Consistent and Efficient Clouds

Robbert van Renesse

Cornell University

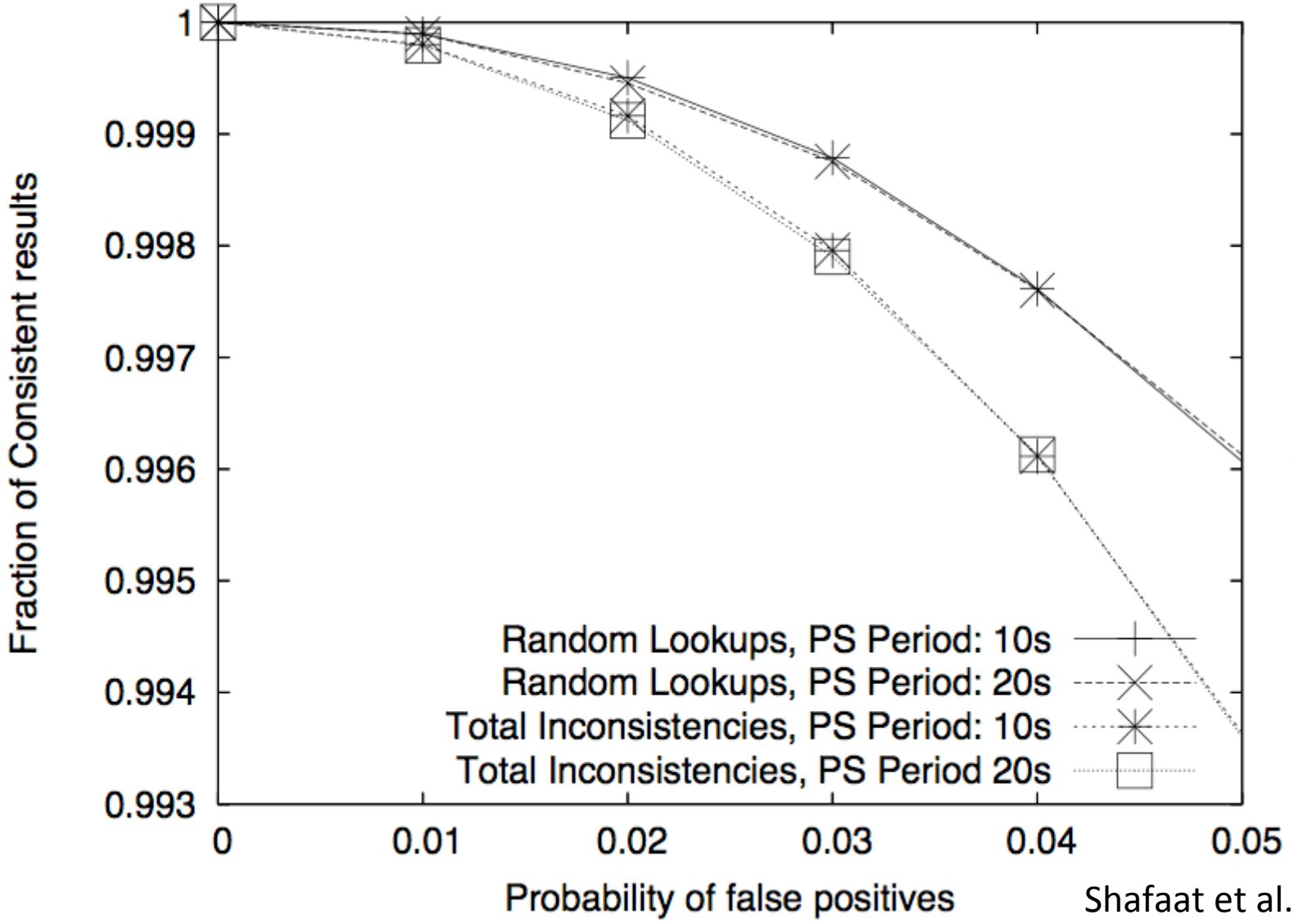
Consistent Clouds?

- If Amazon sends you two copies of the same book, no big deal
- But what if your cloud-enabled insulin delivery device gives you a double dose of insulin?
- Cloud increasingly used to back up small portable compute devices
  - health care, military, farming ("smart cows"), etc.
- Many of these applications will need strong consistency

Building Blocks of Clouds

- DHTs, Dynamo, Memcached, GFS, HDFS, …
  - none are designed for strong consistency
- Chubby, ZooKeeper
  - maintain consistency of configurations only
    - i.e., they are intended for availability, not consistency
- MapReduce
  - consistent but no updates/inserts during processing

Inconsistencies in Chord



Shafaat et al. 2008

What about failures?

- Crash failures vs arbitrary or Byzantine failures
- 8.5% of SATA disks develop silent data errors
  - Bairavasundaram 2007

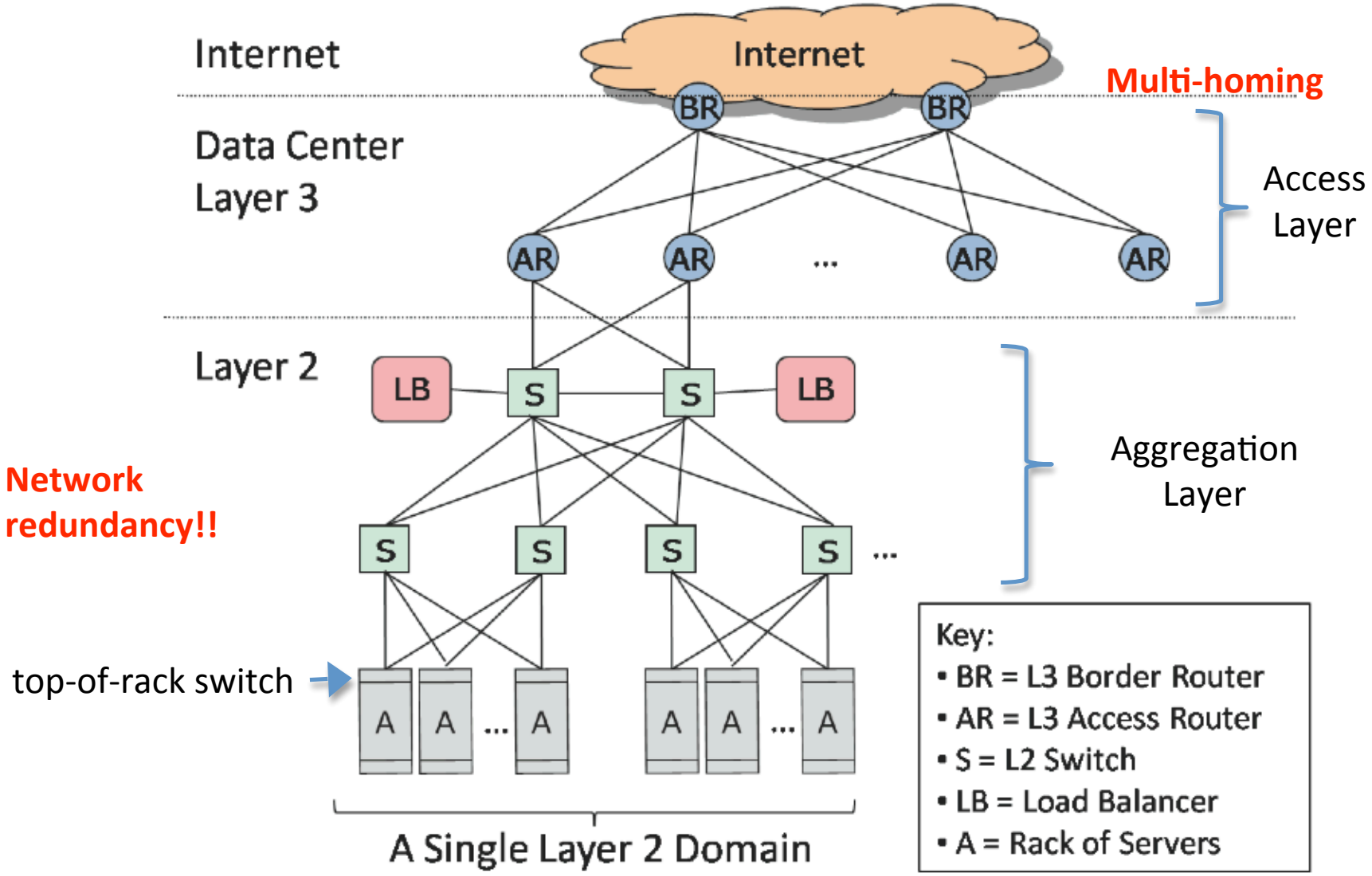
Bug category	DB2 2/03-8/06	Oracle 7/06-11/06	MySQL 8/06-11/06
DBMS crash	**120**	**21**	**60**
Non-crash faults	**131**	**28**	**64**
Incorrect answers	81	24	63
DB corruption	40	4	(inc. above)
Unauth. access	10	unknown	1

Vandiver 2007

Mostly "independent"

But no deployment of Byzantine-tolerant systems in clouds…

Network failures in Datacenters?



Common Misperceptions

- Strong consistency (Byzantine or otherwise) doesn't scale
  - like everything else, partitioning is the key to scaling
- Consensus is impossible
  - only under absence of any timing assumptions
- Consensus allows you to detect failures accurately
  - no, but accurate failure detection allows you to solve consensus
- Byzantine provides stronger consistency than crash-tolerant replication
  - same consistency; crash failures are a different assumption
- Byzantine replication makes weaker assumptions
  - different assumptions, not necessarily weaker
- Byzantine failures must be independent
  - only sort of true, and same applies to crash failures

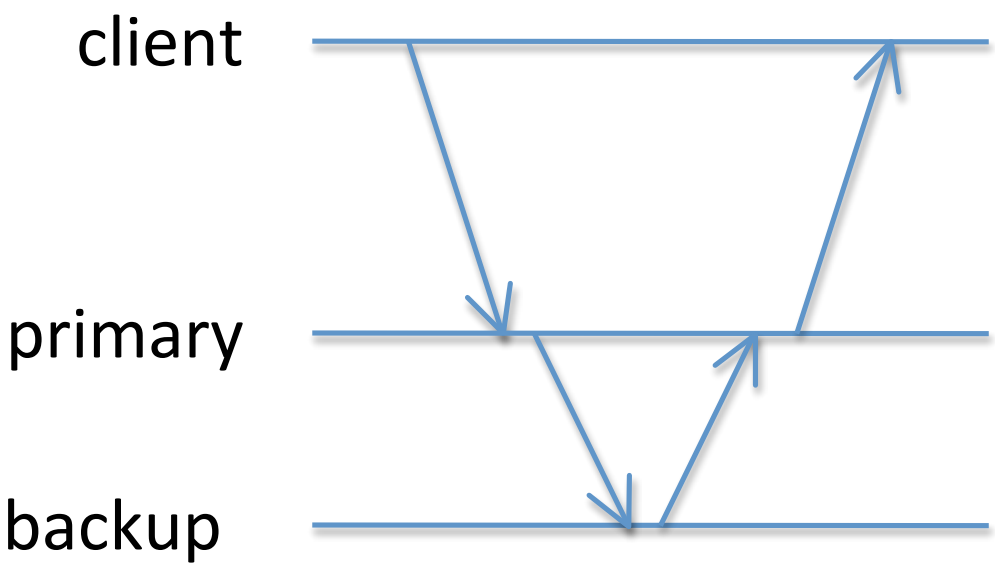
Valid reasons to reject strong consistency

1. Quorum-based protocols are expensive
   - 2f+1 or 3f+1 instead of only f+1 replicas, leads to an increase of acquisition and ownership cost of 1.5x to 3x.
   - Also, as much harder to find sources of diversity
2. In the face of network partitioning, no quorum may be accessible for the duration of the partitioning, leading to a significant outage
   - quite possible between datacenters, as network operators do not provide independently failing network links

This suggests two critical research directions

1. Crash-tolerant and Byzantine-tolerant strongly consistent replication protocols that use only f+1 replicas
2. Internet that provides f+1 independent links between any two mission critical data centers

Byzantine Primary-Backup?



1. Client sends request
2. Primary assigns seq. number and forwards signed operation
3. Backup accepts request if seq# is correct, logs primary's request, and sends signed op. to primary
4. Clients logs backup's operation and sends response
5. Client accepts response if both primary and backup have signed it.

Byzantine Primary-Backup, cont'd

- So:
  - primary has log of completed requests signed by backup and verified by primary
  - backup has log of completed requests signed by primary and verified by backup
  - at least one of the two is correct
- Reconfigure if either becomes unresponsive or is otherwise faulty
  - inspired by Vertical Paxos and work by Birman, Malkhi, and Van Renesse in 2010.

Byzantine Primary-Backup, cont'd

- Three problems
  1. when reconfiguring, may only get the state of one of the replicas. What if this replica is Byzantine and truncates the log?
  2. digital signatures are expensive! Would still need to buy many more machines to handle the same load.
  3. who reconfigures the set of replicas? Doesn't that just push the problem somewhere else?

Much is in the assumptions you make!

- Byzantine ≠ Secure
- Faulty nodes can detect state truncation
  - keep track of hash of state in separate part of the memory --- unlikely that both state and its hash are corrupted to render a correct combination accidentally
- Cheap keyed SHA codes are plenty strong enough to “sign” the requests

Costs can be amortized or distributed

- Use a shared Byzantine configuration service with 3f+1 replicas
- Or organize objects into “rings”, each object responsible for the configuration of its successor on the ring
- Different solutions make different liveness assumptions

ACID?

- Described approach guarantees linearizability, and linearizability composes.
- Still weaker than serializable ACID semantics.
  - and many health care, military, etc. apps will involve transactions involving multiple objects.
- Luckily, transactional components, incl. 2PC transaction manager can be built as Byzantine-tolerant components.

Not covered today

- Efficient Byzantine-tolerant stream processing
- Wide-area concerns
  - providing f+1 independent links
    - good subject for FIA programs
  - minimizing the number of round trip times for good performance
    - many research results exist already

Conclusions

- Future cloud applications need strong consistency while tolerating failures beyond simple crash failures
- But the cost should be no more than f+1
  - economical considerations
  - but also ease of finding sources of diversity
- Requires a deep understanding in what assumptions (and absence of assumptions) are realistic to make