Chain Replication for Supporting High Throughput and Availability

Robbert van Renesse, Fred B. Schneider

OSDI 2004

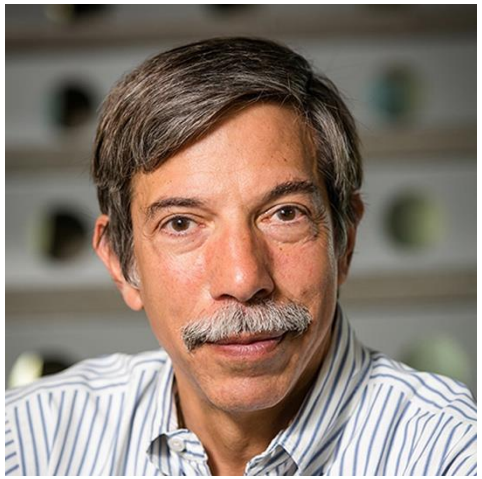
Presented by: Austin Li

10/10/2024

Authors

- Robbert van Renesse

- Fred B. Schneider



Background

- Large scale storage systems *ideally* should be:
  - High availability
  - High throughput
  - Strong consistency

Background

- Why high availability/throughput?
  - AWS, Google, Microsoft, etc. need to provide service at scale
- Why strong consistency?
  - Operations execute in a sequential order
  - Effects of updates are reflected in subsequent queries
  - Easier to reason about and program on

Background

- Google File System (2003), AWS Dynamo (2007)
  - Sacrifice strong consistency

Chain Replication

- Implement SMR
- High availability/throughput
  - Sacrificed in case of network partition
- Strong consistency

Storage Service Interface

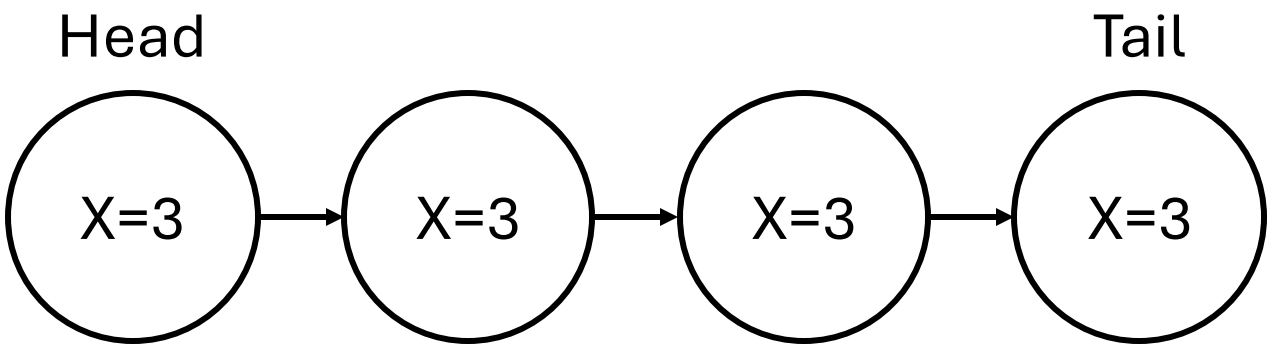
- Object store
- Two request types:
  - `query(objId)`
  - `update(objId, newVal)`

Assumptions

- Servers are fail-stop
- Server failures are detectable
- Reliable FIFO links between servers

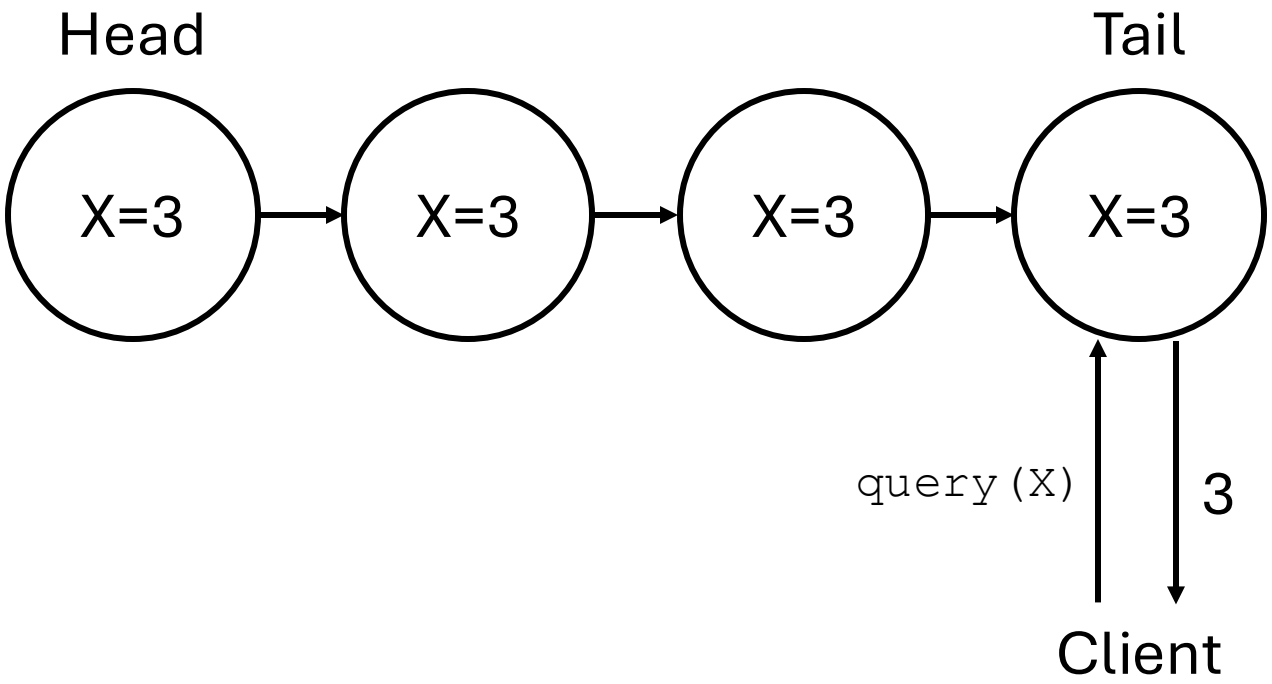
Chain Replication Protocol

`query(X)`



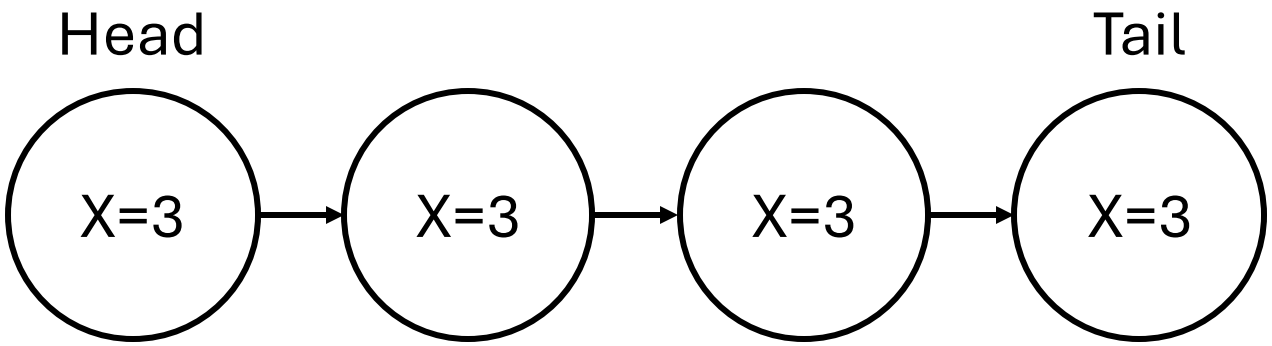
Chain Replication Protocol

`query(X)`



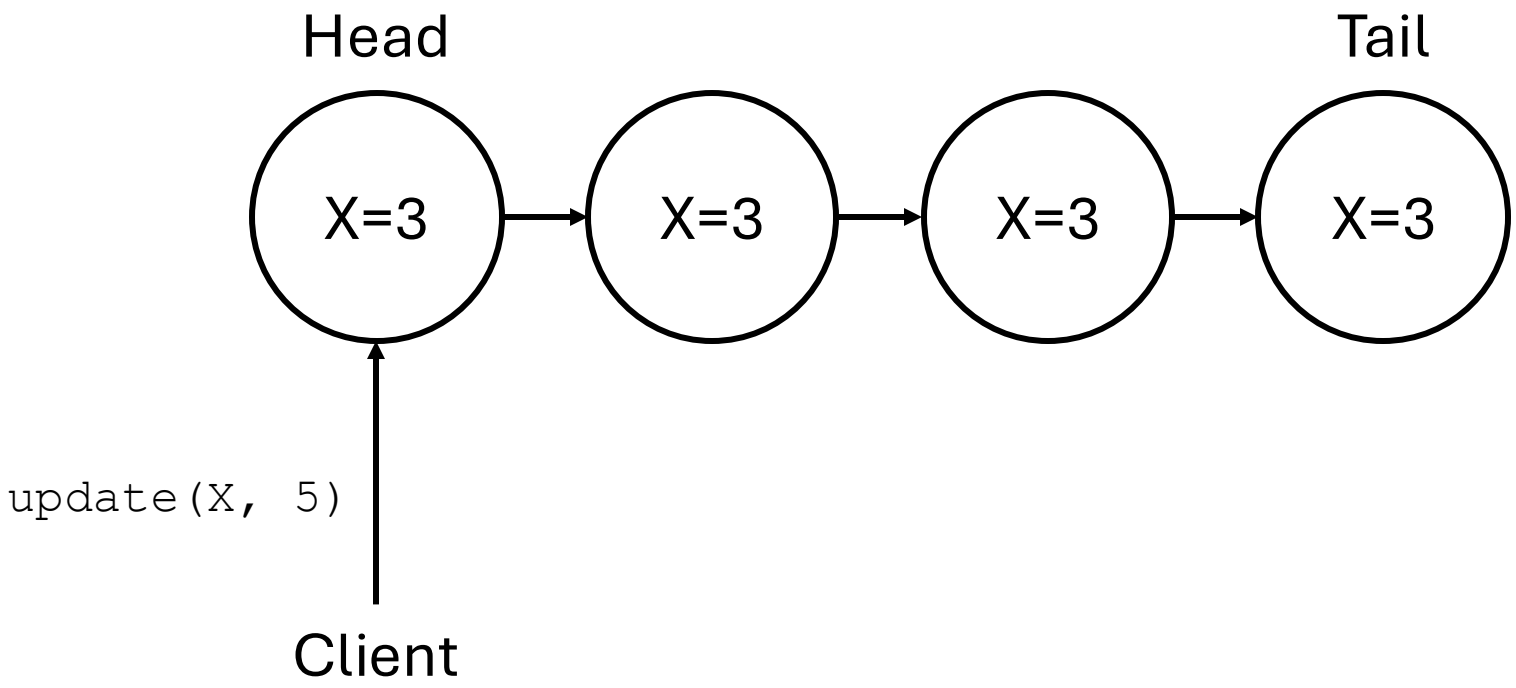
Chain Replication Protocol

```
update(X, 5)
```



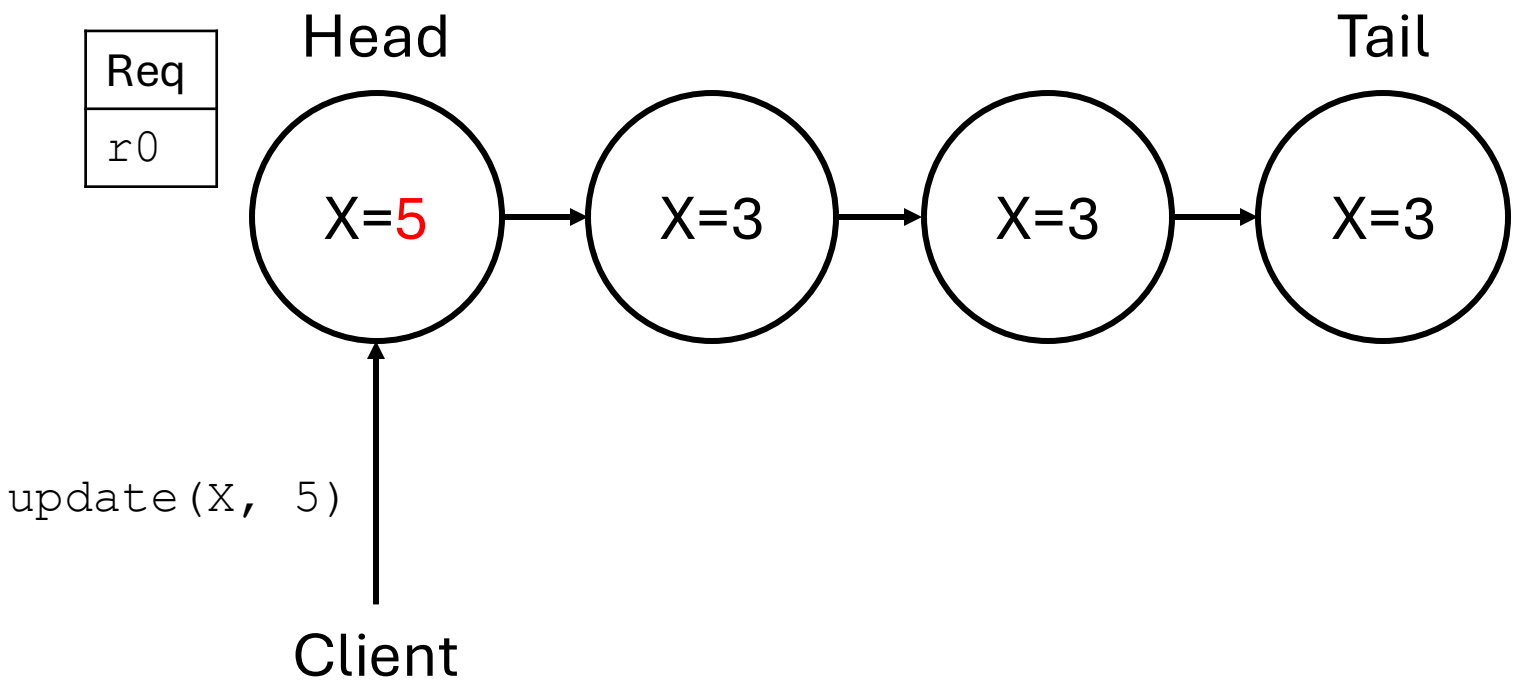
Chain Replication Protocol

`update(X, 5)`



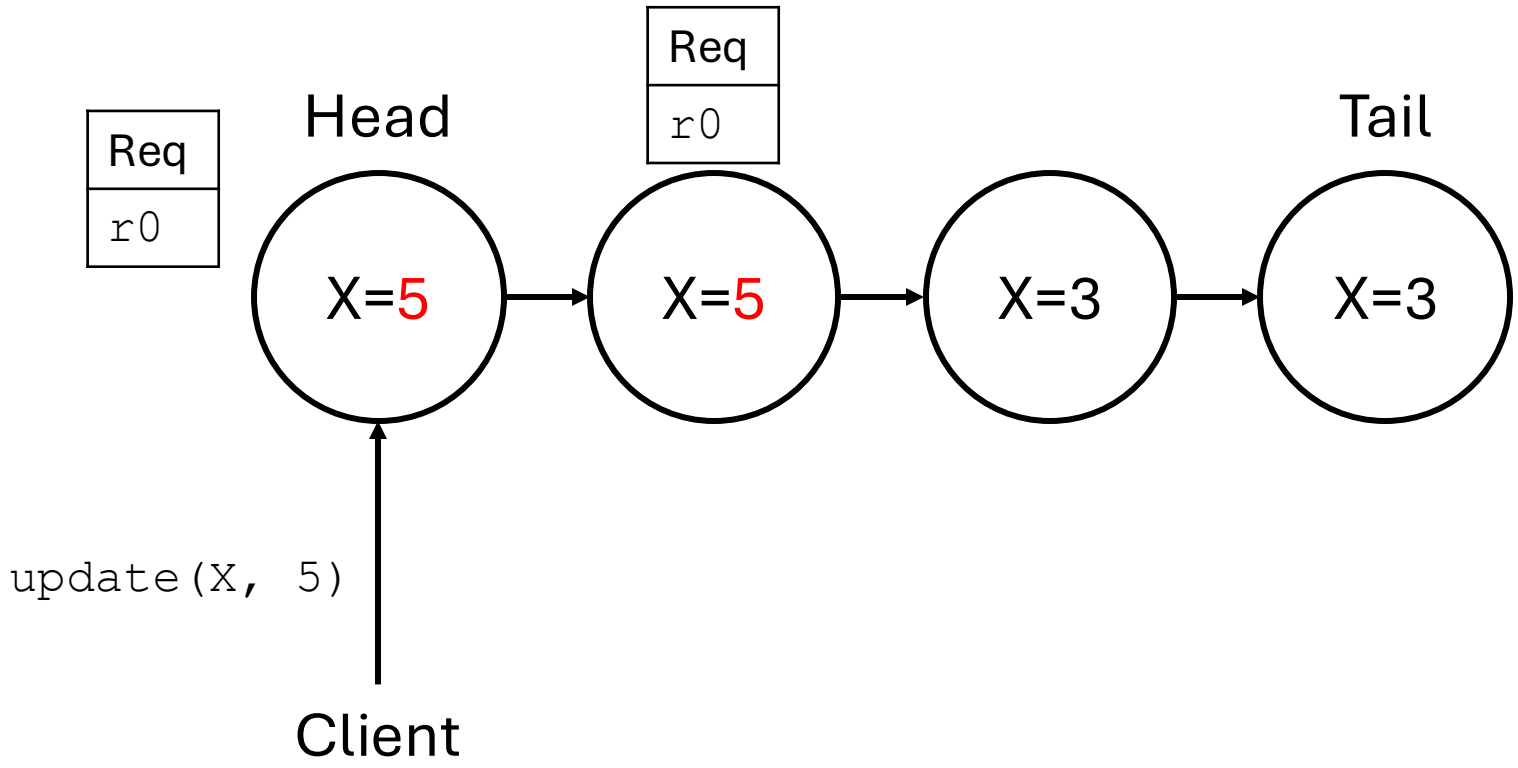
Chain Replication Protocol

`update(X, 5)`



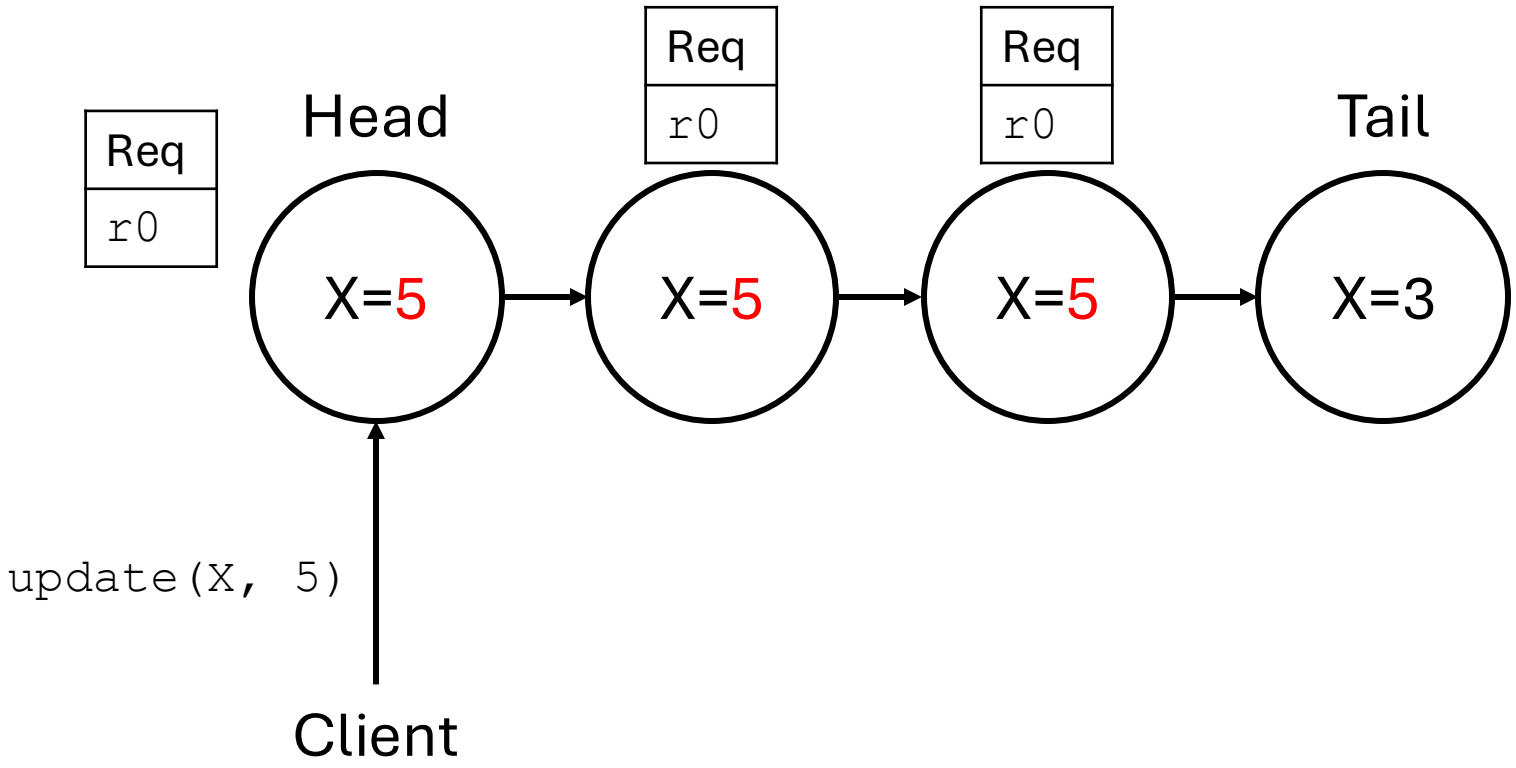
Chain Replication Protocol

`update(X, 5)`



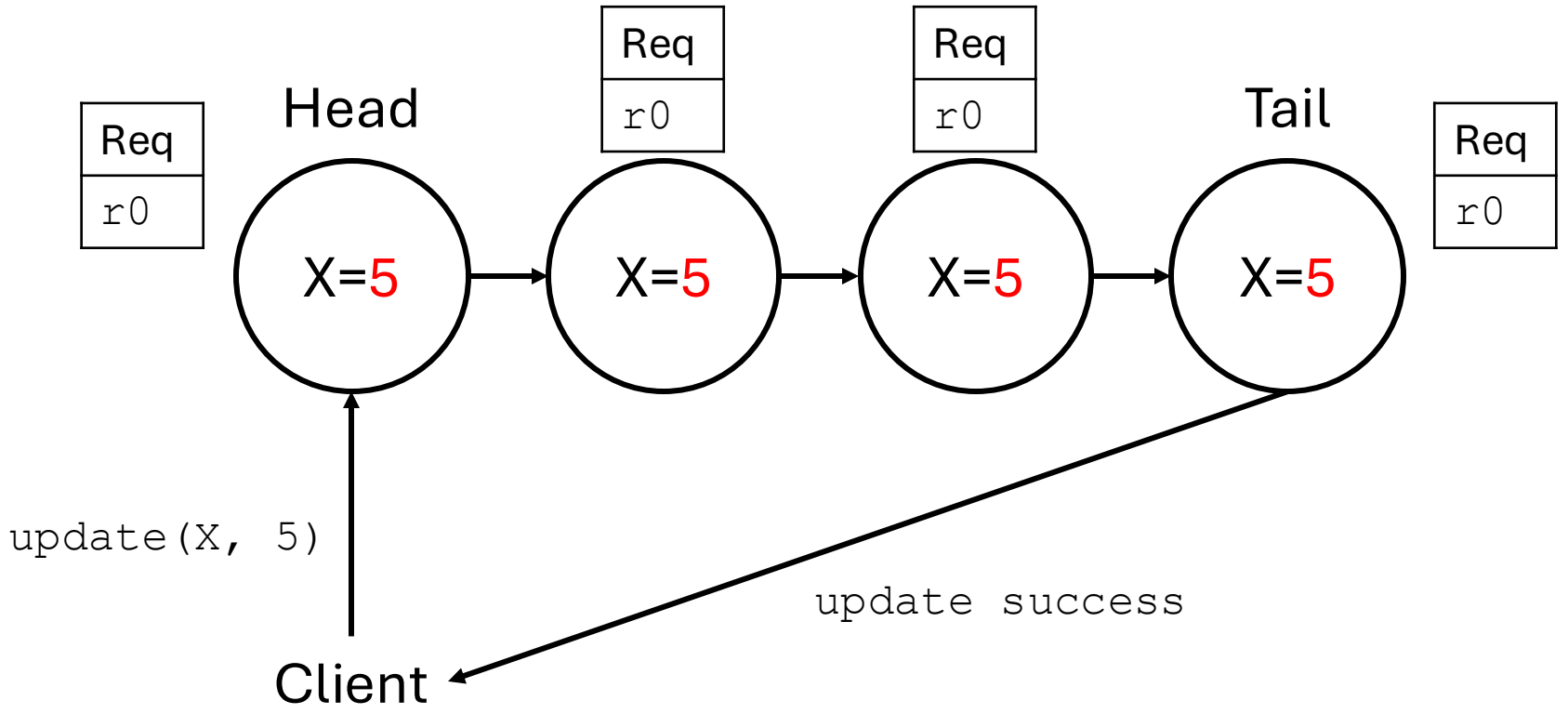
Chain Replication Protocol

`update(X, 5)`



Chain Replication Protocol

`update(X, 5)`



Discussion

- What are some advantages/disadvantages of the protocol?

Strong Consistency

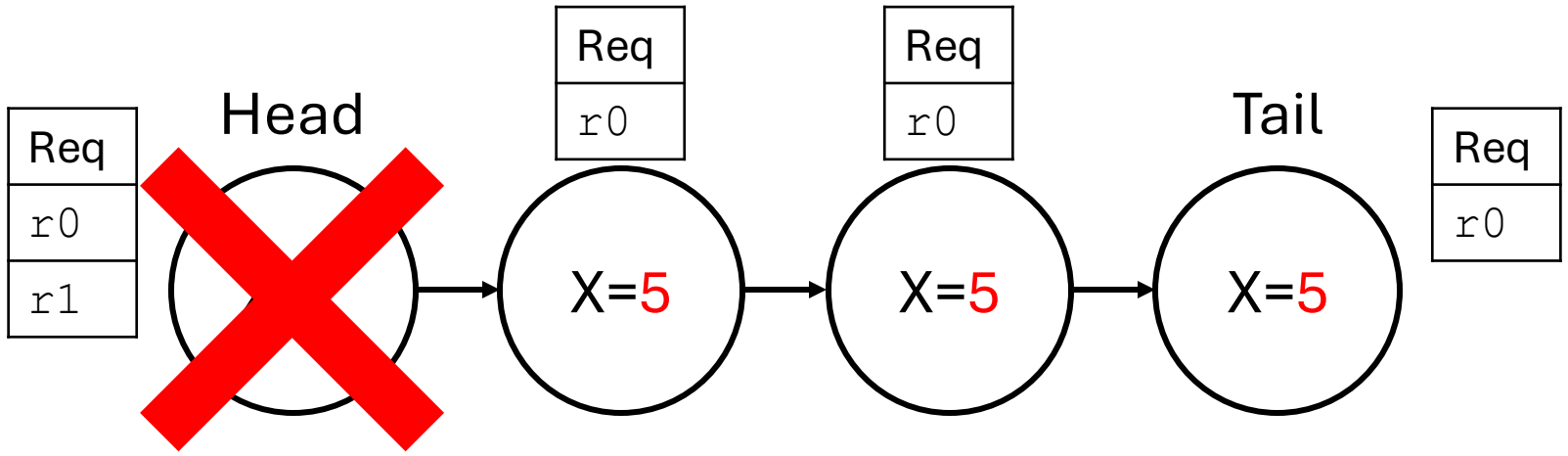
- Comes naturally from the chain design
- Queries and updates all occur sequentially at the tail

Fault Tolerance

- In a chain of length t, can tolerate at most t-1 failures
- Master service
  - Detects failures
  - Informs chain servers of failures and corresponding updates
  - Informs client of who is head/tail
  - Never fails (replicated via Paxos)

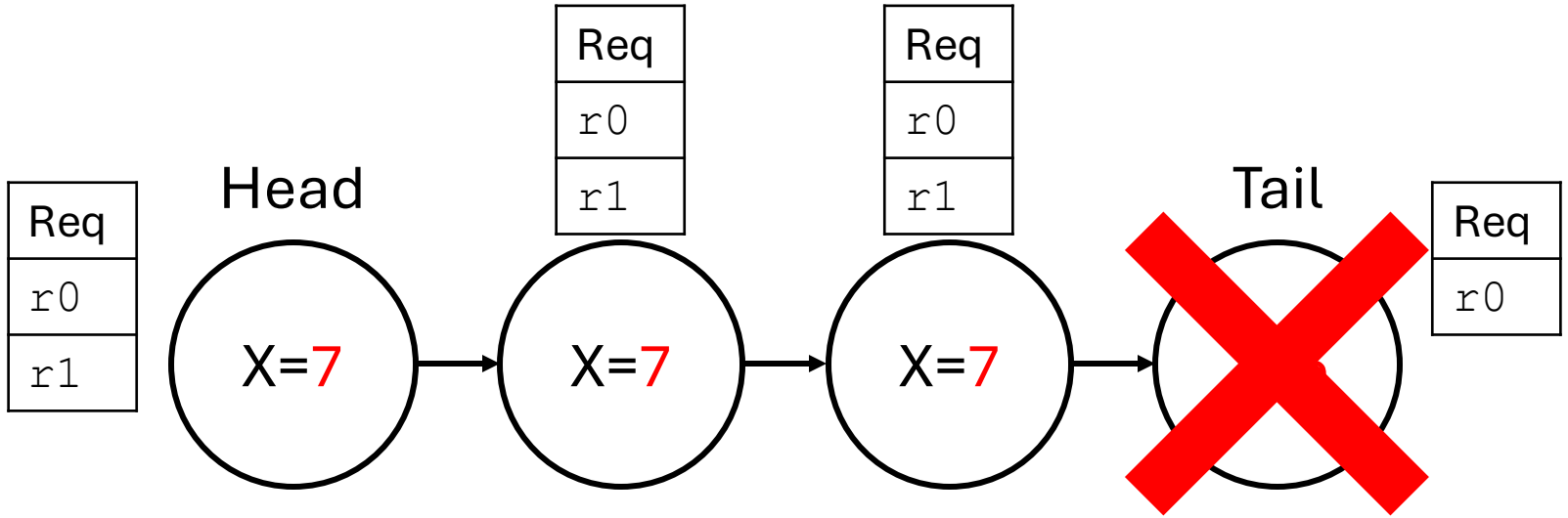
Fault Tolerance

- Head failure
  - Second server is new head
  - Any requests at old head not forwarded yet are dropped



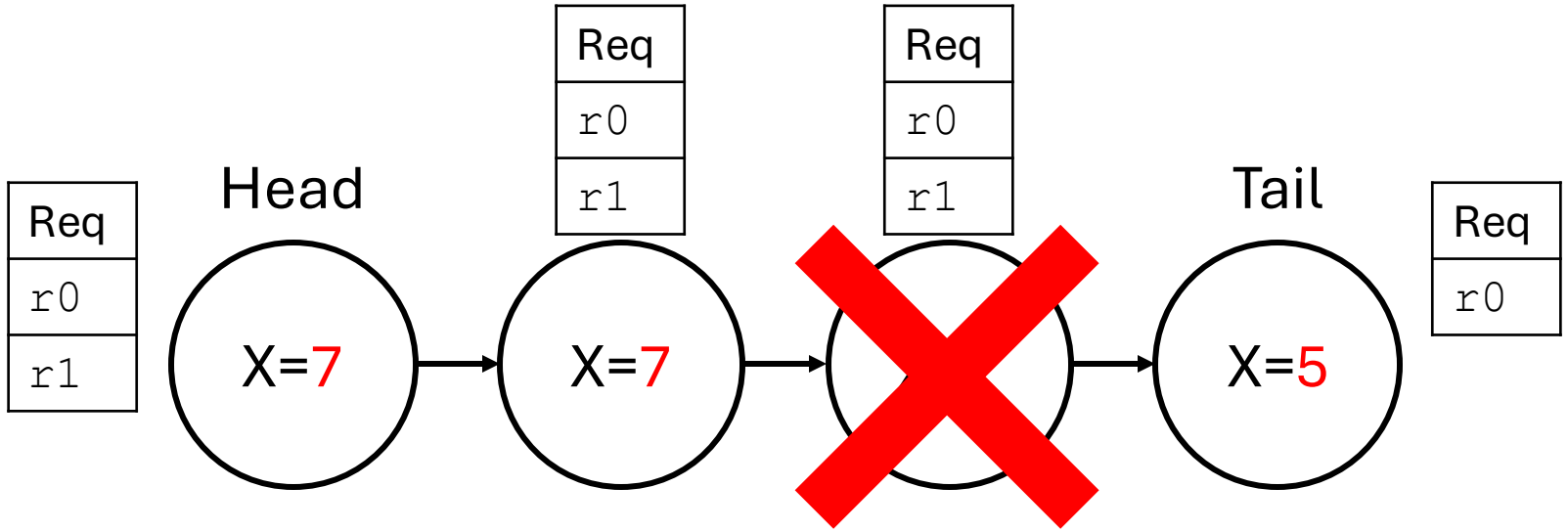
Fault Tolerance

- Tail failure
  - Predecessor of tail is new tail
  - History of updates at predecessor is superset of old tail



Fault Tolerance

- Middle server failure
  - Link around failed server in chain
  - Make sure that any updates that failed server hasn't forwarded get forwarded



Extending the Chain

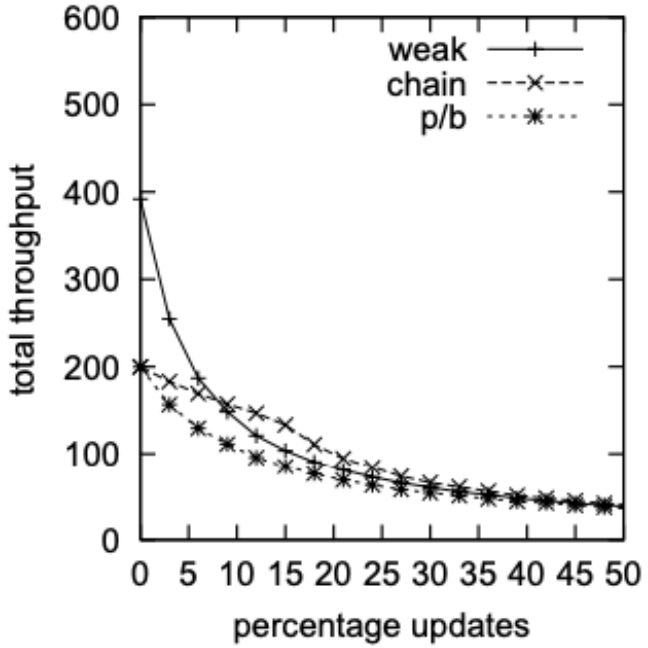
- Add new server to end of chain (new tail)
- Make sure all old tail updates are forwarded before acting as new tail

Evaluation

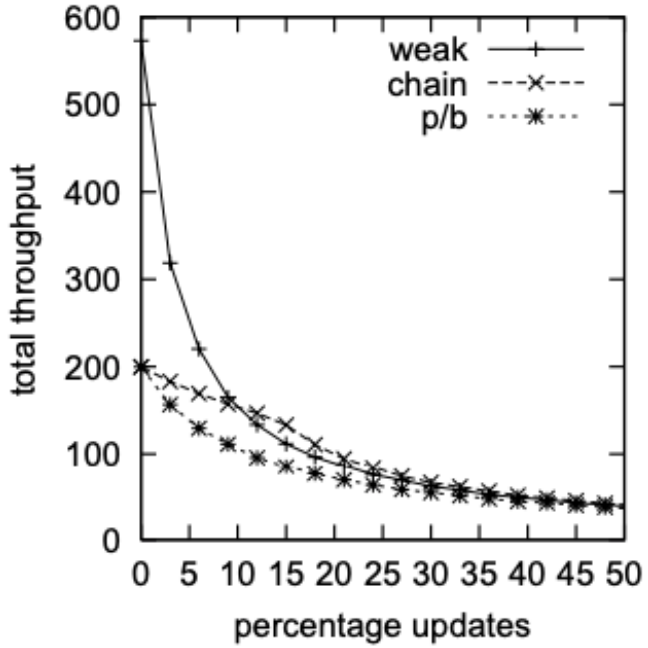
- Compare Chain Replication to primary-backup, weak consistency protocol
- Vary replication factor *t*, measure throughput

Evaluation

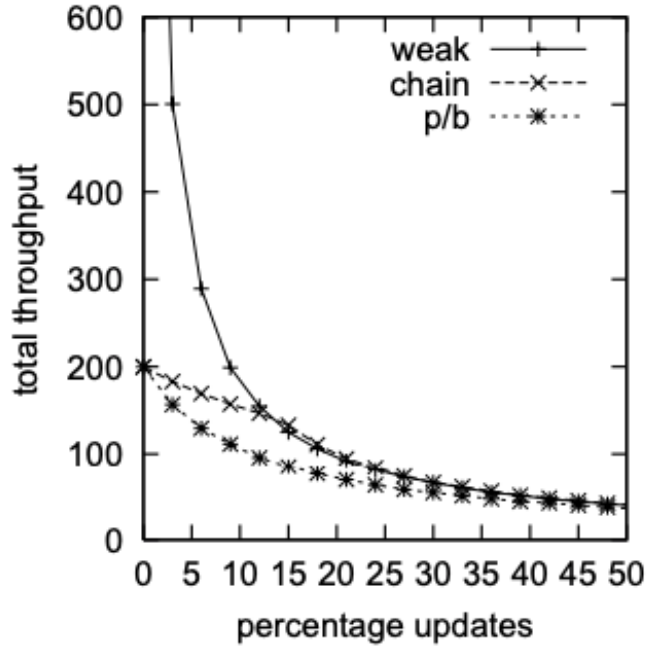
- Chain Replication performs as well or better than primary-backup



(a) $t = 2$



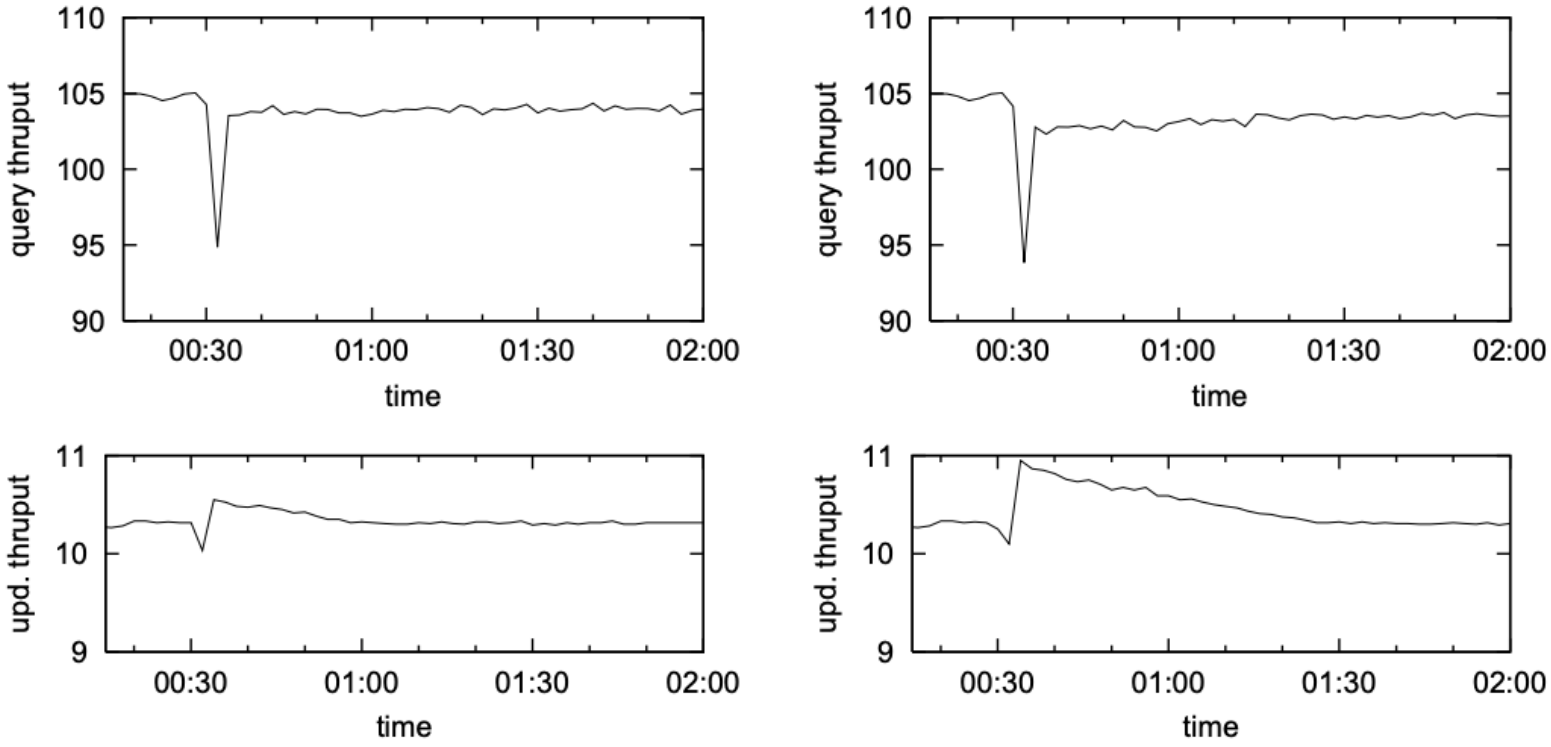
(b) $t = 3$



(c) $t = 10$

Evaluation

- Measure impact of server failures on throughput



(a) one failure

(b) two failures

Figure 6: Query and update throughput with one or two failures at time 00:30.

Discussion

- Did people find the evaluation section convincing?
- Takeaways from Chain Replication?

Conclusion

- Chain Replication is simple yet effective!