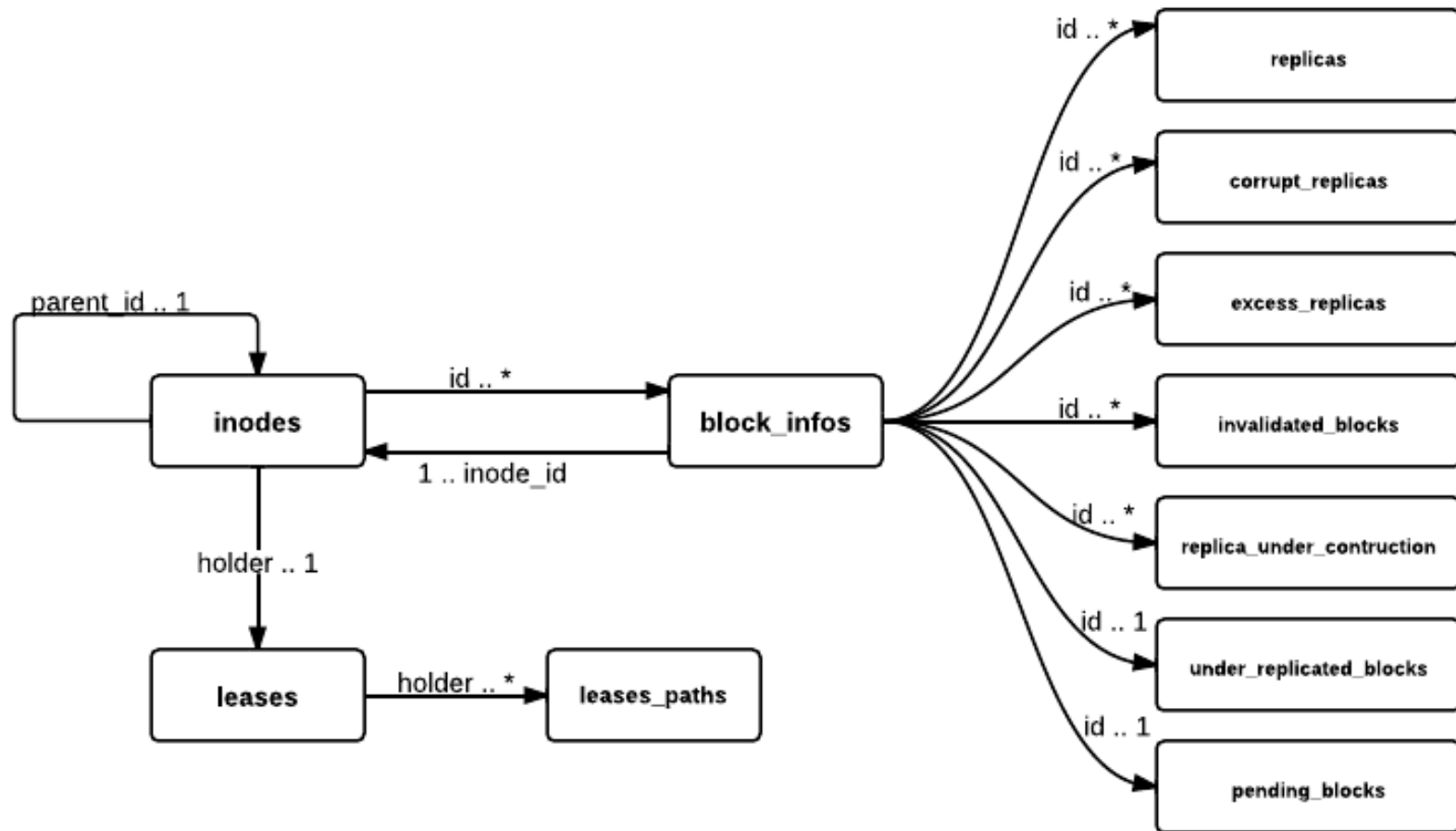
Related Work: Facebook

- Selective Copy-On Appends
 - Copy-On Write
 - Appends followed by Truncate
 - Use Pointers Otherwise.
- Overhead of building Snapshot tree
- Time to take Snapshot Scales linearly.

HOP-HDFS




Entity-Relation Diagram for NameNode state in Hop-HDFS



Hop-Hdfs Operations

snapshot.clear

Operation doOperation

```
tx.begin  
create-snapshot()   
performTask()  
tx.commit
```

Operation create-snapshot

```
S = total_order_sort(op.X)  
for all x in S do  
    if x is a parent then  
        level = x.parent_level_lock  
    else  
        level = x.strongest_lock_type  
        tx.lockLevel(level)  
        snapshot <- tx.find(x.query)  
    end if  
end for
```

Operation performTask

//Operation Body,referring to transaction cache for data

Challenges

- Time to take Snapshot— $O(1)$
- Scale for TeraBytes of Meta-Data(HDFS)
- Efficient Roll-Back
- Low Sapce OverHead on Snapshot meta-data

Solution

- Read-Only(RO) Snapshots
- RO Nested Snapshots
 - General Purpose file/directory level multiple/nested Snapshots.
- RO Root Level Single Snapshot
 - Specific for RollBack on Software Upgrades

Read-Only Root Level Single Snapshot(ROSS)

- **isDeleted**

0 ==> Inode is not deleted.

1 ==> Inode deleted after Root Level snapshot was taken.

- **status**

0 ==> Inode was created before taking Snapshot.

2 ==> Inode created before taking snapshot but modified after that.

3 ==> Inode was created after taking snapshot.

Deletion of Directories

- atomic{
Set isDeleted=1 for this directory
}
- Process the children in depth-first manner.
- If file status=3,
 - atomic{
permanently delete blocks
delete the inode row.
}
- Atomic construct to guarantee consistency of meta-data in case of namenode failure.

Roll Back

For INodes:

- Delete from INodes where status=2 or status=3
- Update INodes set isDeleted=0 where id>0 and isDeleted=1
- Update INodes set id = -id, parent id = -parent id where id<0

For Blocks:

- Delete from Block Info where status=2 or status=3
- Update Block Info set block id = -block id, inode id = -inode id where id<0
- Delete from Block Info where block id<0

Implementation-RollBack

- Take subTreeOpLock on root
- Take read-lock on all rows
- Task1
 - Delete Inodes with status=2 or Status=3
- Task2
 - Set isDeleted=0 for inodes isDelered=1 and id>0
- Task2
 - Update inodes set id=-id, parent_id=-parent_id where id<0;
 - Deleted inodes where id<0;

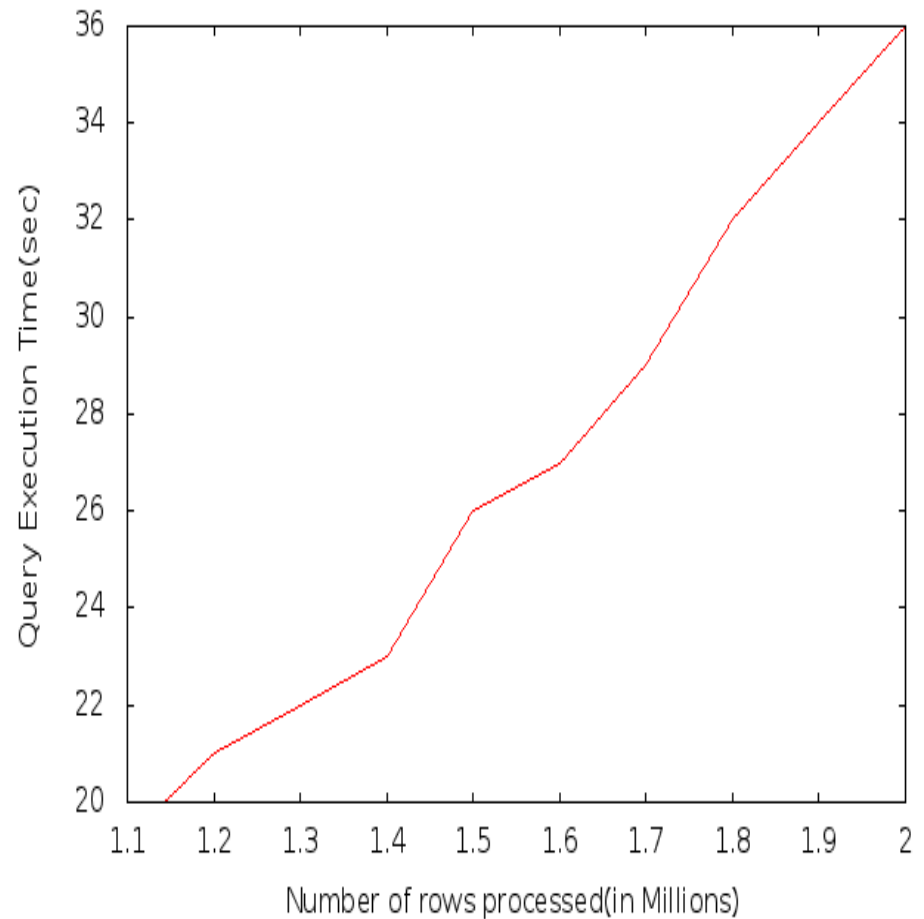
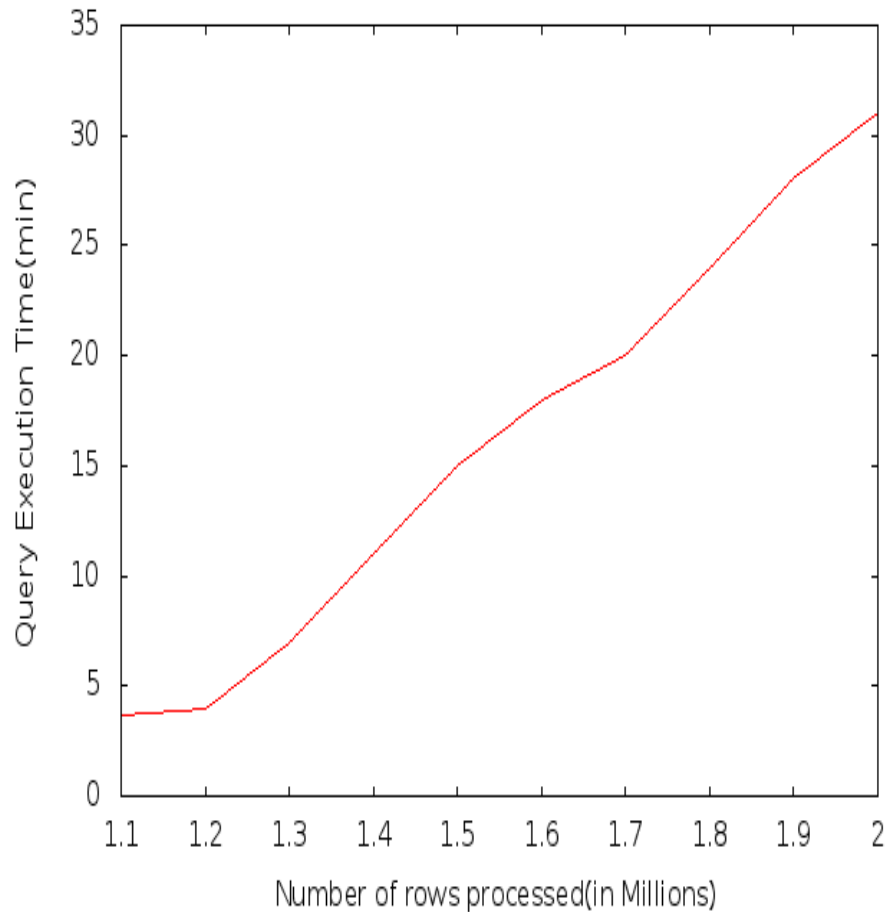
Implementation-RollBack

- Task1, Task2, Task3 are executed on thread pool with batch size 100,000
- Failure Handling:
 - Insert row with Task Id , NameNode Id,status
 - If(status=InProgress && Namenode is dead)
 - Leader assigns the task to another namenode.
 - Since Task1, Task2, Task3 are Idempotent, progress is guaranteed.

Evaluation-RollBack

UnModified inodes that are Deleted	UnModified inodes that are not Deleted	Modified inodes	New Inodes	Total rows Processed	Time	clusterJ
450,000	450,000	100,000	100,000	1,200,000	4 min 35 sec	21 sec
400,000	400,000	200,000	100,000	1,300,000	7 min 37sec	22 sec
350,000	350,000	300,000	100,000	1,400,000	11min 25 sec	23 sec
300,000	300,000	400,000	100,000	1,500,000	14min 40 sec	26sec
250,000	250,000	500,000	100,000	1,600,000	18 min 10sec	27sec
200,000	200,000	600,000	100,000	1,700,000	20 min 5 sec	29sec
150,000	150,000	700,000	100,000	1,800,000	23 min 50sec	32 sec
100,000	100,000	800,000	100,000	1,900,000	27min 40sec	34 sec
50,000	50,000	900,000	100,000	2,000,000	31 min 2 sec	36 sec

RollBack(MySql vs ClusterJ)



Read-Only Nested Snapshots

- SNAPS

–	Inode_Id	Snapshot_Id	User	Time
---	----------	-------------	------	------

- C-List

–	Inode_Id	Time	Created_Inode_Id
---	----------	------	------------------

- D-List

–	Inode_Id	Time	Deleted_Inode_Id
---	----------	------	------------------

- M-List

–	Inode_Id	Time	Modified_Inode_Id	Original_Row
---	----------	------	-------------------	--------------

- MV-List

–	Inode_Id	Time	Modified_Inode_Id	Original_Row
---	----------	------	-------------------	--------------

- MV-In-List

–	Inode_Id	Time	Moved_In_Inode Id
---	----------	------	-------------------

RO-Nested Snapshots(RONS)

- Block-Info C-List

- | | | |
|------------|----------|------|
| – Inode_Id | Block_Id | Time |
|------------|----------|------|

- Block-Info M-List

- | | | | |
|------------|----------|------|--------------|
| – Inode_Id | Block_Id | Time | Original_Row |
|------------|----------|------|--------------|

Listing files in a directory

Void ls(int stime, int pid){

- children={Get children whose parentId=id};
- children = children - { children deleted before stime} - { children created after stime} - {children moved_in after stime};
- children = children + {children moved-out after stime};
- modifiedChildren = { children modified after stime};
- For-each children if it is modified first then moved then print former.
- }

RO-Nested Snapshots

- How to determine whether an inode is in any snapshot ? Ex: /A/B/C/ Is C in any Snapshot?
- Snapshots taken on C , if not
- Check for snapshots on A, B
 - But how do we know whether C was present when those snapshots on A or B taken?

RO-Nested Snapshots

- JoinTime(JT): The time this inode joined its present path from root. Ex: /A/B/C/
- Join-Time are non-decreasing on a given path from root. Ex: $JT(A) \leq JT(B) \leq JT(C)$
- $MaxJT(C) = Max(JT(A), JT(B), JT(C))$

RO-Nested Snapshots

- Move directory /A/B/C/ to /U/V/W/
- $SN = \{\text{Snapshots on A, B after MaxJT(C)}\}$
- For each node in sub-tree at C
 - Insert entry in InodeSnapshotMap table based on node JT. Since some nodes may have $JT > JT(C)$, which moved into C after C moved into B.
- For each node in sub-tree at C update the JT to present time.(The move time).
- InodeSnapshotMap

Inode_Id	Belongs_to _Inode_Id	StartTime	EndTime
C	A	10	12

RO-Nested Snapshots

- How to determine whether an inode is in any snapshot ? Ex: /A/B/C/ Is C in any Snapshot?
- Snapshots on C or
- Snapshots on A or B after MaxJT or
- Entry in InodeSnapshotMap with id=C.

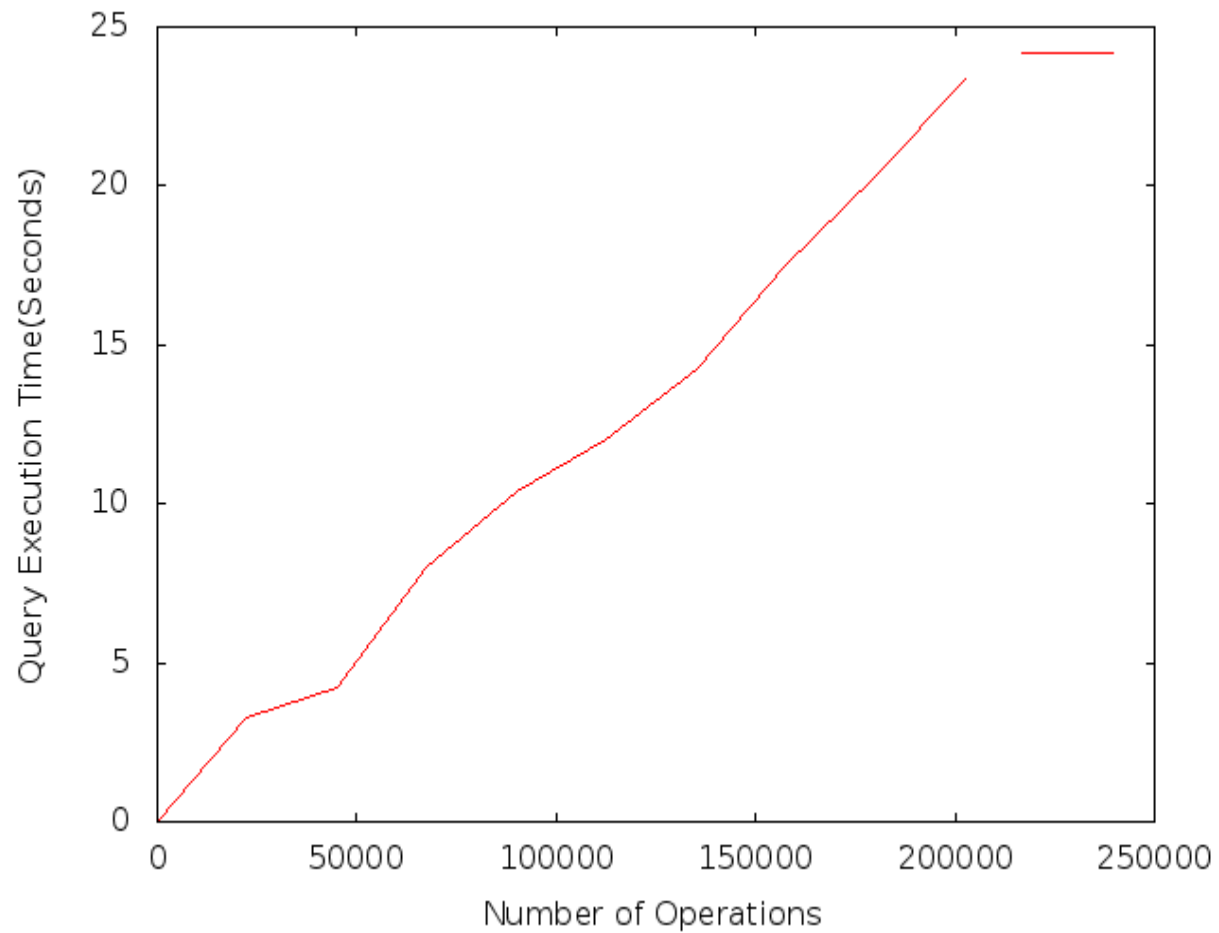
RO-Nested Snapshots

- Delete directory */A/B/C/*
- Check whether C is in any snapshot
- $SN = \{\text{Snapshots on A,B after MaxJT(C)}\}$
- For each node in Sub-tree at C
 - Set `isDeleted=true`
 - Insert entry in `InodeSnapshotMap` based on it JT and SN.
- Background thread deletes nodes with `isDeleted=1` which are not in any snapshot.

Evaluation- Nested Snapshots

ADD ops	DEL ops	MOV ops	MovIN ops	RENAME ops	Total Ops	New Time
5000	5000	5000	2500	5000	22,500	3.72 sec
5000*2	5000*2	5000*2	2500*2	5000*2	45,000	4.25 sec
5000*3	5000*3	5000*3	2500*3	5000*3	67500	7.98sec
5000*4	5000*4	5000*4	2500*4	5000*4	90,000	10.4 sec
5000*5	5000*5	5000*5	2500*5	5000*5	112,500	12.03 sec
5000*6	5000*6	5000*6	2500*6	5000*6	135,000	14.21 sec
5000*7	5000*7	5000*7	2500*7	5000*7	157,500	17.48 sec
5000*8	5000*8	5000*8	2500*8	5000*8	180,000	20.25 sec
5000*9	5000*9	5000*9	2500*9	5000*9	202,500	23.35 sec

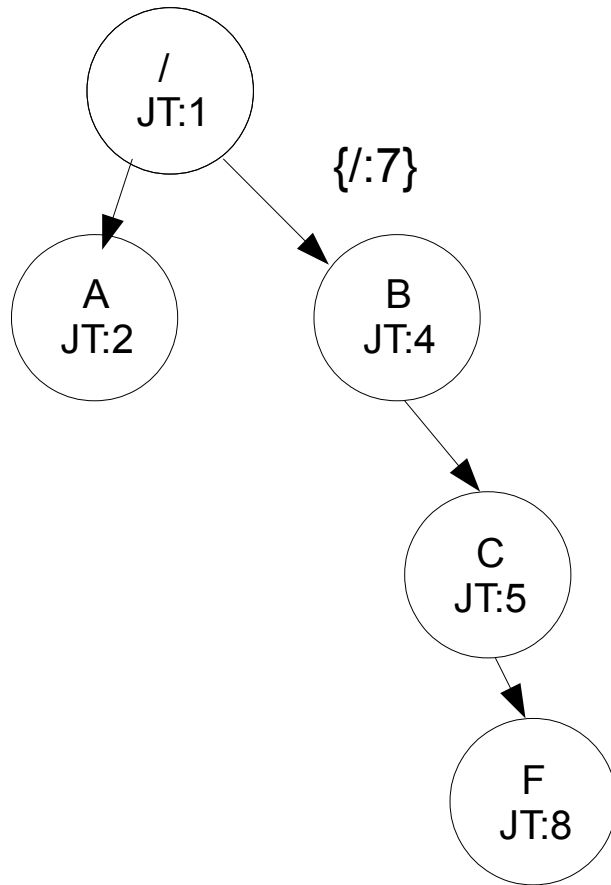
Evaluation-Nested Snapshots



Future Work

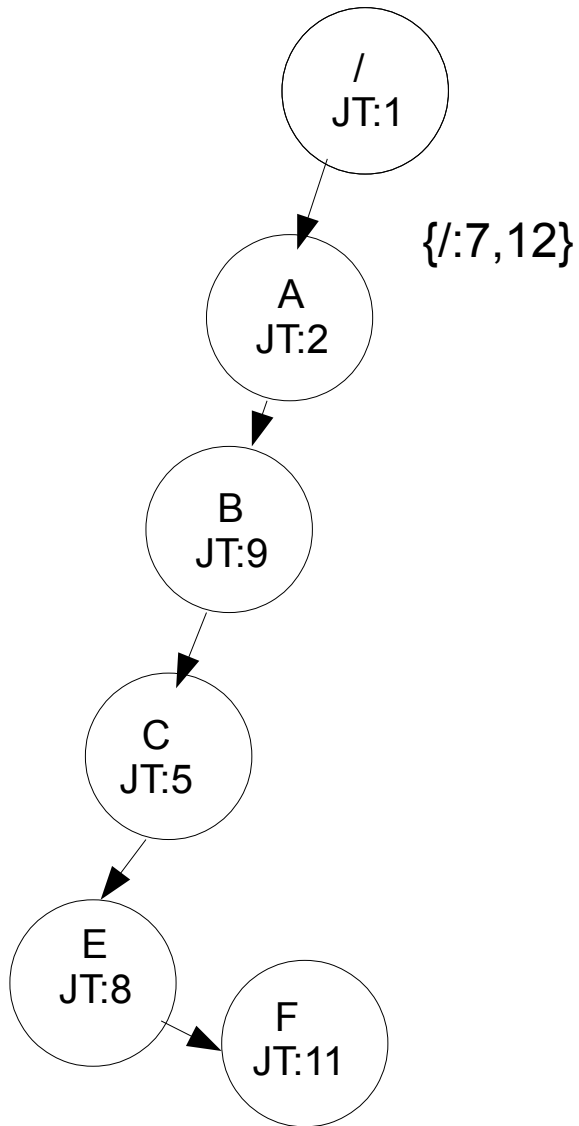
- Implementing Nested Snapshots
- Integrating RO Root Level SingleSnapshot and ROnetsted Snpashot solutions

Move B to A



Id	Snap_Id	Time
A	SA1	3
B	SB1	6
/	S/1	7

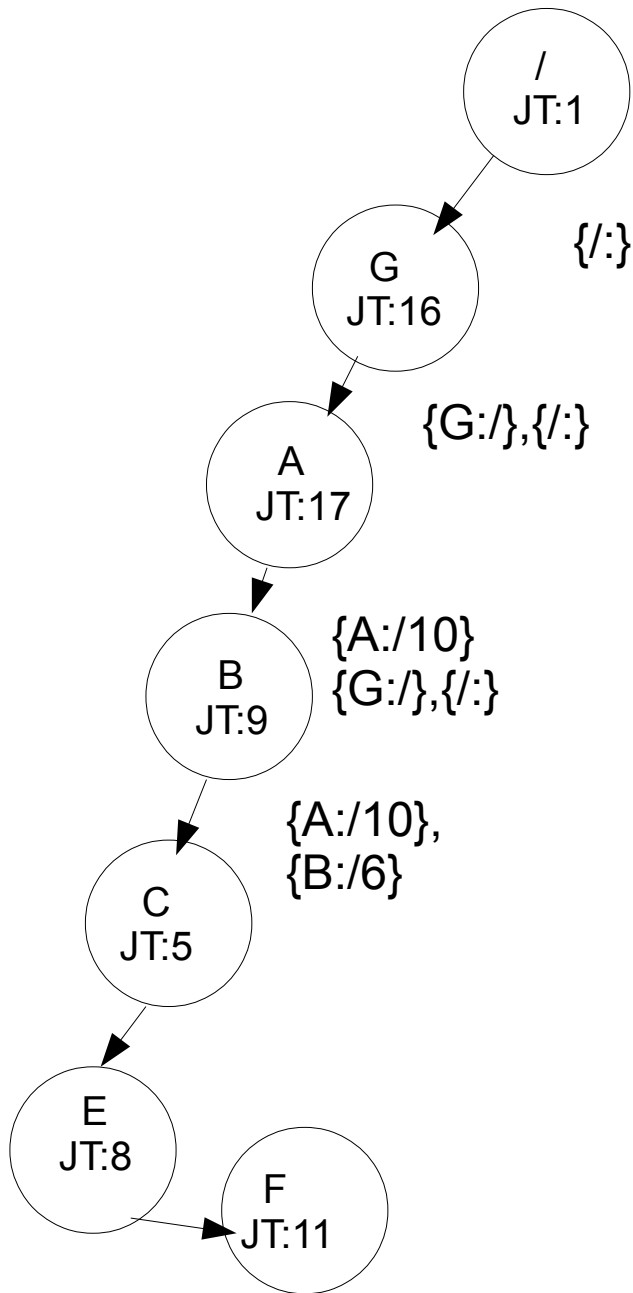
Move A to G



Id	Snap_Id	Time
A	SA1	3
B	SB1	6
/	S/1	7
A	SA2	10
/	S/2	12

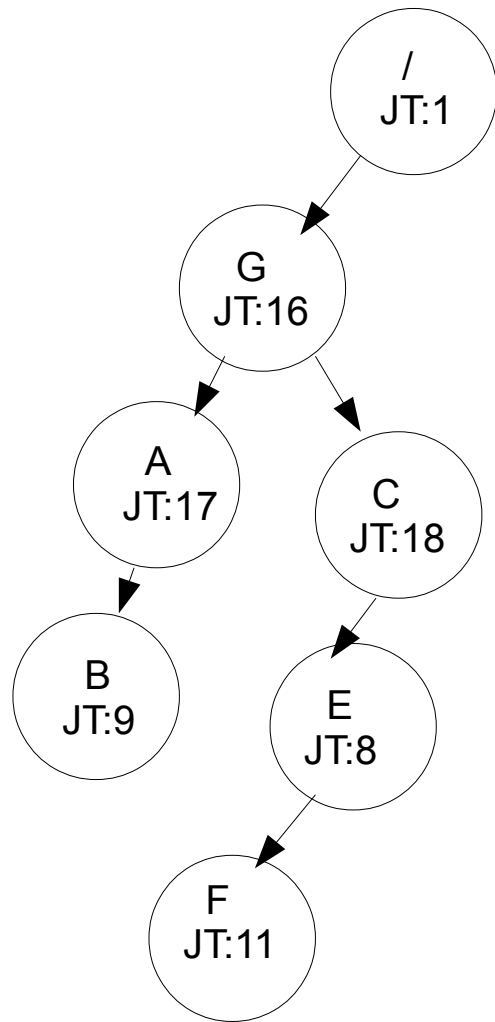
Inode_Id	Belongs_to_Inode_Id	StartTime	EndTime
B	/	7	7
C	/	7	7

Move C to G



Id	Snap_Id	Time
A	SA1	3
B	SB1	6
/	S/1	7
A	SA2	10
/	S/2	12

Inode_Id	Belongs_to_Inode_Id	StartTime	EndTime
B	/	7	7
C	/	7	7
A	/	7	12
B	/	12	12
C	/	12	12
E	/	12	12
F	/	12	12



Id	Snap_Id	Time
A	SA1	3
B	SB1	6
/	S/1	7
A	SA2	10
/	S/2	12

Inode_Id	Belongs_to_Inode_Id	StartTime	EndTime
B	/	7	7
C	/	7	7
A	/	7	12
B	/	12	12
C	/	12	12
E	/	12	12
F	/	12	12
C	A	10	10
C	B	6	6
E	A	10	10

Thank You