UpRight Cluster Services

Karol Ruszczyk
kr248234

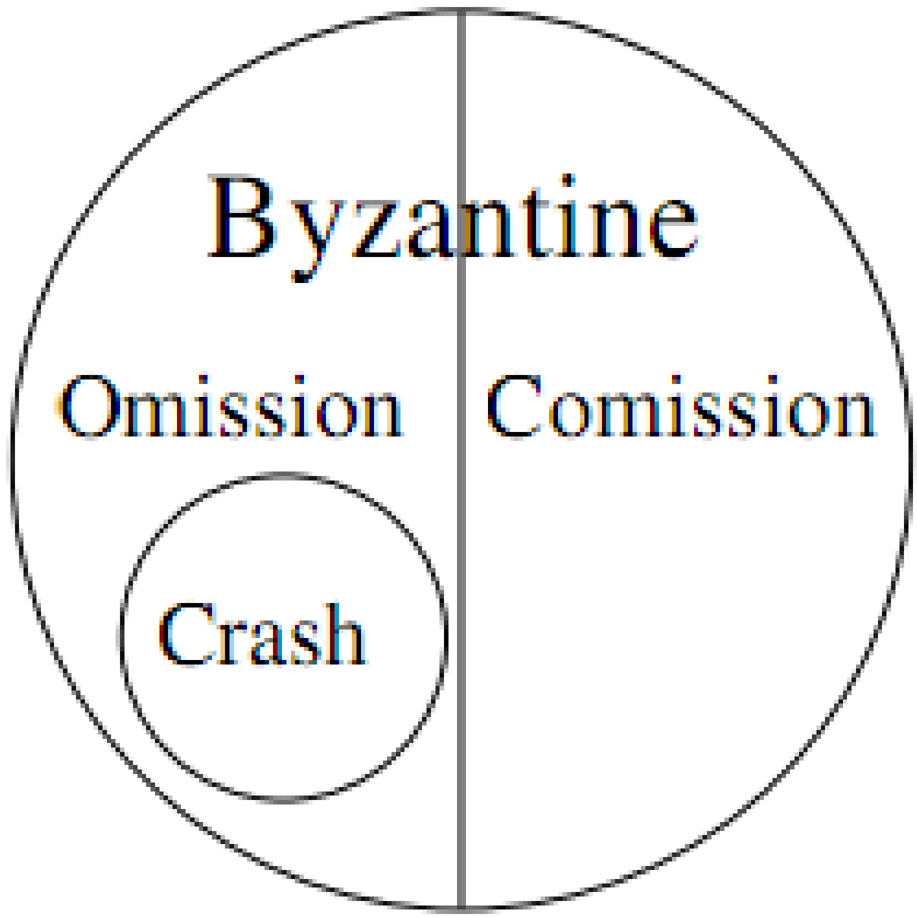
Agenda

- What Byzantine failures are?
- World before UpRight
- UpRight model
- UpRight architecture
- Challenges
  - and possible solutions

UpRight objective

- Make Byzantine fault tolerance (BFT) something that practitioners can easily adopt
  - to safeguard availability (keeping systems **up**)
  - to safeguard correctness (keeping systems **right**)

Byzantine fault – what is it?



Failure hierarchy

Observations

- Practitioners pay non-trivial costs to tolerate crash failures
  - offline backup
  - on-line redundancy
  - Paxos
- Non-crash failures occur with some regularity and can have significant consequence
  - but still deployment of BFT replication remains rare

BFT vs CFT

- practitioners to see BFT as a viable option must be able to use it at low incremental cost
  - compared to the CFT systems they use now

- BFT systems must be competitive with CFT systems in terms of:
  - performance
  - hardware overhead
  - availability
  - **engineering effort**

How is it done now?

- performance, hardware overheads, availability – *DONE*

- engineering effort
  - current state of the art often requires rewriting applications **from scratch**
    - if the cost of BFT is „rewrite your cluster file system" then widespread adoption will not happen

UpRight objectives – part 2

- UpRight design choices
  - favor minimizing intrusiveness to existing applications
  - … over raw performance
  - but try to not loose to much

Model

Model

- Client-Server architecture
- Standard assumptions
  - some faulty nodes (servers or clients) may behave arbitrarily
  - we assume a strong adversary that can coordinate faulty nodes
    - we do, however, assume the adversary cannot break cryptographic techniques
      - collision-resistant hashes
      - encryption
      - signatures

Model

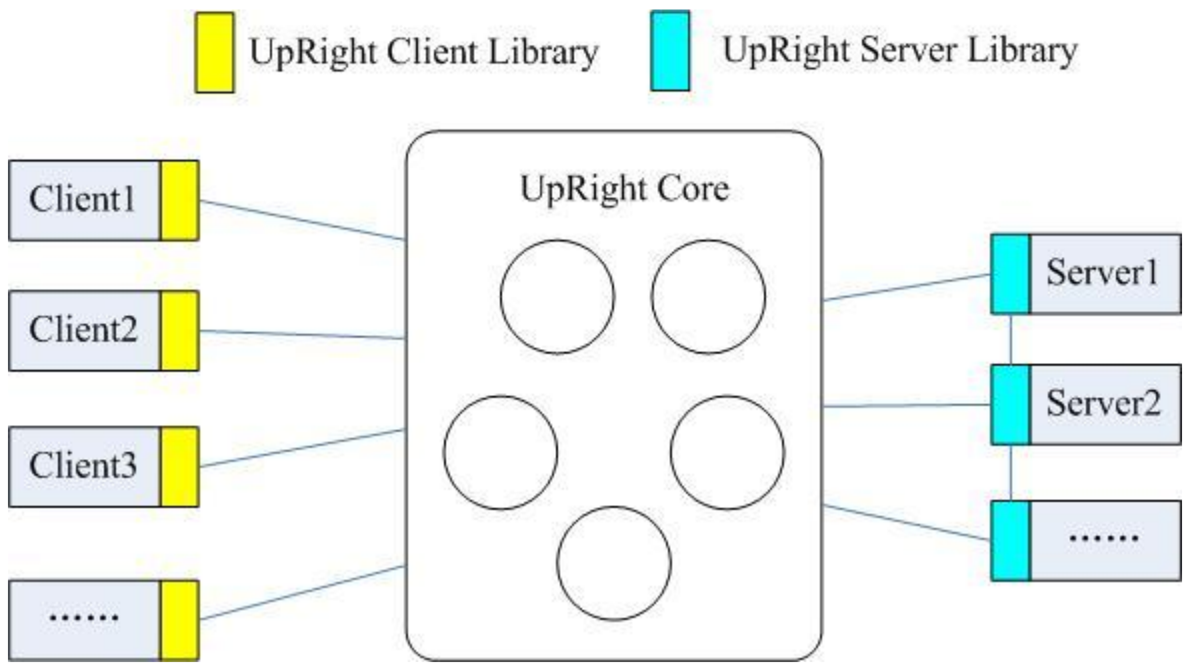
- Tweaks
  - Number of failing nodes
    - u – overall number of failing nodes
    - r – number of nodes failing by commission
  - Crash-recover incidents
    - Formally nodes that crash and recover count as suffering an omission failure during the interval they are crashed and count as correct after they recover
    - Crash/recover nodes are often modelled as correct, but temporarily slow
  - Robust performance
    - „Eventually the system makes progress”

Architecture

UpRight

- implements state machine replication
- client-server architecture
- tries to isolate applications from the details of the replication protocol
  - easy to convert a CFT application into a BFT

Architecture



What UpRight ensures?

- each application server replica sees the same sequence of requests and maintains consistent state
- an application client sees responses consistent with this sequence and state

Challenges

Challenges – Request execution

- Nondeterminism
  - many applications rely on real time or random numbers as part of normal operation
- Multithreading
  - The simplest way: complete execution of request *i* before beginning execution of request *i+1*.
- Spontaneous replies
  - unreliable channels for push events

Challenges – Checkpoints

- Even correct server replicas can fall behind
  - frameworks must provide a way to checkpoint a server replica's state
  - to certify that a quorum of server replicas have produced identical checkpoints
  - to transfer a certified checkpoint to a node that has fallen behind

Challenges – Checkpoints

- Server application checkpoints must be
  - inexpensive to generate
    - checkpoint frequency is relatively high
  - inexpensive to apply
  - deterministic
  - nonintrusive on the codebase

Implemented strategies

- Hybrid checkpoint/delta approach
- Stop and copy
- Helper process
- Copy on write

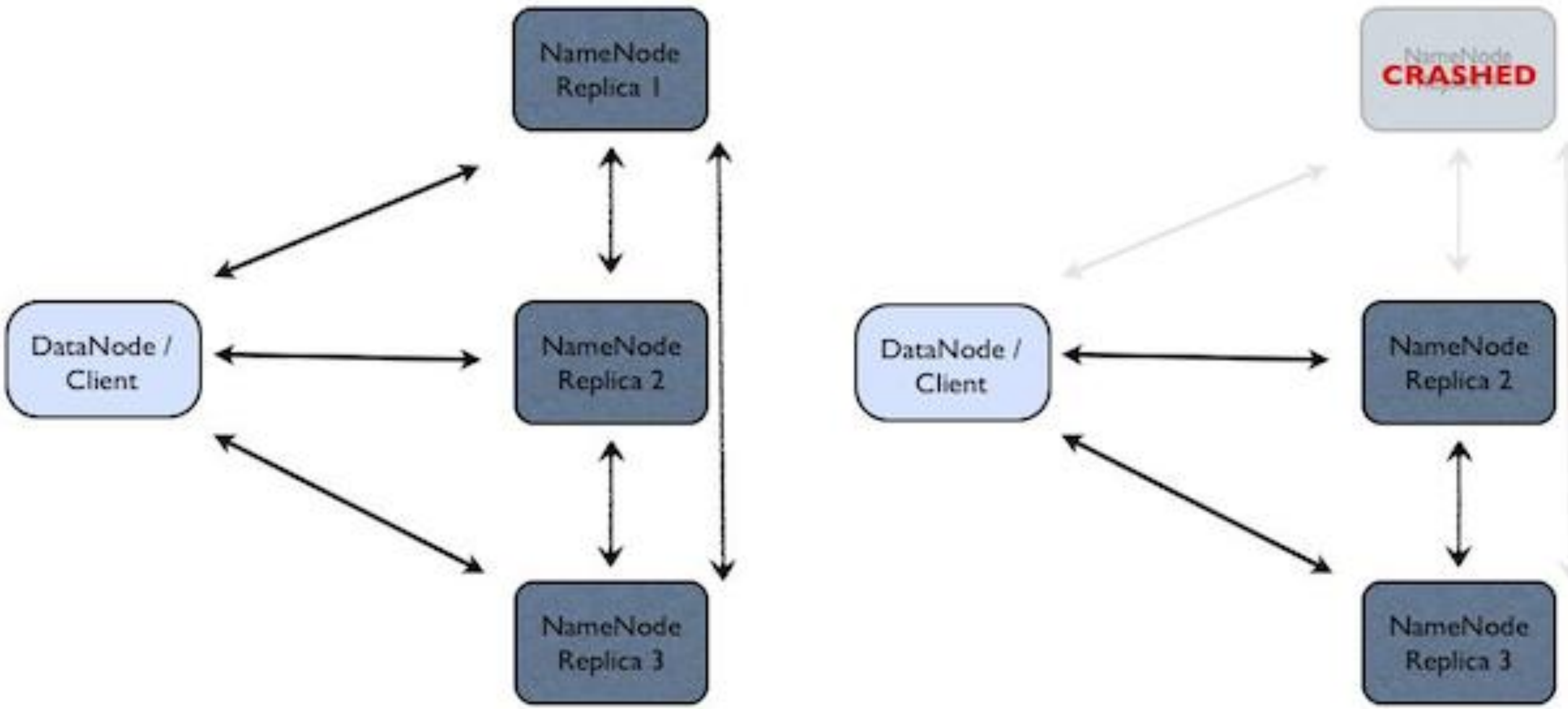
Conclusions

- The purpose of the UpRight library is to make Byzantine fault tolerance (BFT) a viable addition to crash fault tolerance (CFT)
- If a designer has an existing CFT service
  - UpRight can provide an easy way to also tolerate Byzantine faults
- If a designer is building a new service
  - UpRight library makes it easy to provide BFT
    - which can be turned off anytime if not needed ($r = 0$)

Appendix A

- HDFS-UpRight

Idea


NameNode
Replica 1
DataNode /
Client
NameNode
Replica 2
NameNode
Replica 3
CRASHED
DataNode /
Client
NameNode
Replica 2
NameNode
Replica 3

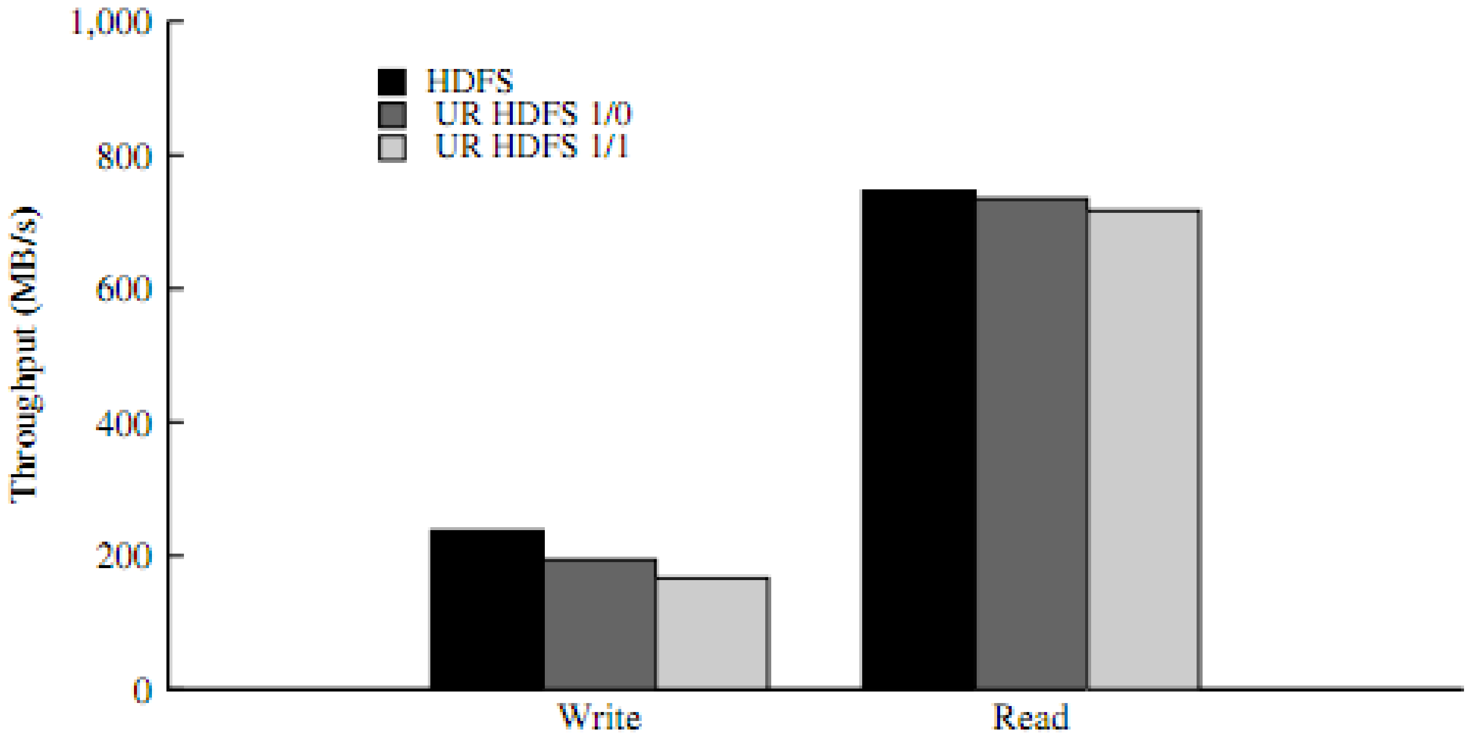


Figure 13: Throughput for HDFS and UpRight-HDFS.

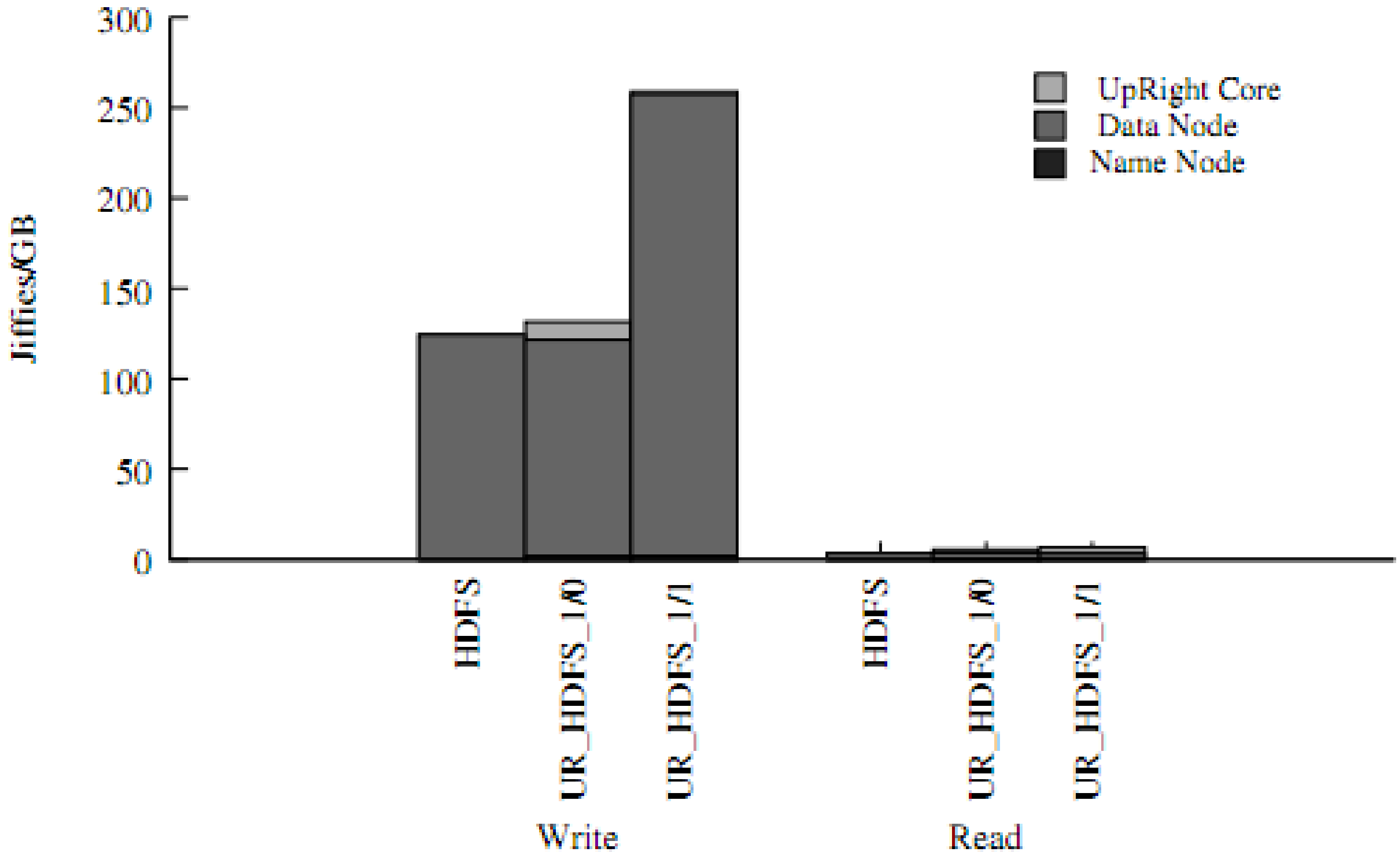
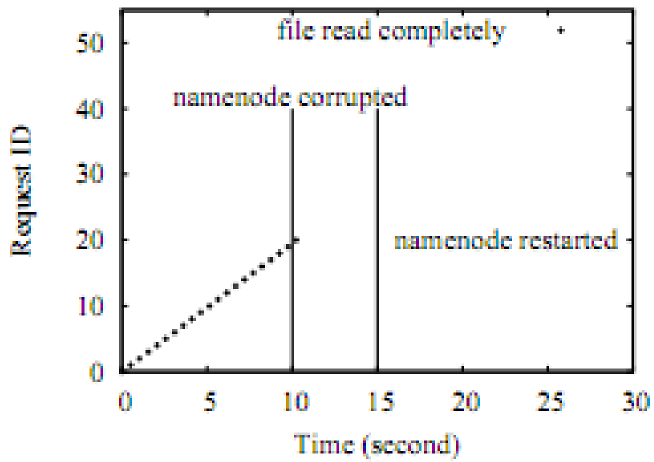
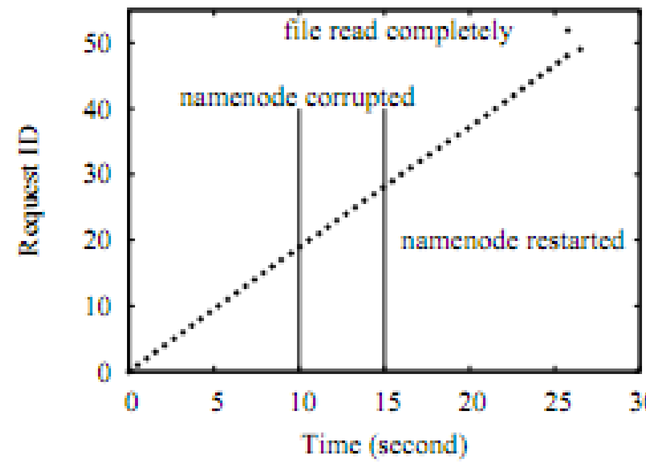


Figure 14: CPU consumption (jiffies per GB of data read or written) for HDFS and UpRight-HDFS.



(a)



(b)

Figure 15: Completion time for requests issued by a single client. In (a), the HDFS NameNode fails and is unable to recover. In (b), a single UpRight-HDFS NameNode fails, and the system continues correctly.