Overview

- Authors present Query/Update (quorum based) protocol for making a system Byzantine Fault-Tolerant that can be scalable in presence of faults
- Other protocols have degraded performance if more faults are tolerated
- Q/U only decreases performance (req/sec) in 36% vs 83% of replicated state machine
Efficiency and scalability

- Q/U protocol requires $5b+1$ servers to tolerate $b$ Byzantine faulty servers.
- Most agreement based approaches require $3b+1$ servers.
- Servers costs are declining but failure costs not.
- Agreement based require servers to process all requests, quorum not.

Query – Update Protocol (1)

- Assumptions
  - Duration, transmission delays, execution rates not considered
  - Clients and servers may be Byzantine
  - Cryptography works
  - Servers have permanent storage, recoverable in case of a crash
- Server types:
  - Benign: work according to specs except when they crash
  - Malevolent: out of specs, non crash behavior
  - Faulty: crash and does not recover, crash recovers constantly or malevolent
Query – Update Protocol (2)

- Quorum: subset
- Preferred Quorum: default
- Clients
- Servers
- Objects: queries (read only) and updates (modify)
- Replica History: version, time stamp

Servers validate OHS with authenticators HMAC.
Servers do not exchange replica histories with one another.
Using timestamps, servers work on the latest version of the object.
*When client receives a quorum of success responses, operation is done*
Server crash: client probes some other servers.
Servers may need to sync from hosts servers.
b+1 responses are needed for validation.

Query – Update Protocol (3)
How many servers in a quorum?

- Smallest possible threshold quorum.
- Order = number of servers replying
- q = quorum size
- t = number of faulty servers
- b = number of malevolent servers
- r = t + b + 1
- Complete quorum if q ≤ Order

Evaluation

Fault Scalability

![Graph showing fault scalability](graph.png)
Evaluation (2)

Scalability and Authenticators

Conclusions

- Q/U protocol supports implementation of arbitrary services that tolerate Byzantine failures in clients and servers.
- Prototype has good performance even if number of faults tolerated increase.
- Prototype is useful for creating substantial services.
Weaknesses

- Authors made 5 repetitions of the experiments. This may not be enough to perform an adequate statistical validation.
- Authors are supposed to work with Byzantine failures, but they never mention how they made the failures so they can be analyzed as Byzantine.

ANY QUESTIONS?