

INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Snapshotting in HDFS for HOPS

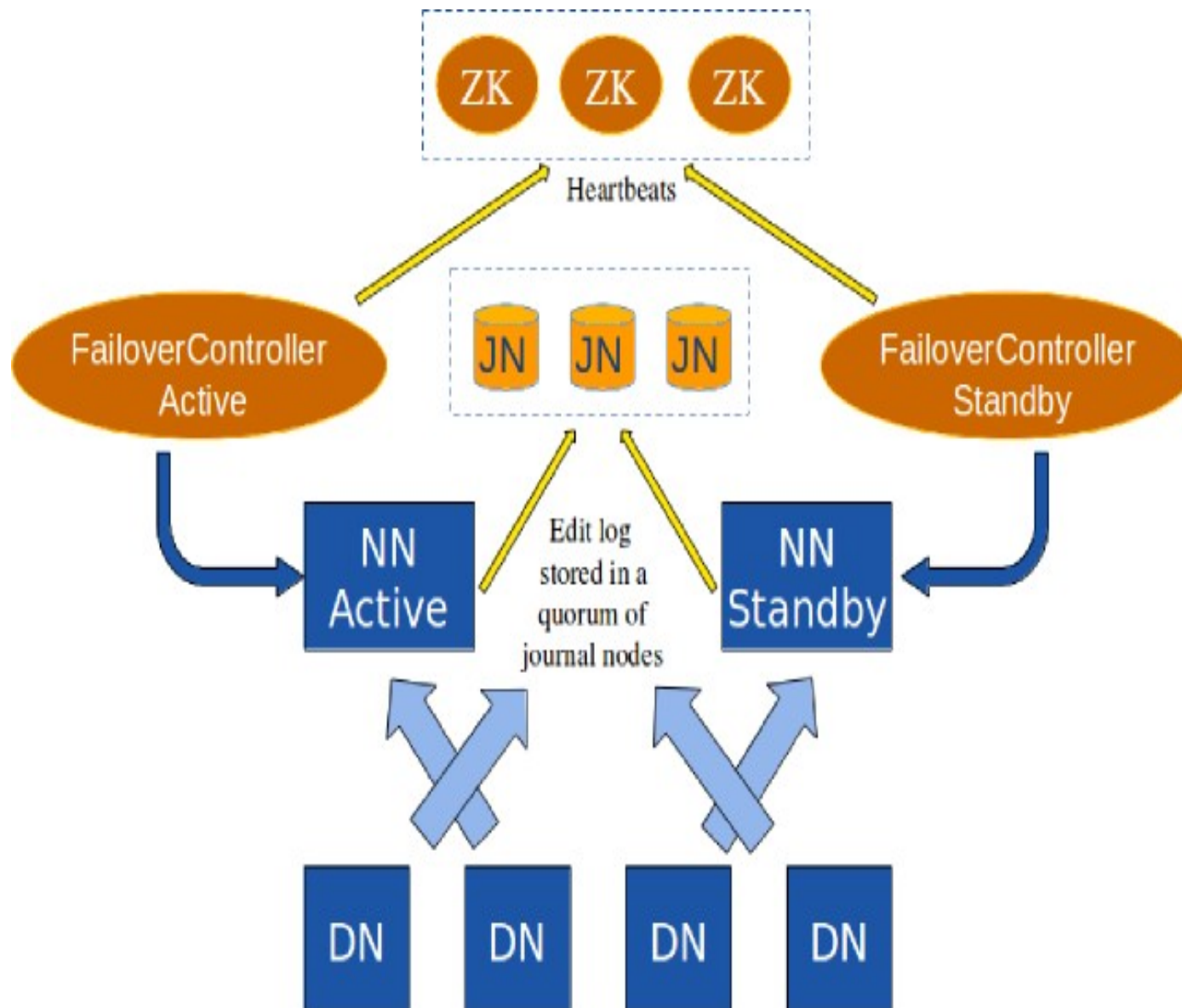
Master of Science Thesis -Pushparaj

Supervisor:Prof. Luís Manuel Antunes Veiga

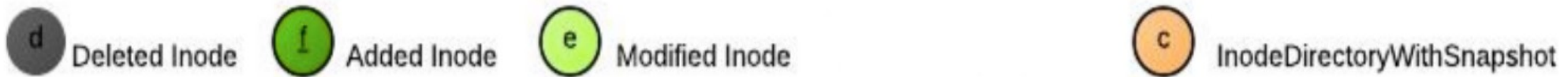
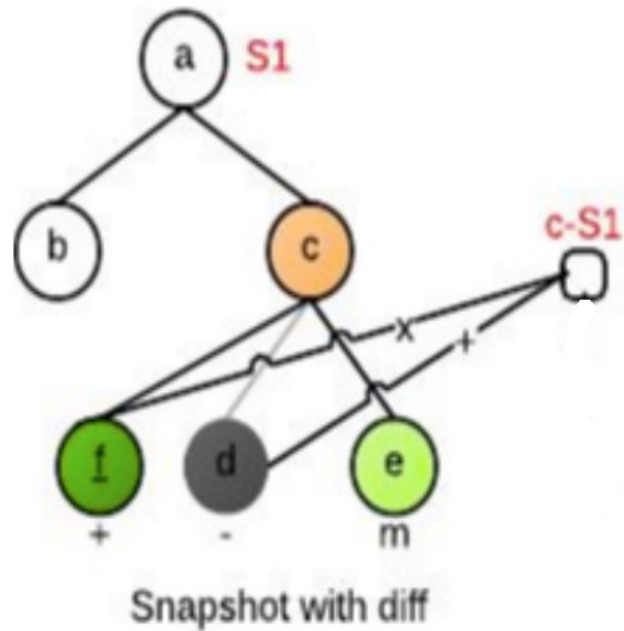
Snapshot -Use Cases

- Software Upgrades
- Rollback from Errors

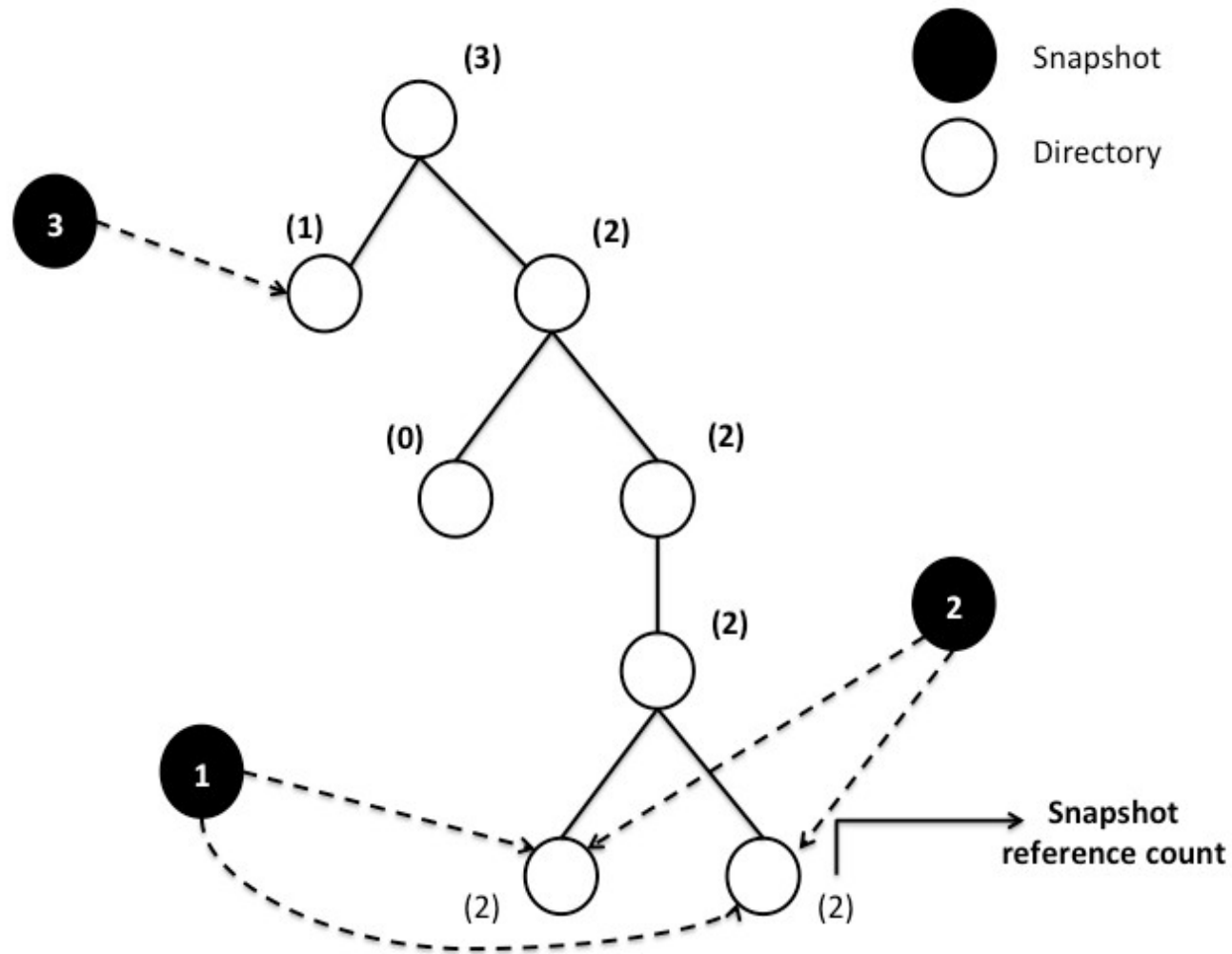
HDFS Architecture



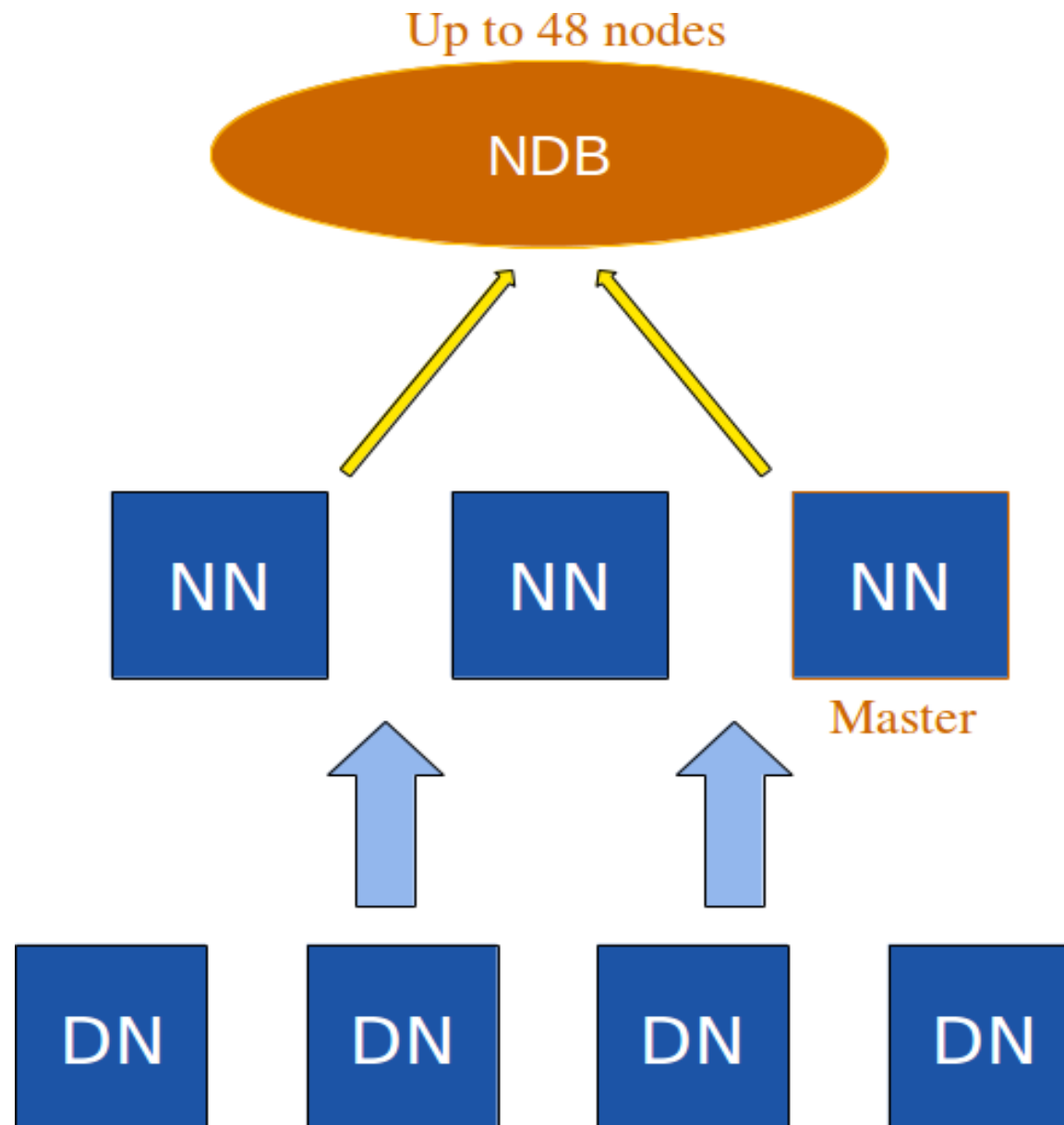
Snapshots in HDFS



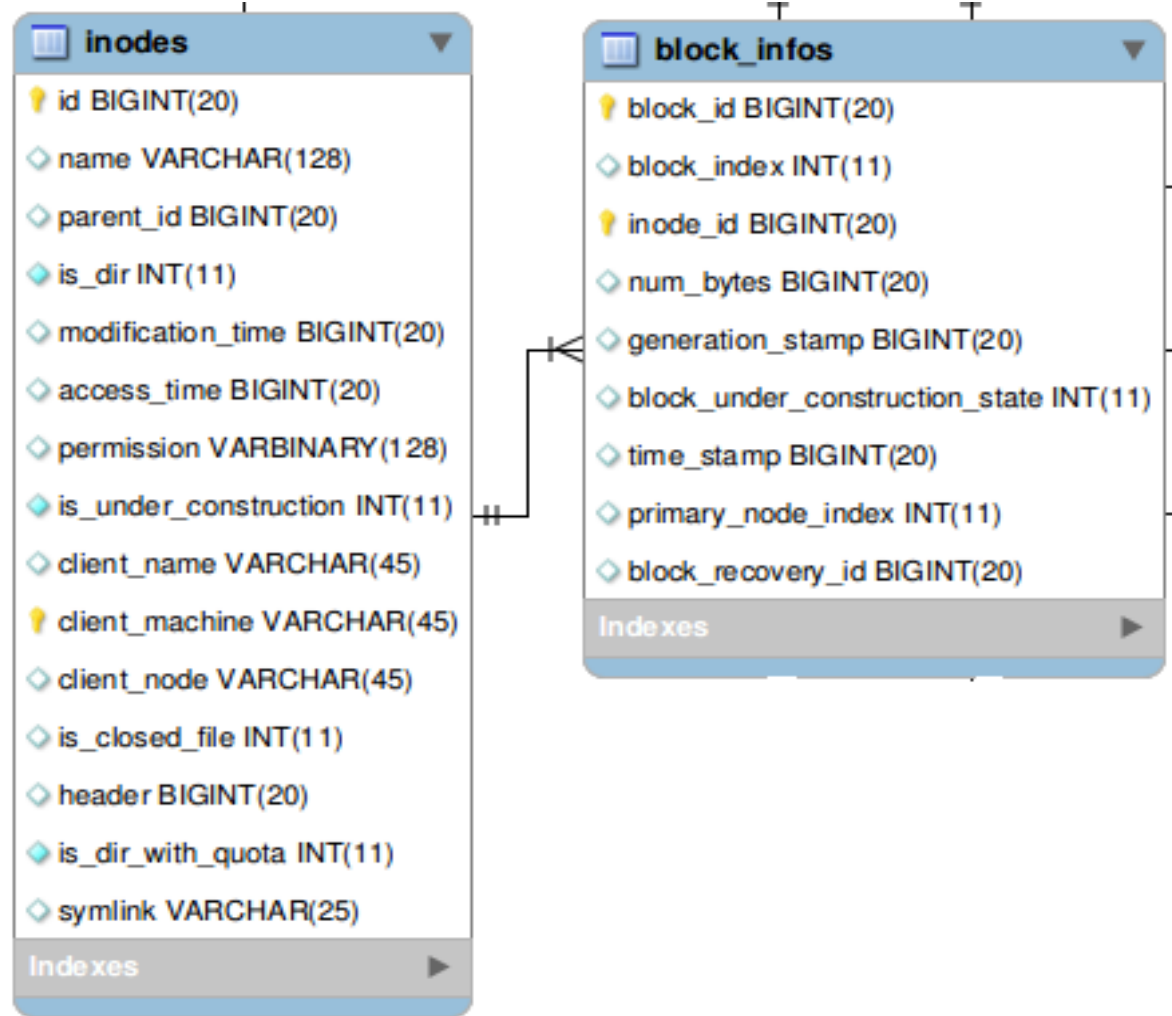
Related Work: Facebook



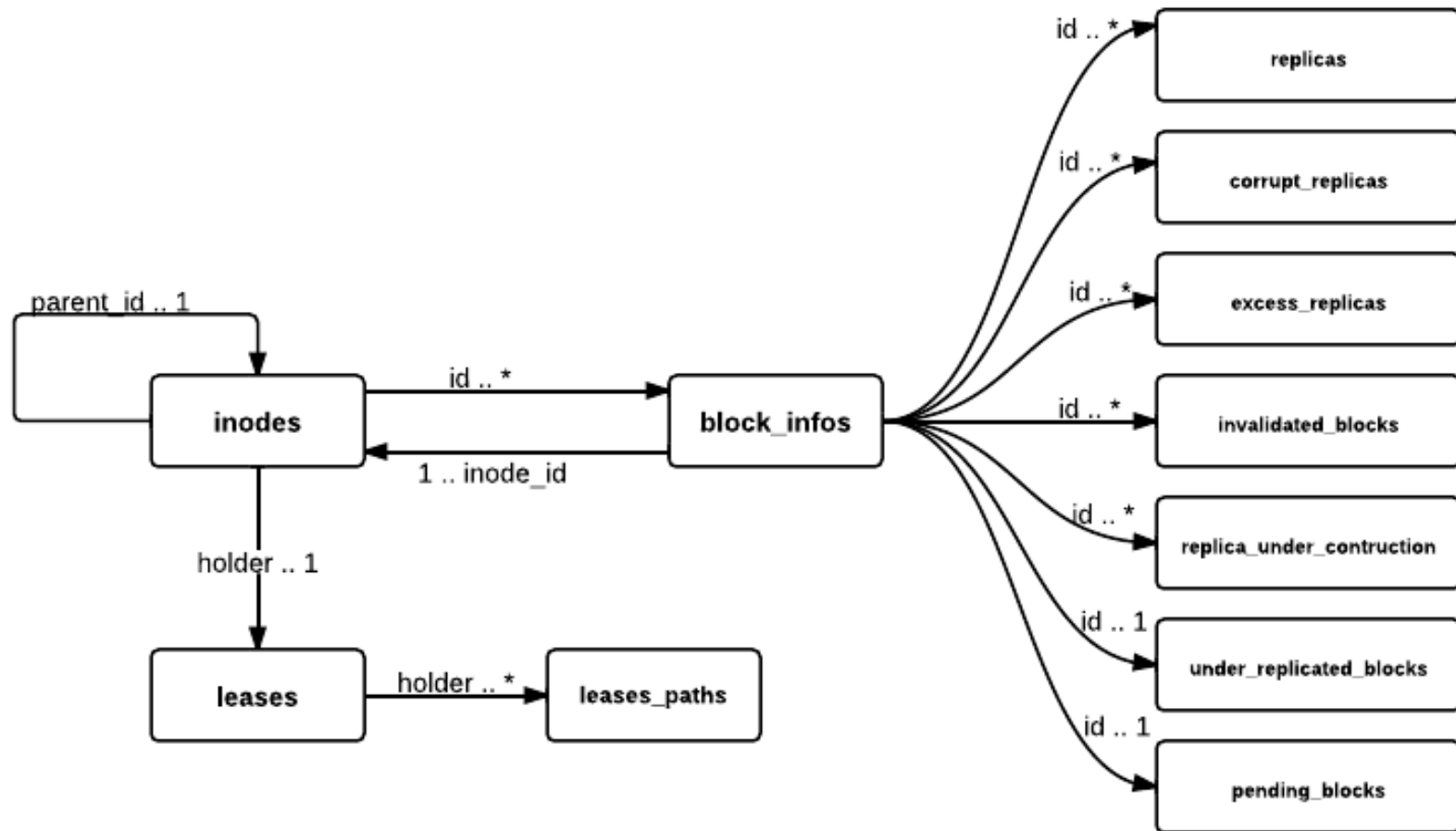
HOP-HDFS



HOP-HDFS



HOP-HDFS



Challenges

- Time to take Snapshot— $O(1)$
- Scale for Tera-bytes of metadata
- Efficient Rollback
- Low Space Overhead on Snapshot metadata

Solutions

- 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 metadata in case of NameNode failure.

RollBack

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

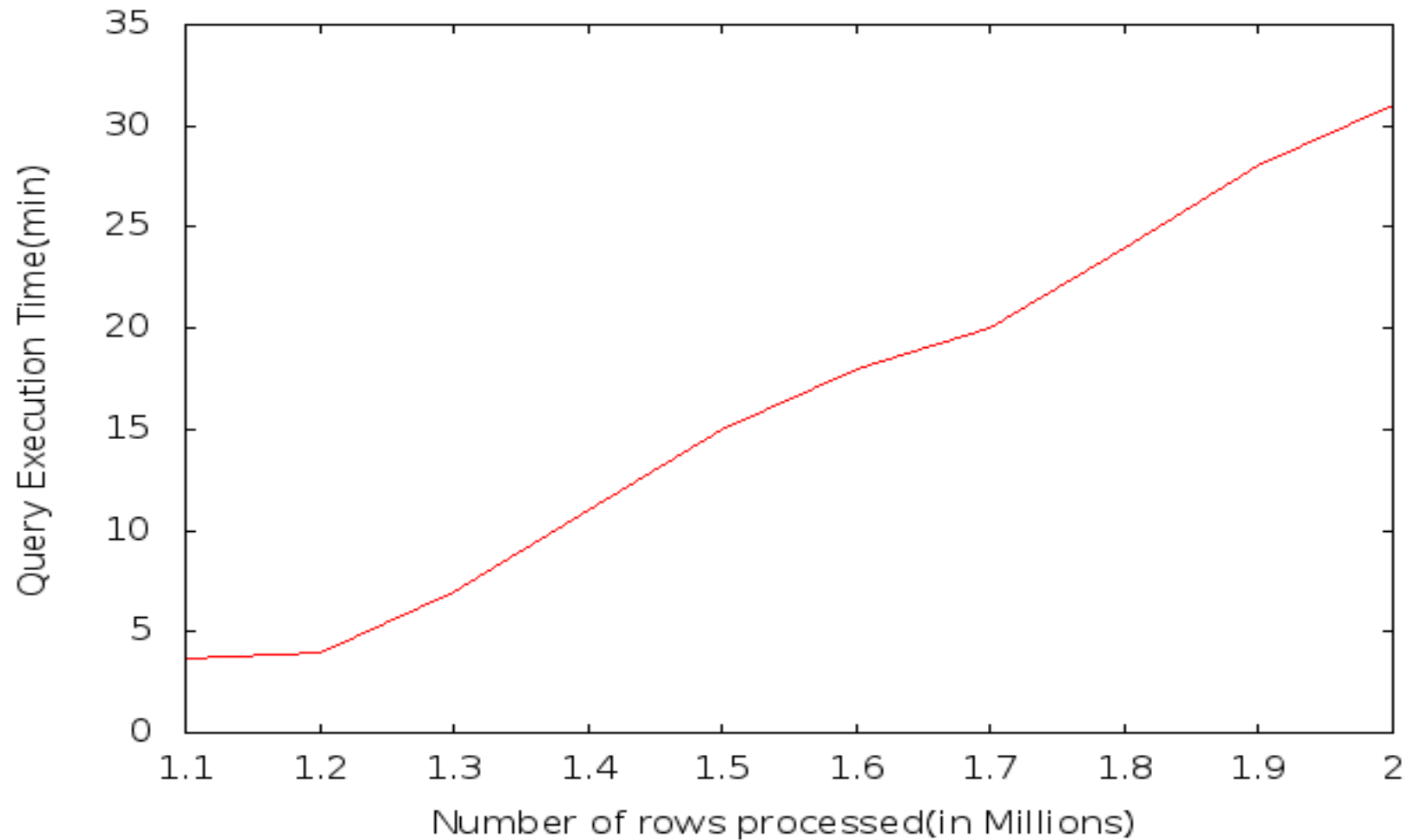
Rollback -Implementation

- 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 isDeleted=1 and id>0
- Task3
 - Update inodes set id=-id, parent_id=-parent_id where id<0;
 - Deleted inodes where id<0;

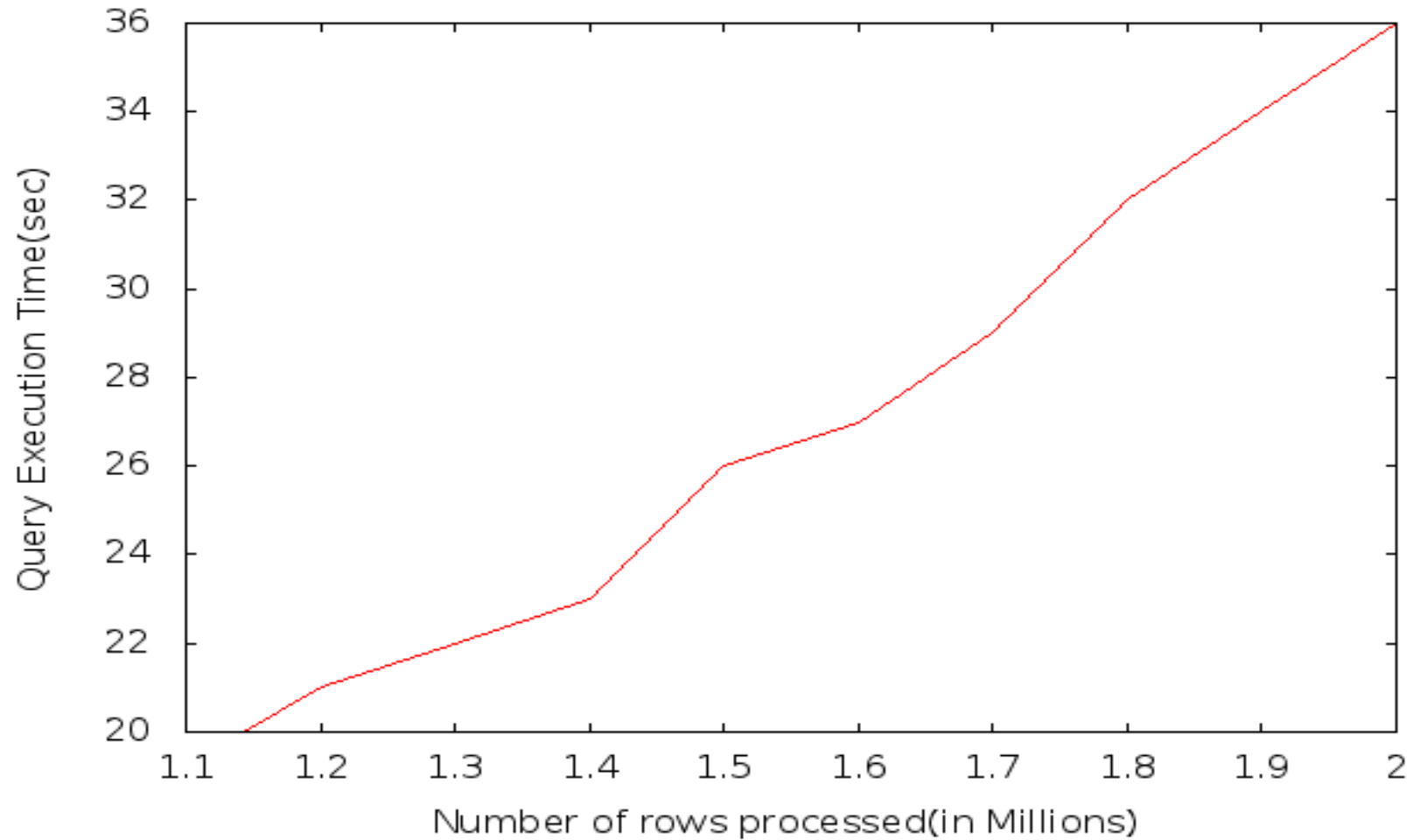
Implementation-RollBack

- 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.

Rollback on MySql Server



Rollback on 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

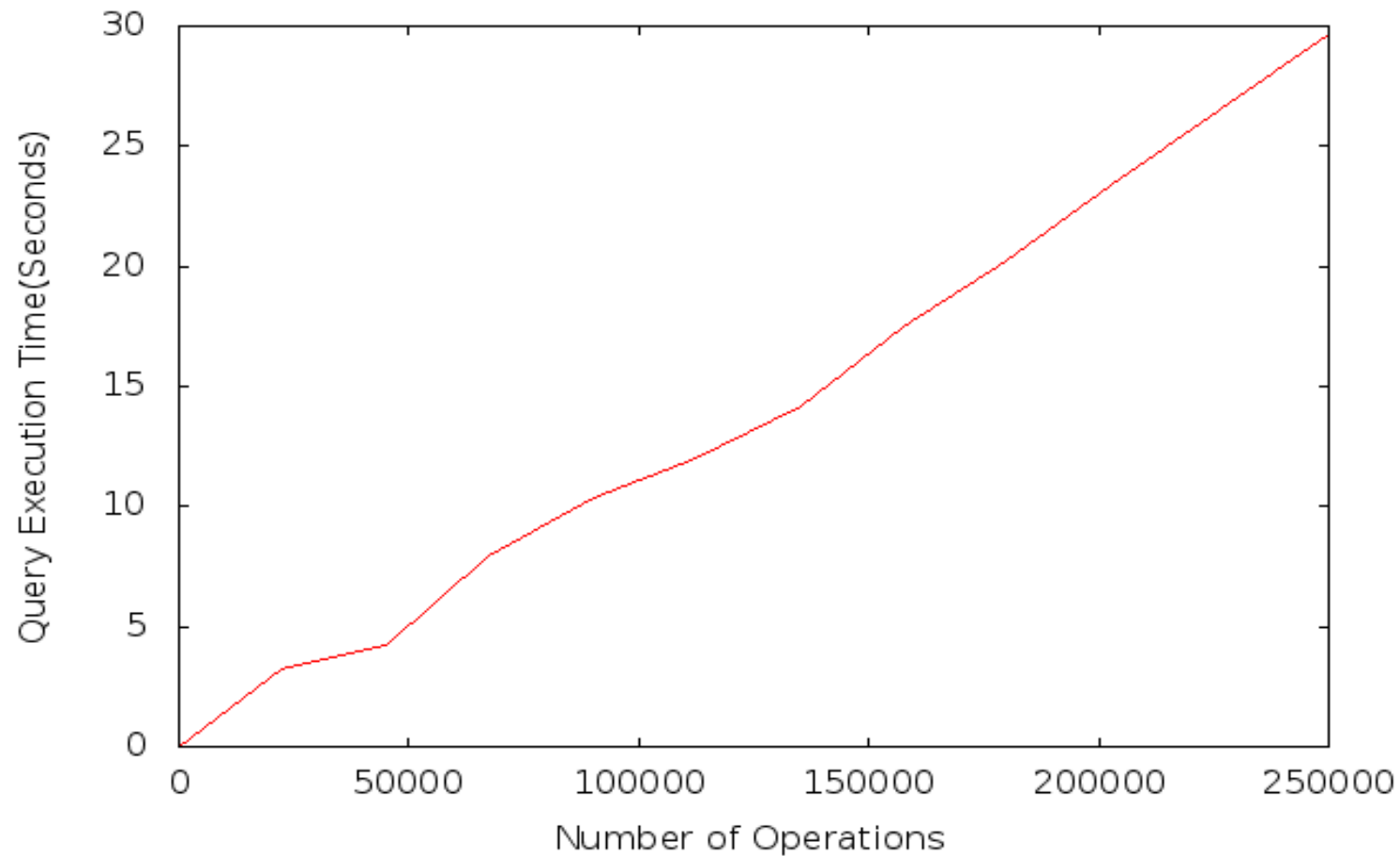
- | | | | | |
|---|----------|----------|------|--------------|
| – | inode_Id | Block_Id | Time | Original_Row |
|---|----------|----------|------|--------------|

Listing files in a directory

```
Void ls(int stime, int id){
```

- 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};
- modified-children = { children modified after stime};
- For-each children if it is modified first then moved then print former.
- }

Evaluation-Nested Snapshots



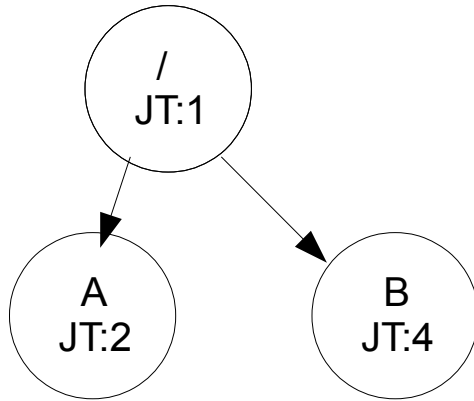
RO-Nested Snapshots

- How to determine whether an inode is in any snapshot ? Ex: /A/B/C/ Is C in any Snapshot?

RONs-Join Time

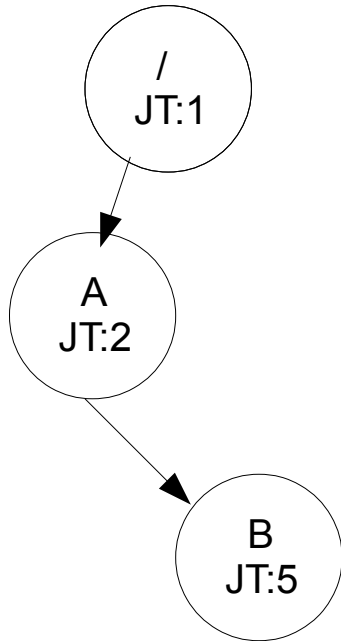
- JoinTime(JT): The time this inode joined its present path from root.

RONs-Join Time



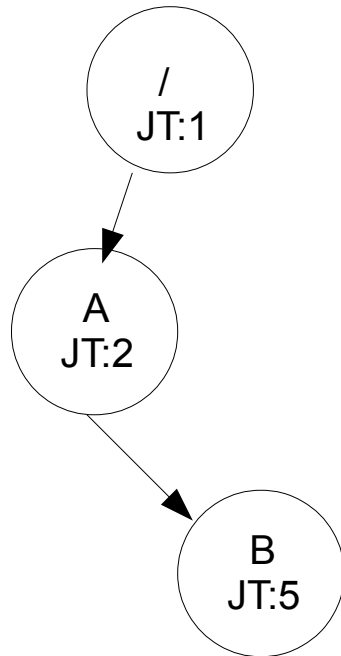
Id	Snap_Id	Time
A	SA1	3

RONs-Join Time



Id	Snap_Id	Time
A	SA1	3
A	SA2	6
B	SB1	7

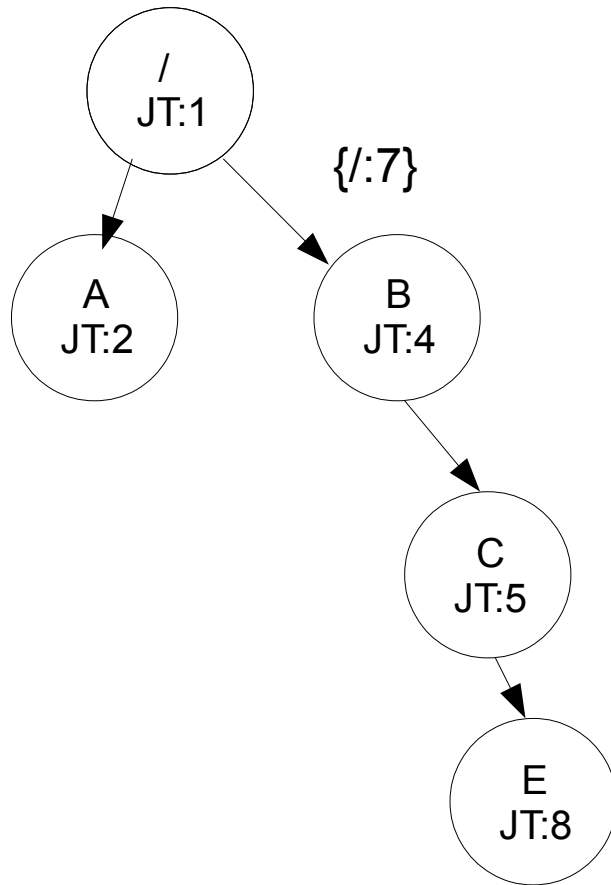
RO-Nested Snapshots



- Snapshots in which B is present
 - Snapshots taken on B = {SB1}
 - Snapshots taken on /[Root] after JT(B) & JT(A) = {}
 - And Snapshots taken on A after JT(B) = {SA2}

Id	Snap_Id	Time
A	SA1	3
A	SA2	6
B	SB1	7

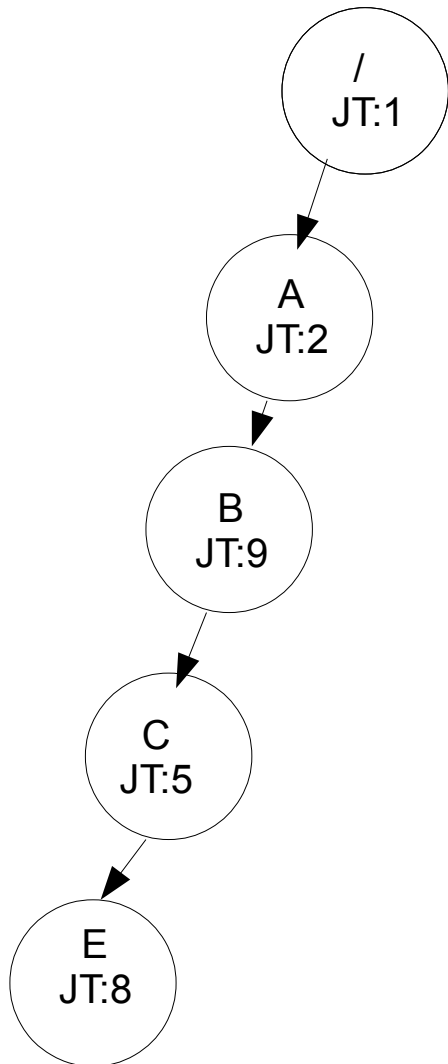
Move B to A



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

Move B to A

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



InodeSnapshotMap Table

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

Conclusion

- Efficient Root Level Snapshot.
- Efficient Nested Snapshot design.

Future Work

- Implementing Nested Snapshots
- Integrating RO Root Level Single Snapshot and RO Nested Snapshot solutions

Thank You