Loosely Time-Synchronized Snapshots in Object-Based File Systems

Jan Stender, Mikael Högqvist, Björn Kolbeck
Zuse Institute Berlin

Outline

- Motivation
- Problem Description
- System Architecture
- Algorithm
- Evaluation
- Conclusion

Motivation

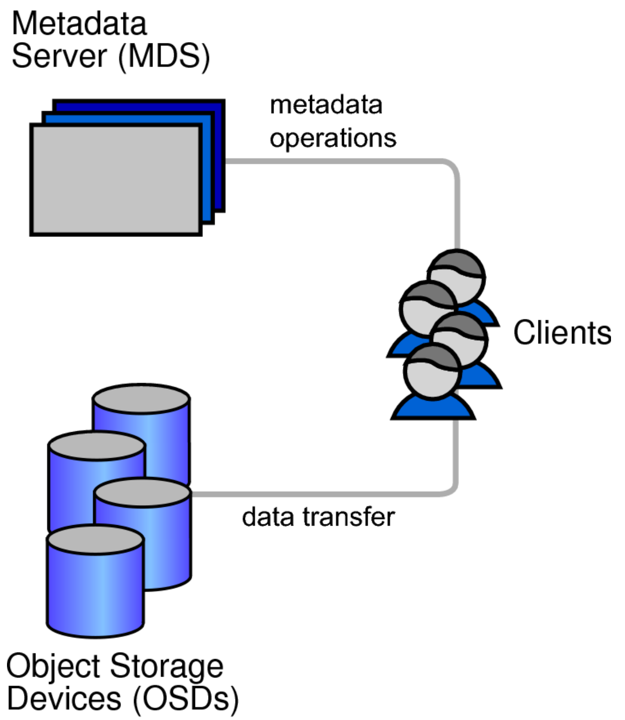
- The “digital universe” is expanding
  - science and industry generate and store huge data volumes
  - large-scale distributed data management gaining in importance
- Data needs to be protected
  - from failures of servers and storage devices,
  - corruption,
  - accidental deletions,
  - virus infections, etc.

Problem Description

- Backups provide for data safety
  - roll-backs and recovery of previous versions
- Typical backup approach:
  - take **snapshot**
  - copy snapshot to backup device
- … but snapshots need to capture all data in a **consistent** state at a **certain point in time**!
  - despite data being physically distributed
  - despite data being concurrently modified
  - despite lack of a global time

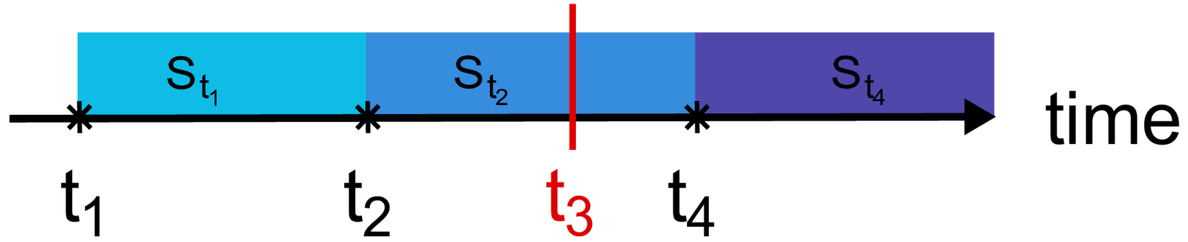
System Architecture

- Object-based storage
  - widely-used design pattern for parallel and distributed file systems
  - metadata servers + intelligent object storage devices
  - file content split into objects
  - easy to scale out by adding new servers
- Object-based file systems
  - examples: Lustre, Panasas Active Scale



File System Snapshots

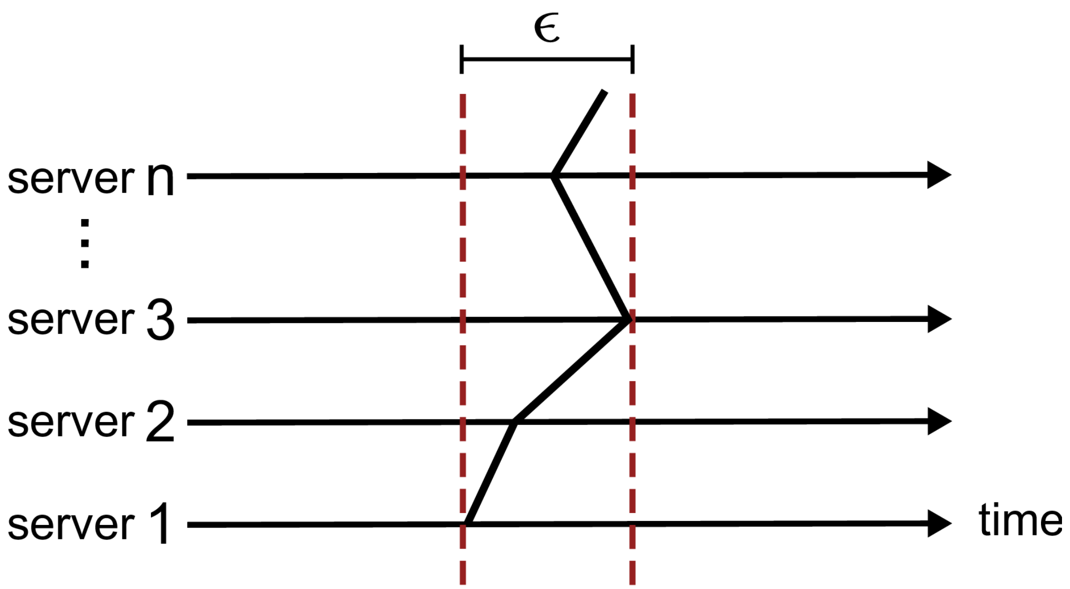
- Stable image of the file system at a given point in time
  - state: all files and directories (data + metadata)
  - **latest** state **before** the point in time



  - **immutable**, regardless of future changes

Algorithm: Assumptions

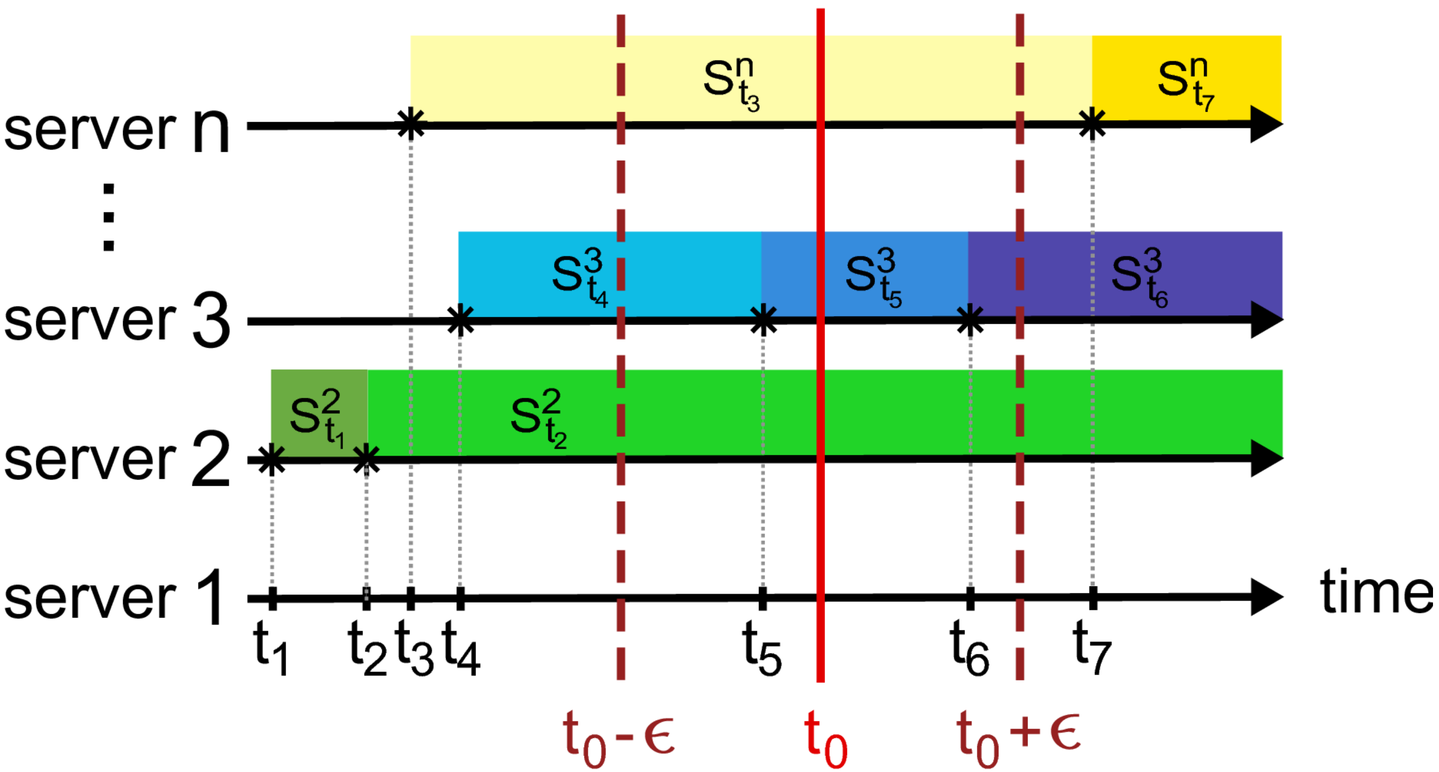
- Servers clocks “loosely” synchronized
  - ϵ bounds clock drift across all servers



  - enforced with NTP or GPS

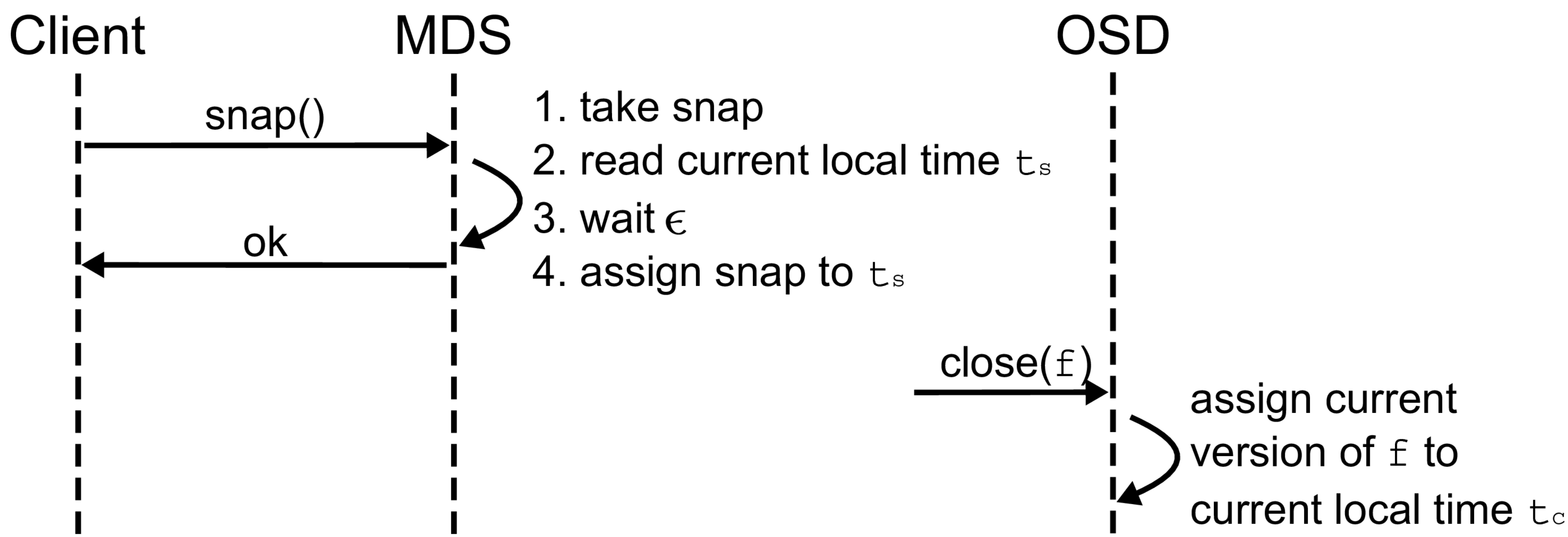
Algorithm: Loose Time Synchrony

- *"Loose time synchrony"*
  - relaxes *point-in-time* guarantees to *time span* guarantees



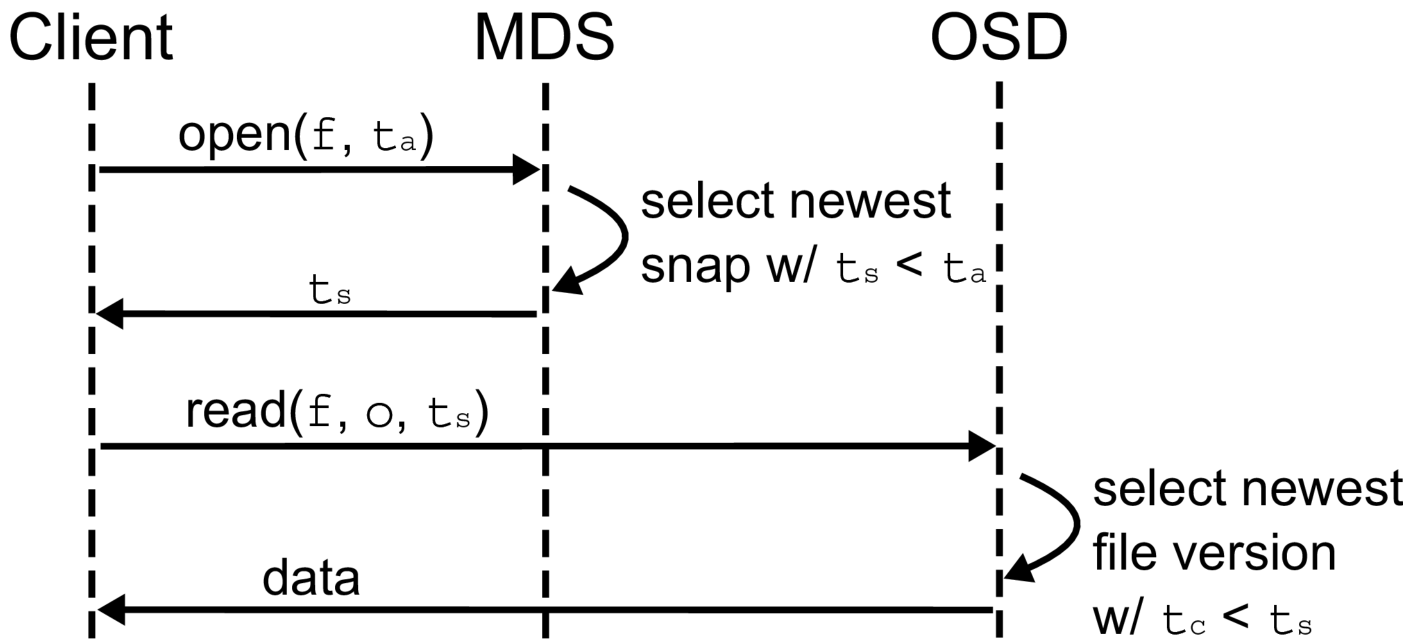
Algorithm: Taking a Snapshot

- Servers take local snapshots
  - MDS: at volume granularity, in response to snapshot requests
  - OSD: at file granularity, in response to `close` events



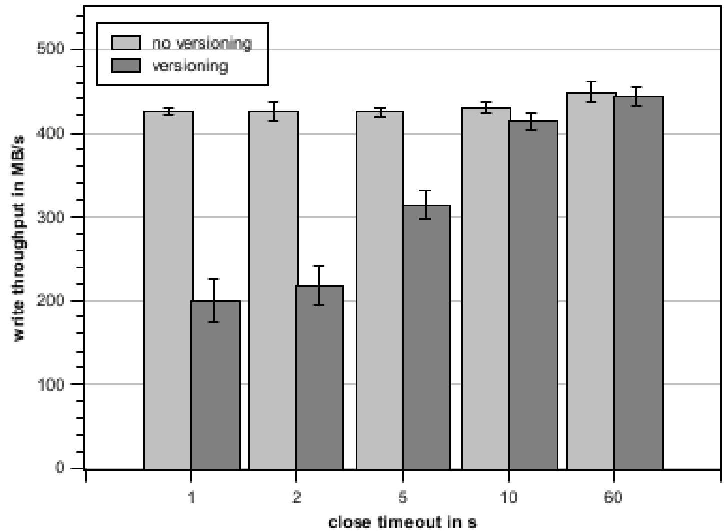
Algorithm: Accessing a Snapshot

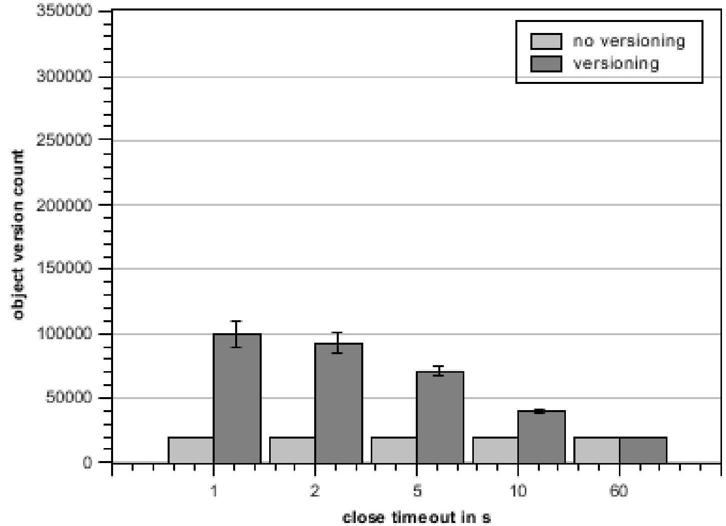
- Accessing a snapshot:



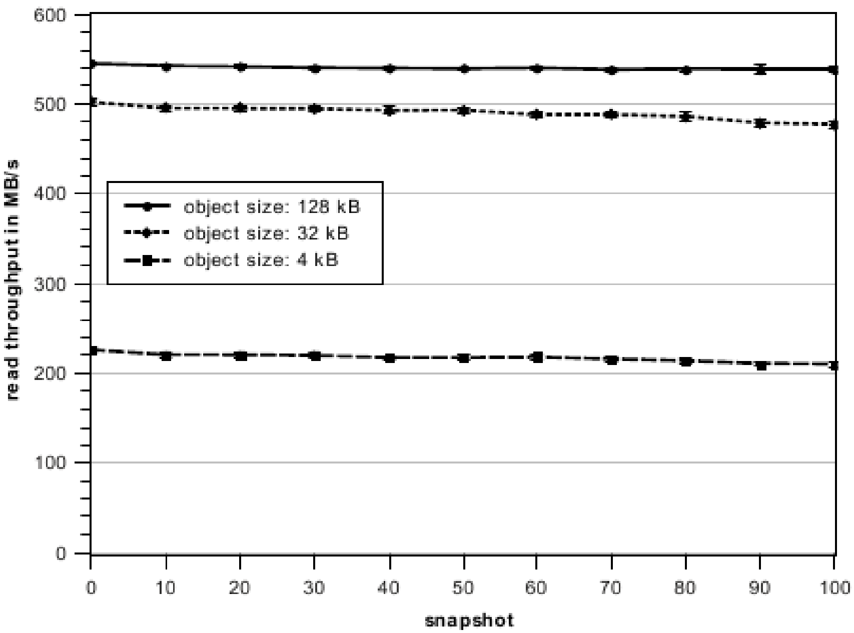
Evaluation

write (128k objects)



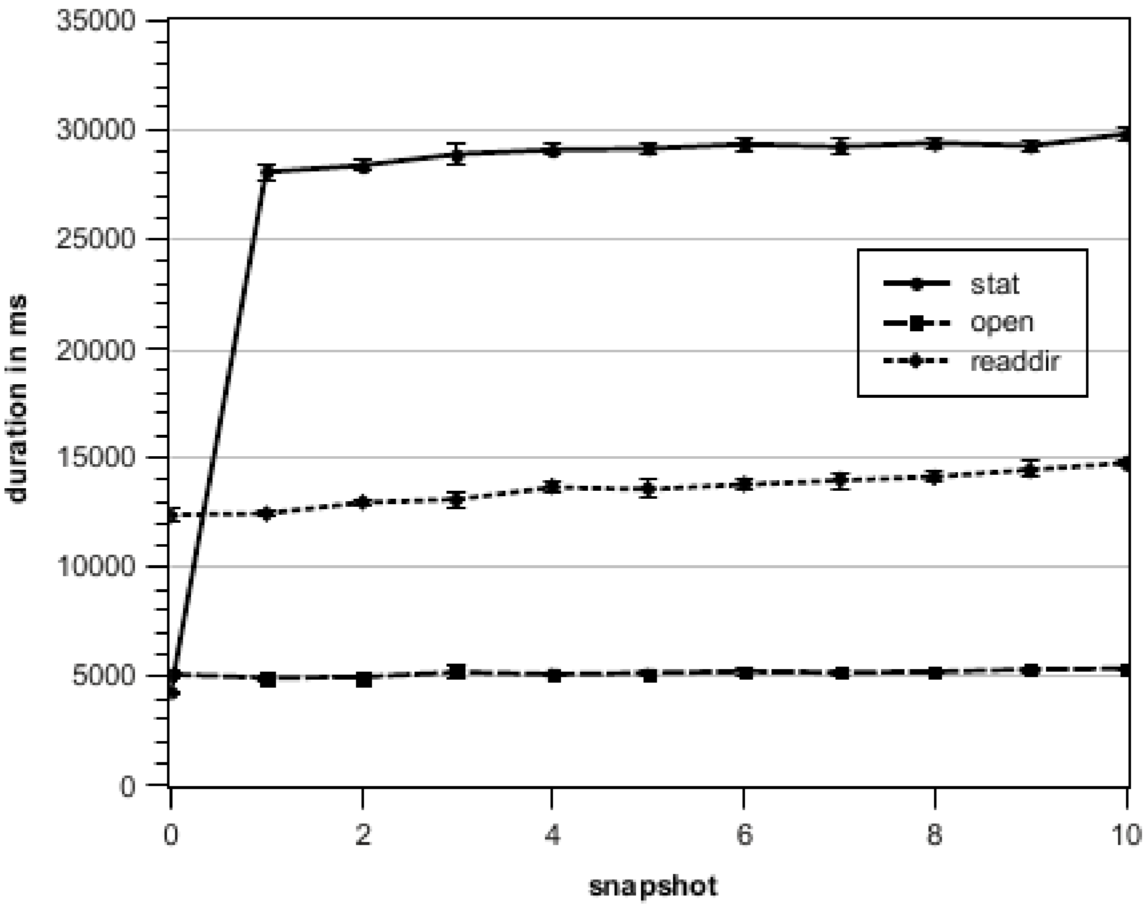


read



Evaluation (2)

metadata



Conclusion

- Snapshots
  - can be accessed ϵ after creation (milliseconds and less)
  - can be taken on-line, w/o disrupting normal file system usage
  - do not require dedicated communication
  - offer unlimited scalability wrt. the number of servers
  - are only partially affected when single servers fail

Thank You!

Questions?

This work was supported by the EU IST program as part of the **XtreemOS** project (contract FP6-033576) and the **Contrail** project (contract FP7-257438).

Implementation (1)

- Metadata versioning:
  - point-in-time snapshots at DB level
  - FS snapshot request triggers MD snapshot
- File content versioning:
  - copy-on-write (COW)
  - object versioning
  - new object versions
    - generated with `write` requests
    - only if object hasn't been written yet since file was opened
  - new file versions
    - generated with `close` requests

Implementation (2)

- Implemented in XtreemFS
  - MDS: BabuDB for metadata snapshots
  - OSDs: COW support
- Clock synchrony
  - NTP, GPS; default: simple NTP-like protocol
- File size consistency
  - “OSD glimpse” to determine correct size of a snapshotted file
- Cleanup
  - many file versions are superseded by later versions
  - cleanup tool removes obsolete object versions